pandas-ml Documentation

Release 0.3.0

sinhrks

Mar 05, 2019

Contents

Contents:

CHAPTER 1

What's new

1.1 v0.6.1

1.1.1 Enhancement

- Support pandas v0.24.0.

1.2 v0.6.0

1.2.1 Enhancement

- Support pandas v0.22.0 and scikit-learn 0.20.0.

1.2.2 API Change

- *ModelFrame.model_selection.describe* now returns *ModelFrame* compat with *GridSearchCV.cv_results_*

1.2.3 Deprecation

- Drop support of pandas v0.18.x or earlier
- Drop support of scikit-learn v0.18.x or earlier.

1.3 v0.5.0

1.3.1 Enhancement

- Support pandas v0.21.0.

1.4 v0.4.0

1.4.1 Enhancement

- Support scikit-learn v0.17.x and v0.18.0.
- Support imbalanced-learn via `.imbalance` accessor. See *Handling imbalanced data*.
- Added `pandas_ml.ConfusionMatrix` class for easier classification results evaluation. See *Confusion matrix*.

1.4.2 Bug Fix

- `ModelFrame.columns` may not be preserved via `.transform` using `FunctionTransformer`, `KernelCenterer`, `MaxAbsScaler` and `RobustScaler`.

1.5 v0.3.1

1.5.1 Enhancement

- `inverse_transform` now reverts original `ModelFrame.columns` information.

1.5.2 Bug Fix

- Assigning `Series` to `ModelFrame.data` property raises `TypeError`

1.6 v0.3.0

1.6.1 Enhancement

- Support `xgboost` via `ModelFrame.xgboost` accessor.

1.7 v0.2.0

1.7.1 Enhancement

- `ModelFrame.transform` can preserve column names for some `sklearn.preprocessing` transformation.

- Added `ModelSeries.fit`, `transform`, `fit_transform` and `inverse_transform` for preprocessing purpose.
- `ModelFrame` can be initialized from `statsmodels` datasets.
- `ModelFrame.cross_validation.iterate` and `ModelFrame.cross_validation.train_test_split` now keep index of original dataset, and added `reset_index` keyword to control this behaviour.

1.7.2 Bug Fix

- `target` kw may be ignored when initializing `ModelFrame` with `np.ndarray` and `columns` kwds.
- `linear_model.enet_path` doesn't accept additional keywords.
- Initializing `ModelFrame` with named `Series` may have duplicated target columns.
- `ModelFrame.target_name` may not be preserved when sliced.

1.8 v0.1.1

1.8.1 Enhancement

- Added `sklearn.learning_curve`, `neural_network`, `random_projection`

1.9 v0.1.0

- Initial Release

CHAPTER 2

Data Handling

2.1 Data Preparation

This section describes how to prepare basic data format named `ModelFrame`. `ModelFrame` defines a metadata to specify target (response variable) and data (explanatory variable / features). Using these metadata, `ModelFrame` can call other statistics/ML functions in more simple way.

You can create `ModelFrame` as the same manner as `pandas.DataFrame`. The below example shows how to create basic `ModelFrame`, which DOESN'T have target values.

```
>>> import pandas_ml as pdml

>>> df = pdml.ModelFrame({'A': [1, 2, 3], 'B': [2, 3, 4],
...                       'C': [3, 4, 5]}, index=['a', 'b', 'c'])
>>> df
   A  B  C
a  1  2  3
b  2  3  4
c  3  4  5

>>> type(df)
<class 'pandas_ml.core.frame.ModelFrame'>
```

You can check whether the created `ModelFrame` has target values using `ModelFrame.has_target()` method.

```
>>> df.has_target()
False
```

Target values can be specifyied via `target` keyword. You can simply pass a column name to be handled as target. Target column name can be confirmed via `target_name` property.

```
>>> df2 = pdml.ModelFrame({'A': [1, 2, 3], 'B': [2, 3, 4],
...                        'C': [3, 4, 5]}, target='A')
>>> df2
```

(continues on next page)

(continued from previous page)

```
   A  B  C
0  1  2  3
1  2  3  4
2  3  4  5

>>> df2.has_target()
True

>>> df2.target_name
'A'
```

Also, you can pass any list-likes to be handled as a target. In this case, target column will be named as ".target".

```
>>> df3 = pdml.ModelFrame({'A': [1, 2, 3], 'B': [2, 3, 4],
...                        'C': [3, 4, 5]}, target=[4, 5, 6])
>>> df3
   .target  A  B  C
0        4  1  2  3
1        5  2  3  4
2        6  3  4  5

>>> df3.has_target()
True

>>> df3.target_name
'.target'
```

Also, you can pass `pandas.DataFrame` and `pandas.Series` as data and target.

```
>>> import pandas as pd
df4 = pdml.ModelFrame({'A': [1, 2, 3], 'B': [2, 3, 4],
...                    'C': [3, 4, 5]}, target=pd.Series([4, 5, 6]))
>>> df4
   .target  A  B  C
0        4  1  2  3
1        5  2  3  4
2        6  3  4  5

>>> df4.has_target()
True

>>> df4.target_name
'.target'
```

Note: Target values are mandatory to perform operations which require response variable, such as regression and supervised learning.

2.2 Data Manipulation

You can maniluplate `ModelFrame` like `pandas.DataFrame`. Because `ModelFrame` inherits `pandas.DataFrame`, all the `pandas` methods / functions can be applied to `ModelFrame`.

Sliced results will be `ModelSeries` (simple wrapper for `pandas.Series` to support some data manipulation) or `ModelFrame`

```
>>> df
   A  B  C
a  1  2  3
b  2  3  4
c  3  4  5

>>> sliced = df['A']
>>> sliced
a    1
b    2
c    3
Name: A, dtype: int64

>>> type(sliced)
<class 'pandas_ml.core.series.ModelSeries'>

>>> subset = df[['A', 'B']]
>>> subset
   A  B
a  1  2
b  2  3
c  3  4

>>> type(subset)
<class 'pandas_ml.core.frame.ModelFrame'>
```

`ModelFrame` has a special properties `data` to access data (features) and `target` to access target.

```
>>> df2
   A  B  C
0  1  2  3
1  2  3  4
2  3  4  5

>>> df2.target_name
'A'

>>> df2.data
   B  C
0  2  3
1  3  4
2  4  5

>>> df2.target
0    1
1    2
2    3
Name: A, dtype: int64
```

You can update data and target via properties. Also, columns / value assignment are supported as the same as `pandas.DataFrame`.

```
>>> df2.target = [9, 9, 9]
>>> df2
   A  B  C
```

(continues on next page)

(continued from previous page)

```
0   9   2   3
1   9   3   4
2   9   4   5

>>> df2.data = pd.DataFrame({'X': [1, 2, 3], 'Y': [4, 5, 6]})
>>> df2
   A   X   Y
0   9   1   4
1   9   2   5
2   9   3   6

>>> df2['X'] = [0, 0, 0]
>>> df2
   A   X   Y
0   9   0   4
1   9   0   5
2   9   0   6
```

You can change target column specifying `target_name` property.

```
>>> df2.target_name
'A'

>>> df2.target_name = 'X'
>>> df2.target_name
'X'
```

If the specified column doesn't exist in `ModelFrame`, it should reset target to `None`. Current target will be regarded as data.

```
>>> df2.target_name
'X'

>>> df2.target_name = 'XXXX'
>>> df2.has_target()
False

>>> df2.data
   A   X   Y
0   9   0   4
1   9   0   5
2   9   0   6
```

CHAPTER 3

Use scikit-learn

This section describes how to use `scikit-learn` functionalities via `pandas-ml`.

3.1 Basics

You can create `ModelFrame` instance from `scikit-learn` datasets directly.

```
>>> import pandas_ml as pdml
>>> import sklearn.datasets as datasets

>>> df = pdml.ModelFrame(datasets.load_iris())
>>> df.head()
   .target  sepal length (cm)  sepal width (cm)  petal length (cm)  \
0        0                5.1               3.5                1.4
1        0                4.9               3.0                1.4
2        0                4.7               3.2                1.3
3        0                4.6               3.1                1.5
4        0                5.0               3.6                1.4

   petal width (cm)
0               0.2
1               0.2
2               0.2
3               0.2
4               0.2

# make columns be readable
>>> df.columns = ['.target', 'sepal length', 'sepal width', 'petal length', 'petal␣
↪width']
```

`ModelFrame` has accessor methods which makes easier access to `scikit-learn` namespace.

```
>>> df.cluster.KMeans
<class 'sklearn.cluster.k_means_.KMeans'>
```

Following table shows `scikit-learn` module and corresponding `ModelFrame` module. Some accessors has its abbreviated versions.

