BigML Documentation

Release 4.21.2

The BigML Team

Apr 13, 2019

Contents

BigML makes machine learning easy by taking care of the details required to add data-driven decisions and predictive power to your company. Unlike other machine learning services, BigML creates beautiful predictive models that can be easily understood and interacted with.

These BigML Python bindings allow you to interact with BigML.io, the API for BigML. You can use it to easily create, retrieve, list, update, and delete BigML resources (i.e., sources, datasets, models and, predictions).

This module is licensed under the Apache License, Version 2.0.

CHAPTER 1

Support

Please report problems and bugs to our BigML.io issue tracker.

Discussions about the different bindings take place in the general BigML mailing list. Or join us in our Campfire chatroom.

CHAPTER 2

Requirements

Python 2.7 and Python 3 are currently supported by these bindings.

The basic third-party dependencies are the requests, poster, unidecode and requests-toolbelt libraries. These libraries are automatically installed during the setup. Support for Google App Engine has been added as of version 3.0.0, using the *urlfetch* package instead of *requests*.

The bindings will also use `simplejson` if you happen to have it installed, but that is optional: we fall back to Python's built-in JSON libraries is `simplejson` is not found.

Additional numpy and scipy libraries are needed in case you want to use local predictions for regression models (including the error information) using proportional missing strategy. As these are quite heavy libraries and they are so seldom used, they are not included in the automatic installation dependencies. The test suite includes some tests that will need these libraries to be installed.

Also in order to use local *Topic Model* predictions, you will need to install pystemmer. Using the *pip install* command for this library can produce an error if your system lacks the correct developer tools to compile it. In Windows, the error message will include a link pointing to the needed Visual Studio version and in OSX you'll need to install the Xcode developer tools.

CHAPTER 3

Installation

To install the latest stable release with pip

```
$ pip install bigml
```

You can also install the development version of the bindings directly from the Git repository

```
$ pip install -e git://github.com/bigmlcom/python.git#egg=bigml_python
```

CHAPTER 4

Importing the module

To import the module:

```
import bigml.api
```

Alternatively you can just import the BigML class:

```
from bigml.api import BigML
```

CHAPTER 5

Authentication

All the requests to BigML.io must be authenticated using your username and API key and are always transmitted over HTTPS.

This module will look for your username and API key in the environment variables `BIGML_USERNAME` and `BIGML_API_KEY` respectively. You can add the following lines to your `.bashrc` or `.bash_profile` to set those variables automatically when you log in:

```
export BIGML_USERNAME=myusername
export BIGML_API_KEY=ae579e7e53fb9abd646a6ff8aa99d4afe83ac291
```

With that environment set up, connecting to BigML is a breeze:

```
from bigml.api import BigML
api = BigML()
```

Otherwise, you can initialize directly when instantiating the BigML class as follows:

```
api = BigML('myusername', 'ae579e7e53fb9abd646a6ff8aa99d4afe83ac291')
```

These credentials will allow you to manage any resource in your user environment.

In BigML a user can also work for an `organization`. In this case, the organization administrator should previously assign permissions for the user to access one or several particular projects in the organization. Once permissions are granted, the user can work with resources in a project according to his permission level by creating a special constructor for each project. The connection constructor in this case should include the `project ID`:

```
api = BigML('myusername', 'ae579e7e53fb9abd646a6ff8aa99d4afe83ac291',
            project='project/53739b98d994972da7001d4a')
```

If the project used in a connection object does not belong to an existing organization but is one of the projects under the user's account, all the resources created or updated with that connection will also be assigned to the specified project.

When the resource to be managed is a `project` itself, the connection needs to include the corresponding``organization ID``:

```
api = BigML('myusername', 'ae579e7e53fb9abd646a6ff8aa99d4afe83ac291',
            organization='organization/53739b98d994972da7025d4a')
```

CHAPTER 6

Alternative domains

The main public domain for the API service is `bigml.io`, but there are some alternative domains, either for Virtual Private Cloud setups or the australian subdomain (`au.bigml.io`). You can change the remote server domain to the VPC particular one by either setting the `BIGML_DOMAIN` environment variable to your VPC subdomain:

```
export BIGML_DOMAIN=my_VPC.bigml.io
```

or setting it when instantiating your connection:

```
api = BigML(domain="my_VPC.bigml.io")
```

The corresponding SSL REST calls will be directed to your private domain henceforth.

You can also set up your connection to use a particular PredictServer only for predictions. In order to do so, you'll need to specify a `Domain` object, where you can set up the general domain name as well as the particular prediction domain name.

```
from bigml.domain import Domain
from bigml.api import BigML

domain_info = Domain(prediction_domain="my_prediction_server.bigml.com",
                     prediction_protocol="http")

api = BigML(domain=domain_info)
```

Finally, you can combine all the options and change both the general domain server, and the prediction domain server.

```
from bigml.domain import Domain
from bigml.api import BigML
domain_info = Domain(domain="my_VPC.bigml.io",
                     prediction_domain="my_prediction_server.bigml.com",
                     prediction_protocol="https")

api = BigML(domain=domain_info)
```

Some arguments for the Domain constructor are more unsual, but they can also be used to set your special service endpoints:

- protocol (string) Protocol for the service (when different from HTTPS)
- verify (boolean) Sets on/off the SSL verification
- prediction_verify (boolean) Sets on/off the SSL verification for the prediction server (when different from the general SSL verification)

Note that the previously existing `dev_mode` flag:

```
api = BigML(dev_mode=True)
```

that caused the connection to work with the Sandbox `Development Environment` has been **deprecated** because this environment does not longer exist. The existing resources that were previously created in this environment have been moved to a special project in the now unique `Production Environment`, so this flag is no longer needed to work with them.

CHAPTER 7

Quick Start

Imagine that you want to use this csv file containing the Iris flower dataset to predict the species of a flower whose `petal length` is `2.45` and whose `petal width` is `1.75`. A preview of the dataset is shown below. It has 4 numeric fields: `sepal length`, `sepal width`, `petal length`, `petal width` and a categorical field: `species`. By default, BigML considers the last field in the dataset as the objective field (i.e., the field that you want to generate predictions for).

```
sepal length,sepal width,petal length,petal width,species
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
...
5.8,2.7,3.9,1.2,Iris-versicolor
6.0,2.7,5.1,1.6,Iris-versicolor
5.4,3.0,4.5,1.5,Iris-versicolor
...
6.8,3.0,5.5,2.1,Iris-virginica
5.7,2.5,5.0,2.0,Iris-virginica
5.8,2.8,5.1,2.4,Iris-virginica
```

You can easily generate a prediction following these steps:

```
from bigml.api import BigML

api = BigML()

source = api.create_source('./data/iris.csv')
dataset = api.create_dataset(source)
model = api.create_model(dataset)
prediction = api.create_prediction(model, \
    {"petal width": 1.75, "petal length": 2.45})
```

You can then print the prediction using the `pprint` method:

```
>>> api.pprint(prediction)
species for {"petal width": 1.75, "petal length": 2.45} is Iris-setosa
```

The `iris` dataset has a small number of instances, and usually will be instantly created, so the `api.create_` calls will probably return the finished resources outright. As BigML's API is asynchronous, in general you will need to ensure that objects are finished before using them by using `api.ok`.

```
from bigml.api import BigML

api = BigML()

source = api.create_source('./data/iris.csv')
api.ok(source)
dataset = api.create_dataset(source)
api.ok(dataset)
model = api.create_model(dataset)
api.ok(model)
prediction = api.create_prediction(model, \
    {"petal width": 1.75, "petal length": 2.45})
```

Note that the prediction call is not followed by the `api.ok` method. Predictions are so quick to be generated that, unlike the rest of resouces, will be generated synchronously as a finished object.

The example assumes that your objective field (the one you want to predict) is the last field in the dataset. If that's not he case, you can explicitly set the name of this field in the creation call using the `objective_field` argument:

```
from bigml.api import BigML

api = BigML()

source = api.create_source('./data/iris.csv')
api.ok(source)
dataset = api.create_dataset(source)
api.ok(dataset)
model = api.create_model(dataset, {"objective_field": "species"})
api.ok(model)
prediction = api.create_prediction(model, \
    {'sepal length': 5, 'sepal width': 2.5})
```

You can also generate an evaluation for the model by using:

```
test_source = api.create_source('./data/test_iris.csv')
api.ok(test_source)
test_dataset = api.create_dataset(test_source)
api.ok(test_dataset)
evaluation = api.create_evaluation(model, test_dataset)
api.ok(evaluation)
```

If you set the `storage` argument in the `api` instantiation:

```
api = BigML(storage='./storage')
```

all the generated, updated or retrieved resources will be automatically saved to the chosen directory.

Alternatively, you can use the `export` method to explicitly download the JSON information that describes any of your resources in BigML to a particular file:

```
api.export('model/5acea49a08b07e14b9001068',
           filename="my_dir/my_model.json")
```

This example downloads the JSON for the model and stores it in the `my_dir/my_model.json` file.

In the case of models that can be represented in a *PMML* syntax, the export method can be used to produce the corresponding *PMML* file.

```
api.export('model/5acea49a08b07e14b9001068',
           filename="my_dir/my_model.pmml",
           pmml=True)
```

You can also retrieve the last resource with some previously given tag:

```
api.export_last("foo",
                resource_type="ensemble",
                filename="my_dir/my_ensemble.json")
```

which selects the last ensemble that has a `foo` tag. This mechanism can be specially useful when retrieving retrained models that have been created with a shared unique keyword as tag.

For a descriptive overview of the steps that you will usually need to follow to model your data and obtain predictions, please see the basic Workflow sketch document. You can also check other simple examples in the following documents:

- model 101
- logistic regression 101
- linear regression 101
- ensemble 101
- cluster 101
- anomaly detector 101
- association 101
- topic model 101
- deepnet 101
- time series 101

CHAPTER 8

Fields Structure

BigML automatically generates idenfiers for each field. To see the fields and the ids and types that have been assigned to a source you can use `get_fields`:

```
>>> source = api.get_source(source)
>>> api.pprint(api.get_fields(source))
{   u'000000': {   u'column_number': 0,
                   u'name': u'sepal length',
                   u'optype': u'numeric'},
    u'000001': {   u'column_number': 1,
                   u'name': u'sepal width',
                   u'optype': u'numeric'},
    u'000002': {   u'column_number': 2,
                   u'name': u'petal length',
                   u'optype': u'numeric'},
    u'000003': {   u'column_number': 3,
                   u'name': u'petal width',
                   u'optype': u'numeric'},
    u'000004': {   u'column_number': 4,
                   u'name': u'species',
                   u'optype': u'categorical'}}
```

When the number of fields becomes very large, it can be useful to exclude or filter them. This can be done using a query string expression, for instance:

```
>>> source = api.get_source(source, "limit=10&order_by=name")
```

would include in the retrieved dictionary the first 10 fields sorted by name.

To handle the field structure you can use the `Fields` class. See the *Fields* section.

CHAPTER 9

Dataset

If you want to get some basic statistics for each field you can retrieve the `fields` from the dataset as follows to get a dictionary keyed by field id:

```
>>> dataset = api.get_dataset(dataset)
>>> api.pprint(api.get_fields(dataset))
{   u'000000': {   u'column_number': 0,
                   u'datatype': u'double',
                   u'name': u'sepal length',
                   u'optype': u'numeric',
                   u'summary': {   u'maximum': 7.9,
                                   u'median': 5.77889,
                                   u'minimum': 4.3,
                                   u'missing_count': 0,
                                   u'population': 150,
                                   u'splits': [   4.51526,
                                                  4.67252,
                                                  4.81113,

                     [... snip ... ]


    u'000004': {   u'column_number': 4,
                   u'datatype': u'string',
                   u'name': u'species',
                   u'optype': u'categorical',
                   u'summary': {   u'categories': [   [   u'Iris-versicolor',
                                                          50],
                                                      [u'Iris-setosa', 50],
                                                      [   u'Iris-virginica',
                                                          50]],
                                   u'missing_count': 0}}}
```

The field filtering options are also available using a query string expression, for instance:

```
>>> dataset = api.get_dataset(dataset, "limit=20")
```

limits the number of fields that will be included in `dataset` to 20.

CHAPTER 10

Model

One of the greatest things about BigML is that the models that it generates for you are fully white-boxed. To get the explicit tree-like predictive model for the example above:

```
>>> model = api.get_model(model)
>>> api.pprint(model['object']['model']['root'])
{u'children': [
  {u'children': [
    {u'children': [{u'count': 38,
                    u'distribution': [[u'Iris-virginica', 38]],
                    u'output': u'Iris-virginica',
                    u'predicate': {u'field': u'000002',
                    u'operator': u'>',
                    u'value': 5.05}},
                    u'children': [

                        [ ... ]

                       {u'count': 50,
                        u'distribution': [[u'Iris-setosa', 50]],
                        u'output': u'Iris-setosa',
                        u'predicate': {u'field': u'000002',
                                       u'operator': u'<=',
                                       u'value': 2.45}}]},
                    {u'count': 150,
                     u'distribution': [[u'Iris-virginica', 50],
                                       [u'Iris-versicolor', 50],
                                       [u'Iris-setosa', 50]],
                     u'output': u'Iris-virginica',
                     u'predicate': True}]}}}
```

(Note that we have abbreviated the output in the snippet above for readability: the full predictive model you'll get is going to contain much more details).

Again, filtering options are also available using a query string expression, for instance:

```
>>> model = api.get_model(model, "limit=5")
```

limits the number of fields that will be included in `model` to 5.

CHAPTER 11

Evaluation

The predictive performance of a model can be measured using many different measures. In BigML these measures can be obtained by creating evaluations. To create an evaluation you need the id of the model you are evaluating and the id of the dataset that contains the data to be tested with. The result is shown as:

```
>>> evaluation = api.get_evaluation(evaluation)
>>> api.pprint(evaluation['object']['result'])
{   'class_names': ['0', '1'],
    'mode': {   'accuracy': 0.9802,
                'average_f_measure': 0.495,
                'average_phi': 0,
                'average_precision': 0.5,
                'average_recall': 0.4901,
                'confusion_matrix': [[99, 0], [2, 0]],
                'per_class_statistics': [   {   'accuracy': 0.9801980198019802,
                                                'class_name': '0',
                                                'f_measure': 0.99,
                                                'phi_coefficient': 0,
                                                'precision': 1.0,
                                                'present_in_test_data': True,
                                                'recall': 0.9801980198019802},
                                            {   'accuracy': 0.9801980198019802,
                                                'class_name': '1',
                                                'f_measure': 0,
                                                'phi_coefficient': 0,
                                                'precision': 0.0,
                                                'present_in_test_data': True,
                                                'recall': 0}]},
    'model': {   'accuracy': 0.9901,
                 'average_f_measure': 0.89746,
                 'average_phi': 0.81236,
                 'average_precision': 0.99495,
                 'average_recall': 0.83333,
                 'confusion_matrix': [[98, 1], [0, 2]],
                 'per_class_statistics': [   {   'accuracy': 0.9900990099009901,
```

(continues on next page)

(continued from previous page)

```
                                                'class_name': '0',
                                                'f_measure': 0.9949238578680203,
                                                'phi_coefficient': 0.
↪8123623944599232,
                                                'precision': 0.98989898989899,
                                                'present_in_test_data': True,
                                                'recall': 1.0},
                                            {   'accuracy': 0.9900990099009901,
                                                'class_name': '1',
                                                'f_measure': 0.8,
                                                'phi_coefficient': 0.
↪8123623944599232,
                                                'precision': 1.0,
                                                'present_in_test_data': True,
                                                'recall': 0.6666666666666666}]},
    'random': {   'accuracy': 0.50495,
                  'average_f_measure': 0.36812,
                  'average_phi': 0.13797,
                  'average_precision': 0.74747,
                  'average_recall': 0.51923,
                  'confusion_matrix': [[49, 50], [0, 2]],
                  'per_class_statistics': [   {   'accuracy': 0.504950495049505,
                                                  'class_name': '0',
                                                  'f_measure': 0.6621621621621622,
                                                  'phi_coefficient': 0.
↪1379728923974526,
                                                  'precision': 0.494949494949495,
                                                  'present_in_test_data': True,
                                                  'recall': 1.0},
                                              {   'accuracy': 0.504950495049505,
                                                  'class_name': '1',
                                                  'f_measure': 0.07407407407407407,
                                                  'phi_coefficient': 0.
↪1379728923974526,
                                                  'precision': 1.0,
                                                  'present_in_test_data': True,
                                                  'recall': 0.038461538461538464}]}}
```

where two levels of detail are easily identified. For classifications, the first level shows these keys:

- **class_names**: A list with the names of all the categories for the objective field (i.e., all the classes)
- **mode**: A detailed result object. Measures of the performance of the classifier that predicts the mode class for all the instances in the dataset
- **model**: A detailed result object.
- **random**: A detailed result object. Measures the performance of the classifier that predicts a random class for all the instances in the dataset.

and the detailed result objects include `accuracy`, `average_f_measure`, `average_phi`, `average_precision`, `average_recall`, `confusion_matrix` and `per_class_statistics`.

For regressions first level will contain these keys:

- **mean**: A detailed result object. Measures the performance of the model that predicts the mean for all the instances in the dataset.
- **model**: A detailed result object.

- **random**: A detailed result object. Measures the performance of the model that predicts a random class for all the instances in the dataset.

where the detailed result objects include `mean_absolute_error`, `mean_squared_error` and `r_squared` (refer to developers documentation for more info on the meaning of these measures.

CHAPTER 12

Cluster

For unsupervised learning problems, the cluster is used to classify in a limited number of groups your training data. The cluster structure is defined by the centers of each group of data, named centroids, and the data enclosed in the group. As for in the model's case, the cluster is a white-box resource and can be retrieved as a JSON:

```
>>> cluster = api.get_cluster(cluster)
>>> api.pprint(cluster['object'])
{   'balance_fields': True,
    'category': 0,
    'cluster_datasets': {   '000000': '', '000001': '', '000002': ''},
    'cluster_datasets_ids': {   '000000': '53739b9ae4b0dad82b0a65e6',
                                '000001': '53739b9ae4b0dad82b0a65e7',
                                '000002': '53739b9ae4b0dad82b0a65e8'},
    'cluster_seed': '2c249dda00fbf54ab4cdd850532a584f286af5b6',
    'clusters': {   'clusters': [   {   'center': {   '000000': 58.5,
                                                      '000001': 26.8314,
                                                      '000002': 44.27907,
                                                      '000003': 14.37209},
                                        'count': 56,
                                        'distance': {   'bins': [   [   0.69602,
                                                                        2],
                                                                    [ ... ]
                                                                    [   3.77052,
                                                                        1]],
                                                        'maximum': 3.77052,
                                                        'mean': 1.61711,
                                                        'median': 1.52146,
                                                        'minimum': 0.69237,
                                                        'population': 56,
                                                        'standard_deviation': 0.6161,
                                                        'sum': 90.55805,
                                                        'sum_squares': 167.31926,
                                                        'variance': 0.37958},
                                        'id': '000000',
                                        'name': 'Cluster 0'},
```

(continues on next page)

(continued from previous page)

```
                                    {   'center': {   '000000': 50.06,
                                                      '000001': 34.28,
                                                      '000002': 14.62,
                                                      '000003': 2.46},
                                        'count': 50,
                                        'distance': {   'bins': [   [   0.16917,
                                                                        1],
                                                                    [ ... ]
                                                                    [   4.94699,
                                                                        1]],
                                                        'maximum': 4.94699,
                                                        'mean': 1.50725,
                                                        'median': 1.3393,
                                                        'minimum': 0.16917,
                                                        'population': 50,
                                                        'standard_deviation': 1.00994,
                                                        'sum': 75.36252,
                                                        'sum_squares': 163.56918,
                                                        'variance': 1.01998},
                                        'id': '000001',
                                        'name': 'Cluster 1'},
                                    {   'center': {   '000000': 68.15625,
                                                      '000001': 31.25781,
                                                      '000002': 55.48438,
                                                      '000003': 19.96875},
                                        'count': 44,
                                        'distance': {   'bins': [   [   0.36825,
                                                                        1],
                                                                    [ ... ]
                                                                    [   3.87216,
                                                                        1]],
                                                        'maximum': 3.87216,
                                                        'mean': 1.67264,
                                                        'median': 1.63705,
                                                        'minimum': 0.36825,
                                                        'population': 44,
                                                        'standard_deviation': 0.78905,
                                                        'sum': 73.59627,
                                                        'sum_squares': 149.87194,
                                                        'variance': 0.6226},
                                        'id': '000002',
                                        'name': 'Cluster 2'}],
                    'fields': {   '000000': {   'column_number': 0,
                                                'datatype': 'int8',
                                                'name': 'sepal length',
                                                'optype': 'numeric',
                                                'order': 0,
                                                'preferred': True,
                                                'summary': {   'bins': [   [   43.75,
                                                                               4],
                                                                           [ ... ]
                                                                           [   79,
                                                                               1]],
                                                               'maximum': 79,
                                                               'mean': 58.43333,
                                                               'median': 57.7889,
                                                               'minimum': 43,
```

(continues on next page)

(continued from previous page)

```
                                                                     'missing_count': 0,
                                                                     'population': 150,
                                                                     'splits': [   45.15258,
                                                                                   46.72525,
                                                                                72.04226,
                                                                                   76.
↪47461],
                                                                     'standard_deviation':␣
↪8.28066,
                                                                     'sum': 8765,
                                                                     'sum_squares': 522385,
                                                                     'variance': 68.56935}},
                                                                      [ ... ]
                                                                                   [   25,
                                                                                       3]],
                                                                     'maximum': 25,
                                                                     'mean': 11.99333,
                                                                     'median': 13.28483,
                                                                     'minimum': 1,
                                                                     'missing_count': 0,
                                                                     'population': 150,
                                                                     'standard_deviation':␣
↪7.62238,
                                                                     'sum': 1799,
                                                                     'sum_squares': 30233,
                                                                     'variance': 58.10063}}}
↪},
    'code': 202,
    'columns': 4,
    'created': '2014-05-14T16:36:40.993000',
    'credits': 0.017578125,
    'credits_per_prediction': 0.0,
    'dataset': 'dataset/53739b88c8db63122b000411',
    'dataset_field_types': {   'categorical': 1,
                               'datetime': 0,
                               'numeric': 4,
                               'preferred': 5,
                               'text': 0,
                               'total': 5},
    'dataset_status': True,
    'dataset_type': 0,
    'description': '',
    'excluded_fields': ['000004'],
    'field_scales': None,
    'fields_meta': {   'count': 4,
                       'limit': 1000,
                       'offset': 0,
                       'query_total': 4,
                       'total': 4},
    'input_fields': ['000000', '000001', '000002', '000003'],
    'k': 3,
    'locale': 'es-ES',
    'max_columns': 5,
    'max_rows': 150,
    'name': 'my iris',
    'number_of_batchcentroids': 0,
    'number_of_centroids': 0,
```

(continues on next page)

(continued from previous page)

```
    'number_of_public_centroids': 0,
    'out_of_bag': False,
    'price': 0.0,
    'private': True,
    'range': [1, 150],
    'replacement': False,
    'resource': 'cluster/53739b98d994972da7001de9',
    'rows': 150,
    'sample_rate': 1.0,
    'scales': {   '000000': 0.22445382597655375,
                  '000001': 0.4264213814821549,
                  '000002': 0.10528680248949522,
                  '000003': 0.2438379900517961},
    'shared': False,
    'size': 4608,
    'source': 'source/53739b24d994972da7001ddd',
    'source_status': True,
    'status': {   'code': 5,
                  'elapsed': 1009,
                  'message': 'The cluster has been created',
                  'progress': 1.0},
    'subscription': True,
    'tags': [],
    'updated': '2014-05-14T16:40:26.234728',
    'white_box': False}
```

(Note that we have abbreviated the output in the snippet above for readability: the full predictive cluster you'll get is going to contain much more details).