Thus, you can instanciate each estimator via `ModelFrame` accessors. Once create an estimator, you can pass it to `ModelFrame.fit` then `predict`. `ModelFrame` automatically uses its data and target properties for each operations.

```
>>> estimator = df.cluster.KMeans(n_clusters=3)
>>> df.fit(estimator)

>>> predicted = df.predict(estimator)
>>> predicted
0     1
1     1
2     1
...
147    2
148    2
149    0
Length: 150, dtype: int32
```

`ModelFrame` preserves the most recently used estimator in `estimator` atribute, and predicted results in `predicted` attibute.

```
>>> df.estimator
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=3, n_init=10,
    n_jobs=1, precompute_distances=True, random_state=None, tol=0.0001,
    verbose=0)

>>> df.predicted
0     1
1     1
2     1
...
147    2
148    2
149    0
Length: 150, dtype: int32
```

`ModelFrame` has following methods corresponding to various `scikit-learn` estimators. The last results are saved as corresponding `ModelFrame` properties.

`ModelFrame` method	`ModelFrame` property
`ModelFrame.fit`	(None)
`ModelFrame.transform`	(None)
`ModelFrame.fit_transform`	(None)
`ModelFrame.inverse_transform`	(None)
`ModelFrame.predict`	`ModelFrame.predicted`
`ModelFrame.fit_predict`	`ModelFrame.predicted`
`ModelFrame.score`	(None)
`ModelFrame.predict_proba`	`ModelFrame.proba`
`ModelFrame.predict_log_proba`	`ModelFrame.log_proba`
`ModelFrame.decision_function`	`ModelFrame.decision`

Note: If you access to a property before calling `ModelFrame` methods, `ModelFrame` automatically calls corresponding method of the latest estimator and return the result.

Following example shows to perform PCA, then revert principal components back to original space. `inverse_transform` should revert the original columns.

```
>>> estimator = df.decomposition.PCA()
>>> df.fit(estimator)

>>> transformed = df.transform(estimator)
>>> transformed.head()
   .target          0         1         2         3
0        0 -2.684207 -0.326607  0.021512  0.001006
1        0 -2.715391  0.169557  0.203521  0.099602
2        0 -2.889820  0.137346 -0.024709  0.019305
3        0 -2.746437  0.311124 -0.037672 -0.075955
4        0 -2.728593 -0.333925 -0.096230 -0.063129

>>> type(transformed)
<class 'pandas_ml.core.frame.ModelFrame'>

>>> transformed.inverse_transform(estimator)
     .target  sepal length  sepal width  petal length  petal width
0          0           5.1          3.5           1.4          0.2
1          0           4.9          3.0           1.4          0.2
2          0           4.7          3.2           1.3          0.2
3          0           4.6          3.1           1.5          0.2
4          0           5.0          3.6           1.4          0.2
..       ...           ...          ...           ...          ...
145        2           6.7          3.0           5.2          2.3
146        2           6.3          2.5           5.0          1.9
147        2           6.5          3.0           5.2          2.0
148        2           6.2          3.4           5.4          2.3
149        2           5.9          3.0           5.1          1.8

[150 rows x 5 columns]
```

If `ModelFrame` both has `target` and `predicted` values, the model evaluation can be performed using functions available in `ModelFrame.metrics`.

```
>>> estimator = df.svm.SVC()
>>> df.fit(estimator)

>>> df.predict(estimator)
0     0
1     0
2     0
...
147    2
148    2
149    2
Length: 150, dtype: int64

>>> df.predicted
0     0
1     0
```

(continues on next page)

(continued from previous page)

```
2    0
...
147    2
148    2
149    2
Length: 150, dtype: int64

>>> df.metrics.confusion_matrix()
Predicted   0   1   2
Target
0          50   0   0
1           0  48   2
2           0   0  50
```

3.2 Use Module Level Functions

Some `scikit-learn` modules define functions which handle data without instanciating estimators. You can call these functions from accessor methods directly, and `ModelFrame` will pass corresponding data on background. Following example shows to use `sklearn.cluster.k_means` function to perform K-means.

Important: When you use module level function, `ModelFrame.predicted` WILL NOT be updated. Thus, using estimator is recommended.

```
# no need to pass data explicitly
# sklearn.cluster.kmeans returns centroids, cluster labels and inertia
>>> c, l, i = df.cluster.k_means(n_clusters=3)
>>> l
0      1
1      1
2      1
...
147    2
148    2
149    0
Length: 150, dtype: int32
```

3.3 Pipeline

`ModelFrame` can handle pipeline as the same as normal estimators.

```
>>> estimators = [('reduce_dim', df.decomposition.PCA()),
...               ('svm', df.svm.SVC())]
>>> pipe = df.pipeline.Pipeline(estimators)
>>> df.fit(pipe)

>>> df.predict(pipe)
0    0
1    0
2    0
```

(continues on next page)

(continued from previous page)

```
...
147    2
148    2
149    2
Length: 150, dtype: int64
```

Above expression is the same as below:

```
>>> df2 = df.copy()
>>> df2 = df2.fit_transform(df2.decomposition.PCA())
>>> svm = df2.svm.SVC()
>>> df2.fit(svm)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
>>> df2.predict(svm)
0      0
1      0
2      0
...
147    2
148    2
149    2
Length: 150, dtype: int64
```

3.4 Cross Validation

`scikit-learn` has some classes for cross validation. `model_selection.train_test_split` splits data to training and test set. You can access to the function via `model_selection` accessor.

```
>>> train_df, test_df = df.model_selection.train_test_split()
>>> train_df
     .target  sepal length  sepal width  petal length  petal width
124        2           6.7          3.3           5.7          2.1
117        2           7.7          3.8           6.7          2.2
123        2           6.3          2.7           4.9          1.8
65         1           6.7          3.1           4.4          1.4
133        2           6.3          2.8           5.1          1.5
..       ...           ...          ...           ...          ...
93         1           5.0          2.3           3.3          1.0
46         0           5.1          3.8           1.6          0.2
121        2           5.6          2.8           4.9          2.0
91         1           6.1          3.0           4.6          1.4
147        2           6.5          3.0           5.2          2.0

[112 rows x 5 columns]


>>> test_df
     .target  sepal length  sepal width  petal length  petal width
146        2           6.3          2.5           5.0          1.9
75         1           6.6          3.0           4.4          1.4
138        2           6.0          3.0           4.8          1.8
77         1           6.7          3.0           5.0          1.7
```

(continues on next page)

(continued from previous page)

```
36          0           5.5          3.5          1.3          0.2
..        ...           ...          ...          ...          ...
14          0           5.8          4.0          1.2          0.2
141         2           6.9          3.1          5.1          2.3
100         2           6.3          3.3          6.0          2.5
83          1           6.0          2.7          5.1          1.6
114         2           5.8          2.8          5.1          2.4

[38 rows x 5 columns]
```

You can iterate over Splitter classes via `ModelFrame.model_selection.split` which returns `ModelFrame` corresponding to training and test data.

```
>>> kf = df.model_selection.KFold(n_splits=3)
>>> for train_df, test_df in df.model_selection.iterate(kf):
...    print('training set shape: ', train_df.shape,
...          'test set shape: ', test_df.shape)
training set shape:  (112, 5) test set shape:  (38, 5)
training set shape:  (112, 5) test set shape:  (38, 5)
training set shape:  (112, 5) test set shape:  (38, 5)
```

3.5 Grid Search

You can perform grid search using `ModelFrame.fit`.

```
>>> tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
...                      'C': [1, 10, 100]},
...                     {'kernel': ['linear'], 'C': [1, 10, 100]}]

>>> df = pdml.ModelFrame(datasets.load_digits())
>>> cv = df.model_selection.GridSearchCV(df.svm.SVC(C=1), tuned_parameters,
...                                      cv=5)

>>> df.fit(cv)

>>> cv.best_estimator_
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.001,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In addition, `ModelFrame.model_selection` has a `describe` function to organize each grid search result as `ModelFrame` accepting estimator.