CHAPTER 13

Anomaly detector

For anomaly detection problems, BigML anomaly detector uses iforest as an unsupervised kind of model that detects anomalous data in a dataset. The information it returns encloses a *top_anomalies* block that contains a list of the most anomalous points. For each, we capture a *score* from 0 to 1. The closer to 1, the more anomalous. We also capture the *row* which gives values for each field in the order defined by *input_fields*. Similarly we give a list of *importances* which match the *row* values. These importances tell us which values contributed most to the anomaly score. Thus, the structure of an anomaly detector is similar to:

```
{   'category': 0,
    'code': 200,
    'columns': 14,
    'constraints': False,
    'created': '2014-09-08T18:51:11.893000',
    'credits': 0.11653518676757812,
    'credits_per_prediction': 0.0,
    'dataset': 'dataset/540dfa9d9841fa5c88000765',
    'dataset_field_types': {   'categorical': 21,
                               'datetime': 0,
                               'numeric': 21,
                               'preferred': 14,
                               'text': 0,
                               'total': 42},
    'dataset_status': True,
    'dataset_type': 0,
    'description': '',
    'excluded_fields': [],
    'fields_meta': {   'count': 14,
                       'limit': 1000,
                       'offset': 0,
                       'query_total': 14,
                       'total': 14},
    'forest_size': 128,
    'input_fields': [   '000004',
                        '000005',
                        '000009',
```

(continues on next page)

(continued from previous page)

```
                          '000016',
                          '000017',
                          '000018',
                          '000019',
                          '00001e',
                          '00001f',
                          '000020',
                          '000023',
                          '000024',
                          '000025',
                          '000026'],
    'locale': 'en_US',
    'max_columns': 42,
    'max_rows': 200,
    'model': {   'fields': {   '000004': {   'column_number': 4,
                                             'datatype': 'int16',
                                             'name': 'src_bytes',
                                             'optype': 'numeric',
                                             'order': 0,
                                             'preferred': True,
                                             'summary': {   'bins': [   [   143,
                                                                            2],
                                                                        ...
                                                                        [   370,
                                                                            2]],
                                                            'maximum': 370,
                                                            'mean': 248.235,
                                                            'median': 234.57157,
                                                            'minimum': 141,
                                                            'missing_count': 0,
                                                            'population': 200,
                                                            'splits': [   159.92462,
                                                                          173.73312,
                                                                          188,
                                                                          ...
                                                                          339.55228],
                                                            'standard_deviation': 49.
↪39869,
                                                            'sum': 49647,
                                                            'sum_squares': 12809729,
                                                            'variance': 2440.23093}},
                               '000005': {   'column_number': 5,
                                             'datatype': 'int32',
                                             'name': 'dst_bytes',
                                             'optype': 'numeric',
                                             'order': 1,
                                             'preferred': True,
                                              ...
                                                            'sum': 1030851,
                                                            'sum_squares':␣
↪22764504759,
                                                            'variance': 87694652.
↪45224}},
                               '000009': {   'column_number': 9,
                                             'datatype': 'string',
                                             'name': 'hot',
                                             'optype': 'categorical',
```

(continues on next page)

(continued from previous page)

```
                                            'order': 2,
                                            'preferred': True,
                                            'summary': {   'categories': [   [   '0',
                                                                                ␣
↪199],
                                                                             [   '1',
                                                                                 1]],
                                                           'missing_count': 0},
                                            'term_analysis': {   'enabled': True}},
                              '000016': {   'column_number': 22,
                                            'datatype': 'int8',
                                            'name': 'count',
                                            'optype': 'numeric',
                                            'order': 3,
                                            'preferred': True,
                                                           ...
                                                           'population': 200,
                                                           'standard_deviation': 5.
↪42421,

                                                           'sum': 1351,
                                                           'sum_squares': 14981,
                                                           'variance': 29.42209}},
                              '000017': { ... }}},
                'kind': 'iforest',
                'mean_depth': 12.314174107142858,
                'top_anomalies': [   {   'importance': [   0.06768,
                                                           0.01667,
                                                           0.00081,
                                                           0.02437,
                                                           0.04773,
                                                           0.22197,
                                                           0.18208,
                                                           0.01868,
                                                           0.11855,
                                                           0.01983,
                                                           0.01898,
                                                           0.05306,
                                                           0.20398,
                                                           0.00562],
                                         'row': [   183.0,
                                                    8654.0,
                                                    '0',
                                                    4.0,
                                                    4.0,
                                                    0.25,
                                                    0.25,
                                                    0.0,
                                                    123.0,
                                                    255.0,
                                                    0.01,
                                                    0.04,
                                                    0.01,
                                                    0.0],
                                         'score': 0.68782},
                                     {   'importance': [   0.05645,
                                                           0.02285,
                                                           0.0015,
```

(continues on next page)

(continued from previous page)

```
                                                                        0.05196,
                                                                        0.04435,
                                                                        0.0005,
                                                                        0.00056,
                                                                        0.18979,
                                                                        0.12402,
                                                                        0.23671,
                                                                        0.20723,
                                                                        0.05651,
                                                                        0.00144,
                                                                        0.00612],
                                                  'row': [   212.0,
                                                             1940.0,
                                                             '0',
                                                             1.0,
                                                             2.0,
                                                             0.0,
                                                             0.0,
                                                             1.0,
                                                             1.0,
                                                             69.0,
                                                             1.0,
                                                             0.04,
                                                             0.0,
                                                             0.0],
                                                  'score': 0.6239},
                                                  ...],
                         'trees': [   {   'root': {   'children': [   {   'children': [   {
↪'children': [   {   'children': [   {   'children':
[   {   'population': 1,
                                                                                        ␣
↪                                         'predicates': [   {   'field': '00001f',
                                                                                        ␣
↪                                                               'op': '>',
                                                                                        ␣
↪                                                               'value': 35.54357}]},

...
                                                                                        ␣
↪                                     {   'population': 1,
                                                                                        ␣
↪                                         'predicates': [   {   'field': '00001f',
                                                                                        ␣
↪                                                               'op': '<=',
                                                                                        ␣
↪                                                               'value': 35.54357}]}],
                                                                                        ␣
↪                   'population': 2,
                                                                                        ␣
↪                   'predicates': [   {   'field': '000005',
                                                                                        ␣
↪                                         'op': '<=',
                                                                                        ␣
↪                                         'value': 1385.5166}]}],

↪'population': 3,

↪'predicates': [   {   'field': '000020',
```

(continues on next page)

(continued from previous page)

```
                                                                                    ␣
↪                         'op': '<=',
                                                                                    ␣
↪                         'value': 65.14308},
                                                                                    ␣
↪                     {   'field': '000019',
                                                                                    ␣
↪                         'op': '=',
                                                                                    ␣
↪                         'value': 0}]}],
                                                                   'population': 105,
                                                                   'predicates': [   {␣
↪  'field': '000017',
                                                                                    ␣
↪  'op': '<=',
                                                                                    ␣
↪  'value': 13.21754},
                                                                                   {␣
↪  'field': '000009',
                                                                                    ␣
↪  'op': 'in',
                                                                                    ␣
↪  'value': [   '0']}]}]}],
                                              'population': 126,
                                              'predicates': [   True,
                                                                {   'field': '000018',
                                                                    'op': '=',
                                                                    'value': 0}]},
                                  'training_mean_depth': 11.071428571428571}]},
    'name': "tiny_kdd's dataset anomaly detector",
    'number_of_batchscores': 0,
    'number_of_public_predictions': 0,
    'number_of_scores': 0,
    'out_of_bag': False,
    'price': 0.0,
    'private': True,
    'project': None,
    'range': [1, 200],
    'replacement': False,
    'resource': 'anomaly/540dfa9f9841fa5c8800076a',
    'rows': 200,
    'sample_rate': 1.0,
    'sample_size': 126,
    'seed': 'BigML',
    'shared': False,
    'size': 30549,
    'source': 'source/540dfa979841fa5c7f000363',
    'source_status': True,
    'status': {   'code': 5,
                  'elapsed': 32397,
                  'message': 'The anomaly detector has been created',
                  'progress': 1.0},
    'subscription': False,
    'tags': [],
    'updated': '2014-09-08T23:54:28.647000',
    'white_box': False}
```

Note that we have abbreviated the output in the snippet above for readability: the full anomaly detector you'll get is going to contain much more details).

The *trees* list contains the actual isolation forest, and it can be quite large usually. That's why, this part of the resource should only be included in downloads when needed. If you are only interested in other properties, such as *top_anomalies*, you'll improve performance by excluding it, using the *excluded=trees* query string in the API call:

```
anomaly = api.get_anomaly('anomaly/540dfa9f9841fa5c8800076a', \
                          query_string='excluded=trees')
```

Each node in an isolation tree can have multiple predicates. For the node to be a valid branch when evaluated with a data point, all of its predicates must be true.

CHAPTER 14

Samples

To provide quick access to your row data you can create a `sample`. Samples are in-memory objects that can be queried for subsets of data by limiting their size, the fields or the rows returned. The structure of a sample would be:

Samples are not permanent objects. Once they are created, they will be available as long as GETs are requested within periods smaller than a pre-established TTL (Time to Live). The expiration timer of a sample is reset every time a new GET is received.

If requested, a sample can also perform linear regression and compute Pearson's and Spearman's correlations for either one numeric field against all other numeric fields or between two specific numeric fields.

CHAPTER 15

Correlations

A `correlation` resource contains a series of computations that reflect the degree of dependence between the field set as objective for your predictions and the rest of fields in your dataset. The dependence degree is obtained by comparing the distributions in every objective and non-objective field pair, as independent fields should have probabilistic independent distributions. Depending on the types of the fields to compare, the metrics used to compute the correlation degree will be:

- for numeric to numeric pairs: Pearson's and Spearman's correlation coefficients.
- for numeric to categorical pairs: One-way Analysis of Variance, with the categorical field as the predictor variable.
- for categorical to categorical pairs: contingency table (or two-way table), Chi-square test of independence , and Cramer's V and Tschuprow's T coefficients.

An example of the correlation resource JSON structure is:

```
>>> from bigml.api import BigML
>>> api = BigML()
>>> correlation = api.create_correlation('dataset/55b7a6749841fa2500000d41')
>>> api.ok(correlation)
>>> api.pprint(correlation['object'])
{   u'category': 0,
    u'clones': 0,
    u'code': 200,
    u'columns': 5,
    u'correlations': {   u'correlations': [   {   u'name': u'one_way_anova',
                                                  u'result': {   u'000000': {   u'eta_
↪square': 0.61871,
                                                                                u'f_
↪ratio': 119.2645,
                                                                                u'p_
↪value': 0,
                                                                                u
↪'significant': [   True,
                                                                                    ␣
↪             True,
```

(continues on next page)

(continued from previous page)

```
                                                                                    ␣
↪              True]},
                                                                u'000001': {    u'eta_
↪square': 0.40078,
                                                                                u'f_
↪ratio': 49.16004,
                                                                                u'p_
↪value': 0,
                                                                                u
↪'significant': [   True,
                                                                                    ␣
↪              True,
                                                                                    ␣
↪              True]},
                                                                u'000002': {    u'eta_
↪square': 0.94137,
                                                                                u'f_
↪ratio': 1180.16118,
                                                                                u'p_
↪value': 0,
                                                                                u
↪'significant': [   True,
                                                                                    ␣
↪              True,
                                                                                    ␣
↪              True]},
                                                                u'000003': {    u'eta_
↪square': 0.92888,
                                                                                u'f_
↪ratio': 960.00715,
                                                                                u'p_
↪value': 0,
                                                                                u
↪'significant': [   True,
                                                                                    ␣
↪              True,
                                                                                    ␣
↪              True]}}}],
                        u'fields': {    u'000000': {    u'column_number': 0,
                                                        u'datatype': u'double',
                                                        u'idx': 0,
                                                        u'name': u'sepal length',
                                                        u'optype': u'numeric',
                                                        u'order': 0,
                                                        u'preferred': True,
                                                        u'summary': {   u'bins': [    [␣
↪  4.3,
                                                                                    ␣
↪  1],
                                                                                   [␣
↪  4.425,
                                                                                    ␣
↪  4],
...
                                                                                   [␣
↪  7.9,
                                                                                    ␣
↪  1]],
```

(continues on next page)

(continued from previous page)

```
                                                                      u'kurtosis': -
↪0.57357,
                                                                      u'maximum': 7.
↪9,
                                                                      u'mean': 5.
↪84333,
                                                                      u'median': 5.8,
                                                                      u'minimum': 4.
↪3,
                                                                      u'missing_count
↪': 0,
                                                                      u'population':␣
↪150,
                                                                      u'skewness': 0.
↪31175,
                                                                      u'splits': [  ␣
↪4.51526,
                                                                                    ␣
↪4.67252,
                                                                                    ␣
↪4.81113,
                                                                                    ␣
↪4.89582,
                                                                                    ␣
↪4.96139,
                                                                                    ␣
↪5.01131,
...
                                                                                    ␣
↪6.92597,
                                                                                    ␣
↪7.20423,
                                                                                    ␣
↪7.64746],
                                                                      u'standard_
↪deviation': 0.82807,
                                                                      u'sum': 876.5,
                                                                      u'sum_squares
↪': 5223.85,
                                                                      u'variance': 0.
↪68569}},
                                       u'000001': {   u'column_number': 1,
                                                      u'datatype': u'double',
                                                      u'idx': 1,
                                                      u'name': u'sepal width',
                                                      u'optype': u'numeric',
                                                      u'order': 1,
                                                      u'preferred': True,
                                                      u'summary': {   u'counts': [  ␣
↪[   2,
                                                                                    ␣
↪    1],
                                                                                    ␣
↪[   2.2,
...
                                       u'000004': {   u'column_number': 4,
                                                      u'datatype': u'string',
```

(continues on next page)

(continued from previous page)

```
                                                                u'idx': 4,
                                                                u'name': u'species',
                                                                u'optype': u'categorical',
                                                                u'order': 4,
                                                                u'preferred': True,
                                                                u'summary': {   u'categories':␣
↪[   [   u'Iris-setosa',
                                                                                                ␣
↪        50],
                                                                                                ␣
↪    [   u'Iris-versicolor',
                                                                                                ␣
↪        50],
                                                                                                ␣
↪    [   u'Iris-virginica',
                                                                                                ␣
↪        50]],
                                                                                u'missing_count
↪': 0},
                                                                u'term_analysis': {   u'enabled
↪': True}}},
                                u'significance_levels': [0.01, 0.05, 0.1]},
    u'created': u'2015-07-28T18:07:37.010000',
    u'credits': 0.017581939697265625,
    u'dataset': u'dataset/55b7a6749841fa2500000d41',
    u'dataset_status': True,
    u'dataset_type': 0,
    u'description': u'',
    u'excluded_fields': [],
    u'fields_meta': {   u'count': 5,
                        u'limit': 1000,
                        u'offset': 0,
                        u'query_total': 5,
                        u'total': 5},
    u'input_fields': [u'000000', u'000001', u'000002', u'000003'],
    u'locale': u'en_US',
    u'max_columns': 5,
    u'max_rows': 150,
    u'name': u"iris' dataset correlation",
    u'objective_field_details': {   u'column_number': 4,
                                    u'datatype': u'string',
                                    u'name': u'species',
                                    u'optype': u'categorical',
                                    u'order': 4},
    u'out_of_bag': False,
    u'price': 0.0,
    u'private': True,
    u'project': None,
    u'range': [1, 150],
    u'replacement': False,
    u'resource': u'correlation/55b7c4e99841fa24f20009bf',
    u'rows': 150,
    u'sample_rate': 1.0,
    u'shared': False,
    u'size': 4609,
    u'source': u'source/55b7a6729841fa24f100036a',
    u'source_status': True,
```

(continues on next page)

(continued from previous page)

```
    u'status': {   u'code': 5,
                   u'elapsed': 274,
                   u'message': u'The correlation has been created',
                   u'progress': 1.0},
    u'subscription': True,
    u'tags': [],
    u'updated': u'2015-07-28T18:07:49.057000',
    u'white_box': False}
```

Note that the output in the snippet above has been abbreviated. As you see, the `correlations` attribute contains the information about each field correlation to the objective field.

CHAPTER 16

Statistical Tests

A `statisticaltest` resource contains a series of tests that compare the distribution of data in each numeric field of a dataset to certain canonical distributions, such as the normal distribution or Benford's law distribution. Statistical test are useful in tasks such as fraud, normality, or outlier detection.

- Fraud Detection Tests:

Benford: This statistical test performs a comparison of the distribution of first significant digits (FSDs) of each value of the field to the Benford's law distribution. Benford's law applies to numerical distributions spanning several orders of magnitude, such as the values found on financial balance sheets. It states that the frequency distribution of leading, or first significant digits (FSD) in such distributions is not uniform. On the contrary, lower digits like 1 and 2 occur disproportionately often as leading significant digits. The test compares the distribution in the field to Bendford's distribution using a Chi-square goodness-of-fit test, and Cho-Gaines d test. If a field has a dissimilar distribution, it may contain anomalous or fraudulent values.

- Normality tests:

These tests can be used to confirm the assumption that the data in each field of a dataset is distributed according to a normal distribution. The results are relevant because many statistical and machine learning techniques rely on this assumption. Anderson-Darling: The Anderson-Darling test computes a test statistic based on the difference between the observed cumulative distribution function (CDF) to that of a normal distribution. A significant result indicates that the assumption of normality is rejected. Jarque-Bera: The Jarque-Bera test computes a test statistic based on the third and fourth central moments (skewness and kurtosis) of the data. Again, a significant result indicates that the normality assumption is rejected. Z-score: For a given sample size, the maximum deviation from the mean that would expected in a sampling of a normal distribution can be computed based on the 68-95-99.7 rule. This test simply reports this expected deviation and the actual deviation observed in the data, as a sort of sanity check.

- Outlier tests:

Grubbs: When the values of a field are normally distributed, a few values may still deviate from the mean distribution. The outlier tests reports whether at least one value in each numeric field differs significantly from the mean using Grubb's test for outliers. If an outlier is found, then its value will be returned.

The JSON structure for `statisticaltest` resources is similar to this one:

```
>>> statistical_test = api.create_statistical_test('dataset/55b7a6749841fa2500000d41')
>>> api.ok(statistical_test)
True
>>> api.pprint(statistical_test['object'])
{   u'category': 0,
    u'clones': 0,
    u'code': 200,
    u'columns': 5,
    u'created': u'2015-07-28T18:16:40.582000',
    u'credits': 0.017581939697265625,
    u'dataset': u'dataset/55b7a6749841fa2500000d41',
    u'dataset_status': True,
    u'dataset_type': 0,
    u'description': u'',
    u'excluded_fields': [],
    u'fields_meta': {   u'count': 5,
                        u'limit': 1000,
                        u'offset': 0,
                        u'query_total': 5,
                        u'total': 5},
    u'input_fields': [u'000000', u'000001', u'000002', u'000003'],
    u'locale': u'en_US',
    u'max_columns': 5,
    u'max_rows': 150,
    u'name': u"iris' dataset test",
    u'out_of_bag': False,
    u'price': 0.0,
    u'private': True,
    u'project': None,
    u'range': [1, 150],
    u'replacement': False,
    u'resource': u'statisticaltest/55b7c7089841fa25000010ad',
    u'rows': 150,
    u'sample_rate': 1.0,
    u'shared': False,
    u'size': 4609,
    u'source': u'source/55b7a6729841fa24f100036a',
    u'source_status': True,
    u'status': {   u'code': 5,
                   u'elapsed': 302,
                   u'message': u'The test has been created',
                   u'progress': 1.0},
    u'subscription': True,
    u'tags': [],
    u'statistical_tests': {   u'ad_sample_size': 1024,
                  u'fields': {   u'000000': {   u'column_number': 0,
                                                u'datatype': u'double',
                                                u'idx': 0,
                                                u'name': u'sepal length',
                                                u'optype': u'numeric',
                                                u'order': 0,
                                                u'preferred': True,
                                                u'summary': {   u'bins': [   [   4.3,
                                                                                 1],
                                                                             [   4.
↪425,
                                                                                 4],
```

(continues on next page)

(continued from previous page)

```
...
                                                                             [   7.9,
                                                                                 1]],
                                                                 u'kurtosis': -0.57357,
                                                                 u'maximum': 7.9,
                                                                 u'mean': 5.84333,
                                                                 u'median': 5.8,
                                                                 u'minimum': 4.3,
                                                                 u'missing_count': 0,
                                                                 u'population': 150,
                                                                 u'skewness': 0.31175,
                                                                 u'splits': [   4.
↪51526,
                                                                                4.
↪67252,
                                                                                4.
↪81113,
                                                                                4.
↪89582,
...
                                                                                7.
↪20423,
                                                                                7.
↪64746],
                                                                 u'standard_deviation
↪': 0.82807,
                                                                 u'sum': 876.5,
                                                                 u'sum_squares': 5223.
↪85,
                                                                 u'variance': 0.68569}}
↪,
...
                                 u'000004': {   u'column_number': 4,
                                                u'datatype': u'string',
                                                u'idx': 4,
                                                u'name': u'species',
                                                u'optype': u'categorical',
                                                u'order': 4,
                                                u'preferred': True,
                                                u'summary': {   u'categories': [   [ ␣
↪ u'Iris-setosa',
                                                                                     ␣
↪ 50],
                                                                                   [ ␣
↪ u'Iris-versicolor',
                                                                                     ␣
↪ 50],
                                                                                   [ ␣
↪ u'Iris-virginica',
                                                                                     ␣
↪ 50]],
                                                                u'missing_count': 0},
                                                u'term_analysis': {   u'enabled':␣
↪True}}},
                  u'fraud': [   {   u'name': u'benford',
                                    u'result': {   u'000000': {   u'chi_square': {   u
↪'chi_square_value': 506.39302,
```

(continues on next page)

(continued from previous page)

```
                                                                                          u
↪'p_value': 0,
                                                                                          u
↪'significant': [   True,
                                                                                          ␣
↪                   True,
                                                                                          ␣
↪                   True]},
                                                                      u'cho_gaines': {    u
↪'d_statistic': 7.124311073683573,
                                                                                          u
↪'significant': [   True,
                                                                                          ␣
↪                   True,
                                                                                          ␣
↪                   True]},
                                                                      u'distribution': [ ␣
↪ 0,
                                                                                          ␣
↪ 0,
                                                                                          ␣
↪ 0,
                                                                                          ␣
↪ 22,
                                                                                          ␣
↪ 61,
                                                                                          ␣
↪ 54,
                                                                                          ␣
↪ 13,
                                                                                          ␣
↪ 0,
                                                                                          ␣
↪ 0],
                                                                      u'negatives': 0,
                                                                      u'zeros': 0},
                                                        u'000001': {   u'chi_square': {   u
↪'chi_square_value': 396.76556,
                                                                                          u
↪'p_value': 0,
                                                                                          u
↪'significant': [   True,
                                                                                          ␣
↪                   True,
                                                                                          ␣
↪                   True]},
                                                                      u'cho_gaines': {    u
↪'d_statistic': 7.503503138331123,
                                                                                          u
↪'significant': [   True,
                                                                                          ␣
↪                   True,
                                                                                          ␣
↪                   True]},
                                                                      u'distribution': [ ␣
↪ 0,
                                                                                          ␣
↪ 57,
```

(continues on next page)

(continued from previous page)

```
                                                                                                    ␣
↪ 89,
                                                                                                    ␣
↪ 4,
                                                                                                    ␣
↪ 0,
                                                                                                    ␣
↪ 0,
                                                                                                    ␣
↪ 0,
                                                                                                    ␣
↪ 0,
                                                                                                    ␣
↪ 0],
                                                                              u'negatives': 0,
                                                                              u'zeros': 0},
                                                               u'000002': {   u'chi_square': {   u
↪'chi_square_value': 154.20728,
                                                                                                 u
↪'p_value': 0,
                                                                                                 u
↪'significant': [   True,
                                                                                                    ␣
↪                   True,
                                                                                                    ␣
↪                   True]},
                                                                              u'cho_gaines': {   u
↪'d_statistic': 3.9229974017266054,
                                                                                                 u
↪'significant': [   True,
                                                                                                    ␣
↪                   True,
                                                                                                    ␣
↪                   True]},
                                                                              u'distribution': [ ␣
↪ 50,
                                                                                                    ␣
↪ 0,
                                                                                                    ␣
↪ 11,
                                                                                                    ␣
↪ 43,
                                                                                                    ␣
↪ 35,
                                                                                                    ␣
↪ 11,
                                                                                                    ␣
↪ 0,
                                                                                                    ␣
↪ 0,
                                                                                                    ␣
↪ 0],
                                                                              u'negatives': 0,
                                                                              u'zeros': 0},
                                                               u'000003': {   u'chi_square': {   u
↪'chi_square_value': 111.4438,
                                                                                                 u
↪'p_value': 0,
```

(continues on next page)

(continued from previous page)

```
                                                                                    u
↪'significant': [   True,
                                                                                    ␣
↪                   True,
                                                                                    ␣
↪                   True]},
                                                                   u'cho_gaines': {   u
↪'d_statistic': 4.103257341299901,
                                                                                    u
↪'significant': [   True,
                                                                                    ␣
↪                   True,
                                                                                    ␣
↪                   True]},
                                                                   u'distribution': [ ␣
↪ 76,
                                                                                    ␣
↪ 58,
                                                                                    ␣
↪ 7,
                                                                                    ␣
↪ 7,
                                                                                    ␣
↪ 1,
                                                                                    ␣
↪ 1,
                                                                                    ␣
↪ 0,
                                                                                    ␣
↪ 0,
                                                                                    ␣
↪ 0],
                                                                   u'negatives': 0,
                                                                   u'zeros': 0}}}],
                  u'normality': [   {   u'name': u'anderson_darling',
                                        u'result': {   u'000000': {   u'p_value': 0.
↪02252,
                                                                      u'significant':␣
↪[   False,
                                                                                    ␣
↪    True,
                                                                                    ␣
↪    True]},
                                                       u'000001': {   u'p_value': 0.
↪02023,
                                                                      u'significant':␣
↪[   False,
                                                                                    ␣
↪    True,
                                                                                    ␣
↪    True]},
                                                       u'000002': {   u'p_value': 0,
                                                                      u'significant':␣
↪[   True,
                                                                                    ␣
↪    True,
                                                                                    ␣
↪    True]},
```

(continues on next page)

(continued from previous page)

```
                                                        u'000003': {   u'p_value': 0,
                                                                       u'significant':␣
↪[   True,
                                                                                      ␣
↪    True,
                                                                                      ␣
↪    True]}}},
                                     {   u'name': u'jarque_bera',
                                         u'result': {   u'000000': {   u'p_value': 0.
↪10615,
                                                                       u'significant':␣
↪[   False,
                                                                                      ␣
↪    False,
                                                                                      ␣
↪    False]},
                                                        u'000001': {   u'p_value': 0.
↪25957,
                                                                       u'significant':␣
↪[   False,
                                                                                      ␣
↪    False,
                                                                                      ␣
↪    False]},
                                                        u'000002': {   u'p_value': 0.
↪0009,
                                                                       u'significant':␣
↪[   True,
                                                                                      ␣
↪    True,
                                                                                      ␣
↪    True]},
                                                        u'000003': {   u'p_value': 0.
↪00332,
                                                                       u'significant':␣
↪[   True,
                                                                                      ␣
↪    True,
                                                                                      ␣
↪    True]}}},
                                     {   u'name': u'z_score',
                                         u'result': {   u'000000': {   u'expected_max_z
↪': 2.71305,
                                                                       u'max_z': 2.
↪48369},
                                                        u'000001': {   u'expected_max_z
↪': 2.71305,
                                                                       u'max_z': 3.
↪08044},
                                                        u'000002': {   u'expected_max_z
↪': 2.71305,
                                                                       u'max_z': 1.
↪77987},
                                                        u'000003': {   u'expected_max_z
↪': 2.71305,
                                                                       u'max_z': 1.
↪70638}}}],
```

(continues on next page)

(continued from previous page)

```
                  u'outliers': [   {   u'name': u'grubbs',
                                       u'result': {   u'000000': {   u'p_value': 1,
                                                                     u'significant':␣
↪[   False,
                                                                                    ␣
↪   False,
                                                                                    ␣
↪   False]},
                                                      u'000001': {   u'p_value': 0.
↪26555,
                                                                     u'significant':␣
↪[   False,
                                                                                    ␣
↪   False,
                                                                                    ␣
↪   False]},
                                                      u'000002': {   u'p_value': 1,
                                                                     u'significant':␣
↪[   False,
                                                                                    ␣
↪   False,
                                                                                    ␣
↪   False]},
                                                      u'000003': {   u'p_value': 1,
                                                                     u'significant':␣
↪[   False,
                                                                                    ␣
↪   False,
                                                                                    ␣
↪   False]}}}}],
                  u'significance_levels': [0.01, 0.05, 0.1]},
    u'updated': u'2015-07-28T18:17:11.829000',
    u'white_box': False}
```

Note that the output in the snippet above has been abbreviated. As you see, the `statistical_tests` attribute contains the `fraud``, ``normality` and `outliers` sections where the information for each field's distribution is stored.

CHAPTER 17

Logistic Regressions

A logistic regression is a supervised machine learning method for solving classification problems. Each of the classes in the field you want to predict, the objective field, is assigned a probability depending on the values of the input fields. The probability is computed as the value of a logistic function, whose argument is a linear combination of the predictors' values. You can create a logistic regression selecting which fields from your dataset you want to use as input fields (or predictors) and which categorical field you want to predict, the objective field. Then the created logistic regression is defined by the set of coefficients in the linear combination of the values. Categorical and text fields need some prior work to be modelled using this method. They are expanded as a set of new fields, one per category or term (respectively) where the number of occurrences of the category or term is store. Thus, the linear combination is made on the frequency of the categories or terms.

The JSON structure for a logistic regression is:

```
>>> api.pprint(logistic_regression['object'])
{   u'balance_objective': False,
    u'category': 0,
    u'code': 200,
    u'columns': 5,
    u'created': u'2015-10-09T16:11:08.444000',
    u'credits': 0.017581939697265625,
    u'credits_per_prediction': 0.0,
    u'dataset': u'dataset/561304f537203f4c930001ca',
    u'dataset_field_types': {   u'categorical': 1,
                                u'datetime': 0,
                                u'effective_fields': 5,
                                u'numeric': 4,
                                u'preferred': 5,
                                u'text': 0,
                                u'total': 5},
    u'dataset_status': True,
    u'description': u'',
    u'excluded_fields': [],
    u'fields_meta': {   u'count': 5,
                        u'limit': 1000,
                        u'offset': 0,
```

(continues on next page)

(continued from previous page)

```
                        u'query_total': 5,
                        u'total': 5},
    u'input_fields': [u'000000', u'000001', u'000002', u'000003'],
    u'locale': u'en_US',
    u'logistic_regression': {   u'bias': 1,
                                u'c': 1,
                                u'coefficients': [   [   u'Iris-virginica',
                                                         [   -1.7074433493289376,
                                                             -1.533662474502423,
                                                             2.47026986670851,
                                                             2.5567582221085563,
                                                             -1.2158200612711925]],
                                                     [   u'Iris-setosa',
                                                         [   0.41021712519841674,
                                                             1.464162165246765,
                                                             -2.26003266131107,
                                                             -1.0210350909174153,
                                                             0.26421852991732514]],
                                                     [   u'Iris-versicolor',
                                                         [   0.42702327817072505,
                                                             -1.611817241669904,
                                                             0.5763832839459982,
                                                             -1.4069842681625884,
                                                             1.0946877732663143]]],
                                u'eps': 1e-05,
                                u'fields': {   u'000000': {   u'column_number': 0,
                                                               u'datatype': u'double',
                                                               u'name': u'sepal length
↪',

                                                               u'optype': u'numeric',
                                                               u'order': 0,
                                                               u'preferred': True,
                                                               u'summary': {   u'bins
↪': [   [   4.3,
                                                                                               ␣
↪          1],
                                                                                               ␣
↪      [   4.425,
                                                                                               ␣
↪          4],
                                                                                               ␣
↪      [   4.6,
                                                                                               ␣
↪          4],
...
                                                                                               ␣
↪      [   7.9,
                                                                                               ␣
↪          1]],
                                                                               u
↪'kurtosis': -0.57357,
                                                                               u
↪'maximum': 7.9,
                                                                               u'mean
↪': 5.84333,
                                                                               u'median
↪': 5.8,
```

(continues on next page)

(continued from previous page)

```
                                                                                u
↪'minimum': 4.3,
                                                                                u
↪'missing_count': 0,
                                                                                u
↪'population': 150,
                                                                                u
↪'skewness': 0.31175,
                                                                                u'splits
↪': [   4.51526,
                                                                                       ␣
↪       4.67252,
                                                                                       ␣
↪       4.81113,
...
                                                                                       ␣
↪       6.92597,
                                                                                       ␣
↪       7.20423,
                                                                                       ␣
↪       7.64746],
                                                                                u
↪'standard_deviation': 0.82807,
                                                                                u'sum':␣
↪876.5,
                                                                                u'sum_
↪squares': 5223.85,
                                                                                u
↪'variance': 0.68569}},
                                                    u'000001': {   u'column_number': 1,
                                                                   u'datatype': u'double',
                                                                   u'name': u'sepal width',
                                                                   u'optype': u'numeric',
                                                                   u'order': 1,
                                                                   u'preferred': True,
                                                                   u'summary': {   u'counts
↪': [   [   2,
                                                                                       ␣
↪           1],
                                                                                       ␣
↪       [   2.2,
                                                                                       ␣
↪           3],
...
                                                                                       ␣
↪       [   4.2,
                                                                                       ␣
↪           1],
                                                                                       ␣
↪       [   4.4,
                                                                                       ␣
↪           1]],
                                                                                u
↪'kurtosis': 0.18098,
                                                                                u
↪'maximum': 4.4,
                                                                                u'mean
↪': 3.05733,
```

(continues on next page)

(continued from previous page)

```
                                                                                     u'median
↪': 3,
                                                                                     u
↪'minimum': 2,
                                                                                     u
↪'missing_count': 0,
                                                                                     u
↪'population': 150,
                                                                                     u
↪'skewness': 0.31577,
                                                                                     u
↪'standard_deviation': 0.43587,
                                                                                     u'sum':␣
↪458.6,
                                                                                     u'sum_
↪squares': 1430.4,
                                                                                     u
↪'variance': 0.18998}},
                                                  u'000002': {   u'column_number': 2,
                                                                 u'datatype': u'double',
                                                                 u'name': u'petal length
↪',
                                                                 u'optype': u'numeric',
                                                                 u'order': 2,
                                                                 u'preferred': True,
                                                                 u'summary': {   u'bins
↪': [   [   1,
                                                                                         ␣
↪         1],
                                                                                         ␣
↪     [   1.16667,
                                                                                         ␣
↪         3],
...
                                                                                         ␣
↪     [   6.6,
                                                                                         ␣
↪         1],
                                                                                         ␣
↪     [   6.7,
                                                                                         ␣
↪         2],
                                                                                         ␣
↪     [   6.9,
                                                                                         ␣
↪         1]],
                                                                                     u
↪'kurtosis': -1.39554,
                                                                                     u
↪'maximum': 6.9,
                                                                                     u'mean
↪': 3.758,
                                                                                     u'median
↪': 4.35,
                                                                                     u
↪'minimum': 1,
                                                                                     u
↪'missing_count': 0,
```

(continues on next page)

(continued from previous page)

```
                                                                                           u
↪'population': 150,
                                                                                           u
↪'skewness': -0.27213,
                                                                                           u'splits
↪': [   1.25138,
                                                                                                  ␣
↪       1.32426,
                                                                                                  ␣
↪       1.37171,
...
                                                                                                  ␣
↪       6.02913,
                                                                                                  ␣
↪       6.38125],
                                                                                           u
↪'standard_deviation': 1.7653,
                                                                                           u'sum':␣
↪563.7,
                                                                                           u'sum_
↪squares': 2582.71,
                                                                                           u
↪'variance': 3.11628}},
                                                  u'000003': {   u'column_number': 3,
                                                                 u'datatype': u'double',
                                                                 u'name': u'petal width',
                                                                 u'optype': u'numeric',
                                                                 u'order': 3,
                                                                 u'preferred': True,
                                                                 u'summary': {   u'counts
↪': [   [   0.1,
                                                                                                  ␣
↪           5],
                                                                                                  ␣
↪       [   0.2,
                                                                                                  ␣
↪           29],
...
                                                                                                  ␣
↪       [   2.4,
                                                                                                  ␣
↪           3],
                                                                                                  ␣
↪       [   2.5,
                                                                                                  ␣
↪           3]],
                                                                                           u
↪'kurtosis': -1.33607,
                                                                                           u
↪'maximum': 2.5,
                                                                                           u'mean
↪': 1.19933,
                                                                                           u'median
↪': 1.3,
                                                                                           u
↪'minimum': 0.1,
                                                                                           u
↪'missing_count': 0,
```

(continues on next page)

(continued from previous page)

```
                                                                                u
↪'population': 150,
                                                                                u
↪'skewness': -0.10193,
                                                                                u
↪'standard_deviation': 0.76224,
                                                                                u'sum':␣
↪179.9,
                                                                                u'sum_
↪squares': 302.33,
                                                                                u
↪'variance': 0.58101}},
                                              u'000004': {   u'column_number': 4,
                                                             u'datatype': u'string',
                                                             u'name': u'species',
                                                             u'optype': u'categorical
↪',
                                                             u'order': 4,
                                                             u'preferred': True,
                                                             u'summary': {   u
↪'categories': [   [   u'Iris-setosa',
                                                                                     ␣
↪                 50],
                                                                                     ␣
↪           [   u'Iris-versicolor',
                                                                                     ␣
↪               50],
                                                                                     ␣
↪           [   u'Iris-virginica',
                                                                                     ␣
↪               50]],
                                                                                u
↪'missing_count': 0},
                                                             u'term_analysis': {   u
↪'enabled': True}}},
                               u'normalize': False,
                               u'regularization': u'l2'},
    u'max_columns': 5,
    u'max_rows': 150,
    u'name': u"iris' dataset's logistic regression",
    u'number_of_batchpredictions': 0,
    u'number_of_evaluations': 0,
    u'number_of_predictions': 1,
    u'objective_field': u'000004',
    u'objective_field_name': u'species',
    u'objective_field_type': u'categorical',
    u'objective_fields': [u'000004'],
    u'out_of_bag': False,
    u'private': True,
    u'project': u'project/561304c137203f4c9300016c',
    u'range': [1, 150],
    u'replacement': False,
    u'resource': u'logisticregression/5617e71c37203f506a000001',
    u'rows': 150,
    u'sample_rate': 1.0,
    u'shared': False,
    u'size': 4609,
```

(continues on next page)

(continued from previous page)

```
    u'source': u'source/561304f437203f4c930001c3',
    u'source_status': True,
    u'status': {   u'code': 5,
                   u'elapsed': 86,
                   u'message': u'The logistic regression has been created',
                   u'progress': 1.0},
    u'subscription': False,
    u'tags': [u'species'],
    u'updated': u'2015-10-09T16:14:02.336000',
    u'white_box': False}
```

Note that the output in the snippet above has been abbreviated. As you see, the `logistic_regression` attribute stores the coefficients used in the logistic function as well as the configuration parameters described in the developers section .

CHAPTER 18

Linear Regressions

A linear regression is a supervised machine learning method for solving regression problems by computing the objective as a linear combination of factors. The implementation is a multiple linear regression that models the output as a linear combination of the predictors. The coefficients are estimated doing a least-squares fit on the training data.

As a linear combination can only be done using numeric values, non-numeric fields need to be transformed to numeric ones following some rules:

- Categorical fields will be encoded and each class appearance in input data will convey a different contribution to the input vector.
- Text and items fields will be expanded to several numeric predictors, each one indicating the number of occurences for a specific term. Text fields without term analysis are excluded from the model.

Therefore, the initial input data is transformed into an input vector with one or may components per field. Also, if a field in the training data contains missing data, the components corresponding to that field will include an additional 1 or 0 value depending on whether the field is missing in the input data or not.

The JSON structure for a linear regression is:

```
>>> api.pprint(linear_regression["object"])
{   u'category': 0,
    u'code': 200,
    u'columns': 4,
    u'composites': None,
    u'configuration': None,
    u'configuration_status': False,
    u'created': u'2019-02-20T21:02:40.027000',
    u'creator': u'merce',
    u'credits': 0.0,
    u'credits_per_prediction': 0.0,
    u'dataset': u'dataset/5c6dc06a983efc18e2000084',
    u'dataset_field_types': {   u'categorical': 0,
                                u'datetime': 0,
                                u'items': 0,
                                u'numeric': 6,
```

(continues on next page)

(continued from previous page)

```
                                u'preferred': 6,
                                u'text': 0,
                                u'total': 6},
    u'dataset_status': True,
    u'datasets': [],
    u'default_numeric_value': None,
    u'description': u'',
    u'excluded_fields': [],
    u'execution_id': None,
    u'execution_status': None,
    u'fields_maps': None,
    u'fields_meta': {   u'count': 4,
                        u'limit': 1000,
                        u'offset': 0,
                        u'query_total': 4,
                        u'total': 4},
    u'fusions': None,
    u'input_fields': [u'000000', u'000001', u'000002'],
    u'linear_regression': {   u'bias': True,
                              u'coefficients': [   [-1.88196],
                                                   [0.475633],
                                                   [0.122468],
                                                   [30.9141]],
                              u'fields': {   u'000000': {   u'column_number': 0,
                                                            u'datatype': u'int8',
                                                            u'name': u'Prefix',
                                                            u'optype': u'numeric',
                                                            u'order': 0,
                                                            u'preferred': True,
                                                            u'summary': {   u'counts
↪': [   [   4,
                                                                                   ␣
↪          1],

                                ...
                              u'stats': {   u'confidence_intervals': [   [   5.63628],
                                                                         [   0.
↪375062],
                                                                         [   0.
↪348577],
                                                                         [   44.
↪4112]],
                                            u'mean_squared_error': 342.206,
                                            u'number_of_parameters': 4,
                                            u'number_of_samples': 77,
                                            u'p_values': [   [0.512831],
                                                             [0.0129362],
                                                             [0.491069],
                                                             [0.172471]],
                                            u'r_squared': 0.136672,
                                            u'standard_errors': [   [   2.87571],
                                                                    [   0.191361],
                                                                    [   0.177849],
                                                                    [   22.6592]],
                                            u'sum_squared_errors': 24981,
                                            u'xtx': [   [   4242,
                                                            48396.9,
```

(continues on next page)

(continued from previous page)

```
                                                             51273.97,
                                                             568],
                                                         [   48396.9,
                                                             570177.6584,
                                                             594274.3274,
                                                             6550.52],
                                                         [   51273.97,
                                                             594274.3274,
                                                             635452.7068,
                                                             6894.24],
                                                         [   568,
                                                             6550.52,
                                                             6894.24,
                                                             77]],
                                            u'z_scores': [   [-0.654436],
                                                              [2.48552],
                                                              [0.688609],
                                                              [1.36431]]}},
    u'locale': u'en_US',
    u'max_columns': 6,
    u'max_rows': 80,
    u'name': u'grades',
    u'name_options': u'bias',
    u'number_of_batchpredictions': 0,
    u'number_of_evaluations': 0,
    u'number_of_predictions': 2,
    u'number_of_public_predictions': 0,
    u'objective_field': u'000005',
    u'objective_field_name': u'Final',
    u'objective_field_type': u'numeric',
    u'objective_fields': [u'000005'],
    u'operating_point': {   },
    u'optiml': None,
    u'optiml_status': False,
    u'ordering': 0,
    u'out_of_bag': False,
    u'out_of_bags': None,
    u'price': 0.0,
    u'private': True,
    u'project': u'project/5c6dc062983efc18d5000129',
    u'range': None,
    u'ranges': None,
    u'replacement': False,
    u'replacements': None,
    u'resource': u'linearregression/5c6dc070983efc18e00001f1',
    u'rows': 80,
    u'sample_rate': 1.0,
    u'sample_rates': None,
    u'seed': None,
    u'seeds': None,
    u'shared': False,
    u'size': 2691,
    u'source': u'source/5c6dc064983efc18e00001ed',
    u'source_status': True,
    u'status': {   u'code': 5,
                   u'elapsed': 62086,
                   u'message': u'The linear regression has been created',
```

(continues on next page)

(continued from previous page)

```
                  u'progress': 1},
    u'subscription': True,
    u'tags': [],
    u'type': 0,
    u'updated': u'2019-02-27T18:01:18.539000',
    u'user_metadata': {    },
    u'webhook': None,
    u'weight_field': None,
    u'white_box': False}
```

Note that the output in the snippet above has been abbreviated. As you see, the `linear_regression` attribute stores the coefficients used in the linear function as well as the configuration parameters described in the developers section .

CHAPTER 19

Associations

Association Discovery is a popular method to find out relations among values in high-dimensional datasets.

A common case where association discovery is often used is market basket analysis. This analysis seeks for customer shopping patterns across large transactional datasets. For instance, do customers who buy hamburgers and ketchup also consume bread?

Businesses use those insights to make decisions on promotions and product placements. Association Discovery can also be used for other purposes such as early incident detection, web usage analysis, or software intrusion detection.

In BigML, the Association resource object can be built from any dataset, and its results are a list of association rules between the items in the dataset. In the example case, the corresponding association rule would have hamburguers and ketchup as the items at the left hand side of the association rule and bread would be the item at the right hand side. Both sides in this association rule are related, in the sense that observing the items in the left hand side implies observing the items in the right hand side. There are some metrics to ponder the quality of these association rules:

- Support: the proportion of instances which contain an itemset.

For an association rule, it means the number of instances in the dataset which contain the rule's antecedent and rule's consequent together over the total number of instances (N) in the dataset.

It gives a measure of the importance of the rule. Association rules have to satisfy a minimum support constraint (i.e., min_support).

- Coverage: the support of the antedecent of an association rule.

It measures how often a rule can be applied.

- Confidence or (strength): The probability of seeing the rule's consequent

under the condition that the instances also contain the rule's antecedent. Confidence is computed using the support of the association rule over the coverage. That is, the percentage of instances which contain the consequent and antecedent together over the number of instances which only contain the antecedent.

Confidence is directed and gives different values for the association rules Antecedent $\rightarrow$ Consequent and Consequent $\rightarrow$ Antecedent. Association rules also need to satisfy a minimum confidence constraint (i.e., min_confidence).

- Leverage: the difference of the support of the association rule (i.e., the antecedent and consequent appearing together) and what would be expected if antecedent and consequent where statistically independent. This is a value between -1 and 1. A positive value suggests a positive relationship and a negative value suggests a negative relationship. 0 indicates independence.

Lift: how many times more often antecedent and consequent occur together than expected if they where statistically independent. A value of 1 suggests that there is no relationship between the antecedent and the consequent. Higher values suggest stronger positive relationships. Lower values suggest stronger negative relationships (the presence of the antecedent reduces the likelihood of the consequent)

As to the items used in association rules, each type of field is parsed to extract items for the rules as follows:

- Categorical: each different value (class) will be considered a separate item.
- Text: each unique term will be considered a separate item.
- Items: each different item in the items summary will be considered.
- Numeric: Values will be converted into categorical by making a

segmentation of the values. For example, a numeric field with values ranging from 0 to 600 split into 3 segments: segment 1 → [0, 200), segment 2 → [200, 400), segment 3 → [400, 600]. You can refine the behavior of the transformation using discretization and field_discretizations.

The JSON structure for an association resource is:

```
>>> api.pprint(association['object'])
{
    "associations":{
        "complement":false,
        "discretization":{
            "pretty":true,
            "size":5,
            "trim":0,
            "type":"width"
        },
        "items":[
            {
                "complement":false,
                "count":32,
                "field_id":"000000",
                "name":"Segment 1",
                "bin_end":5,
                "bin_start":null
            },
            {
                "complement":false,
                "count":49,
                "field_id":"000000",
                "name":"Segment 3",
                "bin_end":7,
                "bin_start":6
            },
            {
                "complement":false,
                "count":12,
                "field_id":"000000",
                "name":"Segment 4",
                "bin_end":null,
                "bin_start":7
            },
```

(continues on next page)

(continued from previous page)

```
            {
                "complement":false,
                "count":19,
                "field_id":"000001",
                "name":"Segment 1",
                "bin_end":2.5,
                "bin_start":null
            },
            ...
            {
                "complement":false,
                "count":50,
                "field_id":"000004",
                "name":"Iris-versicolor"
            },
            {
                "complement":false,
                "count":50,
                "field_id":"000004",
                "name":"Iris-virginica"
            }
        ],
        "max_k": 100,
        "min_confidence":0,
        "min_leverage":0,
        "min_lift":1,
        "min_support":0,
        "rules":[
            {
                "confidence":1,
                "id":"000000",
                "leverage":0.22222,
                "lhs":[
                    13
                ],
                "lhs_cover":[
                    0.33333,
                    50
                ],
                "lift":3,
                "p_value":0.000000000,
                "rhs":[
                    6
                ],
                "rhs_cover":[
                    0.33333,
                    50
                ],
                "support":[
                    0.33333,
                    50
                ]
            },
            {
                "confidence":1,
                "id":"000001",
                "leverage":0.22222,
```

(continues on next page)

(continued from previous page)

```
                "lhs":[
                    6
                ],
                "lhs_cover":[
                    0.33333,
                    50
                ],
                "lift":3,
                "p_value":0.000000000,
                "rhs":[
                    13
                ],
                "rhs_cover":[
                    0.33333,
                    50
                ],
                "support":[
                    0.33333,
                    50
                ]
            },
            ...
            {
                "confidence":0.26,
                "id":"000029",
                "leverage":0.05111,
                "lhs":[
                    13
                ],
                "lhs_cover":[
                    0.33333,
                    50
                ],
                "lift":2.4375,
                "p_value":0.0000454342,
                "rhs":[
                    5
                ],
                "rhs_cover":[
                    0.10667,
                    16
                ],
                "support":[
                    0.08667,
                    13
                ]
            },
            {
                "confidence":0.18,
                "id":"00002a",
                "leverage":0.04,
                "lhs":[
                    15
                ],
                "lhs_cover":[
                    0.33333,
                    50
```

(continues on next page)

(continued from previous page)

```
            ],
            "lift":3,
            "p_value":0.0000302052,
            "rhs":[
                9
            ],
            "rhs_cover":[
                0.06,
                9
            ],
            "support":[
                0.06,
                9
            ]
        },
        {
            "confidence":1,
            "id":"00002b",
            "leverage":0.04,
            "lhs":[
                9
            ],
            "lhs_cover":[
                0.06,
                9
            ],
            "lift":3,
            "p_value":0.0000302052,
            "rhs":[
                15
            ],
            "rhs_cover":[
                0.33333,
                50
            ],
            "support":[
                0.06,
                9
            ]
        }
    ],
    "rules_summary":{
        "confidence":{
            "counts":[
                [
                    0.18,
                    1
                ],
                [
                    0.24,
                    1
                ],
                [
                    0.26,
                    2
                ],
                ...
```

(continues on next page)

(continued from previous page)

```
                [
                    0.97959,
                    1
                ],
                [
                    1,
                    9
                ]
            ],
            "maximum":1,
            "mean":0.70986,
            "median":0.72864,
            "minimum":0.18,
            "population":44,
            "standard_deviation":0.24324,
            "sum":31.23367,
            "sum_squares":24.71548,
            "variance":0.05916
        },
        "k":44,
        "leverage":{
            "counts":[
                [
                    0.04,
                    2
                ],
                [
                    0.05111,
                    4
                ],
                [
                    0.05316,
                    2
                ],
                ...
                [
                    0.22222,
                    2
                ]
            ],
            "maximum":0.22222,
            "mean":0.10603,
            "median":0.10156,
            "minimum":0.04,
            "population":44,
            "standard_deviation":0.0536,
            "sum":4.6651,
            "sum_squares":0.61815,
            "variance":0.00287
        },
        "lhs_cover":{
            "counts":[
                [
                    0.06,
                    2
                ],
                [
```

(continues on next page)

(continued from previous page)

```
                    0.08,
                    2
                ],
                [
                    0.10667,
                    4
                ],
                [
                    0.12667,
                    1
                ],
                ...
                [
                    0.5,
                    4
                ]
            ],
            "maximum":0.5,
            "mean":0.29894,
            "median":0.33213,
            "minimum":0.06,
            "population":44,
            "standard_deviation":0.13386,
            "sum":13.15331,
            "sum_squares":4.70252,
            "variance":0.01792
        },
        "lift":{
            "counts":[
                [
                    1.40625,
                    2
                ],
                [
                    1.5067,
                    2
                ],
                ...
                [
                    2.63158,
                    4
                ],
                [
                    3,
                    10
                ],
                [
                    4.93421,
                    2
                ],
                [
                    12.5,
                    2
                ]
            ],
            "maximum":12.5,
            "mean":2.91963,
```

(continues on next page)

(continued from previous page)

```
                "median":2.58068,
                "minimum":1.40625,
                "population":44,
                "standard_deviation":2.24641,
                "sum":128.46352,
                "sum_squares":592.05855,
                "variance":5.04635
            },
            "p_value":{
                "counts":[
                    [
                        0.000000000,
                        2
                    ],
                    [
                        0.000000000,
                        4
                    ],
                    [
                        0.000000000,
                        2
                    ],
                    ...
                    [
                        0.0000910873,
                        2
                    ]
                ],
                "maximum":0.0000910873,
                "mean":0.0000106114,
                "median":0.00000000,
                "minimum":0.000000000,
                "population":44,
                "standard_deviation":0.0000227364,
                "sum":0.000466903,
                "sum_squares":0.0000000,
                "variance":0.000000001
            },
            "rhs_cover":{
                "counts":[
                    [
                        0.06,
                        2
                    ],
                    [
                        0.08,
                        2
                    ],
                    ...
                    [
                        0.42667,
                        2
                    ],
                    [
                        0.46667,
                        3
                    ],
```

(continues on next page)

(continued from previous page)

```
                [
                    0.5,
                    4
                ]
            ],
            "maximum":0.5,
            "mean":0.29894,
            "median":0.33213,
            "minimum":0.06,
            "population":44,
            "standard_deviation":0.13386,
            "sum":13.15331,
            "sum_squares":4.70252,
            "variance":0.01792
        },
        "support":{
            "counts":[
                [
                    0.06,
                    4
                ],
                [
                    0.06667,
                    2
                ],
                [
                    0.08,
                    2
                ],
                [
                    0.08667,
                    4
                ],
                [
                    0.10667,
                    4
                ],
                [
                    0.15333,
                    2
                ],
                [
                    0.18667,
                    4
                ],
                [
                    0.19333,
                    2
                ],
                [
                    0.20667,
                    2
                ],
                [
                    0.27333,
                    2
                ],
```

(continues on next page)

(continued from previous page)

```
                    [
                        0.28667,
                        2
                    ],
                    [
                        0.3,
                        4
                    ],
                    [
                        0.32,
                        2
                    ],
                    [
                        0.33333,
                        6
                    ],
                    [
                        0.37333,
                        2
                    ]
                ],
                "maximum":0.37333,
                "mean":0.20152,
                "median":0.19057,
                "minimum":0.06,
                "population":44,
                "standard_deviation":0.10734,
                "sum":8.86668,
                "sum_squares":2.28221,
                "variance":0.01152
            }
        },
        "search_strategy":"leverage",
        "significance_level":0.05
    },
    "category":0,
    "clones":0,
    "code":200,
    "columns":5,
    "created":"2015-11-05T08:06:08.184000",
    "credits":0.017581939697265625,
    "dataset":"dataset/562fae3f4e1727141d00004e",
    "dataset_status":true,
    "dataset_type":0,
    "description":"",
    "excluded_fields":[ ],
    "fields_meta":{
        "count":5,
        "limit":1000,
        "offset":0,
        "query_total":5,
        "total":5
    },
    "input_fields":[
        "000000",
        "000001",
        "000002",
```

(continues on next page)

(continued from previous page)

```
        "000003",
        "000004"
    ],
    "locale":"en_US",
    "max_columns":5,
    "max_rows":150,
    "name":"iris' dataset's association",
    "out_of_bag":false,
    "price":0,
    "private":true,
    "project":null,
    "range":[
        1,
        150
    ],
    "replacement":false,
    "resource":"association/5621b70910cb86ae4c000000",
    "rows":150,
    "sample_rate":1,
    "shared":false,
    "size":4609,
    "source":"source/562fae3a4e1727141d000048",
    "source_status":true,
    "status":{
        "code":5,
        "elapsed":1072,
        "message":"The association has been created",
        "progress":1
    },
    "subscription":false,
    "tags":[ ],
    "updated":"2015-11-05T08:06:20.403000",
    "white_box":false
}
```

Note that the output in the snippet above has been abbreviated. As you see, the `associations` attribute stores items, rules and metrics extracted from the datasets as well as the configuration parameters described in the developers section .

CHAPTER 20

Topic Models

A topic model is an unsupervised machine learning method for unveiling all the different topics underlying a collection of documents. BigML uses Latent Dirichlet Allocation (LDA), one of the most popular probabilistic methods for topic modeling. In BigML, each instance (i.e. each row in your dataset) will be considered a document and the contents of all the text fields given as inputs will be automatically concatenated and considered the document bag of words.

Topic model is based on the assumption that any document exhibits a mixture of topics. Each topic is composed of a set of words which are thematically related. The words from a given topic have different probabilities for that topic. At the same time, each word can be attributable to one or several topics. So for example the word "sea" may be found in a topic related with sea transport but also in a topic related to holidays. Topic model automatically discards stop words and high frequency words.

Topic model's main applications include browsing, organizing and understanding large archives of documents. It can been applied for information retrieval, collaborative filtering, assessing document similarity among others. The topics found in the dataset can also be very useful new features before applying other models like classification, clustering, or anomaly detection.