```
>>> df.model_selection.describe(cv)
       mean       std    C   gamma  kernel
0  0.974108  0.013139    1  0.0010     rbf
1  0.951416  0.020010    1  0.0001     rbf
2  0.975372  0.011280   10  0.0010     rbf
3  0.962534  0.020218   10  0.0001     rbf
4  0.975372  0.011280  100  0.0010     rbf
5  0.964695  0.016686  100  0.0001     rbf
6  0.951811  0.018410    1     NaN  linear
7  0.951811  0.018410   10     NaN  linear
8  0.951811  0.018410  100     NaN  linear
```

CHAPTER 4

Handling imbalanced data

This section describes how to use imbalanced-learn functionalities via `pandas-ml` to handle imbalanced data.

4.1 Sampling

Assuming we have `ModelFrame` which has imbalanced target values. The `ModelFrame` has data with 80 observations labeld with `0` and 20 observations labeled with `1`.

```
>>> import numpy as np
>>> import pandas_ml as pdml
>>> df = pdml.ModelFrame(np.random.randn(100, 5),
...                      target=np.array([0, 1]).repeat([80, 20]),
...                      columns=list('ABCDE'))
>>> df
    .target         A         B         C         D         E
0         0  1.467859  1.637449  0.175770  0.189108  0.775139
1         0 -1.706293 -0.598930 -0.343427  0.355235 -1.348378
2         0  0.030542  0.393779 -1.891991  0.041062  0.055530
3         0  0.320321 -1.062963 -0.416418 -0.629776  1.126027
..      ...       ...       ...       ...       ...       ...
96        1 -1.199039  0.055702  0.675555 -0.416601 -1.676259
97        1 -1.264182 -0.167390 -0.939794 -0.638733 -0.806794
98        1 -0.616754  1.667483 -1.858449 -0.259630  1.236777
99        1 -1.374068 -0.400435 -1.825555  0.824052 -0.335694

[100 rows x 6 columns]

>>> df.target.value_counts()
0    80
1    20
Name: .target, dtype: int64
```

You can access `imbalanced-learn` namespace via `.imbalance` accessor. Passing instanciated under-sampling class to `ModelFrame.fit_sample` returns under sampled `ModelFrame` (Note that `.index` is reset).

```
>>> sampler = df.imbalance.under_sampling.ClusterCentroids()
>>> sampler
ClusterCentroids(n_jobs=-1, random_state=None, ratio='auto')

>>> sampled = df.fit_sample(sampler)
>>> sampled
    .target         A         B         C         D         E
0         1  0.232841 -1.364282  1.436854  0.563796 -0.372866
1         1 -0.159551  0.473617 -2.024209  0.760444 -0.820403
2         1  1.495356 -2.144495  0.076485  1.219948  0.382995
3         1 -0.736887  1.399623  0.557098  0.621909 -0.507285
..      ...       ...       ...       ...       ...       ...
36        0  0.429978 -1.421307  0.771368  1.704277  0.645590
37        0  1.408448  0.132760 -1.082301 -1.195149  0.155057
38        0  0.362793 -0.682171  1.026482  0.663343 -2.371229
39        0 -0.796293 -0.196428 -0.747574  2.228031 -0.468669

[40 rows x 6 columns]

>>> sampled.target.value_counts()
1    20
0    20
Name: .target, dtype: int64
```

As the same manner, you can perform over-sampling.

```
>>> sampler = df.imbalance.over_sampling.SMOTE()
>>> sampler
SMOTE(k=5, kind='regular', m=10, n_jobs=-1, out_step=0.5, random_state=None,
ratio='auto')

>>> sampled = df.fit_sample(sampler)
>>> sampled
     .target         A         B         C         D         E
0          0  1.467859  1.637449  0.175770  0.189108  0.775139
1          0 -1.706293 -0.598930 -0.343427  0.355235 -1.348378
2          0  0.030542  0.393779 -1.891991  0.041062  0.055530
3          0  0.320321 -1.062963 -0.416418 -0.629776  1.126027
..       ...       ...       ...       ...       ...       ...
156        1 -1.279399  0.218171 -0.487836 -0.573564  0.582580
157        1 -0.736964  0.239095 -0.422025 -0.841780  0.221591
158        1 -0.273911 -0.305608 -0.886088  0.062414 -0.001241
159        1  0.073145 -0.167884 -0.781611 -0.016734 -0.045330

[160 rows x 6 columns]'

>>> sampled.target.value_counts()
1    80
0    80
Name: .target, dtype: int64
```

Following table shows `imbalanced-learn` module and corresponding `ModelFrame` module.

imbalanced-learn	ModelFrame accessor
imblearn.under_sampling	ModelFrame.imbalance.under_sampling
imblearn.over_sampling	ModelFrame.imbalance.over_sampling
imblearn.combine	ModelFrame.imbalance.combine
imblearn.ensemble	ModelFrame.imbalance.ensemble

CHAPTER 5

Use XGBoost

This section describes how to use `XGBoost` functionalities via `pandas-ml`.

Use `scikit-learn` digits dataset as sample data.

```
>>> import pandas_ml as pdml
>>> import sklearn.datasets as datasets

>>> df = pdml.ModelFrame(datasets.load_digits())
>>> df.head()
   .target  0  1  2 ...  60  61  62  63
0        0  0  0  5 ...  10   0   0   0
1        1  0  0  0 ...  16  10   0   0
2        2  0  0  0 ...  11  16   9   0
3        3  0  0  7 ...  13   9   0   0
4        4  0  0  0 ...  16   4   0   0

[5 rows x 65 columns]
```

As an estimator, `XGBClassifier` and `XGBRegressor` are available via `xgboost` accessor. See XGBoost Scikit-learn API for details.

```
>>> df.xgboost.XGBClassifier
<class 'xgboost.sklearn.XGBClassifier'>

>>> df.xgboost.XGBRegressor
<class 'xgboost.sklearn.XGBRegressor'>
```

You can use these estimators like `scikit-learn` estimators.

```
>>> train_df, test_df = df.model_selection.train_test_split()

>>> estimator = df.xgboost.XGBClassifier()

>>> train_df.fit(estimator)
```

(continues on next page)

(continued from previous page)

```
XGBClassifier(base_score=0.5, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, nthread=-1, objective='multi:softprob', seed=0,
       silent=True, subsample=1)

>>> predicted = test_df.predict(estimator)

>>> predicted
1371    2
1090    3
1299    2
...
1286    8
1632    3
538     2
dtype: int64

>>> test_df.metrics.confusion_matrix()
Predicted   0   1   2   3 ...   6   7   8   9
Target                    ...
0          53   0   0   0 ...   0   0   1   0
1           0  46   0   0 ...   0   0   0   0
2           0   0  51   1 ...   0   0   1   0
3           0   0   0  33 ...   0   0   1   0
4           0   0   0   0 ...   0   0   0   1
5           0   0   0   0 ...   1   0   0   1
6           0   0   0   0 ...  39   0   1   0
7           0   0   0   0 ...   0  40   0   1
8           1   0   0   0 ...   1   0  32   2
9           0   1   0   0 ...   0   1   1  51

[10 rows x 10 columns]
```

Also, plotting functions are available via `xgboost` accessor.

```
>>> train_df.xgboost.plot_importance()
# importance plot will be displayed
```

`XGBoost` estimators can be passed to other `scikit-learn` APIs. Following example shows to perform a grid search.

```
>>> tuned_parameters = [{'max_depth': [3, 4]}]
>>> cv = df.model_selection.GridSearchCV(df.xgb.XGBClassifier(), tuned_parameters,␣
↪cv=5)

>>> df.fit(cv)
>>> df.model_selection.describe(cv)
       mean       std  max_depth
0  0.917641  0.032600          3
1  0.919310  0.026644          4
```

CHAPTER 6

Use patsy

This section describes data transformation using `patsy`. `ModelFrame.transform` can accept `patsy` style formula.

```
>>> import pandas_ml as pdml

# create modelframe which doesn't have target
>>> df = pdml.ModelFrame({'X': [1, 2, 3], 'Y': [2, 3, 4],
...                       'Z': [3, 4, 5]}, index=['a', 'b', 'c'])

>>> df
   X  Y  Z
a  1  2  3
b  2  3  4
c  3  4  5

# transform with patsy formula
>>> transformed = df.transform('Z ~ Y + X')
>>> transformed
   Z  Intercept  Y  X
a  3          1  2  1
b  4          1  3  2
c  5          1  4  3

# transformed data should have target specified by formula
>>> transformed.target
a    3
b    4
c    5
Name: Z, dtype: float64

>>> transformed.data
   Intercept  Y  X
a          1  2  1
b          1  3  2
c          1  4  3
```

If you do not want intercept, specify with 0.