The JSON structure for a topic model is:

```
>>> api.pprint(topic['object'])
{   u'category': 0,
    u'code': 200,
    u'columns': 1,
    u'configuration': None,
    u'configuration_status': False,
    u'created': u'2016-11-23T23:47:54.703000',
    u'credits': 0.0,
    u'credits_per_prediction': 0.0,
    u'dataset': u'dataset/58362aa0983efc45a0000005',
    u'dataset_field_types': {   u'categorical': 1,
                                u'datetime': 0,
                                u'effective_fields': 672,
                                u'items': 0,
                                u'numeric': 0,
                                u'preferred': 2,
```

(continues on next page)

(continued from previous page)

```
                                u'text': 1,
                                u'total': 2},
    u'dataset_status': True,
    u'dataset_type': 0,
    u'description': u'',
    u'excluded_fields': [],
    u'fields_meta': {   u'count': 1,
                        u'limit': 1000,
                        u'offset': 0,
                        u'query_total': 1,
                        u'total': 1},
    u'input_fields': [u'000001'],
    u'locale': u'en_US',
    u'max_columns': 2,
    u'max_rows': 656,
    u'name': u"spam dataset's Topic Model ",
    u'number_of_batchtopicdistributions': 0,
    u'number_of_public_topicdistributions': 0,
    u'number_of_topicdistributions': 0,
    u'ordering': 0,
    u'out_of_bag': False,
    u'price': 0.0,
    u'private': True,
    u'project': None,
    u'range': [1, 656],
    u'replacement': False,
    u'resource': u'topicmodel/58362aaa983efc45a1000007',
    u'rows': 656,
    u'sample_rate': 1.0,
    u'shared': False,
    u'size': 54740,
    u'source': u'source/58362a69983efc459f000001',
    u'source_status': True,
    u'status': {   u'code': 5,
                   u'elapsed': 3222,
                   u'message': u'The topic model has been created',
                   u'progress': 1.0},
    u'subscription': True,
    u'tags': [],
    u'topic_model': {   u'alpha': 4.166666666666667,
                        u'beta': 0.1,
                        u'bigrams': False,
                        u'case_sensitive': False,
                        u'fields': {   u'000001': {   u'column_number': 1,
                                                      u'datatype': u'string',
                                                      u'name': u'Message',
                                                      u'optype': u'text',
                                                      u'order': 0,
                                                      u'preferred': True,
                                                      u'summary': {   u'average_length
↪': 78.14787,
                                                                      u'missing_count
↪': 0,
                                                                      u'tag_cloud': [␣
↪  [   u'call',
                                                                                    ␣
↪       72],
```

(continues on next page)

(continued from previous page)

```
                                                                                   ␣
↪  [   u'ok',
                                                                                   ␣
↪      36],
                                                                                   ␣
↪  [   u'gt',
                                                                                   ␣
↪      34],
...
                                                                                   ␣
↪  [   u'worse',
                                                                                   ␣
↪      2],
                                                                                   ␣
↪  [   u'worth',
                                                                                   ␣
↪      2],
                                                                                   ␣
↪  [   u'write',
                                                                                   ␣
↪      2],
                                                                                   ␣
↪  [   u'yest',
                                                                                   ␣
↪      2],
                                                                                   ␣
↪  [   u'yijue',
                                                                                   ␣
↪      2]],
                                                                   u'term_forms':
↪{   }},
                                                      u'term_analysis': {   u'case_
↪sensitive': False,
                                                                        u'enabled
↪': True,
                                                                        u'language
↪': u'en',
                                                                        u'stem_
↪words': False,
                                                                        u'token_
↪mode': u'all',
                                                                        u'use_
↪stopwords': False}}},
                        u'hashed_seed': 62146850,
                        u'language': u'en',
                        u'number_of_topics': 12,
                        u'term_limit': 4096,
                        u'term_topic_assignments': [   [   0,
                                                           5,
                                                           0,
                                                           1,
                                                           0,
                                                           19,
                                                           0,
                                                           0,
                                                           19,
                                                           0,
```

(continues on next page)

(continued from previous page)

```
                                                         1,
                                                         0],
                                                     [   0,
                                                         0,
                                                         0,
                                                         13,
                                                         0,
                                                         0,
                                                         0,
                                                         0,
                                                         5,
                                                         0,
                                                         0,
                                                         0],
...
                                                     [   0,
                                                         7,
                                                         27,
                                                         0,
                                                         112,
                                                         0,
                                                         0,
                                                         0,
                                                         0,
                                                         0,
                                                         14,
                                                         2]],
                        u'termset': [   u'000',
                                        u'03',
                                        u'04',
                                        u'06',
                                        u'08000839402',
                                        u'08712460324',
...

                                        u'yes',
                                        u'yest',
                                        u'yesterday',
                                        u'yijue',
                                        u'yo',
                                        u'yr',
                                        u'yup',
                                        u'\xfc'],
                        u'top_n_terms': 10,
                        u'topicmodel_seed': u'26c386d781963ca1ea5c90dab8a6b023b5e1d180
↪',
                        u'topics': [   {   u'id': u'000000',
                                           u'name': u'Topic 00',
                                           u'probability': 0.09375,
                                           u'top_terms': [   [   u'im',
                                                                 0.04849],
                                                             [   u'hi',
                                                                 0.04717],
                                                             [   u'love',
                                                                 0.04585],
                                                             [   u'please',
                                                                 0.02867],
```

(continues on next page)

(continued from previous page)

```
                                                            [   u'tomorrow',
                                                                0.02867],
                                                            [   u'cos',
                                                                0.02823],
                                                            [   u'sent',
                                                                0.02647],
                                                            [   u'da',
                                                                0.02383],
                                                            [   u'meet',
                                                                0.02207],
                                                            [   u'dinner',
                                                                0.01898]]},
                                      {   u'id': u'000001',
                                          u'name': u'Topic 01',
                                          u'probability': 0.08215,
                                          u'top_terms': [   [   u'lt',
                                                                0.1015],
                                                            [   u'gt',
                                                                0.1007],
                                                            [   u'wish',
                                                                0.03958],
                                                            [   u'feel',
                                                                0.0272],
                                                            [   u'shit',
                                                                0.02361],
                                                            [   u'waiting',
                                                                0.02281],
                                                            [   u'stuff',
                                                                0.02001],
                                                            [   u'name',
                                                                0.01921],
                                                            [   u'comp',
                                                                0.01522],
                                                            [   u'forgot',
                                                                0.01482]]},
...
                                      {   u'id': u'00000b',
                                          u'name': u'Topic 11',
                                          u'probability': 0.0826,
                                          u'top_terms': [   [   u'call',
                                                                0.15084],
                                                            [   u'min',
                                                                0.05003],
                                                            [   u'msg',
                                                                0.03185],
                                                            [   u'home',
                                                                0.02648],
                                                            [   u'mind',
                                                                0.02152],
                                                            [   u'lt',
                                                                0.01987],
                                                            [   u'bring',
                                                                0.01946],
                                                            [   u'camera',
                                                                0.01905],
                                                            [   u'set',
                                                                0.01905],
```

(continues on next page)

(continued from previous page)

```
                                                            [   u'contact',
                                                                0.01781]]}],
                    u'use_stopwords': False},
    u'updated': u'2016-11-23T23:48:03.336000',
    u'white_box': False}
```

Note that the output in the snippet above has been abbreviated.

The topic model returns a list of top terms for each topic found in the data. Note that topics are not labeled, so you have to infer their meaning according to the words they are composed of.

Once you build the topic model you can calculate each topic probability for a given document by using Topic Distribution. This information can be useful to find documents similarities based on their thematic.

As you see, the `topic_model` attribute stores the topics and termset and term to topic assignment, as well as the configuration parameters described in the developers section .

CHAPTER 21

Time Series

A time series model is a supervised learning method to forecast the future values of a field based on its previously observed values. It is used to analyze time based data when historical patterns can explain the future behavior such as stock prices, sales forecasting, website traffic, production and inventory analysis, weather forecasting, etc. A time series model needs to be trained with time series data, i.e., a field containing a sequence of equally distributed data points in time.

BigML implements exponential smoothing to train time series models. Time series data is modeled as a level component and it can optionally include a trend (damped or not damped) and a seasonality components. You can learn more about how to include these components and their use in the API documentation page.

You can create a time series model selecting one or several fields from your dataset, that will be the ojective fields. The forecast will compute their future values.

The JSON structure for a time series is:

```
>>> api.pprint(time_series['object'])
{   u'category': 0,
    u'clones': 0,
    u'code': 200,
    u'columns': 1,
    u'configuration': None,
    u'configuration_status': False,
    u'created': u'2017-07-15T12:49:42.601000',
    u'credits': 0.0,
    u'dataset': u'dataset/5968ec42983efc21b0000016',
    u'dataset_field_types': {   u'categorical': 0,
                                u'datetime': 0,
                                u'effective_fields': 6,
                                u'items': 0,
                                u'numeric': 6,
                                u'preferred': 6,
                                u'text': 0,
                                u'total': 6},
    u'dataset_status': True,
    u'dataset_type': 0,
```

(continues on next page)

(continued from previous page)

```
    u'description': u'',
    u'fields_meta': {   u'count': 1,
                        u'limit': 1000,
                        u'offset': 0,
                        u'query_total': 1,
                        u'total': 1},
    u'forecast': {   u'000005': [   {   u'lower_bound': [   30.14111,
                                                            30.14111,
                                                            ...
                                                            30.14111],
                                        u'model': u'A,N,N',
                                        u'point_forecast': [   68.53181,
                                                               68.53181,
                                                               ...
                                                               68.53181,
                                                               68.53181],
                                        u'time_range': {   u'end': 129,
                                                           u'interval': 1,
                                                           u'interval_unit': u
↪'milliseconds',
                                                           u'start': 80},
                                        u'upper_bound': [   106.92251,
                                                            106.92251,
                                                            ...
                                                            106.92251,
                                                            106.92251]},
                                    {   u'lower_bound': [   35.44118,
                                                            35.5032,
                                                            ...
                                                            35.28083],
                                        u'model': u'A,Ad,N',
                        ...
                                                               66.83537,
                                                               66.9465],
                                        u'time_range': {   u'end': 129,
                                                           u'interval': 1,
                                                           u'interval_unit': u
↪'milliseconds',
                                                           u'start': 80}}]},
    u'horizon': 50,
    u'locale': u'en_US',
    u'max_columns': 6,
    u'max_rows': 80,
    u'name': u'my_ts_data',
    u'name_options': u'period=1, range=[1, 80]',
    u'number_of_evaluations': 0,
    u'number_of_forecasts': 0,
    u'number_of_public_forecasts': 0,
    u'objective_field': u'000005',
    u'objective_field_name': u'Final',
    u'objective_field_type': u'numeric',
    u'objective_fields': [u'000005'],
    u'objective_fields_names': [u'Final'],
    u'price': 0.0,
    u'private': True,
    u'project': None,
    u'range': [1, 80],
```

(continues on next page)

(continued from previous page)

```
    u'resource': u'timeseries/596a0f66983efc53f3000000',
    u'rows': 80,
    u'shared': False,
    u'short_url': u'',
    u'size': 2691,
    u'source': u'source/5968ec3c983efc218c000006',
    u'source_status': True,
    u'status': {   u'code': 5,
                   u'elapsed': 8358,
                   u'message': u'The time series has been created',
                   u'progress': 1.0},
    u'subscription': True,
    u'tags': [],
    u'time_series': {   u'all_numeric_objectives': False,
                        u'datasets': {   u'000005': u'dataset/596a0f70983efc53f3000003
↪'},
                        u'ets_models': {   u'000005': [   {   u'aic': 831.30903,
                                                              u'aicc': 831.84236,
                                                              u'alpha': 0.00012,
                                                              u'beta': 0,
                                                              u'bic': 840.83713,
                                                              u'final_state': {   u'b
↪': 0,
                                                                                  u'l
↪': 68.53181,
                                                                                  u's
↪': [   0]},
                                                              u'gamma': 0,
                                                              u'initial_state': {   u
↪'b': 0,
                                                                                    u
↪'l': 68.53217,
                                                                                    u
↪'s': [   0]},
                                                              u'name': u'A,N,N',
                                                              u'period': 1,
                                                              u'phi': 1,
                                                              u'r_squared': -0.0187,
                                                              u'sigma': 19.19535},
                                                          {   u'aic': 834.43049,
                                                              ...
                                                              u'slope': 0.11113,
                                                              u'value': 61.39}]},
                        u'fields': {   u'000005': {   u'column_number': 5,
                                                      u'datatype': u'double',
                                                      u'name': u'Final',
                                                      u'optype': u'numeric',
                                                      u'order': 0,
                                                      u'preferred': True,
                                                      u'summary': {   u'bins': [   [ ␣
↪ 28.06,
                                                                                     ␣
↪ 1],
                                                                                   [ ␣
↪ 34.44,
                                                                                    ..
↪.
```

(continues on next page)

(continued from previous page)

```
                                                                                          [ ␣
↪ 108.335,
                                                                                            ␣
↪ 2]],
                                                                              ...
                                                                              u'sum_squares':␣
↪389814.3944,
                                                                              u'variance':␣
↪380.73315}}},
                        u'period': 1,
                        u'time_range': {   u'end': 79,
                                           u'interval': 1,
                                           u'interval_unit': u'milliseconds',
                                           u'start': 0}},
    u'type': 0,
    u'updated': u'2017-07-15T12:49:52.549000',
    u'white_box': False}
```

CHAPTER 22

OptiMLs

An OptiML is the result of an automated optimization process to find the best model (type and configuration) to solve a particular classification or regression problem.

The selection process automates the usual time-consuming task of trying different models and parameters and evaluating their results to find the best one. Using the OptiML, non-experts can build top-performing models.

You can create an OptiML selecting the ojective field to be predicted, the evaluation metric to be used to rank the models tested in the process and a maximum time for the task to be run.

The JSON structure for an OptiML is:

```
>>> api.pprint(optiml["object"])
{   u'category': 0,
    u'code': 200,
    u'configuration': None,
    u'configuration_status': False,
    u'created': u'2018-05-17T20:23:00.060000',
    u'creator': u'mmartin',
    u'dataset': u'dataset/5afdb7009252732d930009e8',
    u'dataset_status': True,
    u'datasets': [   u'dataset/5afde6488bf7d551ee00081c',
                     u'dataset/5afde6488bf7d551fd00511f',
                     u'dataset/5afde6488bf7d551fe002e0f',
                        ...
                     u'dataset/5afde64d8bf7d551fd00512e'],
    u'description': u'',
    u'evaluations': [   u'evaluation/5afde65c8bf7d551fd00514c',
                        u'evaluation/5afde65c8bf7d551fd00514f',
                        ...
                        u'evaluation/5afde6628bf7d551fd005161'],
    u'excluded_fields': [],
    u'fields_meta': {   u'count': 5,
                        u'limit': 1000,
                        u'offset': 0,
                        u'query_total': 5,
```

(continues on next page)

(continued from previous page)

```
                     u'total': 5},
   u'input_fields': [u'000000', u'000001', u'000002', u'000003'],
   u'model_count': {   u'logisticregression': 1, u'model': 8, u'total': 9},
   u'models': [   u'model/5afde64e8bf7d551fd005131',
                  u'model/5afde64f8bf7d551fd005134',
                  u'model/5afde6518bf7d551fd005137',
                  u'model/5afde6538bf7d551fd00513a',
                  u'logisticregression/5afde6558bf7d551fd00513d',
                  ...
                  u'model/5afde65a8bf7d551fd005149'],
   u'models_meta': {   u'count': 9, u'limit': 1000, u'offset': 0, u'total': 9},
   u'name': u'iris',
   u'name_options': u'9 total models (logisticregression: 1, model: 8), metric=max_
↪phi, model candidates=18, max. training time=300',
   u'objective_field': u'000004',
   u'objective_field_details': {   u'column_number': 4,
                                   u'datatype': u'string',
                                   u'name': u'species',
                                   u'optype': u'categorical',
                                   u'order': 4},
   u'objective_field_name': u'species',
   u'objective_field_type': u'categorical',
   u'objective_fields': [u'000004'],
   u'optiml': {   u'created_resources': {   u'dataset': 10,
                                            u'logisticregression': 11,
                                            u'logisticregression_evaluation': 11,
                                            u'model': 29,
                                            u'model_evaluation': 29},
                  u'datasets': [   {   u'id': u'dataset/5afde6488bf7d551ee00081c',
                                       u'name': u'iris',
                                       u'name_options': u'120 instances, 5 fields (1␣
↪categorical, 4 numeric), sample rate=0.8'},
                                   {   u'id': u'dataset/5afde6488bf7d551fd00511f',
                                       u'name': u'iris',
                                       u'name_options': u'30 instances, 5 fields (1␣
↪categorical, 4 numeric), sample rate=0.2, out of bag'},
                                   {   u'id': u'dataset/5afde6488bf7d551fe002e0f',
                                       u'name': u'iris',
                                       u'name_options': u'120 instances, 5 fields (1␣
↪categorical, 4 numeric), sample rate=0.8'},
                                   ...
                                   {   u'id': u'dataset/5afde64d8bf7d551fd00512e',
                                       u'name': u'iris',
                                       u'name_options': u'120 instances, 5 fields (1␣
↪categorical, 4 numeric), sample rate=0.8'}],
                  u'fields': {   u'000000': {   u'column_number': 0,
                                                u'datatype': u'double',
                                                u'name': u'sepal length',
                                                u'optype': u'numeric',
                                                u'order': 0,
                                                u'preferred': True,
                                                u'summary': {   u'bins': [   [   4.3,
                                                                                 1],
                                                                               ...
                                                                             [   7.9,
                                                                                 1]],
                                                                ...
```

(continues on next page)

(continued from previous page)

```
                                                                u'sum': 179.9,
                                                                u'sum_squares': 302.
↪33,
                                                                u'variance': 0.58101}
↪},
                                  u'000004': {   u'column_number': 4,
                                                 u'datatype': u'string',
                                                 u'name': u'species',
                                                 u'optype': u'categorical',
                                                 u'order': 4,
                                                 u'preferred': True,
                                                 u'summary': {   u'categories': [   [␣
↪  u'Iris-setosa',
                                                                                    ␣
↪  50],
                                                                                   [␣
↪  u'Iris-versicolor',
                                                                                    ␣
↪  50],
                                                                                   [␣
↪  u'Iris-virginica',
                                                                                    ␣
↪  50]],
                                                                u'missing_count': 0},
                                                 u'term_analysis': {   u'enabled':␣
↪True}}},
                  u'max_training_time': 300,
                  u'metric': u'max_phi',
                  u'model_types': [u'model', u'logisticregression'],
                  u'models': [   {   u'evaluation': {   u'id': u'evaluation/
↪5afde65c8bf7d551fd00514c',
                                                        u'info': {   u'accuracy': 0.
↪96667,
                                                                     u'average_area_
↪under_pr_curve': 0.97867,
                                                                     ...
                                                                     u'per_class_
↪statistics': [   {   u'accuracy': 1,
                                                                                    ␣
↪                u'area_under_pr_curve': 1,
                                                                                    ␣
↪                ...
                                                                                    ␣
↪                u'spearmans_rho': 0.82005}]},
                                                        u'metric_value': 0.95356,
                                                        u'metric_variance': 0.00079,
                                                        u'name': u'iris vs. iris',
                                                        u'name_options': u'279-node,␣
↪deterministic order, operating kind=probability'},
                                 u'evaluation_count': 3,
                                 u'id': u'model/5afde64e8bf7d551fd005131',
                                 u'importance': [   [   u'000002',
                                                        0.70997],
                                                    [   u'000003',
                                                        0.27289],
                                                    [   u'000000',
                                                        0.0106],
```

(continues on next page)

(continued from previous page)

```
                                                         [   u'000001',
                                                             0.00654]],
                                     u'kind': u'model',
                                     u'name': u'iris',
                                     u'name_options': u'279-node, deterministic order
↪'},
                                 {   u'evaluation': {   u'id': u'evaluation/
↪5afde65c8bf7d551fd00514f',
                                                        u'info': {   u'accuracy': 0.
↪93333,

                                                        ...
                                                         [   u'000001',
                                                             0.02133]],
                                     u'kind': u'model',
                                     u'name': u'iris',
                                     u'name_options': u'12-node, randomize,␣
↪deterministic order, balanced'}],
                  u'number_of_model_candidates': 18,
                  u'recent_evaluations': [   0.90764,
                                             0.94952,
                                             ...
                                             0.90427],
                  u'search_complete': True,
                  u'summary': {   u'logisticregression': {   u'best': u
↪'logisticregression/5afde6558bf7d551fd00513d',
                                                             u'count': 1},
                                  u'model': {   u'best': u'model/
↪5afde64e8bf7d551fd005131',
                                                u'count': 8}}},
    u'private': True,
    u'project': None,
    u'resource': u'optiml/5afde4a42a83475c1b0008a2',
    u'shared': False,
    u'size': 3686,
    u'source': u'source/5afdb6fb9252732d930009e5',
    u'source_status': True,
    u'status': {   u'code': 5,
                   u'elapsed': 448878.0,
                   u'message': u'The optiml has been created',
                   u'progress': 1},
    u'subscription': False,
    u'tags': [],
    u'test_dataset': None,
    u'type': 0,
    u'updated': u'2018-05-17T20:30:29.063000'}
```

CHAPTER 23

Fusions

A Fusion is a special type of composed resource for which all submodels satisfy the following constraints: they're all either classifications or regressions over the same kind of data or compatible fields, with the same objective field. Given those properties, a fusion can be considered a supervised model, and therefore one can predict with fusions and evaluate them. Ensembles can be viewed as a kind of fusion subject to the additional constraints that all its submodels are tree models that, moreover, have been built from the same base input data, but sampled in particular ways.

The model types allowed to be a submodel of a fusion are: deepnet, ensemble, fusion, model, logistic regression and linear regression.

The JSON structure for an Fusion is:

```
>>> api.pprint(fusion["object"])
{
    "category": 0,
    "code": 200,
    "configuration": null,
    "configuration_status": false,
    "created": "2018-05-09T20:11:05.821000",
    "credits_per_prediction": 0,
    "description": "",
    "fields_meta": {
        "count": 5,
        "limit": 1000,
        "offset": 0,
        "query_total": 5,
        "total": 5
    },
    "fusion": {
        "models": [
            {
                "id": "ensemble/5af272eb4e1727d378000050",
                "kind": "ensemble",
                "name": "Iris ensemble",
                "name_options": "boosted trees, 1999-node, 16-iteration,␣
↪deterministic order, balanced"
```

(continues on next page)

(continued from previous page)

```
            },
            {
                "id": "model/5af272fe4e1727d3780000d6",
                "kind": "model",
                "name": "Iris model",
                "name_options": "1999-node, pruned, deterministic order, balanced"
            },
            {
                "id": "logisticregression/5af272ff4e1727d3780000d9",
                "kind": "logisticregression",
                "name": "Iris LR",
                "name_options": "L2 regularized (c=1), bias, auto-scaled, missing␣
↪values, eps=0.001"
            }
        ]
    },
    "importance": {
        "000000": 0.05847,
        "000001": 0.03028,
        "000002": 0.13582,
        "000003": 0.4421
    },
    "model_count": {
        "ensemble": 1,
        "logisticregression": 1,
        "model": 1,
        "total": 3
    },
    "models": [
        "ensemble/5af272eb4e1727d378000050",
        "model/5af272fe4e1727d3780000d6",
        "logisticregression/5af272ff4e1727d3780000d9"
    ],
    "models_meta": {
        "count": 3,
        "limit": 1000,
        "offset": 0,
        "total": 3
    },
    "name": "iris",
    "name_options": "3 total models (ensemble: 1, logisticregression: 1, model: 1)",
    "number_of_batchpredictions": 0,
    "number_of_evaluations": 0,
    "number_of_predictions": 0,
    "number_of_public_predictions": 0,
    "objective_field": "000004",
    "objective_field_details": {
        "column_number": 4,
        "datatype": "string",
        "name": "species",
        "optype": "categorical",
        "order": 4
    },
    "objective_field_name": "species",
    "objective_field_type": "categorical",
    "objective_fields": [
        "000004"
```

(continues on next page)

(continued from previous page)

```
    ],
    "private": true,
    "project": null,
    "resource":"fusion/59af8107b8aa0965d5b61138",
    "shared": false,
    "status": {
        "code": 5,
        "elapsed": 8420,
        "message": "The fusion has been created",
        "progress": 1
    },
    "subscription": false,
    "tags": [],
    "type": 0,
    "updated": "2018-05-09T20:11:14.258000"
}
```

CHAPTER 24

PCAs

A PCA (Principal Component Analysis) resource fits a number of orthogonal projections (components) to maximally capture the variance in a dataset. This is a dimensional reduction technique, as it can be used to reduce the number of inputs for the modeling step. PCA models belong to the unsupervised class of models (there is no objective field).

The JSON structure for an PCA is:

```
{'code': 200,
 'error': None,
 'location': 'https://strato.dev.bigml.io/andromeda/pca/5c002572983efc0ac5000003',
 'object': {u'category': 0,
            u'code': 200,
            u'columns': 2,
            u'configuration': None,
            u'configuration_status': False,
            u'created': u'2018-11-29T17:44:18.359000',
            u'creator': u'merce',
            u'credits': 0.0,
            u'credits_per_prediction': 0.0,
            u'dataset': u'dataset/5c00256a983efc0acf000000',
            u'dataset_field_types': {u'categorical': 1,
                                     u'datetime': 0,
                                     u'items': 0,
                                     u'numeric': 0,
                                     u'preferred': 2,
                                     u'text': 1,
                                     u'total': 2},
            u'dataset_status': True,
            u'description': u'',
            u'excluded_fields': [],
            u'fields_meta': {u'count': 2,
                             u'limit': 1000,
                             u'offset': 0,
                             u'query_total': 2,
                             u'total': 2},
```

(continues on next page)

(continued from previous page)

```
            u'input_fields': [u'000000', u'000001'],
            u'locale': u'en-us',
            u'max_columns': 2,
            u'max_rows': 7,
            u'name': u'spam 4 words',
            u'name_options': u'standardized',
            u'number_of_batchprojections': 2,
            u'number_of_projections': 0,
            u'number_of_public_projections': 0,
            u'ordering': 0,
            u'out_of_bag': False,
            u'pca': {u'components': [[-0.64757,
                                      0.83392,
                                      0.1158,
                                      0.83481,
                                      ...
                                      -0.09426,
                                      -0.08544,
                                      -0.03457]],
                     u'cumulative_variance': [0.43667,
                                              0.74066,
                                              0.87902,
                                              0.98488,
                                              0.99561,
                                              1],
                     u'eigenvectors': [[-0.3894,
                                        0.50146,
                                        0.06963,
                                        ...
                                        -0.56542,
                                        -0.5125,
                                        -0.20734]],
                     u'fields': {u'000000': {u'column_number': 0,
                                             u'datatype': u'string',
                                             u'name': u'Type',
                                             ...
                                                                  u'token_mode': u'all',
                                                                  u'use_stopwords':␣
↪False}}},
                     u'pca_seed': u'2c249dda00fbf54ab4cdd850532a584f286af5b6',
                     u'standardized': True,
                     u'text_stats': {u'000001': {u'means': [0.71429,
                                                            0.71429,
                                                            0.42857,
                                                            0.28571],
                                                 u'standard_deviations': [0.75593,
                                                                          0.75593,
                                                                          0.53452,
                                                                          0.48795]}},
                     u'variance': [0.43667,
                                   0.30399,
                                   0.13837,
                                   0.10585,
                                   0.01073,
                                   0.00439]},
            u'price': 0.0,
            u'private': True,
```

(continues on next page)

(continued from previous page)

```
            u'project': None,
            u'range': None,
            u'replacement': False,
            u'resource': u'pca/5c002572983efc0ac5000003',
            u'rows': 7,
            u'sample_rate': 1.0,
            u'shared': False,
            u'size': 127,
            u'source': u'source/5c00255e983efc0acd00001b',
            u'source_status': True,
            u'status': {u'code': 5,
                        u'elapsed': 1571,
                        u'message': u'The pca has been created',
                        u'progress': 1},
            u'subscription': True,
            u'tags': [],
            u'type': 0,
            u'updated': u'2018-11-29T18:13:19.714000',
            u'white_box': False},
 'resource': u'pca/5c002572983efc0ac5000003'}
```

CHAPTER 25

WhizzML Resources

WhizzML is a Domain Specific Language that allows the definition and execution of ML-centric workflows. Its objective is allowing BigML users to define their own composite tasks, using as building blocks the basic resources provided by BigML itself. Using Whizzml they can be glued together using a higher order, functional, Turing-complete language. The WhizzML code can be stored and executed in BigML using three kinds of resources: `Scripts`, `Libraries` and `Executions`.

WhizzML `Scripts` can be executed in BigML's servers, that is, in a controlled, fully-scalable environment which takes care of their parallelization and fail-safe operation. Each execution uses an `Execution` resource to store the arguments and results of the process. WhizzML `Libraries` store generic code to be shared of reused in other WhizzML `Scripts`.

CHAPTER 26

Scripts

In BigML a `Script` resource stores WhizzML source code, and the results of its compilation. Once a WhizzML script is created, it's automatically compiled; if compilation succeeds, the script can be run, that is, used as the input for a WhizzML execution resource.

An example of a `script` that would create a `source` in BigML using the contents of a remote file is:

```
>>> from bigml.api import BigML
>>> api = BigML()
# creating a script directly from the source code. This script creates
# a source uploading data from an s3 repo. You could also create a
# a script by using as first argument the path to a .whizzml file which
# contains your source code.
>>> script = api.create_script( \
        "(create-source {\"remote\" \"s3://bigml-public/csv/iris.csv\"})")
>>> api.ok(script) # waiting for the script compilation to finish
>>> api.pprint(script['object'])
{   u'approval_status': 0,
    u'category': 0,
    u'code': 200,
    u'created': u'2016-05-18T16:54:05.666000',
    u'description': u'',
    u'imports': [],
    u'inputs': None,
    u'line_count': 1,
    u'locale': u'en-US',
    u'name': u'Script',
    u'number_of_executions': 0,
    u'outputs': None,
    u'price': 0.0,
    u'private': True,
    u'project': None,
    u'provider': None,
    u'resource': u'script/573c9e2db85eee23cd000489',
    u'shared': False,
```

(continues on next page)

(continued from previous page)

```
    u'size': 59,
    u'source_code': u'(create-source {"remote" "s3://bigml-public/csv/iris.csv"})',
    u'status': {   u'code': 5,
                   u'elapsed': 4,
                   u'message': u'The script has been created',
                   u'progress': 1.0},
    u'subscription': True,
    u'tags': [],
    u'updated': u'2016-05-18T16:54:05.850000',
    u'white_box': False}
```

A `script` allows to define some variables as `inputs`. In the previous example, no input has been defined, but we could modify our code to allow the user to set the remote file name as input:

```
>>> from bigml.api import BigML
>>> api = BigML()
>>> script = api.create_script( \
        "(create-source {\"remote\" my_remote_data})",
        {"inputs": [{"name": "my_remote_data",
                     "type": "string",
                     "default": "s3://bigml-public/csv/iris.csv",
                     "description": "Location of the remote data"}]})
```

The `script` can also use a `library` resource (please, see the `Libraries` section below for more details) by including its id in the `imports` attribute. Other attributes can be checked at the API Developers documentation for Scripts.

CHAPTER 27

Executions

To execute in BigML a compiled WhizzML `script` you need to create an `execution` resource. It's also possible to execute a pipeline of many compiled scripts in one request.

Each `execution` is run under its associated user credentials and its particular environment constaints. As `scripts` can be shared, you can execute the same `script` several times under different usernames by creating different `executions`.

As an example of `execution` resource, let's create one for the script in the previous section:

```
>>> from bigml.api import BigML
>>> api = BigML()
>>> execution = api.create_execution('script/573c9e2db85eee23cd000489')
>>> api.ok(execution) # waiting for the execution to finish
>>> api.pprint(execution['object'])
{   u'category': 0,
    u'code': 200,
    u'created': u'2016-05-18T16:58:01.613000',
    u'creation_defaults': {   },
    u'description': u'',
    u'execution': {   u'output_resources': [   {   u'code': 1,
                                                       u'id': u'source/
↪573c9f19b85eee23c600024a',
                                                       u'last_update': 1463590681854,
                                                       u'progress': 0.0,
                                                       u'state': u'queued',
                                                       u'task': u'Queuing job',
                                                       u'variable': u''}],
                      u'outputs': [],
                      u'result': u'source/573c9f19b85eee23c600024a',
                      u'results': [u'source/573c9f19b85eee23c600024a'],
                      u'sources': [[   u'script/573c9e2db85eee23cd000489',
                                         u'']],
                      u'steps': 16},
    u'inputs': None,
    u'locale': u'en-US',
```

(continues on next page)

(continued from previous page)

```
    u'name': u"Script's Execution",
    u'project': None,
    u'resource': u'execution/573c9f19b85eee23bd000125',
    u'script': u'script/573c9e2db85eee23cd000489',
    u'script_status': True,
    u'shared': False,
    u'status': {   u'code': 5,
                   u'elapsed': 249,
                   u'elapsed_times': {   u'in-progress': 247,
                                         u'queued': 62,
                                         u'started': 2},
                   u'message': u'The execution has been created',
                   u'progress': 1.0},
    u'subscription': True,
    u'tags': [],
    u'updated': u'2016-05-18T16:58:02.035000'}
```

An `execution` receives inputs, the ones defined in the `script` chosen to be executed, and generates a result. It can also generate outputs. As you can see, the execution resource contains information about the result of the execution, the resources that have been generated while executing and users can define some variables in the code to be exported as outputs. Please refer to the Developers documentation for Executions for details on how to define execution outputs.

CHAPTER 28

Libraries

The `library` resource in BigML stores a special kind of compiled Whizzml source code that only defines functions and constants. The `library` is intended as an import for executable scripts. Thus, a compiled library cannot be executed, just used as an import in other `libraries` and `scripts` (which then have access to all identifiers defined in the `library`).

As an example, we build a `library` to store the definition of two functions: `mu` and `g`. The first one adds one to the value set as argument and the second one adds two variables and increments the result by one.

```
>>> from bigml.api import BigML
>>> api = BigML()
>>> library = api.create_library( \
        "(define (mu x) (+ x 1)) (define (g z y) (mu (+ y z)))")
>>> api.ok(library) # waiting for the library compilation to finish
>>> api.pprint(library['object'])
{   u'approval_status': 0,
    u'category': 0,
    u'code': 200,
    u'created': u'2016-05-18T18:58:50.838000',
    u'description': u'',
    u'exports': [   {   u'name': u'mu', u'signature': [u'x']},
                    {   u'name': u'g', u'signature': [u'z', u'y']}],
    u'imports': [],
    u'line_count': 1,
    u'name': u'Library',
    u'price': 0.0,
    u'private': True,
    u'project': None,
    u'provider': None,
    u'resource': u'library/573cbb6ab85eee23c300018e',
    u'shared': False,
    u'size': 53,
    u'source_code': u'(define (mu x) (+ x 1)) (define (g z y) (mu (+ y z)))',
    u'status': {   u'code': 5,
                   u'elapsed': 2,
```

(continues on next page)

(continued from previous page)

```
                  u'message': u'The library has been created',
                  u'progress': 1.0},
    u'subscription': True,
    u'tags': [],
    u'updated': u'2016-05-18T18:58:52.432000',
    u'white_box': False}
```

Libraries can be imported in scripts. The `imports` attribute of a `script` can contain a list of `library` IDs whose defined functions and constants will be ready to be used throughout the `script`. Please, refer to the API Developers documentation for Libraries for more details.

CHAPTER 29

Creating Resources

Newly-created resources are returned in a dictionary with the following keys:

- **code**: If the request is successful you will get a `bigml.api.HTTP_CREATED` (201) status code. In asynchronous file uploading `api.create_source` calls, it will contain `bigml.api.HTTP_ACCEPTED` (202) status code. Otherwise, it will be one of the standard HTTP error codes detailed in the documentation.
- **resource**: The identifier of the new resource.
- **location**: The location of the new resource.
- **object**: The resource itself, as computed by BigML.
- **error**: If an error occurs and the resource cannot be created, it will contain an additional code and a description of the error. In this case, **location**, and **resource** will be `None`.

29.1 Statuses

Please, bear in mind that resource creation is almost always asynchronous (**predictions** are the only exception). Therefore, when you create a new source, a new dataset or a new model, even if you receive an immediate response from the BigML servers, the full creation of the resource can take from a few seconds to a few days, depending on the size of the resource and BigML's load. A resource is not fully created until its status is `bigml.api.FINISHED`. See the documentation on status codes for the listing of potential states and their semantics. So depending on your application you might need to import the following constants:

```
from bigml.api import WAITING
from bigml.api import QUEUED
from bigml.api import STARTED
from bigml.api import IN_PROGRESS
from bigml.api import SUMMARIZED
from bigml.api import FINISHED
from bigml.api import UPLOADING
from bigml.api import FAULTY
from bigml.api import UNKNOWN
from bigml.api import RUNNABLE
```

Usually, you will simply need to wait until the resource is in the `bigml.api.FINISHED` state for further processing. If that's the case, the easiest way is calling the `api.ok` method and passing as first argument the object that contains your resource:

```
from bigml.api import BigML
api = BigML() # creates a connection to BigML's API
source = api.create_source('my_file.csv') # creates a source object
api.ok(source) # checks that the source is finished and updates ``source``
```

In this code, `api.create_source` will probably return a non-finished `source` object. Then, `api.ok` will query its status and update the contents of the `source` variable with the retrieved information until it reaches a `bigml.api.FINISHED` or `bigml.api.FAILED` status.

If you don't want the contents of the variable to be updated, you can also use the `check_resource` function:

```
check_resource(resource, api.get_source)
```

that will constantly query the API until the resource gets to a FINISHED or FAULTY state, or can also be used with `wait_time` (in seconds) and `retries` arguments to control the polling:

```
check_resource(resource, api.get_source, wait_time=2, retries=20)
```

The `wait_time` value is used as seed to a wait interval that grows exponentially with the number of retries up to the given `retries` limit.

However, in other scenarios you might need to control the complete evolution of the resource, not only its final states. There, you can query the status of any resource with the `status` method, which simply returns its value and does not update the contents of the associated variable:

```
api.status(source)
api.status(dataset)
api.status(model)
api.status(prediction)
api.status(evaluation)
api.status(ensemble)
api.status(batch_prediction)
api.status(cluster)
api.status(centroid)
api.status(batch_centroid)
api.status(anomaly)
api.status(anomaly_score)
api.status(batch_anomaly_score)
api.status(sample)
api.status(correlation)
api.status(statistical_test)
api.status(logistic_regression)
api.status(association)
api.status(association_set)
api.status(topic_model)
api.status(topic_distribution)
api.status(batch_topic_distribution)
api.status(time_series)
api.status(forecast)
api.status(optiml)
api.status(fusion)
api.status(pca)
api.status(projection)
api.status(batch_projection)
```

(continues on next page)

(continued from previous page)

```
api.status(linear_regression)
api.status(script)
api.status(execution)
api.status(library)
```

Remember that, consequently, you will need to retrieve the resources explicitly in your code to get the updated information.

29.2 Projects

A special kind of resource is `project`. Projects are repositories for resources, intended to fulfill organizational purposes. Each project can contain any other kind of resource, but the project that a certain resource belongs to is determined by the one used in the `source` they are generated from. Thus, when a source is created and assigned a certain `project_id`, the rest of resources generated from this source will remain in this project.

The REST calls to manage the `project` resemble the ones used to manage the rest of resources. When you create a `project`:

```
from bigml.api import BigML
api = BigML()

project = api.create_project({'name': 'my first project'})
```

the resulting resource is similar to the rest of resources, although shorter:

```
{'code': 201,
 'resource': u'project/54a1bd0958a27e3c4c0002f0',
 'location': 'http://bigml.io/andromeda/project/54a1bd0958a27e3c4c0002f0',
 'object': {u'category': 0,
            u'updated': u'2014-12-29T20:43:53.060045',
            u'resource': u'project/54a1bd0958a27e3c4c0002f0',
            u'name': u'my first project',
            u'created': u'2014-12-29T20:43:53.060013',
            u'tags': [],
            u'private': True,
            u'dev': None,
            u'description': u''},
 'error': None}
```

and you can use its project id to get, update or delete it:

```
project = api.get_project('project/54a1bd0958a27e3c4c0002f0')
api.update_project(project['resource'],
                   {'description': 'This is my first project'})

api.delete_project(project['resource'])
```

Important: Deleting a non-empty project will also delete **all resources** assigned to it, so please be extra-careful when using the `api.delete_project` call.

29.3 Creating sources

To create a source from a local data file, you can use the `create_source` method. The only required parameter is the path to the data file (or file-like object). You can use a second optional parameter to specify any of the options for source creation described in the BigML API documentation.

Here's a sample invocation:

```
from bigml.api import BigML
api = BigML()

source = api.create_source('./data/iris.csv',
    {'name': 'my source', 'source_parser': {'missing_tokens': ['?']}})
```

or you may want to create a source from a file in a remote location:

```
source = api.create_source('s3://bigml-public/csv/iris.csv',
    {'name': 'my remote source', 'source_parser': {'missing_tokens': ['?']}})
```

or maybe reading the content from stdin:

```
content = StringIO.StringIO(sys.stdin.read())
source = api.create_source(content,
    {'name': 'my stdin source', 'source_parser': {'missing_tokens': ['?']}})
```

or using data stored in a local python variable. The following example shows the two accepted formats:

```
local = [['a', 'b', 'c'], [1, 2, 3], [4, 5, 6]]
local2 = [{'a': 1, 'b': 2, 'c': 3}, {'a': 4, 'b': 5, 'c': 6}]
source = api.create_source(local, {'name': 'inline source'})
```

As already mentioned, source creation is asynchronous. In both these examples, the `api.create_source` call returns once the file is uploaded. Then `source` will contain a resource whose status code will be either `WAITING` or `QUEUED`.

For local data files you can go one step further and use asynchronous uploading:

```
source = api.create_source('./data/iris.csv',
    {'name': 'my source', 'source_parser': {'missing_tokens': ['?']}},
    async=True)
```

In this case, the call fills *source* immediately with a primary resource like:

```
{'code': 202,
 'resource': None,
 'location': None,
 'object': {'status':
               {'progress': 0.99,
                'message': 'The upload is in progress',
                'code': 6}},
 'error': None}
```

where the `source['object']` status is set to `UPLOADING` and its `progress` is periodically updated with the current uploading progress ranging from 0 to 1. When upload completes, this structure will be replaced by the real resource info as computed by BigML. Therefore source's status will eventually be (as it is in the synchronous upload case) `WAITING` or `QUEUED`.

You can retrieve the updated status at any time using the corresponding get method. For example, to get the status of our source we would use:

```
api.status(source)
```

29.4 Creating datasets

Once you have created a source, you can create a dataset. The only required argument to create a dataset is a source id. You can add all the additional arguments accepted by BigML and documented in the Datasets section of the Developer's documentation.

For example, to create a dataset named "my dataset" with the first 1024 bytes of a source, you can submit the following request:

```
dataset = api.create_dataset(source, {"name": "my dataset", "size": 1024})
```

Upon success, the dataset creation job will be queued for execution, and you can follow its evolution using `api.status(dataset)`.

As for the rest of resources, the create method will return an incomplete object, that can be updated by issuing the corresponding `api.get_dataset` call until it reaches a `FINISHED` status. Then you can export the dataset data to a CSV file using:

```
api.download_dataset('dataset/526fc344035d071ea3031d75',
    filename='my_dir/my_dataset.csv')
```

You can also extract samples from an existing dataset and generate a new one with them using the `api.create_dataset` method. The first argument should be the origin dataset and the rest of arguments that set the range or the sampling rate should be passed as a dictionary. For instance, to create a new dataset extracting the 80% of instances from an existing one, you could use:

```
dataset = api.create_dataset(origin_dataset, {"sample_rate": 0.8})
```

Similarly, if you want to split your source into training and test datasets, you can set the *sample_rate* as before to create the training dataset and use the *out_of_bag* option to assign the complementary subset of data to the test dataset. If you set the *seed* argument to a value of your choice, you will ensure a deterministic sampling, so that each time you execute this call you will get the same datasets as a result and they will be complementary:

```
origin_dataset = api.create_dataset(source)
train_dataset = api.create_dataset(
    origin_dataset, {"name": "Dataset Name | Training",
                     "sample_rate": 0.8, "seed": "my seed"})
test_dataset = api.create_dataset(
    origin_dataset, {"name": "Dataset Name | Test",
                     "sample_rate": 0.8, "seed": "my seed",
                     "out_of_bag": True})
```

Sometimes, like for time series evaluations, it's important that the data in your train and test datasets is ordered. In this case, the split cannot be done at random. You will need to start from an ordered dataset and decide the ranges devoted to training and testing using the `range` attribute:

```
origin_dataset = api.create_dataset(source)
train_dataset = api.create_dataset(
    origin_dataset, {"name": "Dataset Name | Training",
                     "range": [1, 80]})
```

(continues on next page)

(continued from previous page)

```
test_dataset = api.create_dataset(
    origin_dataset, {"name": "Dataset Name | Test",
                     "range": [81, 100]})
```

It is also possible to generate a dataset from a list of datasets (multidataset):

```
dataset1 = api.create_dataset(source1)
dataset2 = api.create_dataset(source2)
multidataset = api.create_dataset([dataset1, dataset2])
```

Clusters can also be used to generate datasets containing the instances grouped around each centroid. You will need the cluster id and the centroid id to reference the dataset to be created. For instance,

```
cluster = api.create_cluster(dataset)
cluster_dataset_1 = api.create_dataset(cluster,
                                       args={'centroid': '000000'})
```

would generate a new dataset containing the subset of instances in the cluster associated to the centroid id `000000`.

29.5 Creating models

Once you have created a dataset you can create a model from it. If you don't select one, the model will use the last field of the dataset as objective field. The only required argument to create a model is a dataset id. You can also include in the request all the additional arguments accepted by BigML and documented in the Models section of the Developer's documentation.

For example, to create a model only including the first two fields and the first 10 instances in the dataset, you can use the following invocation:

```
model = api.create_model(dataset, {
    "name": "my model", "input_fields": ["000000", "000001"], "range": [1, 10]})
```

Again, the model is scheduled for creation, and you can retrieve its status at any time by means of `api.status(model)`.

Models can also be created from lists of datasets. Just use the list of ids as the first argument in the api call

```
model = api.create_model([dataset1, dataset2], {
    "name": "my model", "input_fields": ["000000", "000001"], "range": [1, 10]})
```

And they can also be generated as the result of a clustering procedure. When a cluster is created, a model that predicts if a certain instance belongs to a concrete centroid can be built by providing the cluster and centroid ids:

```
model = api.create_model(cluster, {
    "name": "model for centroid 000001", "centroid": "000001"})
```

if no centroid id is provided, the first one appearing in the cluster is used.

29.6 Creating clusters

If your dataset has no fields showing the objective information to predict for the training data, you can still build a cluster that will group similar data around some automatically chosen points (centroids). Again, the only required argument to create a cluster is the dataset id. You can also include in the request all the additional arguments accepted by BigML and documented in the Clusters section of the Developer's documentation.

Let's create a cluster from a given dataset:

```
cluster = api.create_cluster(dataset, {"name": "my cluster",
                                       "k": 5})
```

that will create a cluster with 5 centroids.

29.7 Creating anomaly detectors

If your problem is finding the anomalous data in your dataset, you can build an anomaly detector, that will use iforest to single out the anomalous records. Again, the only required argument to create an anomaly detector is the dataset id. You can also include in the request all the additional arguments accepted by BigML and documented in the Anomaly detectors section of the Developer's documentation.

Let's create an anomaly detector from a given dataset:

```
anomaly = api.create_anomaly(dataset, {"name": "my anomaly"})
```

that will create an anomaly resource with a *top_anomalies* block of the most anomalous points.

29.8 Creating associations

To find relations between the field values you can create an association discovery resource. The only required argument to create an association is a dataset id. You can also include in the request all the additional arguments accepted by BigML and documented in the Association section of the Developer's documentation.

For example, to create an association only including the first two fields and the first 10 instances in the dataset, you can use the following invocation:

```
association = api.create_association(dataset, { \
    "name": "my association", "input_fields": ["000000", "000001"], \
    "range": [1, 10]})
```

Again, the association is scheduled for creation, and you can retrieve its status at any time by means of `api.status(association)`.

Associations can also be created from lists of datasets. Just use the list of ids as the first argument in the api call

```
association = api.create_association([dataset1, dataset2], { \
    "name": "my association", "input_fields": ["000000", "000001"], \
    "range": [1, 10]})
```

29.9 Creating topic models

To find which topics do your documents refer to you can create a topic model. The only required argument to create a topic model is a dataset id. You can also include in the request all the additional arguments accepted by BigML and documented in the Topic Model section of the Developer's documentation.

For example, to create a topic model including exactly 32 topics you can use the following invocation:

```
topic_model = api.create_topic_model(dataset, { \
    "name": "my topics", "number_of_topics": 32})
```

Again, the topic model is scheduled for creation, and you can retrieve its status at any time by means of `api.status(topic_model)`.

Topic models can also be created from lists of datasets. Just use the list of ids as the first argument in the api call

```
topic_model = api.create_topic_model([dataset1, dataset2], { \
    "name": "my topics", "number_of_topics": 32})
```

29.10 Creating time series

To forecast the behaviour of any numeric variable that depends on its historical records you can use a time series. The only required argument to create a time series is a dataset id. You can also include in the request all the additional arguments accepted by BigML and documented in the Time Series section of the Developer's documentation.

For example, to create a time series including a forecast of 10 points for the numeric values you can use the following invocation:

```
time_series = api.create_time_series(dataset, { \
    "name": "my time series", "horizon": 10})
```

Again, the time series is scheduled for creation, and you can retrieve its status at any time by means of `api.status(time_series)`.

Time series also be created from lists of datasets. Just use the list of ids as the first argument in the api call

```
time_series = api.create_time_series([dataset1, dataset2], { \
    "name": "my time series", "horizon": 10})
```

29.11 Creating OptiML

To create an OptiML, the only required argument is a dataset id. You can also include in the request all the additional arguments accepted by BigML and documented in the OptiML section of the Developer's documentation.

For example, to create an OptiML which optimizes the accuracy of the model you can use the following method

```
optiml = api.create_optiml(dataset, { \
    "name": "my optiml", "metric": "accuracy"})
```

The OptiML is then scheduled for creation, and you can retrieve its status at any time by means of `api.status(optiml)`.

29.12 Creating Fusion

To create a Fusion, the only required argument is a list of models. You can also include in the request all the additional arguments accepted by BigML and documented in the Fusion section of the Developer's documentation.

For example, to create a Fusion you can use this connection method:

```
fusion = api.create_fusion(["model/5af06df94e17277501000010",
                            "model/5af06df84e17277502000019",
                            "deepnet/5af06df84e17277502000016",
                            "ensemble/5af06df74e1727750100000d"],
                            {"name": "my fusion"})
```

The Fusion is then scheduled for creation, and you can retrieve its status at any time by means of `api.status(fusion)`.

Fusions can also be created by assigning some weights to each model in the list. In this case, the argument for the create call will be a list of dictionaries that contain the `id` and `weight` keys:

```
fusion = api.create_fusion([{"id": "model/5af06df94e17277501000010",
                             "weight": 10},
                            {"id": "model/5af06df84e17277502000019",
                             "weight": 20},
                            {"id": "deepnet/5af06df84e17277502000016",
                             "weight": 5}],
                            {"name": "my weighted fusion"})
```

29.13 Creating predictions

You can now use the model resource identifier together with some input parameters to ask for predictions, using the `create_prediction` method. You can also give the prediction a name:

```
prediction = api.create_prediction(model,
                                   {"sepal length": 5,
                                    "sepal width": 2.5},
                                    {"name": "my prediction"})
```

To see the prediction you can use `pprint`:

```
api.pprint(prediction)
```

Predictions can be created using any supervised model (model, ensemble, logistic regression, linear regression, deepnet and fusion) as first argument.

29.14 Creating centroids

To obtain the centroid associated to new input data, you can now use the `create_centroid` method. Give the method a cluster identifier and the input data to obtain the centroid. You can also give the centroid predicition a name:

```
centroid = api.create_centroid(cluster,
                               {"pregnancies": 0,
                                "plasma glucose": 118,
                                "blood pressure": 84,
                                "triceps skin thickness": 47,
                                "insulin": 230,
                                "bmi": 45.8,
                                "diabetes pedigree": 0.551,
                                "age": 31,
                                "diabetes": "true"},
                                {"name": "my centroid"})
```

29.15 Creating anomaly scores

To obtain the anomaly score associated to new input data, you can now use the `create_anomaly_score` method. Give the method an anomaly detector identifier and the input data to obtain the score:

```
anomaly_score = api.create_anomaly_score(anomaly, {"src_bytes": 350},
                                         args={"name": "my score"})
```

29.16 Creating association sets

Using the association resource, you can obtain the consequent items associated by its rules to your input data. These association sets can be obtained calling the `create_association_set` method. The first argument is the association ID or object and the next one is the input data.

```
association_set = api.create_association_set( \
    association, {"genres": "Action$Adventure"}, \
    args={"name": "my association set"})
```

29.17 Creating topic distributions

To obtain the topic distributions associated to new input data, you can now use the `create_topic_distribution` method. Give the method a topic model identifier and the input data to obtain the score:

```
topic_distribution = api.create_topic_distribution( \
    topic_model,
    {"Message": "The bubble exploded in 2007."},
    args={"name": "my topic distribution"})
```

29.18 Creating forecasts

To obtain the forecast associated to a numeric variable, you can now use the `create_forecast` method. Give the method a time series identifier and the input data to obtain the forecast:

```
forecast = api.create_forecast( \
    time_series,
    {"Final": {"horizon": 10}})
```

29.19 Creating projections

You can now use the PCA resource identifier together with some input parameters to ask for the corresponding projections, using the `create_projection` method. You can also give the projection a name:

```
projection = api.create_projection(pca,
                                   {"sepal length": 5,
                                    "sepal width": 2.5},
                                    {"name": "my projection"})
```

29.20 Creating evaluations

Once you have created a supervised learning model, you can measure its perfomance by running a dataset of test data through it and comparing its predictions to the objective field real values. Thus, the required arguments to create an evaluation are model id and a dataset id. You can also include in the request all the additional arguments accepted by BigML and documented in the Evaluations section of the Developer's documentation.

For instance, to evaluate a previously created model using an existing dataset you can use the following call:

```
evaluation = api.create_evaluation(model, dataset, {
    "name": "my model"})
```

Again, the evaluation is scheduled for creation and `api.status(evaluation)` will show its state.

Evaluations can also check the ensembles' performance. To evaluate an ensemble you can do exactly what we just did for the model case, using the ensemble object instead of the model as first argument:

```
evaluation = api.create_evaluation(ensemble, dataset)
```

Evaluations can be created using any supervised model (including time series) as first argument.

29.21 Creating ensembles

To improve the performance of your predictions, you can create an ensemble of models and combine their individual predictions. The only required argument to create an ensemble is the dataset id:

```
ensemble = api.create_ensemble('dataset/5143a51a37203f2cf7000972')
```

BigML offers three kinds of ensembles. Two of them are known as `Decision Forests` because they are built as collections of `Decision trees` whose predictions are aggregated using different combiners (`plurality`, `confidence weighted`, `probability weighted`) or setting a `threshold` to issue the ensemble's prediction. All `Decision Forests` use bagging to sample the data used to build the underlying models.

As an example of how to create a `Decision Forest` with *20* models, you only need to provide the dataset ID that you want to build the ensemble from and the number of models:

```
args = {'number_of_models': 20}
ensemble = api.create_ensemble('dataset/5143a51a37203f2cf7000972', args)
```

If no `number_of_models` is provided, the ensemble will contain 10 models.