```
>>> df.transform('Z ~ Y + 0')
   Z  Y
a  3  2
b  4  3
c  5  4
```

Also, you can use formula which doesn't have left side.

```
# create modelframe which has target
>>> df2 = pdml.ModelFrame({'X': [1, 2, 3], 'Y': [2, 3, 4],'Z': [3, 4, 5]},
...                       target =[7, 8, 9], index=['a', 'b', 'c'])

>>> df2
   .target  X  Y  Z
a        7  1  2  3
b        8  2  3  4
c        9  3  4  5

# overwrite data with transformed data
>>> df2.data = df2.transform('Y + Z')
>>> df2
   .target  Intercept  Y  Z
a        7          1  2  3
b        8          1  3  4
c        9          1  4  5

# data has been updated based on formula
>>> df2.data
   Intercept  Y  Z
a          1  2  3
b          1  3  4
c          1  4  5

# target is not changed
>>> df2.target
a    7
b    8
c    9
Name: .target, dtype: int64
```

Below example is performing deviation coding via patsy formula.

```
>>> df3 = pdml.ModelFrame({'X': [1, 2, 3, 4, 5], 'Y': [1, 3, 2, 2, 1],
...                        'Z': [1, 1, 1, 2, 2]}, target='Z',
...                        index=['a', 'b', 'c', 'd', 'e'])
```

```
>>> df3
   X  Y  Z
a  1  1  1
b  2  3  1
c  3  2  1
d  4  2  2
e  5  1  2
```

```
>>> df3.transform('C(X, Sum)')
```

(continues on next page)

(continued from previous page)

```
   Intercept  C(X, Sum)[S.1]  C(X, Sum)[S.2]  C(X, Sum)[S.3]  C(X, Sum)[S.4]
a          1               1               0               0               0
b          1               0               1               0               0
c          1               0               0               1               0
d          1               0               0               0               1
e          1              -1              -1              -1              -1
```

```
>>> df3.transform('C(Y, Sum)')
   Intercept  C(Y, Sum)[S.1]  C(Y, Sum)[S.2]
a          1               1               0
b          1              -1              -1
c          1               0               1
d          1               0               1
e          1               1               0
```

CHAPTER 7

Confusion matrix

Import `ConfusionMatrix`

```
from pandas_ml import ConfusionMatrix
```

Define actual values (`y_true`) and predicted values (`y_pred`)

```
y_true = ['rabbit', 'cat', 'rabbit', 'rabbit', 'cat', 'dog', 'dog', 'rabbit', 'rabbit
↪', 'cat', 'dog', 'rabbit']
y_pred = ['cat', 'cat', 'rabbit', 'dog', 'cat', 'rabbit', 'dog', 'cat', 'rabbit', 'cat
↪', 'rabbit', 'rabbit']
```

Let's define a (non binary) confusion matrix

```
confusion_matrix = ConfusionMatrix(y_true, y_pred)
print("Confusion matrix:\n%s" % confusion_matrix)
```

You can see it

```
Predicted  cat  dog  rabbit  __all__
Actual
cat          3    0       0        3
dog          0    1       2        3
rabbit       2    1       3        6
__all__      5    2       5       12
```

7.1 Matplotlib plot of a confusion matrix

Inside a IPython notebook add this line as first cell

```
%matplotlib inline
```

You can plot confusion matrix using:

```
import matplotlib.pyplot as plt

confusion_matrix.plot()
```

If you are not using inline mode, you need to use to show confusion matrix plot.

```
plt.show()
```

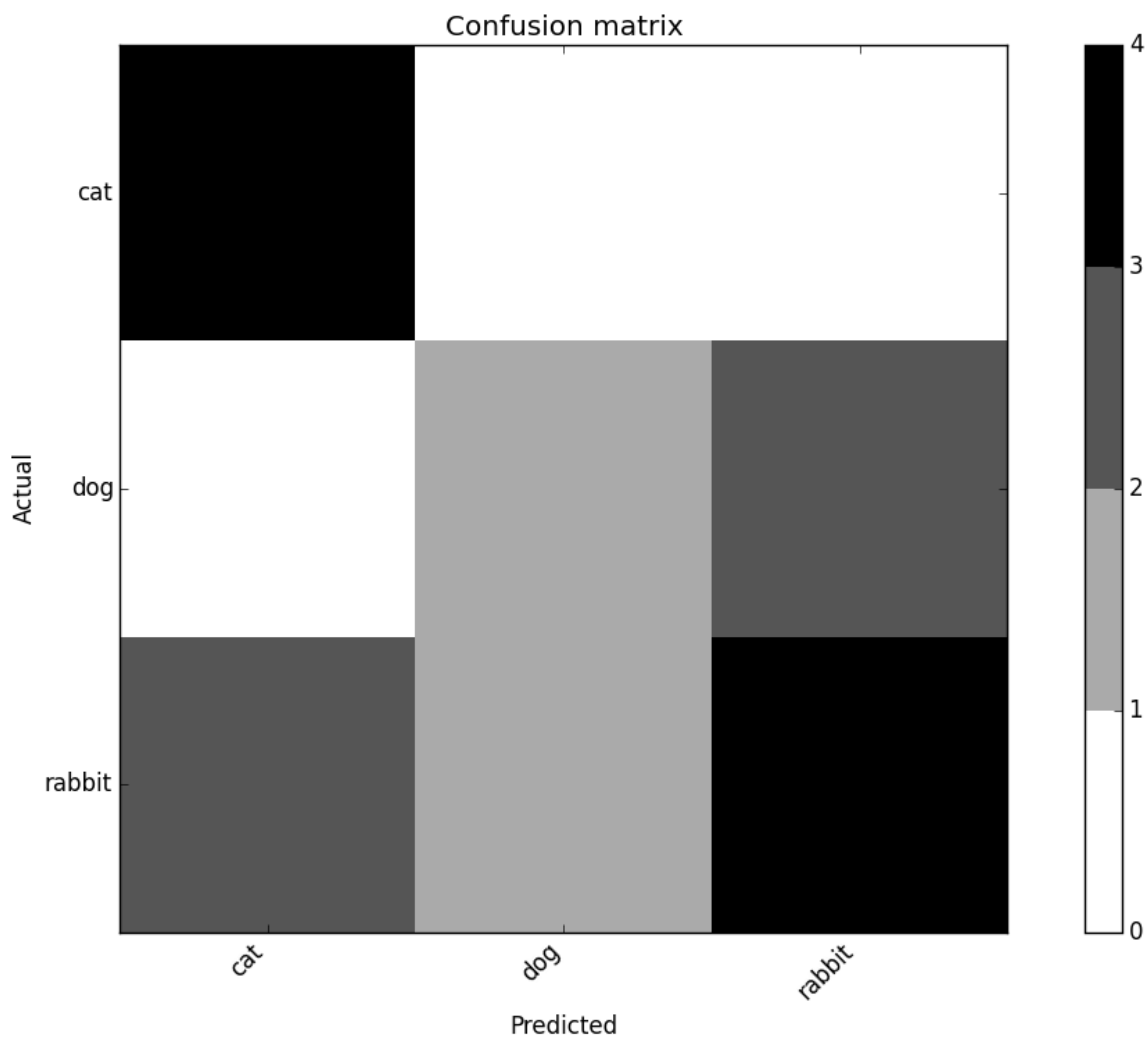

Fig. 1: confusion_matrix

7.2 Matplotlib plot of a normalized confusion matrix

```
confusion_matrix.plot(normalized=True)
plt.show()
```

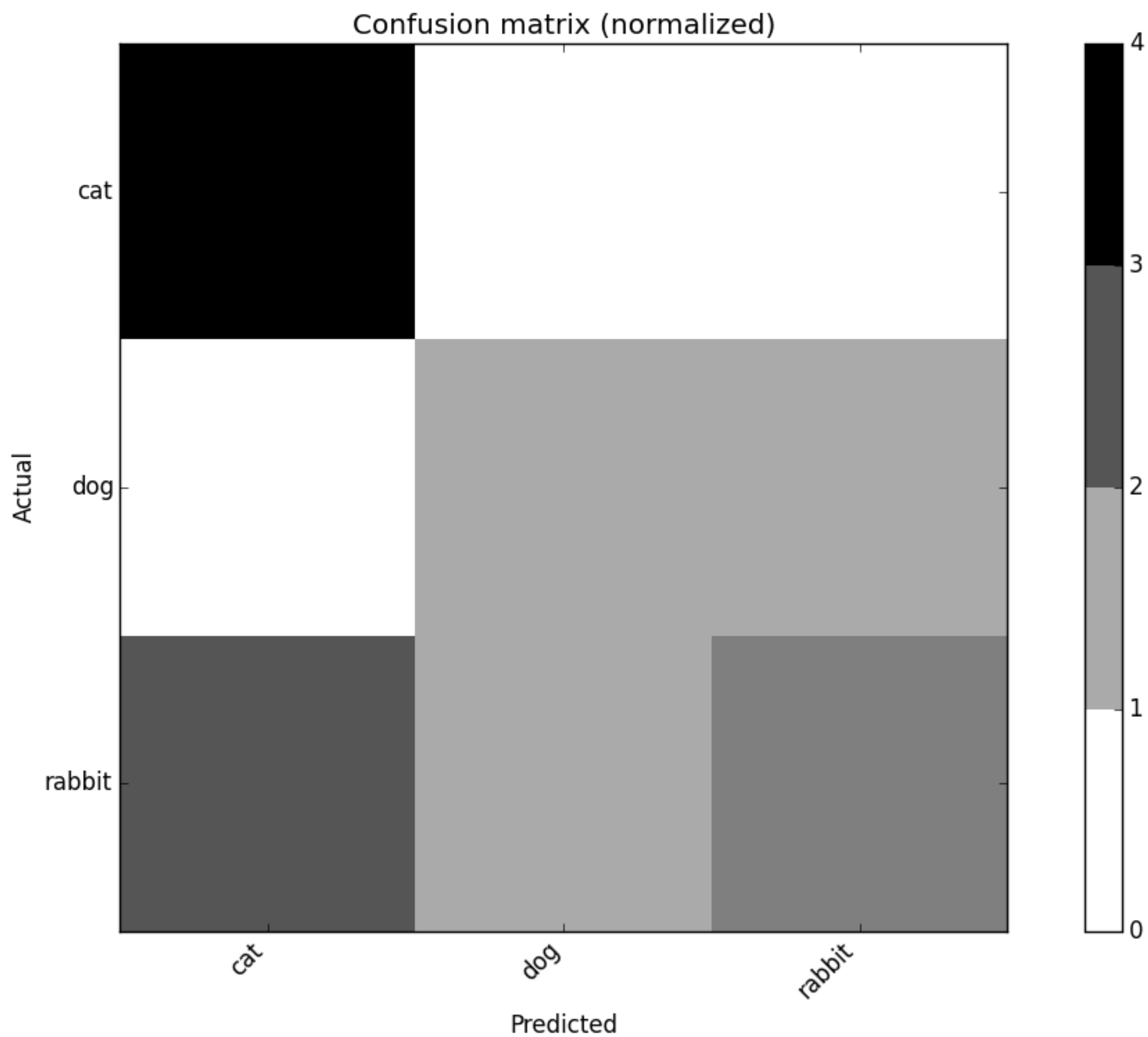

Fig. 2: confusion_matrix_norm

7.3 Binary confusion matrix

If actual values (y_true) and predicted values (y_pred) are bool, ConfusionMatrix outputs binary confusion matrix.

```
y_true = [ True,  True, False, False, False,  True, False,  True,  True,
           False,  True, False, False, False, False, False,  True, False,
            True,  True,  True,  True, False, False, False,  True, False,
            True, False, False, False, False,  True,  True, False, False,
           False,  True,  True,  True,  True, False, False, False, False,
            True, False, False, False, False, False, False, False, False,
           False,  True,  True, False,  True, False,  True,  True,  True,
           False, False,  True, False,  True, False, False,  True, False,
           False, False, False, False, False, False, False,  True, False,
            True,  True,  True,  True, False, False,  True, False,  True,
            True, False,  True, False,  True, False, False,  True,  True,
           False, False,  True,  True, False, False, False, False, False,
           False,  True,  True, False]

y_pred = [False, False, False, False, False,  True, False, False,  True,
       False,  True, False, False, False, False, False, False, False,
        True,  True,  True,  True, False, False, False, False, False,
       False, False, False, False, False,  True, False, False, False,
       False,  True, False, False, False, False, False, False, False,
        True, False, False, False, False, False, False, False, False,
       False,  True, False, False, False, False, False, False, False,
       False, False,  True, False, False, False, False,  True, False,
       False, False, False, False, False, False, False,  True, False,
       False,  True, False, False, False, False,  True, False,  True,
        True, False, False, False,  True, False, False,  True,  True,
       False, False,  True,  True, False, False, False, False, False,
       False,  True, False, False]

binary_confusion_matrix = ConfusionMatrix(y_true, y_pred)
print("Binary confusion matrix:\n%s" % binary_confusion_matrix)
```

It display as a nicely labeled Pandas DataFrame

```
Binary confusion matrix:
Predicted  False  True  __all__
Actual
False         67     0       67
True          21    24       45
__all__       88    24      112
```

You can get useful attributes such as True Positive (TP), True Negative (TN) . . .

```
print(binary_confusion_matrix.TP)
```

7.4 Matplotlib plot of a binary confusion matrix

```
binary_confusion_matrix.plot()
plt.show()
```

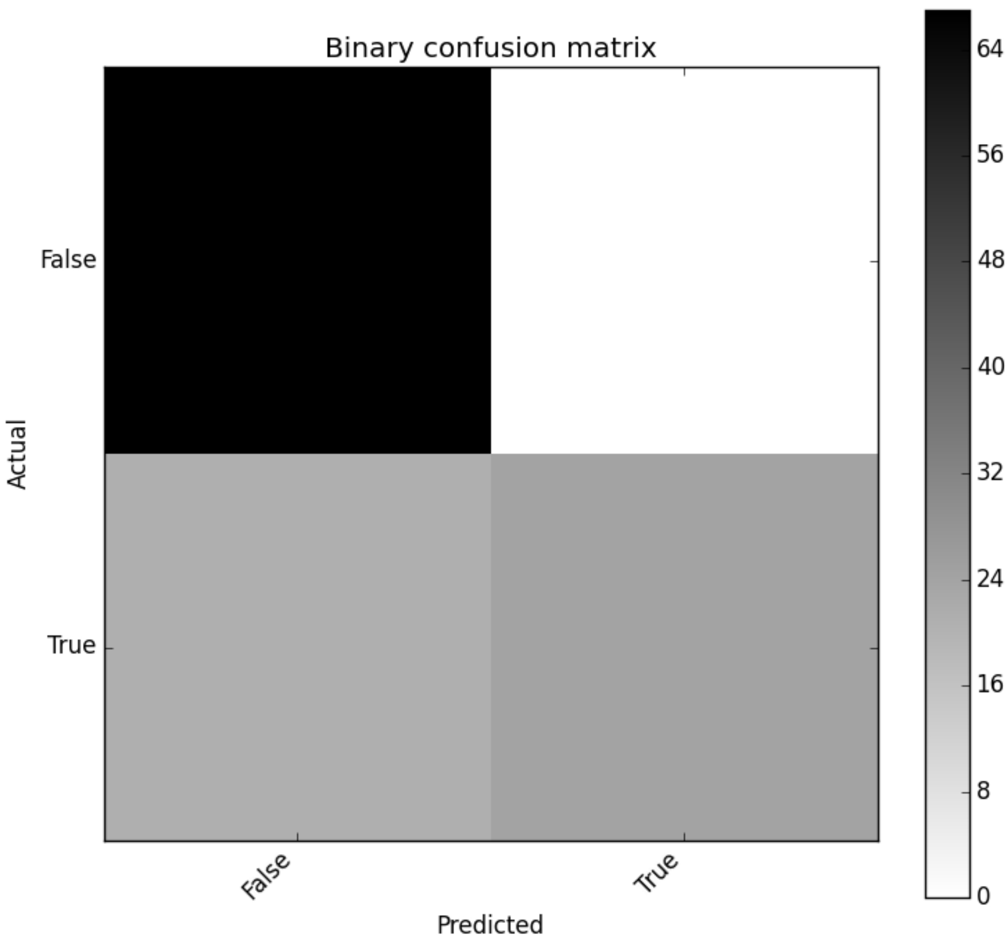

Fig. 3: binary_confusion_matrix

7.5 Matplotlib plot of a normalized binary confusion matrix