`Random Decision Forests` fall also into the `Decision Forest` category, but they only use a subset of the fields chosen at random at each split. To create this kind of ensemble, just use the `randomize` option:

```
args = {'number_of_models': 20, 'randomize': True}
ensemble = api.create_ensemble('dataset/5143a51a37203f2cf7000972', args)
```

The third kind of ensemble is `Boosted Trees`. This type of ensemble uses quite a different algorithm. The trees used in the ensemble don't have as objective field the one you want to predict, and they don't aggregate the underlying models' votes. Instead, the goal is adjusting the coefficients of a function that will be used to predict. The models' objective is, therefore, the gradient that minimizes the error of the predicting function (when comparing its output with the real values). The process starts with some initial values and computes these gradients. Next step uses the previous fields plus the last computed gradient field as the new initial state for the next iteration. Finally, it stops when the error is smaller than a certain threshold or iterations reach a user-defined limit. In classification problems, every category in the ensemble's objective field would be associated with a subset of the `Boosted Trees`. The objective of each subset of trees is adjustig the function to the probability of belonging to this particular category.

In order to build an ensemble of `Boosted Trees` you need to provide the `boosting` attributes. You can learn about the existing attributes in the ensembles' section of the API documentation, but a typical attribute to be set would be the maximum number of iterations:

```
args = {'boosting': {'iterations': 20}}
ensemble = api.create_ensemble('dataset/5143a51a37203f2cf7000972', args)
```

29.22 Creating linear regressions

For regression problems, you can choose also linear regressions to model your data. Linear regressions expect the predicted value for the objective field to be computable as a linear combination of the predictors.

As the rest of models, linear regressions can be created from a dataset by calling the corresponding create method:

```
linear_regression = api.create_linear_regression( \
    'dataset/5143a51a37203f2cf7000972',
    {"name": "my linear regression",
     "objective_field": "my_objective_field"})
```

In this example, we created a linear regression named `my linear regression` and set the objective field to be `my_objective_field`. Other arguments, like `bias`, can also be specified as attributes in arguments dictionary at creation time. Particularly for categorical fields, there are three different available *field_codings*' options (`contrast`, `other` or the `dummy` default coding). For a more detailed description of the `field_codings` attribute and its syntax, please see the Developers API Documentation.

29.23 Creating logistic regressions

For classification problems, you can choose also logistic regressions to model your data. Logistic regressions compute a probability associated to each class in the objective field. The probability is obtained using a logistic function, whose argument is a linear combination of the field values.

As the rest of models, logistic regressions can be created from a dataset by calling the corresponding create method:

```
logistic_regression = api.create_logistic_regression( \
    'dataset/5143a51a37203f2cf7000972',
    {"name": "my logistic regression",
     "objective_field": "my_objective_field"})
```

In this example, we created a logistic regression named `my logistic regression` and set the objective field to be `my_objective_field`. Other arguments, like `bias`, `missing_numerics` and `c` can also be specified as attributes in arguments dictionary at creation time. Particularly for categorical fields, there are four different available *field_codings*' options (`dummy`, `contrast`, `other` or the `one-hot` default coding). For a more detailed description of the `field_codings` attribute and its syntax, please see the Developers API Documentation.

29.24 Creating deepnets

Deepnets can also solve classification and regression problems. Deepnets are an optimized version of Deep Neural Networks, a class of machine-learned models inspired by the neural circuitry of the human brain. In these classifiers, the input features are fed to a group of "nodes" called a "layer". Each node is essentially a function on the input that transforms the input features into another value or collection of values. Then the entire layer transforms an input vector into a new "intermediate" feature vector. This new vector is fed as input to another layer of nodes. This process continues layer by layer, until we reach the final "output" layer of nodes, where the output is the network's prediction: an array of per-class probabilities for classification problems or a single, real value for regression problems.

Deepnets predictions compute a probability associated to each class in the objective field for classification problems. As the rest of models, deepnets can be created from a dataset by calling the corresponding create method:

```
deepnet = api.create_deepnet( \
    'dataset/5143a51a37203f2cf7000972',
    {"name": "my deepnet",
     "objective_field": "my_objective_field"})
```

In this example, we created a deepnet named `my deepnet` and set the objective field to be `my_objective_field`. Other arguments, like `number_of_hidden_layers`, `learning_rate` and `missing_numerics` can also be specified as attributes in an arguments dictionary at creation time. For a more detailed description of the available attributes and its syntax, please see the Developers API Documentation.

29.25 Creating PCAs

In order to reduce the number of features used in the modeling steps, you can use a PCA (Principal Component Analysis) to find out the best combination of features that describe the variance of your data. As the rest of models, PCAs can be created from a dataset by calling the corresponding create method:

```
pca = api.create_pca( \
    'dataset/5143a51a37203f2cf7000972',
    {"name": "my PCA"})
```

In this example, we created a PCA named `my PCA`. Other arguments, like `standardized` can also be specified as attributes in arguments dictionary at creation time. Please see the Developers API Documentation.

29.26 Creating batch predictions

We have shown how to create predictions individually, but when the amount of predictions to make increases, this procedure is far from optimal. In this case, the more efficient way of predicting remotely is to create a dataset containing the input data you want your model to predict from and to give its id and the one of the model to the `create_batch_prediction` api call:

```
batch_prediction = api.create_batch_prediction(model, dataset, {
    "name": "my batch prediction", "all_fields": True,
    "header": True,
    "confidence": True})
```

In this example, setting `all_fields` to true causes the input data to be included in the prediction output, `header` controls whether a headers line is included in the file or not and `confidence` set to true causes the confidence of the prediction to be appended. If none of these arguments is given, the resulting file will contain the name of the objective field as a header row followed by the predictions.

As for the rest of resources, the create method will return an incomplete object, that can be updated by issuing the corresponding `api.get_batch_prediction` call until it reaches a `FINISHED` status. Then you can download the created predictions file using:

```
api.download_batch_prediction('batchprediction/526fc344035d071ea3031d70',
    filename='my_dir/my_predictions.csv')
```

that will copy the output predictions to the local file given in `filename`. If no `filename` is provided, the method returns a file-like object that can be read as a stream:

```
CHUNK_SIZE = 1024
response = api.download_batch_prediction(
    'batchprediction/526fc344035d071ea3031d70')
chunk = response.read(CHUNK_SIZE)
if chunk:
    print chunk
```

The output of a batch prediction can also be transformed to a source object using the `source_from_batch_prediction` method in the api:

```
api.source_from_batch_prediction(
    'batchprediction/526fc344035d071ea3031d70',
    args={'name': 'my_batch_prediction_source'})
```

This code will create a new source object, that can be used again as starting point to generate datasets.

29.27 Creating batch centroids

As described in the previous section, it is also possible to make centroids' predictions in batch. First you create a dataset containing the input data you want your cluster to relate to a centroid. The `create_batch_centroid` call will need the id of the input data dataset and the cluster used to assign a centroid to each instance:

```
batch_centroid = api.create_batch_centroid(cluster, dataset, {
    "name": "my batch centroid", "all_fields": True,
    "header": True})
```

29.28 Creating batch anomaly scores

Input data can also be assigned an anomaly score in batch. You train an anomaly detector with your training data and then build a dataset from your input data. The `create_batch_anomaly_score` call will need the id of the dataset and of the anomaly detector to assign an anomaly score to each input data instance:

```
batch_anomaly_score = api.create_batch_anomaly_score(anomaly, dataset, {
    "name": "my batch anomaly score", "all_fields": True,
    "header": True})
```

29.29 Creating batch topic distributions

Input data can also be assigned a topic distribution in batch. You train a topic model with your training data and then build a dataset from your input data. The `create_batch_topic_distribution` call will need the id of the dataset and of the topic model to assign a topic distribution to each input data instance:

```
batch_topic_distribution = api.create_batch_topic_distribution( \
    topic_model, dataset, {
    "name": "my batch topic distribution", "all_fields": True,
    "header": True})
```

29.30 Creating batch projections

Input data can also be assigned a projection in batch. You train a PCA with your training data and then build a dataset from your input data. The `create_batch_projection` call will need the id of the input data dataset and of the PCA to compute the projection that corresponds to each input data instance:

```
batch_projection = api.create_batch_projection( \
    pca, dataset, {
    "name": "my batch pca", "all_fields": True,
    "header": True})
```

CHAPTER 30

Reading Resources

When retrieved individually, resources are returned as a dictionary identical to the one you get when you create a new resource. However, the status code will be `bigml.api.HTTP_OK` if the resource can be retrieved without problems, or one of the HTTP standard error codes otherwise.

CHAPTER 31

Listing Resources

You can list resources with the appropriate api method:

```
api.list_sources()
api.list_datasets()
api.list_models()
api.list_predictions()
api.list_evaluations()
api.list_ensembles()
api.list_batch_predictions()
api.list_clusters()
api.list_centroids()
api.list_batch_centroids()
api.list_anomalies()
api.list_anomaly_scores()
api.list_batch_anomaly_scores()
api.list_projects()
api.list_samples()
api.list_correlations()
api.list_statistical_tests()
api.list_logistic_regressions()
api.list_linear_regressions()
api.list_associations()
api.list_association_sets()
api.list_topic_models()
api.list_topic_distributions()
api.list_batch_topic_distributions()
api.list_time_series()
api.list_deepnets()
api.list_fusions()
api.list_pcas()
api.list_projections()
api.list_batch_projections()
api.list_forecasts()
api.list_scripts()
```

(continues on next page)

(continued from previous page)

```
api.list_libraries()
api.list_executions()
```

you will receive a dictionary with the following keys:

- **code**: If the request is successful you will get a `bigml.api.HTTP_OK` (200) status code. Otherwise, it will be one of the standard HTTP error codes. See BigML documentation on status codes for more info.
- **meta**: A dictionary including the following keys that can help you paginate listings:
  - **previous**: Path to get the previous page or `None` if there is no previous page.
  - **next**: Path to get the next page or `None` if there is no next page.
  - **offset**: How far off from the first entry in the resources is the first one listed in the resources key.
  - **limit**: Maximum number of resources that you will get listed in the resources key.
  - **total_count**: The total number of resources in BigML.
- **objects**: A list of resources as returned by BigML.
- **error**: If an error occurs and the resource cannot be created, it will contain an additional code and a description of the error. In this case, **meta**, and **resources** will be `None`.

31.1 Filtering Resources

You can filter resources in listings using the syntax and fields labeled as *filterable* in the BigML documentation for each resource.

A few examples:

31.1.1 Ids of the first 5 sources created before April 1st, 2012

```
[source['resource'] for source in
  api.list_sources("limit=5;created__lt=2012-04-1")['objects']]
```

31.1.2 Name of the first 10 datasets bigger than 1MB

```
[dataset['name'] for dataset in
  api.list_datasets("limit=10;size__gt=1048576")['objects']]
```

31.1.3 Name of models with more than 5 fields (columns)

```
[model['name'] for model in api.list_models("columns__gt=5")['objects']]
```

31.1.4 Ids of predictions whose model has not been deleted

```
[prediction['resource'] for prediction in
  api.list_predictions("model_status=true")['objects']]
```

31.2 Ordering Resources

You can order resources in listings using the syntax and fields labeled as *sortable* in the BigML documentation for each resource.

A few examples:

31.2.1 Name of sources ordered by size

```
[source['name'] for source in api.list_sources("order_by=size")['objects']]
```

31.2.2 Number of instances in datasets created before April 1st, 2012 ordered by size

```
[dataset['rows'] for dataset in
  api.list_datasets("created__lt=2012-04-1;order_by=size")['objects']]
```

31.2.3 Model ids ordered by number of predictions (in descending order).

```
[model['resource'] for model in
  api.list_models("order_by=-number_of_predictions")['objects']]
```

31.2.4 Name of predictions ordered by name.

```
[prediction['name'] for prediction in
  api.list_predictions("order_by=name")['objects']]
```

CHAPTER 32

Updating Resources

When you update a resource, it is returned in a dictionary exactly like the one you get when you create a new one. However the status code will be `bigml.api.HTTP_ACCEPTED` if the resource can be updated without problems or one of the HTTP standard error codes otherwise.

```
api.update_source(source, {"name": "new name"})
api.update_dataset(dataset, {"name": "new name"})
api.update_model(model, {"name": "new name"})
api.update_prediction(prediction, {"name": "new name"})
api.update_evaluation(evaluation, {"name": "new name"})
api.update_ensemble(ensemble, {"name": "new name"})
api.update_batch_prediction(batch_prediction, {"name": "new name"})
api.update_cluster(cluster, {"name": "new name"})
api.update_centroid(centroid, {"name": "new name"})
api.update_batch_centroid(batch_centroid, {"name": "new name"})
api.update_anomaly(anomaly, {"name": "new name"})
api.update_anomaly_score(anomaly_score, {"name": "new name"})
api.update_batch_anomaly_score(batch_anomaly_score, {"name": "new name"})
api.update_project(project, {"name": "new name"})
api.update_correlation(correlation, {"name": "new name"})
api.update_statistical_test(statistical_test, {"name": "new name"})
api.update_logistic_regression(logistic_regression, {"name": "new name"})
api.update_linear_regression(linear_regression, {"name": "new name"})
api.update_association(association, {"name": "new name"})
api.update_association_set(association_set, {"name": "new name"})
api.update_topic_model(topic_model, {"name": "new name"})
api.update_topic_distribution(topic_distribution, {"name": "new name"})
api.update_batch_topic_distribution(\
    batch_topic_distribution, {"name": "new name"})
api.update_time_series(\
    time_series, {"name": "new name"})
api.update_forecast(\
    forecast, {"name": "new name"})
api.update_deepnet(deepnet, {"name": "new name"})
api.update_fusion(fusion, {"name": "new name"})
```

(continues on next page)

(continued from previous page)

```
api.update_pca(pca, {"name": "new name"})
api.update_projection(projection, {"name": "new name"})
api.update_batch_projection(batch_projection, {"name": "new name"})
api.update_script(script, {"name": "new name"})
api.update_library(library, {"name": "new name"})
api.update_execution(execution, {"name": "new name"})
```

Updates can change resource general properties, such as the `name` or `description` attributes of a dataset, or specific properties, like the `missing tokens` (strings considered as missing values). As an example, let's say that your source has a certain field whose contents are numeric integers. BigML will assign a numeric type to the field, but you might want it to be used as a categorical field. You could change its type to `categorical` by calling:

```
api.update_source(source, \
    {"fields": {"000001": {"optype": "categorical"}}})
```

where `000001` is the field id that corresponds to the updated field.

Another usually needed update is changing a fields' `non-preferred` attribute, so that it can be used in the modeling process:

```
api.update_dataset(dataset, {"fields": {"000001": {"preferred": True}}})
```

where you would be setting as `preferred` the field whose id is `000001`.

You may also want to change the name of one of the clusters found in your clustering:

```
api.update_cluster(cluster, \
    {"clusters": {"000001": {"name": "my cluster"}}})
```

which is changing the name of the cluster whose centroid id is `000001` to `my_cluster`. Or, similarly, changing the name of one detected topic:

```
api.update_topic_model(topic_model, \
    {"topics": {"000001": {"name": "my topic"}}})
```

You will find detailed information about the updatable attributes of each resource in BigML developer's documentation.

CHAPTER 33

Deleting Resources

Resources can be deleted individually using the corresponding method for each type of resource.

```
api.delete_source(source)
api.delete_dataset(dataset)
api.delete_model(model)
api.delete_prediction(prediction)
api.delete_evaluation(evaluation)
api.delete_ensemble(ensemble)
api.delete_batch_prediction(batch_prediction)
api.delete_cluster(cluster)
api.delete_centroid(centroid)
api.delete_batch_centroid(batch_centroid)
api.delete_anomaly(anomaly)
api.delete_anomaly_score(anomaly_score)
api.delete_batch_anomaly_score(batch_anomaly_score)
api.delete_sample(sample)
api.delete_correlation(correlation)
api.delete_statistical_test(statistical_test)
api.delete_logistic_regression(logistic_regression)
api.delete_linear_regression(linear_regression)
api.delete_association(association)
api.delete_association_set(association_set)
api.delete_topic_model(topic_model)
api.delete_topic_distribution(topic_distribution)
api.delete_batch_topic_distribution(batch_topic_distribution)
api.delete_time_series(time_series)
api.delete_forecast(forecast)
api.delete_fusion(fusion)
api.delete_pca(pca)
api.delete_deepnet(deepnet)
api.delete_projection(projection)
api.delete_batch_projection(batch_projection)
api.delete_project(project)
api.delete_script(script)
```

(continues on next page)

(continued from previous page)

```
api.delete_library(library)
api.delete_execution(execution)
```

Each of the calls above will return a dictionary with the following keys:

- **code** If the request is successful, the code will be a `bigml.api.HTTP_NO_CONTENT` (204) status code. Otherwise, it wil be one of the standard HTTP error codes. See the documentation on status codes for more info.
- **error** If the request does not succeed, it will contain a dictionary with an error code and a message. It will be `None` otherwise.

CHAPTER 34

Public and shared resources

The previous examples use resources that were created by the same user that asks for their retrieval or modification. If a user wants to share one of her resources, she can make them public or share them. Declaring a resource public means that anyone can see the resource. This can be applied to datasets and models. To turn a dataset public, just update its `private` property:

```
api.update_dataset('dataset/5143a51a37203f2cf7000972', {'private': false})
```

and any user will be able to download it using its id prepended by `public`:

```
api.get_dataset('public/dataset/5143a51a37203f2cf7000972')
```

In the models' case, you can also choose if you want the model to be fully downloadable or just accesible to make predictions. This is controlled with the `white_box` property. If you want to publish your model completely, just use:

```
api.update_model('model/5143a51a37203f2cf7000956', {'private': false,
                 'white_box': true})
```

Both public models and datasets, will be openly accessible for anyone, registered or not, from the web gallery.

Still, you may want to share your models with other users, but without making them public for everyone. This can be achieved by setting the `shared` property:

```
api.update_model('model/5143a51a37203f2cf7000956', {'shared': true})
```

Shared models can be accessed using their share hash (propery `shared_hash` in the original model):

```
api.get_model('shared/model/d53iw39euTdjsgesj7382ufhwnD')
```

or by using their original id with the creator user as username and a specific sharing api_key you will find as property `sharing_api_key` in the updated model:

```
api.get_model('model/5143a51a37203f2cf7000956', shared_username='creator',
              shared_api_key='c972018dc5f2789e65c74ba3170fda31d02e00c3')
```

Only users with the share link or credentials information will be able to access your shared models.

CHAPTER 35

Local Resources

All the resources in BigML can be downloaded and used locally with no connection whatsoever to BigML's servers. This is specially important for all Supervised and Unsupervised models, that can be used to generate predictions in any programmable device. The next sections describe how to do that for each type of resource, but as a general rule, resources can be exported to a JSON file in your file system using the `export` method.

```
api.export('model/5143a51a37203f2cf7000956',
           'filename': 'my_dir/my_model.json')
```

The contents of the generated file can be used just as the remote model to generate predictions. As you'll see in next section, the local `Model` object can be instantiated by giving the path to this file as first argument:

```
from bigml.model import Model
local_model = Model("my_dir/my_model.json")
local_model.predict({"petal length": 3, "petal width": 1})
Iris-versicolor
```

If you use a tag to label the resource, you can also ask for the last resource that has the tag:

```
api.export_last('my_tag',
                resource_type='ensemble',
                'filename': 'my_dir/my_ensemble.json')
```

and even for a resource inside a project:

```
api.export_last('my_tag',
                resource_type='dataset',
                project='project/5143a51a37203f2cf7000959',
                'filename': 'my_dir/my_dataset.json')
```

CHAPTER 36

Local Models

You can instantiate a local version of a remote model.

```
from bigml.model import Model
local_model = Model('model/502fdbff15526876610002615')
```

This will retrieve the remote model information, using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials) and return a Model object that will be stored in the `./storage` directory and you can use to make local predictions. If you want to use a specific connection object for the remote retrieval or a different storage directory, you can set it as second parameter:

```
from bigml.model import Model
from bigml.api import BigML

local_model = Model('model/502fdbff15526876610002615',
                    api=BigML(my_username,
                              my_api_key,
                              storage="my_storage"))
```

or even use the remote model information previously retrieved to build the local model object:

```
from bigml.model import Model
from bigml.api import BigML
api = BigML()
model = api.get_model('model/502fdbff15526876610002615',
                      query_string='only_model=true;limit=-1')

local_model = Model(model)
```

As you can see, the `query_string` used to retrieve the model has two parts. They both act on the `fields` information that is added to the JSON response. First `only_model=true` is used to restrict the fields described in the `fields` structure of the response to those used as predictors in the model. Also `limit=-1` avoids the pagination of fields which is used by default and includes them all at once. These details are already taken care of in the two previous examples, where the model ID is used as argument.

Any of these methods will return a `Model` object that you can use to make local predictions, generate IF-THEN rules, Tableau rules or a Python function that implements the model.

You can also build a local model from a model previously retrieved and stored in a JSON file:

```
from bigml.model import Model
local_model = Model('./my_model.json')
```

CHAPTER 37

Local Predictions

Once you have a local model you can use to generate predictions locally.

```
local_model.predict({"petal length": 3, "petal width": 1})
Iris-versicolor
```

Local predictions have three clear advantages:

- Removing the dependency from BigML to make new predictions.
- No cost (i.e., you do not spend BigML credits).
- Extremely low latency to generate predictions for huge volumes of data.

The default output for local predictions is the prediction itself, but you can also add other properties associated to the prediction, like its confidence or probability, the distribution of values in the predicted node (for decision tree models), and the number of instances supporting the prediction. To obtain a dictionary with the prediction and the available additional properties use the `full=True` argument:

```
local_model.predict({"petal length": 3, "petal width": 1}, full=True)
```

that will return:

```
{'count': 47,
 'confidence': 0.92444,
 'probability': 0.9861111111111112,
 'prediction': u'Iris-versicolor',
 'distribution_unit': 'categories',
 'path': [u'petal length > 2.45',
          u'petal width <= 1.75',
          u'petal length <= 4.95',
          u'petal width <= 1.65'],
 'distribution': [[u'Iris-versicolor', 47]]}
```

Note that the `path` attribute for the `proportional` missing strategy shows the path leading to a final unique node, that gives the prediction, or to the first split where a missing value is found. Other optional attributes are `next` which contains the field that determines the next split after the prediction node and `distribution` that adds the distribution that leads to the prediction. For regression models, min and max will add the limit values for the data that supports the prediction.

When your test data has missing values, you can choose between last prediction or proportional strategy to compute the prediction. The last prediction strategy is the one used by default. To compute a prediction, the algorithm goes down the model's decision tree and checks the condition it finds at each node (e.g.: 'sepal length' > 2). If the field checked is missing in your input data you have two options: by default (last prediction strategy) the algorithm will stop and issue the last prediction it computed in the previous node. If you chose proportional strategy instead, the algorithm will continue to go down the tree considering both branches from that node on. Thus, it will store a list of possible predictions from then on, one per valid node. In this case, the final prediction will be the majority (for categorical models) or the average (for regressions) of values predicted by the list of predicted values.

You can set this strategy by using the missing_strategy argument with code 0 to use last prediction and 1 for proportional.

```
from bigml.model import LAST_PREDICTION, PROPORTIONAL
# LAST_PREDICTION = 0; PROPORTIONAL = 1
local_model.predict({"petal length": 3, "petal width": 1},
                    missing_strategy=PROPORTIONAL)
```

For classification models, it is sometimes useful to obtain a probability or confidence prediction for each possible class of the objective field. To do this, you can use the predict_probability and predict_confidence methods respectively. The former gives a prediction based on the distribution of instances at the appropriate leaf node, with a Laplace correction based on the root node distribution. The latter returns a lower confidence bound on the leaf node probability based on the Wilson score interval.

Each of these methods take the missing_strategy argument that functions as it does in predict, and one additional argument, compact. If compact is False (the default), the output of these functions is a list of maps, each with the keys prediction and probability (or confidence) mapped to the class name and its associated probability (or confidence). Note that these methods substitute the deprecated multiple parameter in the predict method functionallity.

So, for example, the following:

```
local_model.predict_probability({"petal length": 3})
```

would result in

```
[{'prediction': u'Iris-setosa',
  'probability': 0.0033003300330033},
 {'prediction': u'Iris-versicolor',
  'probability': 0.4983498349834984},
 {'prediction': u'Iris-virginica',
  'probability': 0.4983498349834984}]
```

If compact is True, only the probabilities themselves are returned, as a list in class name order. Note that, for reference, the attribute Model.class_names contains the class names in the appropriate ordering.

To illustrate, the following:

```
local_model.predict_probability({"petal length": 3}, compact=True)
```

would result in

```
[0.0033003300330033, 0.4983498349834984, 0.4983498349834984]
```

The output of predict_confidence is the same, except that the output maps are keyed with confidence instead of probability.

For classifications, the prediction of a local model will be one of the available categories in the objective field and an associated `confidence` or `probability` that is used to decide which is the predicted category. If you prefer the model predictions to be operated using any of them, you can use the `operating_kind` argument in the `predict` method. Here's the example to use predictions based on `confidence`:

```
local_model.predict({"petal length": 3, "petal width": 1},
                    {"operating_kind": "confidence"})
```

Previous versions of the bindings had additional arguments in the `predict` method that were used to format the prediction attributes. The signature of the method has been changed to accept only arguments that affect the prediction itself, (like `missing_strategy`, `operating_kind` and `opreating_point`) and `full` which is a boolean that controls whether the output is the prediction itself or a dictionary will all the available properties associated to the prediction. Formatting can be achieved by using the `cast_prediction` function:

```
def cast_prediction(full_prediction, to=None,
                    confidence=False, probability=False,
                    path=False, distribution=False,
                    count=False, next=False, d_min=False,
                    d_max=False, median=False,
                    unused_fields=False):
```

whose first argument is the prediction obtained with the `full=True` argument, the second one defines the type of output (‘‘None‘‘‘to obtain the prediction output only, “list” or “dict”) and the rest of booleans cause the corresponding property to be included or not.

CHAPTER 38

Operating point's predictions

In classification problems, Models, Ensembles and Logistic Regressions can be used at different operating points, that is, associated to particular thresholds. Each operating point is then defined by the kind of property you use as threshold, its value and a the class that is supposed to be predicted if the threshold is reached.

Let's assume you decide that you have a binary problem, with classes `True` and `False` as possible outcomes. Imagine you want to be very sure to predict the *True* outcome, so you don't want to predict that unless the probability associated to it is over `0,8`. You can achieve this with any classification model by creating an operating point:

```
operating_point = {"kind": "probability",
                   "positive_class": "True",
                   "threshold": 0.8};
```

to predict using this restriction, you can use the `operating_point` parameter:

```
prediction = local_model.predict(inputData,
                                 operating_point=operating_point)
```

where `inputData` should contain the values for which you want to predict. Local models allow two kinds of operating points: `probability` and `confidence`. For both of them, the threshold can be set to any number in the `[0, 1]` range.

CHAPTER 39

Local Clusters

You can also instantiate a local version of a remote cluster.

```
from bigml.cluster import Cluster
local_cluster = Cluster('cluster/502fdbff15526876610002435')
```

This will retrieve the remote cluster information, using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials) and return a `Cluster` object that will be stored in the `./storage` directory and you can use to make local centroid predictions. If you want to use a specific connection object for the remote retrieval or a different storage directory, you can set it as second parameter:

```
from bigml.cluster import Cluster
from bigml.api import BigML

local_cluster = Cluster('cluster/502fdbff15526876610002435',
                        api=BigML(my_username,
                                  my_api_key
                                  storage="my_storage"))
```

or even use the remote cluster information previously retrieved to build the local cluster object:

```
from bigml.cluster import Cluster
from bigml.api import BigML
api = BigML()
cluster = api.get_cluster('cluster/502fdbff15526876610002435',
                          query_string='limit=-1')

local_cluster = Cluster(cluster)
```

Note that in this example we used a `limit=-1` query string for the cluster retrieval. This ensures that all fields are retrieved by the get method in the same call (unlike in the standard calls where the number of fields returned is limited).

Local clusters provide also methods for the significant operations that can be done using clusters: finding the centroid assigned to a certain data point, sorting centroids according to their distance to a data point, summarizing the centroids intra-distances and inter-distances and also finding the closest points to a given one. The *Local Centroids* and the *Summary generation* sections will explain these methods.

CHAPTER 40

Local Centroids

Using the local cluster object, you can predict the centroid associated to an input data set:

```
local_cluster.centroid({"pregnancies": 0, "plasma glucose": 118,
                        "blood pressure": 84, "triceps skin thickness": 47,
                        "insulin": 230, "bmi": 45.8,
                        "diabetes pedigree": 0.551, "age": 31,
                        "diabetes": "true"})
{'distance': 0.454110207355, 'centroid_name': 'Cluster 4',
 'centroid_id': '000004'}
```

You must keep in mind, though, that to obtain a centroid prediction, input data must have values for all the numeric fields. No missing values for the numeric fields are allowed unless you provided a `default_numeric_value` in the cluster construction configuration. If so, this value will be used to fill the missing numeric fields.

As in the local model predictions, producing local centroids can be done independently of BigML servers, so no cost or connection latencies are involved.

Another interesting method in the cluster object is `local_cluster.closests_in_cluster`, which given a reference data point will provide the rest of points that fall into the same cluster sorted in an ascending order according to their distance to this point. You can limit the maximum number of points returned by setting the `number_of_points` argument to any positive integer.

```
local_cluster.closests_in_cluster( \
    {"pregnancies": 0, "plasma glucose": 118,
     "blood pressure": 84, "triceps skin thickness": 47,
     "insulin": 230, "bmi": 45.8,
     "diabetes pedigree": 0.551, "age": 31,
     "diabetes": "true"}, number_of_points=2)
```

The response will be a dictionary with the centroid id of the cluster an the list of closest points and their distances to the reference point.

```
{'closest': [ \
    {'distance': 0.06912270988567025,
```

(continues on next page)

(continued from previous page)

```
     'data': {'plasma glucose': '115', 'blood pressure': '70',
              'triceps skin thickness': '30', 'pregnancies': '1',
              'bmi': '34.6', 'diabetes pedigree': '0.529',
              'insulin': '96', 'age': '32', 'diabetes': 'true'}},
    {'distance': 0.10396456577958413,
     'data': {'plasma glucose': '167', 'blood pressure': '74',
     'triceps skin thickness': '17', 'pregnancies': '1', 'bmi': '23.4',
     'diabetes pedigree': '0.447', 'insulin': '144', 'age': '33',
     'diabetes': 'true'}}],
'reference': {'age': 31, 'bmi': 45.8, 'plasma glucose': 118,
              'insulin': 230, 'blood pressure': 84,
              'pregnancies': 0, 'triceps skin thickness': 47,
              'diabetes pedigree': 0.551, 'diabetes': 'true'},
'centroid_id': u'000000'}
```

No missing numeric values are allowed either in the reference data point. If you want the data points to belong to a different cluster, you can provide the `centroid_id` for the cluster as an additional argument.

Other utility methods are `local_cluster.sorted_centroids` which given a reference data point will provide the list of centroids sorted according to the distance to it

```
local_cluster.sorted_centroids( \
{'plasma glucose': '115', 'blood pressure': '70',
 'triceps skin thickness': '30', 'pregnancies': '1',
 'bmi': '34.6', 'diabetes pedigree': '0.529',
 'insulin': '96', 'age': '32', 'diabetes': 'true'})
{'centroids': [{'distance': 0.31656890408929705,
                'data': {u'000006': 0.34571, u'000007': 30.7619,
                         u'000000': 3.79592, u'000008': u'false'},
                'centroid_id': u'000000'},
               {'distance': 0.4424198506958207,
                'data': {u'000006': 0.77087, u'000007': 45.50943,
                         u'000000': 5.90566, u'000008': u'true'},
                'centroid_id': u'000001'}],
 'reference': {'age': '32', 'bmi': '34.6', 'plasma glucose': '115',
               'insulin': '96', 'blood pressure': '70',
               'pregnancies': '1', 'triceps skin thickness': '30',
               'diabetes pedigree': '0.529', 'diabetes': 'true'}}
```

or `points_in_cluster` that returns the list of data points assigned to a certain cluster, given its `centroid_id`.

```
centroid_id = "000000"
local_cluster.points_in_cluster(centroid_id)
```

CHAPTER 41

Local Anomaly Detector

You can also instantiate a local version of a remote anomaly.

```
from bigml.anomaly import Anomaly
local_anomaly = Anomaly('anomaly/502fcbff15526876610002435')
```

This will retrieve the remote anomaly detector information, using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials) and return an `Anomaly` object that will be stored in the `./storage` directory and you can use to make local anomaly scores. If you want to use a specific connection object for the remote retrieval or a different storage directory, you can set it as second parameter:

```
from bigml.anomaly import Anomaly
from bigml.api import BigML

local_anomaly = Anomaly('anomaly/502fcbff15526876610002435',
                        api=BigML(my_username,
                                  my_api_key))
```

or even use the remote anomaly information retrieved previously to build the local anomaly detector object:

```
from bigml.anomaly import Anomaly
from bigml.api import BigML
api = BigML()
anomaly = api.get_anomaly('anomaly/502fcbff15526876610002435',
                          query_string='limit=-1')

local_anomaly = Anomaly(anomaly)
```

Note that in this example we used a `limit=-1` query string for the anomaly retrieval. This ensures that all fields are retrieved by the get method in the same call (unlike in the standard calls where the number of fields returned is limited).

The anomaly detector object has also the method `anomalies_filter` that will build the LISP filter you would need to filter the original dataset and create a new one excluding the top anomalies. Setting the `include` parameter to True you can do the inverse and create a dataset with only the most anomalous data points.

CHAPTER 42

Local Anomaly Scores

Using the local anomaly detector object, you can predict the anomaly score associated to an input data set:

```
local_anomaly.anomaly_score({"src_bytes": 350})
0.9268527808726705
```

As in the local model predictions, producing local anomaly scores can be done independently of BigML servers, so no cost or connection latencies are involved.

CHAPTER 43

Local Logistic Regression

You can also instantiate a local version of a remote logistic regression.

```
from bigml.logistic import LogisticRegression
local_log_regression = LogisticRegression(
    'logisticregression/502fdbff15526876610042435')
```

This will retrieve the remote logistic regression information, using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials) and return a `LogisticRegression` object that will be stored in the `./storage` directory and you can use to make local predictions. If you want to use a specific connection object for the remote retrieval or a different storage directory, you can set it as second parameter:

```
from bigml.logistic import LogisticRegression
from bigml.api import BigML

local_log_regression = LogisticRegression(
    'logisticregression/502fdbff15526876610602435',
    api=BigML(my_username, my_api_key, storage="my_storage"))
```

You can also reuse a remote logistic regression JSON structure as previously retrieved to build the local logistic regression object:

```
from bigml.logistic import LogisticRegression
from bigml.api import BigML
api = BigML()
logistic_regression = api.get_logistic_regression(
    'logisticregression/502fdbff15526876610002435',
    query_string='limit=-1')

local_log_regression = LogisticRegression(logistic_regression)
```

Note that in this example we used a `limit=-1` query string for the logistic regression retrieval. This ensures that all fields are retrieved by the get method in the same call (unlike in the standard calls where the number of fields returned is limited).

CHAPTER 44

Local Logistic Regression Predictions

Using the local logistic regression object, you can predict the prediction for an input data set:

```
local_log_regression.predict({"petal length": 2, "sepal length": 1.5,
                              "petal width": 0.5, "sepal width": 0.7},
                              full=True)
{'distribution': [
    {'category': u'Iris-virginica', 'probability': 0.5041444478857267},
    {'category': u'Iris-versicolor', 'probability': 0.46926542042788333},
    {'category': u'Iris-setosa', 'probability': 0.02659013168639014}],
    'prediction': u'Iris-virginica', 'probability': 0.5041444478857267}
```

As you can see, the prediction contains the predicted category and the associated probability. It also shows the distribution of probabilities for all the possible categories in the objective field. If you only need the predicted value, you can remove the ‘‘full‘‘argument.

You must keep in mind, though, that to obtain a logistic regression prediction, input data must have values for all the numeric fields. No missing values for the numeric fields are allowed.

For consistency of interface with the `Model` class, logistic regressions again have a `predict_probability` method, which takes the same argument as `Model.predict`: `compact`. As stated above, missing values are not allowed, and so there is no `missing_strategy` argument.

As with local Models, if `compact` is `False` (the default), the output is a list of maps, each with the keys `prediction` and `probability` mapped to the class name and its associated probability.

So, for example

```
local_log_regression.predict_probability({"petal length": 2, "sepal length": 1.5,
                                          "petal width": 0.5, "sepal width": 0.7})

[{'category': u'Iris-setosa', 'probability': 0.02659013168639014},
 {'category': u'Iris-versicolor', 'probability': 0.46926542042788333},
 {'category': u'Iris-virginica', 'probability': 0.5041444478857267}]
```

If `compact` is `True`, only the probabilities themselves are returned, as a list in class name order, again, as is the case with local Models.

Operating point predictions are also available for local logistic regressions and an example of it would be:

```
operating_point = {"kind": "probability",
                   "positive_class": "True",
                   "threshold": 0.8}
local_logistic.predict(inputData, operating_point=operating_point)
```

You can check the *Operating point's predictions* section to learn about operating points. For logistic regressions, the only available kind is `probability`, that sets the threshold of probability to be reached for the prediction to be the positive class.

CHAPTER 45

Local Logistic Regression

You can also instantiate a local version of a remote logistic regression.

```
from bigml.logistic import LogisticRegression
local_log_regression = LogisticRegression(
    'logisticregression/502fdbff15526876610042435')
```

This will retrieve the remote logistic regression information, using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials) and return a `LogisticRegression` object that will be stored in the `./storage` directory and you can use to make local predictions. If you want to use a specific connection object for the remote retrieval or a different storage directory, you can set it as second parameter:

```
from bigml.logistic import LogisticRegression
from bigml.api import BigML

local_log_regression = LogisticRegression(
    'logisticregression/502fdbff15526876610602435',
    api=BigML(my_username, my_api_key, storage="my_storage"))
```

You can also reuse a remote logistic regression JSON structure as previously retrieved to build the local logistic regression object:

```
from bigml.logistic import LogisticRegression
from bigml.api import BigML
api = BigML()
logistic_regression = api.get_logistic_regression(
    'logisticregression/502fdbff15526876610002435',
    query_string='limit=-1')

local_log_regression = LogisticRegression(logistic_regression)
```

Note that in this example we used a `limit=-1` query string for the logistic regression retrieval. This ensures that all fields are retrieved by the get method in the same call (unlike in the standard calls where the number of fields returned is limited).

CHAPTER 46

Local Linear Regression Predictions

Using the local linear regression object, you can predict the prediction for an input data set:

```
local_linear_regression.predict({"petal length": 2, "sepal length": 1.5,
                                 "species": "Iris-setosa",
                                 "sepal width": 0.7},
                                 full=True)
{'confidence_bounds': {
    'prediction_interval': 0.43783924497784293,
    'confidence_interval': 0.2561542783257394},
 'prediction': -0.6109005499999999, 'unused_fields': ['petal length']}
```

To obtain a linear regression prediction, input data can only have missing values for fields that had already some missings in training data.

CHAPTER 47

Local Deepnet

You can also instantiate a local version of a remote Deepnet.

```
from bigml.deepnet import Deepnet
local_deepnet = Deepnet(
    'deepnet/502fdbff15526876610022435')
```

This will retrieve the remote deepnet information, using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials) and return a `Deepnet` object that will be stored in the `./storage` directory and you can use to make local predictions. If you want to use a specific connection object for the remote retrieval or a different storage directory, you can set it as second parameter:

```
from bigml.deepnet import Deepnet
from bigml.api import BigML

local_deepnet = Deepnet(
    'deepnet/502fdbff15526876610602435',
    api=BigML(my_username, my_api_key, storage="my_storage"))
```

You can also reuse a remote Deepnet JSON structure as previously retrieved to build the local Deepnet object:

```
from bigml.deepnet import Deepnet
from bigml.api import BigML
api = BigML()
deepnet = api.get_deepnet(
    'deepnet/502fdbff15526876610002435',
    query_string='limit=-1')

local_deepnet = Deepnet(deepnet)
```

Note that in this example we used a `limit=-1` query string for the deepnet retrieval. This ensures that all fields are retrieved by the get method in the same call (unlike in the standard calls where the number of fields returned is limited).

CHAPTER 48

Local Deepnet Predictions

Using the local deepnet object, you can predict the prediction for an input data set:

```
local_deepnet.predict({"petal length": 2, "sepal length": 1.5,
                       "petal width": 0.5, "sepal width": 0.7},
                       full=True)
{'distribution': [
    {'category': u'Iris-virginica', 'probability': 0.5041444478857267},
    {'category': u'Iris-versicolor', 'probability': 0.46926542042788333},
    {'category': u'Iris-setosa', 'probability': 0.02659013168639014}],
    'prediction': u'Iris-virginica', 'probability': 0.5041444478857267}
```

As you can see, the full prediction contains the predicted category and the associated probability. It also shows the distribution of probabilities for all the possible categories in the objective field. If you only need the predicted value, you can remove the `full` argument.

To be consistent with the `Model` class interface, deepnets have also a `predict_probability` method, which takes the same argument as `Model.predict`: `compact`.

As with local Models, if `compact` is `False` (the default), the output is a list of maps, each with the keys `prediction` and `probability` mapped to the class name and its associated probability.

So, for example

```
local_deepnet.predict_probability({"petal length": 2, "sepal length": 1.5,
                                   "petal width": 0.5, "sepal width": 0.7})

[{'category': u'Iris-setosa', 'probability': 0.02659013168639014},
 {'category': u'Iris-versicolor', 'probability': 0.46926542042788333},
 {'category': u'Iris-virginica', 'probability': 0.5041444478857267}]
```

If `compact` is `True`, only the probabilities themselves are returned, as a list in class name order, again, as is the case with local Models.

Operating point predictions are also available for local deepnets and an example of it would be:

```
operating_point = {"kind": "probability",
                   "positive_class": "True",
                   "threshold": 0.8};
prediction = local_deepnet.predict(inputData,
                                   operating_point=operating_point)
```

CHAPTER 49

Local Fusion

You can also instantiate a local version of a remote Fusion.

```
from bigml.fusion import Fusion
local_fusion = Fusion(
    'fusion/502fdbff15526876610022438')
```

This will retrieve the remote fusion information, using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials) and return a `Fusion` object that will be stored in the `./storage` directory and you can use to make local predictions. If you want to use a specific connection object for the remote retrieval or a different storage directory, you can set it as second parameter:

```
from bigml.fusion import Fusion
from bigml.api import BigML

local_fusion = Fusion(
    'fusion/502fdbff15526876610602435',
    api=BigML(my_username, my_api_key, storage="my_storage"))
```

You can also reuse a remote Fusion JSON structure as previously retrieved to build the local Fusion object:

```
from bigml.fusion import Fusion
from bigml.api import BigML
api = BigML()
fusion = api.get_fusion(
    'fusion/502fdbff15526876610002435',
    query_string='limit=-1')

local_fusion = Fusion(fusion)
```

Note that in this example we used a `limit=-1` query string for the fusion retrieval. This ensures that all fields are retrieved by the get method in the same call (unlike in the standard calls where the number of fields returned is limited).

CHAPTER 50

Local Fusion Predictions

Using the local fusion object, you can predict the prediction for an input data set:

```
local_fusion.predict({"petal length": 2, "sepal length": 1.5,
                      "petal width": 0.5, "sepal width": 0.7},
                      full=True)
{'prediction': u'Iris-setosa', 'probability': 0.45224}
```

As you can see, the full prediction contains the predicted category and the associated probability. If you only need the predicted value, you can remove the `full` argument.

To be consistent with the `Model` class interface, fusions have also a `predict_probability` method, which takes the same argument as `Model.predict`: `compact`.

As with local Models, if `compact` is `False` (the default), the output is a list of maps, each with the keys `prediction` and `probability` mapped to the class name and its associated probability.

So, for example

```
local_fusion.predict_probability({"petal length": 2, "sepal length": 1.5,
                                  "petal width": 0.5, "sepal width": 0.7})

[{'category': u'Iris-setosa', 'probability': 0.45224},
 {'category': u'Iris-versicolor', 'probability': 0.2854},
 {'category': u'Iris-virginica', 'probability': 0.26236}]
```

If `compact` is `True`, only the probabilities themselves are returned, as a list in class name order, again, as is the case with local Models.

Operating point predictions are also available with probability as threshold for local fusions and an example of it would be:

```
operating_point = {"kind": "probability",
                   "positive_class": "True",
                   "threshold": 0.8};
prediction = local_fusion.predict(inputData,
                                  operating_point=operating_point)
```

CHAPTER 51

Local Association

You can also instantiate a local version of a remote association resource.

```
from bigml.association import Association
local_association = Association('association/502fdcff15526876610002435')
```

This will retrieve the remote association information, using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials) and return an `Association` object that will be stored in the `./storage` directory and you can use to extract the rules found in the original dataset. If you want to use a specific connection object for the remote retrieval or a different storage directory, you can set it as second parameter:

```
from bigml.association import Association
from bigml.api import BigML

local_association = Association('association/502fdcff15526876610002435',
                                api=BigML(my_username,
                                          my_api_key
                                          storage="my_storage"))
```

or even use the remote association information retrieved previously to build the local association object:

```
from bigml.association import Association
from bigml.api import BigML
api = BigML()
association = api.get_association('association/502fdcff15526876610002435',
                                  query_string='limit=-1')

local_association = Association(association)
```

Note that in this example we used a `limit=-1` query string for the association retrieval. This ensures that all fields are retrieved by the get method in the same call (unlike in the standard calls where the number of fields returned is limited).

The created `Association` object has some methods to help retrieving the association rules found in the original data. The `get_rules` method will return the association rules. Arguments can be set to filter the rules returned according to its `leverage`, `strength`, `support`, `p_value`, a list of items involved in the rule or a user-given filter function.

```
from bigml.association import Association
local_association = Association('association/502fdcff15526876610002435')
local_association.get_rules(item_list=["Edible"], min_p_value=0.3)
```

In this example, the only rules that will be returned by the `get_rules` method will be the ones that mention `Edible` and their `p_value` is greater or equal to `0.3`.

The rules can also be stored in a CSV file using `rules_CSV`:

```
from bigml.association import Association
local_association = Association('association/502fdcff15526876610002435')
local_association.rules_CSV(file_name='/tmp/my_rules.csv',
                            min_strength=0.1)
```

This example will store the rules whose strength is bigger or equal to 0.1 in the `/tmp/my_rules.csv` file.

You can also obtain the list of `items` parsed in the dataset using the `get_items` method. You can also filter the results by field name, by item names and by a user-given function:

```
from bigml.association import Association
local_association = Association('association/502fdcff15526876610002435')
local_association.get_items(field="Cap Color",
                            names=["Brown cap", "White cap", "Yellow cap"])
```

This will recover the `Item` objects found in the `Cap Color` field for the names in the list, with their properties as described in the developers section

CHAPTER 52

Local Association Sets

Using the local association object, you can predict the association sets related to an input data set:

```
local_association.association_set( \
    {"gender": "Female", "genres": "Adventure$Action", \
     "timestamp": 993906291, "occupation": "K-12 student",
     "zipcode": 59583, "rating": 3})
[{'item': {'complement': False,
           'count': 70,
           'field_id': u'000002',
           'name': u'Under 18'},
  'rules': ['000000'],
  'score': 0.0969181441561211},
 {'item': {'complement': False,
           'count': 216,
           'field_id': u'000007',
           'name': u'Drama'},
  'score': 0.025050115102862636},
 {'item': {'complement': False,
           'count': 108,
           'field_id': u'000007',
           'name': u'Sci-Fi'},
  'rules': ['000003'],
  'score': 0.02384578264599424},
 {'item': {'complement': False,
           'count': 40,
           'field_id': u'000002',
           'name': u'56+'},
  'rules': ['000008',
            '000020'],
  'score': 0.021845366022721312},
 {'item': {'complement': False,
           'count': 66,
           'field_id': u'000002',
           'name': u'45-49'},
```

(continues on next page)

(continued from previous page)

```
  'rules': ['00000e'],
  'score': 0.019657155185835006}]
```

As in the local model predictions, producing local association sets can be done independently of BigML servers, so no cost or connection latencies are involved.

CHAPTER 53

Local Topic Model

You can also instantiate a local version of a remote topic model.

```
from bigml.topicmodel import TopicModel
local_topic_model = TopicModel(
    'topicmodel/502fdbcf15526876210042435')
```

This will retrieve the remote topic model information, using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials) and return a `TopicModel` object that will be stored in the `./storage` directory and you can use to obtain local topic distributions. If you want to use a specific connection object for the remote retrieval or a different storage directory, you can set it as second parameter:

```
from bigml.topicmodel import TopicModel
from bigml.api import BigML

local_topic_model = TopicModel(
    'topicmodel/502fdbcf15526876210042435',
    api=BigML(my_username, my_api_key, storage="my_storage"))
```

You can also reuse a remote topic model JSON structure as previously retrieved to build the local topic model object:

```
from bigml.topicmodel import TopicModel
from bigml.api import BigML
api = BigML()
topic_model = api.get_topic_model(
    'topicmodel/502fdbcf15526876210042435',
    query_string='limit=-1')

local_topic_model = TopicModel(topic_model)
```

Note that in this example we used a `limit=-1` query string for the topic model retrieval. This ensures that all fields are retrieved by the get method in the same call (unlike in the standard calls where the number of fields returned is limited).

CHAPTER 54

Local Topic Distributions

Using the local topic model object, you can predict the local topic distribution for an input data set:

```
local_topic_model.distribution({"Message": "Our mobile phone is free"})
[   {   'name': u'Topic 00', 'probability': 0.002627154266498529},
    {   'name': u'Topic 01', 'probability': 0.003257671290458176},
    {   'name': u'Topic 02', 'probability': 0.002627154266498529},
    {   'name': u'Topic 03', 'probability': 0.1968263976460698},
    {   'name': u'Topic 04', 'probability': 0.002627154266498529},
    {   'name': u'Topic 05', 'probability': 0.002627154266498529},
    {   'name': u'Topic 06', 'probability': 0.13692728036990331},
    {   'name': u'Topic 07', 'probability': 0.6419714165615805},
    {   'name': u'Topic 08', 'probability': 0.002627154266498529},
    {   'name': u'Topic 09', 'probability': 0.002627154266498529},
    {   'name': u'Topic 10', 'probability': 0.002627154266498529},
    {   'name': u'Topic 11', 'probability': 0.002627154266498529}]
```

As you can see, the topic distribution contains the name of the possible topics in the model and the associated probabilities.

CHAPTER 55

Local Time Series

You can also instantiate a local version of a remote time series.

```
from bigml.timeseries import TimeSeries
local_time_series = TimeSeries(
    'timeseries/502fdbcf15526876210042435')
```

This will create a series of models from the remote time series information, using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials) and return a `TimeSeries` object that will be stored in the `./storage` directory and you can use to obtain local forecasts. If you want to use a specific connection object for the remote retrieval or a different storage directory, you can set it as second parameter:

```
from bigml.timeseries import TimeSeries
from bigml.api import BigML

local_time_series = TimeSeries( \
    'timeseries/502fdbcf15526876210042435',
    api=BigML(my_username, my_api_key, storage="my_storage"))
```

You can also reuse a remote time series JSON structure as previously retrieved to build the local time series object:

```
from bigml.timeseries import TimeSeries
from bigml.api import BigML
api = BigML()
time_series = api.get_time_series( \
    'timeseries/502fdbcf15526876210042435',
    query_string='limit=-1')

local_time_series = TimeSeries(time_series)
```

Note that in this example we used a `limit=-1` query string for the time series retrieval. This ensures that all fields are retrieved by the get method in the same call (unlike in the standard calls where the number of fields returned is limited).

CHAPTER 56

Local Forecasts

Using the local time series object, you can forecast any of the objective field values:

```
local_time_series.forecast({"Final": {"horizon": 5}, "Assignment": { \
    "horizon": 10, "ets_models": {"criterion": "aic", "limit": 2}}})
{u'000005': [
    {'point_forecast': [68.53181, 68.53181, 68.53181, 68.53181, 68.53181],
     'model': u'A,N,N'}],
 u'000001': [{'point_forecast': [54.776650000000004, 90.00943000000001,
                                 83.59285000000001, 85.72403000000001,
                                 72.87196, 93.85872, 84.80786, 84.65522,
                                 92.52545, 88.78403],
              'model': u'A,N,A'},
             {'point_forecast': [55.882820120000005, 90.5255466567616,
                                 83.44908577909621, 87.64524353046498,
                                 74.32914583152592, 95.12372848262932,
                                 86.69298716626228, 85.31630744944385,
                                 93.62385478607113, 89.06905451921818],
              'model': u'A,Ad,A'}]}
```

As you can see, the forecast contains the ID of the forecasted field, the computed points and the name of the models meeting the criterion. For more details about the available parameters, please check the API documentation.

CHAPTER 57

Local PCAs

The *PCA* class will create a local version of a remote PCA.

```
from bigml.pca import PCA
local_pca = PCA(
    'pca/502fdbcf15526876210042435')
```

This will create an object that stores the remote information that defines the PCA, needed to generate projections to the new dimensionally reduced components. The remote resource is automatically downloaded the first time the the PCA is instantiated by using an implicitly built `BigML()` connection object (see the *Authentication* section for more details on how to set your credentials). The JSON that contains this information is stored in a `./storage` directory, which is the default choice. If you want to use a specific connection object to define the credentials for the authentication in BigML or the directory where the JSON information is stored, you can set it as the second parameter:

```
from bigml.pca import PCA
from bigml.api import BigML

local_pca = PCA( \
    'timeseries/502fdbcf15526876210042435',
    api=BigML(my_username, my_api_key, storage="my_storage"))
```

You can also reuse a remote PCA JSON structure as previously retrieved to build the local PCA object:

```
from bigml.pca import PCA
from bigml.api import BigML
api = BigML()
time_series = api.get_pca( \
    'pca/502fdbcf15526876210042435',
    query_string='limit=-1')

local_pca = PCA(pca)
```

Note that in this example we used a `limit=-1` query string for the PCA retrieval. This ensures that all fields are retrieved by the get method in the same call (unlike in the standard calls where the number of fields returned is limited).

CHAPTER 58

Local Projections

Using the local PCA object, you can compute the projection of an input dataset into the new components:

```
local_pca.projection({"species": "Iris-versicolor"})
[6.03852, 8.35456, 5.04432, 0.75338, 0.06787, 0.03018]
```

You can use the max_components and variance_threshold arguments to limit the number of components generated. You can also use the full argument to produce a dictionary whose keys are the names of the generated components.

```
local_pca.projection({"species": "Iris-versicolor"}, full=yes)
{'PCA1': 6.03852, 'PCA2': 8.35456, 'PCA3': 5.04432, 'PCA4': 0.75338,
 'PCA5': 0.06787, 'PCA6': 0.03018}
```

As in the local model predictions, producing local projections can be done independently of BigML servers, so no cost or connection latencies are involved.

CHAPTER 59

Local Forecasts

Using the local time series object, you can forecast any of the objective field values:

```
local_time_series.forecast({"Final": {"horizon": 5}, "Assignment": { \
    "horizon": 10, "ets_models": {"criterion": "aic", "limit": 2}}})
{u'000005': [
    {'point_forecast': [68.53181, 68.53181, 68.53181, 68.53181, 68.53181],
     'model': u'A,N,N'}],
 u'000001': [{'point_forecast': [54.776650000000004, 90.00943000000001,
                                 83.59285000000001, 85.72403000000001,
                                 72.87196, 93.85872, 84.80786, 84.65522,
                                 92.52545, 88.78403],
              'model': u'A,N,A'},
             {'point_forecast': [55.882820120000005, 90.5255466567616,
                                 83.44908577909621, 87.64524353046498,
                                 74.32914583152592, 95.12372848262932,
                                 86.69298716626228, 85.31630744944385,
                                 93.62385478607113, 89.06905451921818],
              'model': u'A,Ad,A'}]}
```

As you can see, the forecast contains the ID of the forecasted field, the computed points and the name of the models meeting the criterion. For more details about the available parameters, please check the API documentation.

CHAPTER 60

Multi Models

Multi Models use a numbers of BigML remote models to build a local version that can be used to generate predictions locally. Predictions are generated combining the outputs of each model.

```
from bigml.api import BigML
from bigml.multimodel import MultiModel

api = BigML()

model = MultiModel([api.get_model(model['resource']) for model in
                    api.list_models(query_string="tags__in=my_tag")
                    ['objects']])

model.predict({"petal length": 3, "petal width": 1})
```

This will create a multi model using all the models that have been previously tagged with `my_tag` and predict by combining each model's prediction. The combination method used by default is `plurality` for categorical predictions and mean value for numerical ones. You can also use `confidence weighted`:

```
model.predict({"petal length": 3, "petal width": 1}, method=1)
```

that will weight each vote using the confidence/error given by the model to each prediction, or even `probability weighted`:

```
model.predict({"petal length": 3, "petal width": 1}, method=2)
```

that weights each vote by using the probability associated to the training distribution at the prediction node.

There's also a `threshold` method that uses an additional set of options: threshold and category. The category is predicted if and only if the number of predictions for that category is at least the threshold value. Otherwise, the prediction is plurality for the rest of predicted values.

An example of `threshold` combination method would be:

```
model.predict({'petal length': 0.9, 'petal width': 3.0}, method=3,
              options={'threshold': 3, 'category': 'Iris-virginica'})
```

When making predictions on a test set with a large number of models, `batch_predict` can be useful to log each model's predictions in a separated file. It expects a list of input data values and the directory path to save the prediction files in.

```
model.batch_predict([{"petal length": 3, "petal width": 1},
                     {"petal length": 1, "petal width": 5.1}],
                    "data/predictions")
```

The predictions generated for each model will be stored in an output file in *data/predictions* using the syntax *model_[id of the model]__predictions.csv*. For instance, when using *model/50c0de043b563519830001c2* to predict, the output file name will be *model_50c0de043b563519830001c2__predictions.csv*. An additional feature is that using `reuse=True` as argument will force the function to skip the creation of the file if it already exists. This can be helpful when using repeatedly a bunch of models on the same test set.

```
model.batch_predict([{"petal length": 3, "petal width": 1},
                     {"petal length": 1, "petal width": 5.1}],
                    "data/predictions", reuse=True)
```

Prediction files can be subsequently retrieved and converted into a votes list using `batch_votes`:

```
model.batch_votes("data/predictions")
```

which will return a list of MultiVote objects. Each MultiVote contains a list of predictions (e.g. `[{'prediction': u'Iris-versicolor', 'confidence': 0.34, 'order': 0}, {'prediction': u'Iris-setosa', 'confidence': 0.25, 'order': 1}]`). These votes can be further combined to issue a final prediction for each input data element using the method `combine`

```
for multivote in model.batch_votes("data/predictions"):
    prediction = multivote.combine()
```

Again, the default method of combination is `plurality` for categorical predictions and mean value for numerical ones. You can also use `confidence weighted`:

```
prediction = multivote.combine(1)
```

or `probability weighted`:

```
prediction = multivote.combine(2)
```

You can also get a confidence measure for the combined prediction:

```
prediction = multivolte.combine(0, with_confidence=True)
```

For classification, the confidence associated to the combined prediction is derived by first selecting the model's predictions that voted for the resulting prediction and computing the weighted average of their individual confidence. Nevertheless, when `probability weighted` is used, the confidence is obtained by using each model's distribution at the prediction node to build a probability distribution and combining them. The confidence is then computed as the wilson score interval of the combined distribution (using as total number of instances the sum of all the model's distributions original instances at the prediction node)

In regression, all the models predictions' confidences contribute to the weighted average confidence.

CHAPTER 61

Local Ensembles

Remote ensembles can also be used locally through the `Ensemble` class. The simplest way to access an existing ensemble and using it to predict locally is:

```
from bigml.ensemble import Ensemble
ensemble = Ensemble('ensemble/5143a51a37203f2cf7020351')
ensemble.predict({"petal length": 3, "petal width": 1})
```

This call will download all the ensemble related info and store it in a `./storage` directory ready to be used to predict, but you can choose another storage directory or even avoid storing at all.

The local ensemble object can be used to manage the three types of ensembles: `Decision Forests` (bagging or random) and the ones using `Boosted Trees`.

```
from bigml.api import BigML
from bigml.ensemble import Ensemble

# api connection using a user-selected storage
api = BigML(storage='./my_storage')

# creating ensemble
ensemble = api.create_ensemble('dataset/5143a51a37203f2cf7000972')

# Ensemble object to predict
ensemble = Ensemble(ensemble, api)
ensemble.predict({"petal length": 3, "petal width": 1},
                 operating_kind="votes")
```

In this example, we create a new ensemble and store its information in the `./my_storage` folder. Then this information is used to predict locally using the number of votes (one per model) backing each category.

The `operating_kind` argument overrides the legacy `method` argument, which was previously used to define the combiner for the models predictions.

Similarly, local ensembles can also be created by giving a list of models to be combined to issue the final prediction (note: only random decision forests and bagging ensembles can be built using this method):

```
from bigml.ensemble import Ensemble
ensemble = Ensemble(['model/50c0de043b563519830001c2', \
                     'model/50c0de043b5635198300031b')]
ensemble.predict({"petal length": 3, "petal width": 1})
```

or even a JSON file that contains the ensemble resource:

```
import json
from bigml.ensemble import Ensemble
from bigml.api import BigML
api = api.BigML()
ensemble_info = api.get_ensemble('ensemble/50c0de043b5635198300033c')
with open("./my_ensemble", "w") as ensemble_file:
    ensemble_file.write(json.dumps(ensemble_info))
local_ensemble = Ensemble("./my_ensemble")
```

Note: the ensemble JSON structure is not self-contained, meaning that it contains references to the models that the ensemble is build of, but not the information of the models themselves. To use an ensemble locally with no connection to the internet, you must make sure that not only a local copy of the ensemble JSON file is available in your computer, but also the JSON files corresponding to the models in it. This is automatically achieved when you use the `Ensemble('ensemble/50c0de043b5635198300033c')` constructor, that fetches all the related JSON files and stores them in an `./storage` directory. Next calls to `Ensemble('ensemble/50c0de043b5635198300033c')` will retrieve the files from this local storage, so that internet connection will only be needed the first time an `Ensemble` is built.

On the contrary, if you have no memory limitations and want to increase prediction speed, you can create the ensemble from a list of local model objects. Then, local model objects will only be instantiated once, and this could increase performance for large ensembles:

```
from bigml.model import Model
model_ids = ['model/50c0de043b563519830001c2', \
             'model/50c0de043b5635198300031b']
local_models = [Model(model_id) for model_id in model_ids]
local_ensemble = Ensemble(local_models)
```

The `Ensemble` object can also be instantiated using local models previously stored in disks or memory object caching systems. To retrieve these models provide a list of model ids as first argument and an extra argument named `cache_get` that should be a function receiving the model id to be retrieved and returning a local model object.

```
from bigml.model import Model
model_ids = ['model/50c0de043b563519830001c2', \
             'model/50c0de043b5635198300031b']
def cache_get(model_id):
    """ Retrieves a JSON model structure and builds a local model object

    """
    model_file = model_id.replace("/", "_")
    return Model(json.load(open(model_file)))
local_ensemble = Ensemble(model_ids, cache_get=cache_get)
```

CHAPTER 62

Local Ensemble's Predictions

As in the local model's case, you can use the local ensemble to create new predictions for your test data, and set some arguments to configure the final output of the `predict` method.

The predictions' structure will vary depending on the kind of ensemble used. For `Decision Forests` local predictions will just contain the ensemble's final prediction if no other argument is used.

```
from bigml.ensemble import Ensemble
ensemble = Ensemble('ensemble/5143a51a37203f2cf7020351')
ensemble.predict({"petal length": 3, "petal width": 1})
u'Iris-versicolor'
```

The final prediction of an ensemble is determined by aggregating or selecting the predictions of the individual models therein. For classifications, the most probable class is returned if no especial operating method is set. Using `full=True` you can see both the predicted output and the associated probability:

```
from bigml.ensemble import Ensemble
ensemble = Ensemble('ensemble/5143a51a37203f2cf7020351')
ensemble.predict({"petal length": 3, "petal width": 1}, \
                 full=True)

{'prediction': u'Iris-versicolor',
 'probability': 0.98566}
```

In general, the prediction in a classification will be one amongst the list of categories in the objective field. When each model in the ensemble is used to predict, each category has a confidence, a probability or a vote associated to this prediction. Then, through the collection of models in the ensemble, each category gets an averaged confidence, probabiity and number of votes. Thus you can decide whether to operate the ensemble using the `confidence`, the `probability` or the `votes` so that the predicted category is the one that scores higher in any of these quantities. The criteria can be set using the *operating_kind* option (default is set to `probability`):

```
ensemble.predict({"petal length": 3, "petal width": 1}, \
                 operating_kind="votes")
```

Regression will generate a predictiona and an associated error, however `Boosted Trees` don't have an associated confidence measure, so only the prediction will be obtained in this case.

For consistency of interface with the `Model` class, as well as between boosted and non-boosted ensembles, local Ensembles again have a `predict_probability` method. This takes the same optional arguments as `Model.predict`: `missing_strategy` and `compact`. As with local Models, if `compact` is `False` (the default), the output is a list of maps, each with the keys `prediction` and `probability` mapped to the class name and its associated probability.

So, for example:

```
ensemble.predict_probability({"petal length": 3, "petal width": 1})

[{'category': u'Iris-setosa', 'probability': 0.006733220044732548},
 {'category': u'Iris-versicolor', 'probability': 0.9824478534614787},
 {'category': u'Iris-virginica', 'probability': 0.0108189264937886}]
```

If `compact` is `True`, only the probabilities themselves are returned, as a list in class name order, again, as is the case with local Models.

Operating point predictions are also available for local ensembles and an example of it would be:

```
operating_point = {"kind": "probability",
                   "positive_class": "True",
                   "threshold": 0.8};
prediction = local_ensemble.predict(inputData,
                                    operating_point=operating_point)
```

You can check the *Operating point's predictions* section to learn about operating points. For ensembles, three kinds of operating points are available: `votes`, `probability` and `confidence`. `Votes` will use as threshold the number of models in the ensemble that vote for the positive class. The other two are already explained in the above mentioned section.

CHAPTER 63

Local Ensemble Predictor

Predictions can take longer when the ensemble is formed by a large number of models or when its models have a high number of nodes. In these cases, predictions' speed can be increased and memory usage minimized by using the `EnsemblePredictor` object. The basic example to build it is:

```
from bigml.ensemblepredictor import EnsemblePredictor
ensemble = EnsemblePredictor('ensemble/5143a51a37203f2cf7020351',
                             "./model_fns_directory")
ensemble.predict({"petal length": 3, "petal width": 1})
(u'Iris-versicolor', 0.91519)
```

This constructor has two compulsory attributes: then ensemble ID (or the corresponding API response) and the path to a directory that contains a file per each of the ensemble models. Each file stores the `predict` function needed to obtain the model's predictions. As in the `Ensemble` object, you can also add an `api` argument with the connection to be used to download the ensemble's JSON information.

The functions stored in this directory are generated automatically the first time you instantiate the ensemble. Once they are generated, the functions are retrieved from the directory.

Note that only last prediction missings strategy is available for these predictions.

CHAPTER 64

Local Supervised Model

There's a general class that will allow you to predict using any supervised model resource, regardless of its particular type (model, ensemble, logistic regression or deepnet).

The `SupervisedModel` object will retrieve the resource information and instantiate the corresponding local object, so that you can use its `predict` method to produce local predictions:

CHAPTER 65

Fields

Once you have a resource, you can use the Fields class to generate a representation that will allow you to easily list fields, get fields ids, get a field id by name, column number, etc.

```
from bigml.api import BigML
from bigml.fields import Fields
api = BigML()
source = api.get_source("source/5143a51a37203f2cf7000974")

fields = Fields(source)
```

you can also instantiate the Fields object from the fields dict itself:

```
from bigml.api import BigML
from bigml.fields import Fields
api = BigML()
source = api.get_source("source/5143a51a37203f2cf7000974")

fields = Fields(source['object']['fields'])
```

The newly instantiated Fields object will give direct methods to retrieve different fields properties:

```
# Internal id of the 'sepal length' field
fields.field_id('sepal length')

# Field name of field with column number 0
fields.field_name(0)

# Column number of field name 'petal length'
fields.field_column_number('petal length')

# Statistics of values in field name 'petal length')
fields.stats('petal length')
```

Depending on the resource type, Fields information will vary. Sources will have only the name, label, description, type of field (optype) while dataset resources will have also the preferred (whether a field will is selectable as predictor), `missing_count`, `errors` and a summary of the values found in each field. This is due to the fact that the `source` object is built by inspecting the contents of a sample of the uploaded file, while the `dataset` resource really reads all the uploaded information. Thus, dataset's fields structure will always be more complete than source's.

In both cases, you can extract the summarized information available using the `summary_csv` method:

```
from bigml.api import BigML
from bigml.fields import Fields
api = BigML()
dataset = api.get_dataset("dataset/5143a51a37203f2cf7300974")

fields = Fields(dataset)
fields.summary_csv("my_fields_summary.csv")
```

In this example, the information will be stored in the `my_fields_summary.csv` file. For the typical `iris.csv` data file, the summary will read:

field column	field ID	field name	field label	field description	field type	preferred	missing count	errors	contents summary	errors summary
0	000000	sepal length			numeric	true	0	0	[4.3, 7.9], mean: 5.84333	
1	000001	sepal width			numeric	false	0	0	[2, 4.4], mean: 3.05733	
2	000002	petal length			numeric	true	0	0	[1, 6.9], mean: 3.758	
3	000003	petal width			numeric	true	0	0	[0.1, 2.5], mean: 1.19933	
4	000004	species			categorical	true	0	0	3 categories: Iris-setosa (50), Iris-versicolor (50), Iris-virginica (50)	

Another utility in the `Fields` object will help you update the updatable attributes of your source or dataset fields. For instance, if you need to update the type associated to one field in your dataset, you can change the `field type` values in the previous file and use it to obtain the fields structure needed to update your source:

```
from bigml.api import BigML
from bigml.fields import Fields
api = BigML()
source = api.get_source("source/5143a51a37203f2cf7000974")

fields = Fields(source)
fields_update_info = fields.new_fields_structure("my_fields_summary.csv")
source = api.update_source(source, fields_update_info)
```

For both sources and datasets, the updatable attributes are name, label and description. In `sources` you can also update the type of the field, and in `datasets` you can update the `preferred` attribute.

In addition to that, you can also easily `pair` a list of values with fields ids what is very useful to make predictions.

For example, the following snippet may be useful to create local predictions using a csv file as input:

```
test_reader = csv.reader(open(dir + test_set))
local_model = Model(model)
for row in test_reader:
    input_data = fields.pair([float(val) for val in row], objective_field)
    prediction = local_model.predict(input_data)
```

If missing values are present, the `Fields` object can return a dict with the ids of the fields that contain missing values and its count. The following example:

```
from bigml.fields import Fields
from bigml.api import BigML
api = BigML()
dataset = api.get_dataset("dataset/5339d42337203f233e000015")

fields = Fields(dataset)
fields.missing_counts()
```

would output:

```
{'000003': 1, '000000': 1, '000001': 1}
```

if the there was a missing value in each of the fields whose ids are `000003`, `000000`, `000001`.

You can also obtain the counts of errors per field using the `errors_count` method of the api:

```
from bigml.api import BigML
api = BigML()
dataset = api.get_dataset("dataset/5339d42337203f233e000015")
api.error_counts(dataset)
```

The generated output is like the one in `missing_counts`, that is, the error counts per field:

```
{'000000': 1}
```

CHAPTER 66

Rule Generation

You can also use a local model to generate a IF-THEN rule set that can be very helpful to understand how the model works internally.

```
local_model.rules()
IF petal_length > 2.45 AND
    IF petal_width > 1.65 AND
        IF petal_length > 5.05 THEN
            species = Iris-virginica
        IF petal_length <= 5.05 AND
            IF sepal_width > 2.9 AND
                IF sepal_length > 5.95 AND
                    IF petal_length > 4.95 THEN
                        species = Iris-versicolor
                    IF petal_length <= 4.95 THEN
                        species = Iris-virginica
                IF sepal_length <= 5.95 THEN
                    species = Iris-versicolor
            IF sepal_width <= 2.9 THEN
                species = Iris-virginica
    IF petal_width <= 1.65 AND
        IF petal_length > 4.95 AND
            IF sepal_length > 6.05 THEN
                species = Iris-virginica
            IF sepal_length <= 6.05 AND
                IF sepal_width > 2.45 THEN
                    species = Iris-versicolor
                IF sepal_width <= 2.45 THEN
                    species = Iris-virginica
        IF petal_length <= 4.95 THEN
            species = Iris-versicolor
IF petal_length <= 2.45 THEN
    species = Iris-setosa
```

CHAPTER 67

Python, Tableau and Hadoop-ready Generation

If you prefer, you can also generate a Python function that implements the model and that can be useful to make the model actionable right away with `local_model.python()`.

```
local_model.python()
def predict_species(sepal_length=None,
                    sepal_width=None,
                    petal_length=None,
                    petal_width=None):
    """ Predictor for species from model/50a8e2d9eabcb404d2000293

        Predictive model by BigML - Machine Learning Made Easy
    """
    if (petal_length is None):
        return 'Iris-virginica'
    if (petal_length <= 2.45):
        return 'Iris-setosa'
    if (petal_length > 2.45):
        if (petal_width is None):
            return 'Iris-virginica'
        if (petal_width <= 1.65):
            if (petal_length <= 4.95):
                return 'Iris-versicolor'
            if (petal_length > 4.95):
                if (sepal_length is None):
                    return 'Iris-virginica'
                if (sepal_length <= 6.05):
                    if (petal_width <= 1.55):
                        return 'Iris-virginica'
                    if (petal_width > 1.55):
                        return 'Iris-versicolor'
                if (sepal_length > 6.05):
                    return 'Iris-virginica'
        if (petal_width > 1.65):
            if (petal_length <= 5.05):
```

(continues on next page)

(continued from previous page)

```
                if (sepal_width is None):
                    return 'Iris-virginica'
                if (sepal_width <= 2.9):
                    return 'Iris-virginica'
                if (sepal_width > 2.9):
                    if (sepal_length is None):
                        return 'Iris-virginica'
                    if (sepal_length <= 6.4):
                        if (sepal_length <= 5.95):
                            return 'Iris-versicolor'
                        if (sepal_length > 5.95):
                            return 'Iris-virginica'
                    if (sepal_length > 6.4):
                        return 'Iris-versicolor'
            if (petal_length > 5.05):
                return 'Iris-virginica'
```

The `local.python(hadoop=True)` call will generate the code that you need for the Hadoop map-reduce engine to produce batch predictions using Hadoop streaming . Saving the mapper and reducer generated functions in their corresponding files (let's say `/home/hduser/hadoop_mapper.py` and `/home/hduser/hadoop_reducer.py`) you can start a Hadoop job to generate predictions by issuing the following Hadoop command in your system console:

```
bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar \
-file /home/hduser/hadoop_mapper.py -mapper hadoop_mapper.py \
-file /home/hduser/hadoop_reducer.py -reducer hadoop_reducer.py \
-input /home/hduser/hadoop/input.csv \
-output /home/hduser/hadoop/output_dir
```

assuming you are in the Hadoop home directory, your input file is in the corresponding dfs directory (`/home/hduser/hadoop/input.csv` in this example) and the output will be placed at `/home/hduser/hadoop/output_dir` (inside the dfs directory).

Tableau-ready rules are also available through `local_model.tableau()` for all the models except those that use text predictors.

```
local_model.tableau()
IF ISNULL([petal width]) THEN 'Iris-virginica'
ELSEIF [petal width]>0.8 AND [petal width]>1.75 AND ISNULL([petal length]) THEN 'Iris-
↪virginica'
ELSEIF [petal width]>0.8 AND [petal width]>1.75 AND [petal length]>4.85 THEN 'Iris-
↪virginica'
ELSEIF [petal width]>0.8 AND [petal width]>1.75 AND [petal length]<=4.85 AND␣
↪ISNULL([sepal width]) THEN 'Iris-virginica'
ELSEIF [petal width]>0.8 AND [petal width]>1.75 AND [petal length]<=4.85 AND [sepal␣
↪width]>3.1 THEN 'Iris-versicolor'
ELSEIF [petal width]>0.8 AND [petal width]>1.75 AND [petal length]<=4.85 AND [sepal␣
↪width]<=3.1 THEN 'Iris-virginica'
ELSEIF [petal width]>0.8 AND [petal width]<=1.75 AND ISNULL([petal length]) THEN
↪'Iris-versicolor'
ELSEIF [petal width]>0.8 AND [petal width]<=1.75 AND [petal length]>4.95 AND [petal␣
↪width]>1.55 AND [petal length]>5.45 THEN 'Iris-virginica'
ELSEIF [petal width]>0.8 AND [petal width]<=1.75 AND [petal length]>4.95 AND [petal␣
↪width]>1.55 AND [petal length]<=5.45 THEN 'Iris-versicolor'
ELSEIF [petal width]>0.8 AND [petal width]<=1.75 AND [petal length]>4.95 AND [petal␣
↪width]<=1.55 THEN 'Iris-virginica'
```

(continues on next page)

(continued from previous page)

```
ELSEIF [petal width]>0.8 AND [petal width]<=1.75 AND [petal length]<=4.95 AND [petal␣
↪width]>1.65 THEN 'Iris-virginica'
ELSEIF [petal width]>0.8 AND [petal width]<=1.75 AND [petal length]<=4.95 AND [petal␣
↪width]<=1.65 THEN 'Iris-versicolor'
ELSEIF [petal width]<=0.8 THEN 'Iris-setosa'
END
```

CHAPTER 68

Summary generation

You can also print the model from the point of view of the classes it predicts with `local_model.summarize()`. It shows a header section with the training data initial distribution per class (instances and percentage) and the final predicted distribution per class.

Then each class distribution is detailed. First a header section shows the percentage of the total data that belongs to the class (in the training set and in the predicted results) and the rules applicable to all the the instances of that class (if any). Just after that, a detail section shows each of the leaves in which the class members are distributed. They are sorted in descending order by the percentage of predictions of the class that fall into that leaf and also show the full rule chain that leads to it.

```
Data distribution:
    Iris-setosa: 33.33% (50 instances)
    Iris-versicolor: 33.33% (50 instances)
    Iris-virginica: 33.33% (50 instances)


Predicted distribution:
    Iris-setosa: 33.33% (50 instances)
    Iris-versicolor: 33.33% (50 instances)
    Iris-virginica: 33.33% (50 instances)


Field importance:
    1. petal length: 53.16%
    2. petal width: 46.33%
    3. sepal length: 0.51%
    4. sepal width: 0.00%


Iris-setosa : (data 33.33% / prediction 33.33%) petal length <= 2.45
    · 100.00%: petal length <= 2.45 [Confidence: 92.86%]


Iris-versicolor : (data 33.33% / prediction 33.33%) petal length > 2.45
```

(continues on next page)

(continued from previous page)

```
    · 94.00%: petal length > 2.45 and petal width <= 1.65 and petal length <= 4.95␣
↪[Confidence: 92.44%]
    · 2.00%: petal length > 2.45 and petal width <= 1.65 and petal length > 4.95 and␣
↪sepal length <= 6.05 and petal width > 1.55 [Confidence: 20.65%]
    · 2.00%: petal length > 2.45 and petal width > 1.65 and petal length <= 5.05 and␣
↪sepal width > 2.9 and sepal length > 6.4 [Confidence: 20.65%]
    · 2.00%: petal length > 2.45 and petal width > 1.65 and petal length <= 5.05 and␣
↪sepal width > 2.9 and sepal length <= 6.4 and sepal length <= 5.95 [Confidence: 20.
↪65%]


Iris-virginica : (data 33.33% / prediction 33.33%) petal length > 2.45
    · 76.00%: petal length > 2.45 and petal width > 1.65 and petal length > 5.05␣
↪[Confidence: 90.82%]
    · 12.00%: petal length > 2.45 and petal width > 1.65 and petal length <= 5.05 and␣
↪sepal width <= 2.9 [Confidence: 60.97%]
    · 6.00%: petal length > 2.45 and petal width <= 1.65 and petal length > 4.95 and␣
↪sepal length > 6.05 [Confidence: 43.85%]
    · 4.00%: petal length > 2.45 and petal width > 1.65 and petal length <= 5.05 and␣
↪sepal width > 2.9 and sepal length <= 6.4 and sepal length > 5.95 [Confidence: 34.24
↪%]
    · 2.00%: petal length > 2.45 and petal width <= 1.65 and petal length > 4.95 and␣
↪sepal length <= 6.05 and petal width <= 1.55 [Confidence: 20.65%]
```

You can also use `local_model.get_data_distribution()` and `local_model.get_prediction_distribution()` to obtain the training and prediction basic distribution information as a list (suitable to draw histograms or any further processing). The tree nodes' information (prediction, confidence, impurity and distribution) can also be retrieved in a CSV format using the method `local_model.tree_CSV()`. The output can be sent to a file by providing a `file_name` argument or used as a list.

Local ensembles have a `local_ensemble.summarize()` method too, the output in this case shows only the data distribution (only available in `Decision Forests`) and field importance sections.

For local clusters, the `local_cluster.summarize()` method prints also the data distribution, the training data statistics per cluster and the basic intercentroid distance statistics. There's also a `local_cluster.statistics_CSV(file_name)` method that store in a CSV format the values shown by the `summarize()` method. If no file name is provided, the function returns the rows that would have been stored in the file as a list.

CHAPTER 69

Running the Tests

The test will be run using nose , that is installed on setup, and you'll need to set up your authentication via environment variables, as explained below. With that in place, you can run the test suite simply by issuing .. code-block:: bash

> $ python setup.py nosetests

Some tests need the numpy and scipy libraries to be installed too. They are not automatically installed as a dependency, as they are quite heavy and very seldom used.

Additionally, Tox can be used to automatically run the test suite in virtual environments for all supported Python versions. To install Tox:

```
$ pip install tox
```

Then run the tests from the top-level project directory:

```
$ tox
```

CHAPTER 70

Building the Documentation

Install the tools required to build the documentation:

```
$ pip install sphinx
```

To build the HTML version of the documentation:

```
$ cd docs/
$ make html
```

Then launch `docs/_build/html/index.html` in your browser.

CHAPTER 71

Additional Information

For additional information about the API, see the BigML developer's documentation.