```
binary_confusion_matrix.plot(normalized=True)
plt.show()
```

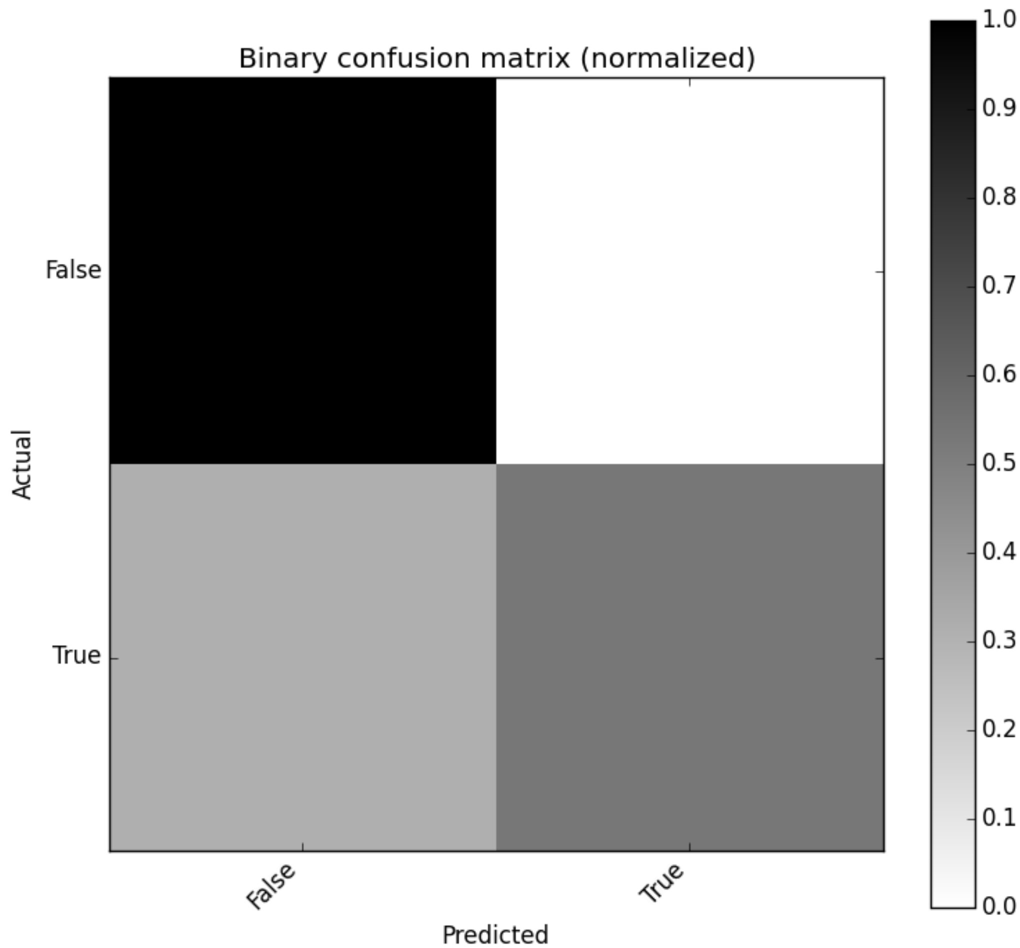

Fig. 4: binary_confusion_matrix_norm

7.6 Seaborn plot of a binary confusion matrix (ToDo)

```
binary_confusion_matrix.plot(backend='seaborn')
```

7.7 Confusion matrix and class statistics

Overall statistics and class statistics of confusion matrix can be easily displayed.

```
y_true = [600, 200, 200, 200, 200, 200, 200, 200, 500, 500, 500, 200, 200, 200, 200,␣
↪200, 200, 200, 200, 200]
y_pred = [100, 200, 200, 100, 100, 200, 200, 200, 100, 200, 500, 100, 100, 100, 100,␣
↪100, 100, 100, 500, 200]
cm = ConfusionMatrix(y_true, y_pred)
cm.print_stats()
```

You should get:

```
Confusion Matrix:

Classes  100  200  500  600  __all__
Actual
100        0    0    0    0        0
200        9    6    1    0       16
500        1    1    1    0        3
600        1    0    0    0        1
__all__   11    7    2    0       20


Overall Statistics:

Accuracy: 0.35
95% CI: (0.1539092047845412, 0.59218853453282805)
No Information Rate: ToDo
P-Value [Acc > NIR]: 0.978585644357
Kappa: 0.0780141843972
Mcnemar's Test P-Value: ToDo


Class Statistics:

Classes                                 100         200         500   600
Population                               20          20          20    20
Condition positive                        0          16           3     1
Condition negative                       20           4          17    19
Test outcome positive                    11           7           2     0
Test outcome negative                     9          13          18    20
TP: True Positive                         0           6           1     0
TN: True Negative                         9           3          16    19
FP: False Positive                       11           1           1     0
FN: False Negative                        0          10           2     1
TPR: Sensivity                          NaN       0.375   0.3333333     0
TNR=SPC: Specificity                   0.45        0.75   0.9411765     1
PPV: Pos Pred Value = Precision           0   0.8571429         0.5   NaN
NPV: Neg Pred Value                       1   0.2307692   0.8888889  0.95
FPR: False-out                         0.55        0.25  0.05882353     0
FDR: False Discovery Rate                 1   0.1428571         0.5   NaN
FNR: Miss Rate                          NaN       0.625   0.6666667     1
```

(continues on next page)

(continued from previous page)

```
ACC: Accuracy                              0.45        0.45        0.85  0.95
F1 score                                      0   0.5217391         0.4     0
MCC: Matthews correlation coefficient       NaN   0.1048285    0.326732   NaN
Informedness                                NaN       0.125   0.2745098     0
Markedness                                    0  0.08791209   0.3888889   NaN
Prevalence                                    0         0.8        0.15  0.05
LR+: Positive likelihood ratio              NaN         1.5    5.666667   NaN
LR-: Negative likelihood ratio              NaN   0.8333333   0.7083333     1
DOR: Diagnostic odds ratio                  NaN         1.8           8   NaN
FOR: False omission rate                      0   0.7692308   0.1111111  0.05
```

Statistics are also available as an OrderedDict using:

```
cm.stats()
```

API:

CHAPTER 8

pandas_ml.core package

8.1 Submodules

class pandas_ml.core.frame.**ModelFrame**(*data*, *target=None*, **args*, ***kwargs*)

Bases: *pandas_ml.core.generic.ModelPredictor*, pandas.core.frame.DataFrame

Data structure subclassing pandas.DataFrame to define a metadata to specify target (response variable) and data (explanatory variable / features).

Parameters

data [same as pandas.DataFrame]

target [str or array-like] Column name or values to be used as target

args [arguments passed to pandas.DataFrame]

kwargs [keyword arguments passed to pandas.DataFrame]

calibration
Property to access sklearn.calibration

cls
alias of *pandas_ml.skaccessors.gaussian_process.GaussianProcessMethods*

cluster
Property to access sklearn.cluster. See *pandas_ml.skaccessors.cluster*

covariance
Property to access sklearn.covariance. See *pandas_ml.skaccessors.covariance*

cross_decomposition
Property to access sklearn.cross_decomposition

da
Property to access sklearn.discriminant_analysis

data
Return data (explanatory variable / features)

Returns

data [`ModelFrame`]

`decision_function`(*estimator*, **args*, ***kwargs*)
Call estimator's decision_function method.

Parameters

args [arguments passed to decision_function method]

kwargs [keyword arguments passed to decision_function method]

Returns

returned [decisions]

`decomposition`
Property to access `sklearn.decomposition`

`discriminant_analysis`
Property to access `sklearn.discriminant_analysis`

`dummy`
Property to access `sklearn.dummy`

`ensemble`
Property to access `sklearn.ensemble`. See *`pandas_ml.skaccessors.ensemble`*

`feature_extraction`
Property to access `sklearn.feature_extraction`. See *`pandas_ml.skaccessors.feature_extraction`*

`feature_selection`
Property to access `sklearn.feature_selection`. See *`pandas_ml.skaccessors.feature_selection`*

`fit_predict`(*estimator*, **args*, ***kwargs*)
Call estimator's fit_predict method.

Parameters

args [arguments passed to fit_predict method]

kwargs [keyword arguments passed to fit_predict method]

Returns

returned [predicted result]

`fit_resample`(*estimator*, **args*, ***kwargs*)
Call estimator's fit_resample method.

Parameters

args [arguments passed to fit_resample method]

kwargs [keyword arguments passed to fit_resample method]

Returns

returned [resampling result]

`fit_sample`(*estimator*, **args*, ***kwargs*)
Call estimator's fit_sample method.

Parameters

args [arguments passed to fit_sample method]

kwargs [keyword arguments passed to fit_sample method]

Returns

returned [sampling result]

fit_transform(*estimator*, **args*, ***kwargs*)
Call estimator's fit_transform method.

Parameters

args [arguments passed to fit_transform method]

kwargs [keyword arguments passed to fit_transform method]

Returns

returned [transformed result]

gaussian_process
Property to access `sklearn.gaussian_process`. See *pandas_ml.skaccessors.gaussian_process*

gp
Property to access `sklearn.gaussian_process`. See *pandas_ml.skaccessors.gaussian_process*

groupby(*by=None*, *axis=0*, *level=None*, *as_index=True*, *sort=True*, *group_keys=True*, *squeeze=False*)
Group DataFrame or Series using a mapper or by a Series of columns.

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

Parameters

by [mapping, function, label, or list of labels] Used to determine the groups for the groupby. If `by` is a function, it's called on each value of the object's index. If a dict or Series is passed, the Series or dict VALUES will be used to determine the groups (the Series' values are first aligned; see `.align()` method). If an ndarray is passed, the values are used as-is determine the groups. A label or list of labels may be passed to group by the columns in `self`. Notice that a tuple is interpreted a (single) key.

axis [{0 or 'index', 1 or 'columns'}, default 0] Split along rows (0) or columns (1).

level [int, level name, or sequence of such, default None] If the axis is a MultiIndex (hierarchical), group by a particular level or levels.

as_index [bool, default True] For aggregated output, return object with group labels as the index. Only relevant for DataFrame input. as_index=False is effectively "SQL-style" grouped output.

sort [bool, default True] Sort group keys. Get better performance by turning this off. Note this does not influence the order of observations within each group. Groupby preserves the order of rows within each group.

group_keys [bool, default True] When calling apply, add group keys to index to identify pieces.

squeeze [bool, default False] Reduce the dimensionality of the return type if possible, otherwise return a consistent type.

observed [bool, default False] This only applies if any of the groupers are Categoricals. If True: only show observed values for categorical groupers. If False: show all values for categorical groupers.

New in version 0.23.0.

****kwargs** Optional, only accepts keyword argument 'mutated' and is passed to groupby.

Returns

DataFrameGroupBy or SeriesGroupBy Depends on the calling object and returns groupby object that contains information about the groups.

See also:

resample Convenience method for frequency conversion and resampling of time series.

Notes

See the user guide for more.

Examples

```
>>> df = pd.DataFrame({'Animal' : ['Falcon', 'Falcon',
...                               'Parrot', 'Parrot'],
...                    'Max Speed' : [380., 370., 24., 26.]})
>>> df
   Animal  Max Speed
0  Falcon      380.0
1  Falcon      370.0
2  Parrot       24.0
3  Parrot       26.0
>>> df.groupby(['Animal']).mean()
        Max Speed
Animal
Falcon      375.0
Parrot       25.0
```

Hierarchical Indexes

We can groupby different levels of a hierarchical index using the *level* parameter:

```
>>> arrays = [['Falcon', 'Falcon', 'Parrot', 'Parrot'],
...           ['Capitve', 'Wild', 'Capitve', 'Wild']]
>>> index = pd.MultiIndex.from_arrays(arrays, names=('Animal', 'Type'))
>>> df = pd.DataFrame({'Max Speed' : [390., 350., 30., 20.]},
...                   index=index)
>>> df
                Max Speed
Animal Type
Falcon Capitve      390.0
       Wild         350.0
Parrot Capitve       30.0
       Wild          20.0
>>> df.groupby(level=0).mean()
        Max Speed
Animal
```

(continues on next page)

(continued from previous page)

```
Falcon      370.0
Parrot       25.0
>>> df.groupby(level=1).mean()
         Max Speed
Type
Capitve      210.0
Wild         185.0
```

has_data()
Return whether ModelFrame has data

Returns

has_data [bool]

has_multi_targets()
Return whether ModelFrame has multiple target columns

Returns

has_multi_targets [bool]

has_target()
Return whether ModelFrame has target

Returns

has_target [bool]

imbalance
Property to access imblearn

inverse_transform(*estimator*, **args*, ***kwargs*)
Call estimator's inverse_transform method.

Parameters

args [arguments passed to inverse_transform method]

kwargs [keyword arguments passed to inverse_transform method]

Returns

returned [transformed result]

isotonic
Property to access sklearn.isotonic. See *pandas_ml.skaccessors.isotonic*

kernel_approximation
Property to access sklearn.kernel_approximation

kernel_ridge
Property to access sklearn.kernel_ridge

lda
Property to access sklearn.lda

linear_model
Property to access sklearn.linear_model. See *pandas_ml.skaccessors.linear_model*

lm
Property to access sklearn.linear_model. See *pandas_ml.skaccessors.linear_model*

manifold
Property to access `sklearn.manifold`. See *pandas_ml.skaccessors.manifold*

metrics
Property to access `sklearn.metrics`. See *pandas_ml.skaccessors.metrics*

mixture
Property to access `sklearn.mixture`

model_selection
Property to access `sklearn.model_selection`. See *pandas_ml.skaccessors.model_selection*

ms
Property to access `sklearn.model_selection`. See *pandas_ml.skaccessors.model_selection*

multiclass
Property to access `sklearn.multiclass`. See `pandas_ml.skaccessors.multiclass`

multioutput
Property to access `sklearn.multioutput`. See `pandas_ml.skaccessors.multioutput`

naive_bayes
Property to access `sklearn.naive_bayes`

neighbors
Property to access `sklearn.neighbors`. See *pandas_ml.skaccessors.neighbors*

neural_network
Property to access `sklearn.neural_network`

pipeline
Property to access `sklearn.pipeline`. See *pandas_ml.skaccessors.pipeline*

pp
Property to access `sklearn.preprocessing`. See *pandas_ml.skaccessors.preprocessing*

predict_log_proba (*estimator*, **args*, ***kwargs*)
Call estimator's predict_log_proba method.

Parameters

args [arguments passed to predict_log_proba method]

kwargs [keyword arguments passed to predict_log_proba method]

Returns

returned [probabilities]

predict_proba (*estimator*, **args*, ***kwargs*)
Call estimator's predict_proba method.

Parameters

args [arguments passed to predict_proba method]

kwargs [keyword arguments passed to predict_proba method]

Returns

returned [probabilities]

preprocessing
Property to access `sklearn.preprocessing`. See *pandas_ml.skaccessors.preprocessing*

qda
Property to access `sklearn.qda`

random_projection
Property to access `sklearn.random_projection`. See `pandas_ml.skaccessors.random_projection`

sample (*estimator*, **args*, ***kwargs*)
Call estimator's sample method.

Parameters

args [arguments passed to sample method]

kwargs [keyword arguments passed to sample method]

Returns

returned [sampling result]

score (*estimator*, **args*, ***kwargs*)
Call estimator's score method.

Parameters

args [arguments passed to score method]

kwargs [keyword arguments passed to score method]

Returns

returned [score]

seaborn
Property to access `seaborn` API

semi_supervised
Property to access `sklearn.semi_supervised`. See `pandas_ml.skaccessors.semi_supervised`

sns
Property to access `seaborn` API

svm
Property to access `sklearn.svm`. See *pandas_ml.skaccessors.svm*

target
Return target (response variable)

Returns

target [`ModelSeries`]

target_name
Return target column name

Returns

target [object]

transform (*estimator*, **args*, ***kwargs*)
Call estimator's transform method.

Parameters

args [arguments passed to transform method]

kwargs [keyword arguments passed to transform method]

Returns

returned [transformed result]

tree
Property to access `sklearn.tree`

xgb
Property to access `xgboost.sklearn` API

xgboost
Property to access `xgboost.sklearn` API

class `pandas_ml.core.generic.`**ModelPredictor**
Bases: *pandas_ml.core.generic.ModelTransformer*

Base class for `ModelFrame` and `ModelFrameGroupBy`

decision
Return current estimator's decision function

Returns

decisions [`ModelFrame`]

estimator
Return most recently used estimator

Returns

estimator [estimator]

log_proba
Return current estimator's log probabilities

Returns

probabilities [`ModelFrame`]

predict (*estimator*, **args*, ***kwargs*)
Call estimator's predict method.

Parameters

args [arguments passed to predict method]

kwargs [keyword arguments passed to predict method]

Returns

returned [predicted result]

predicted
Return current estimator's predicted results

Returns

predicted [`ModelSeries`]

proba
Return current estimator's probabilities

Returns

probabilities [ModelFrame]

class pandas_ml.core.generic.**ModelTransformer**

Bases: object

Base class for ModelFrame and ModelFrame

fit (*estimator*, **args*, ***kwargs*)

Call estimator's fit method.

Parameters

args [arguments passed to fit method]

kwargs [keyword arguments passed to fit method]

Returns

returned [None or fitted estimator]

fit_transform (*estimator*, **args*, ***kwargs*)

Call estimator's fit_transform method.

Parameters

args [arguments passed to fit_transform method]

kwargs [keyword arguments passed to fit_transform method]

Returns

returned [transformed result]

inverse_transform (*estimator*, **args*, ***kwargs*)

Call estimator's inverse_transform method.

Parameters

args [arguments passed to inverse_transform method]

kwargs [keyword arguments passed to inverse_transform method]

Returns

returned [transformed result]

transform (*estimator*, **args*, ***kwargs*)

Call estimator's transform method.

Parameters

args [arguments passed to transform method]

kwargs [keyword arguments passed to transform method]

Returns

returned [transformed result]

class pandas_ml.core.groupby.**GroupedEstimator** (*estimator*, *grouped*)

Bases: pandas_ml.core.base._BaseEstimator

Create grouped estimators based on passed estimator

class pandas_ml.core.groupby.**ModelFrameGroupBy**(*obj, keys=None, axis=0, level=None, grouper=None, exclusions=None, selection=None, as_index=True, sort=True, group_keys=True, squeeze=False, observed=False, **kwargs*)

Bases: pandas.core.groupby.generic.DataFrameGroupBy, *pandas_ml.core.generic.ModelPredictor*

transform(*func, *args, **kwargs*)

Call estimator's transform method.

Parameters

args [arguments passed to transform method]

kwargs [keyword arguments passed to transform method]

Returns

returned [transformed result]

class pandas_ml.core.groupby.**ModelSeriesGroupBy**(*obj, keys=None, axis=0, level=None, grouper=None, exclusions=None, selection=None, as_index=True, sort=True, group_keys=True, squeeze=False, observed=False, **kwargs*)

Bases: pandas.core.groupby.generic.SeriesGroupBy

pandas_ml.core.groupby.**groupby**(*obj, by, **kwds*)

Class for grouping and aggregating relational data.

See aggregate, transform, and apply functions on this object.

It's easiest to use obj.groupby(...) to use GroupBy, but you can also do:

```
grouped = groupby(obj, ...)
```

Parameters

obj [pandas object]

axis [int, default 0]

level [int, default None] Level of MultiIndex

groupings [list of Grouping objects] Most users should ignore this

exclusions [array-like, optional] List of columns to exclude

name [string] Most users should ignore this

Returns

****Attributes****

groups [dict] {group name -> group labels}

len(grouped) [int] Number of groups

Notes

After grouping, see aggregate, apply, and transform functions. Here are some other brief notes about usage. When grouping by multiple groups, the result index will be a MultiIndex (hierarchical) by default.

Iteration produces (key, group) tuples, i.e. chunking the data by group. So you can write code like:

```
grouped = obj.groupby(keys, axis=axis)
for key, group in grouped:
    # do something with the data
```

Function calls on GroupBy, if not specially implemented, "dispatch" to the grouped data. So if you group a DataFrame and wish to invoke the std() method on each group, you can simply do:

```
df.groupby(mapper).std()
```

rather than

```
df.groupby(mapper).aggregate(np.std)
```

You can pass arguments to these "wrapped" functions, too.

See the online documentation for full exposition on these topics and much more

class `pandas_ml.core.series.`**`ModelSeries`**(*data=None*, *index=None*, *dtype=None*, *name=None*, *copy=False*, *fastpath=False*)

Bases: `pandas_ml.core.generic.ModelTransformer`, `pandas.core.series.Series`

Wrapper for `pandas.Series` to support `sklearn.preprocessing`

`groupby`(*by=None*, *axis=0*, *level=None*, *as_index=True*, *sort=True*, *group_keys=True*, *squeeze=False*)

Group DataFrame or Series using a mapper or by a Series of columns.

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

Parameters

by [mapping, function, label, or list of labels] Used to determine the groups for the groupby. If `by` is a function, it's called on each value of the object's index. If a dict or Series is passed, the Series or dict VALUES will be used to determine the groups (the Series' values are first aligned; see `.align()` method). If an ndarray is passed, the values are used as-is determine the groups. A label or list of labels may be passed to group by the columns in `self`. Notice that a tuple is interpreted a (single) key.

axis [{0 or 'index', 1 or 'columns'}, default 0] Split along rows (0) or columns (1).

level [int, level name, or sequence of such, default None] If the axis is a MultiIndex (hierarchical), group by a particular level or levels.

as_index [bool, default True] For aggregated output, return object with group labels as the index. Only relevant for DataFrame input. as_index=False is effectively "SQL-style" grouped output.

sort [bool, default True] Sort group keys. Get better performance by turning this off. Note this does not influence the order of observations within each group. Groupby preserves the order of rows within each group.

group_keys [bool, default True] When calling apply, add group keys to index to identify pieces.

squeeze [bool, default False] Reduce the dimensionality of the return type if possible, otherwise return a consistent type.

observed [bool, default False] This only applies if any of the groupers are Categoricals. If True: only show observed values for categorical groupers. If False: show all values for categorical groupers.

New in version 0.23.0.

****kwargs** Optional, only accepts keyword argument 'mutated' and is passed to groupby.

Returns

DataFrameGroupBy or SeriesGroupBy Depends on the calling object and returns groupby object that contains information about the groups.

See also:

`resample` Convenience method for frequency conversion and resampling of time series.

Notes

See the user guide for more.

Examples

```
>>> df = pd.DataFrame({'Animal' : ['Falcon', 'Falcon',
...                                'Parrot', 'Parrot'],
...                    'Max Speed' : [380., 370., 24., 26.]})
>>> df
   Animal  Max Speed
0  Falcon      380.0
1  Falcon      370.0
2  Parrot       24.0
3  Parrot       26.0
>>> df.groupby(['Animal']).mean()
        Max Speed
Animal
Falcon      375.0
Parrot       25.0
```

Hierarchical Indexes

We can groupby different levels of a hierarchical index using the *level* parameter:

```
>>> arrays = [['Falcon', 'Falcon', 'Parrot', 'Parrot'],
...           ['Capitve', 'Wild', 'Capitve', 'Wild']]
>>> index = pd.MultiIndex.from_arrays(arrays, names=('Animal', 'Type'))
>>> df = pd.DataFrame({'Max Speed' : [390., 350., 30., 20.]},
...                    index=index)
>>> df
                Max Speed
Animal Type
Falcon Capitve      390.0
       Wild         350.0
Parrot Capitve       30.0
       Wild          20.0
```

(continues on next page)

(continued from previous page)

```
>>> df.groupby(level=0).mean()
        Max Speed
Animal
Falcon      370.0
Parrot       25.0
>>> df.groupby(level=1).mean()
         Max Speed
Type
Capitve      210.0
Wild         185.0
```

pp
Property to access sklearn.preprocessing. See *pandas_ml.skaccessors.preprocessing*

preprocessing
Property to access sklearn.preprocessing. See *pandas_ml.skaccessors.preprocessing*

to_frame(*name=None*)
Convert Series to DataFrame.

Parameters

name [object, default None] The passed name should substitute for the series name (if it has one).

Returns

data_frame [DataFrame]

transform(*estimator*, **args*, ***kwargs*)
Call estimator's transform method.

Parameters

args [arguments passed to transform method]

kwargs [keyword arguments passed to transform method]

Returns

returned [transformed result]

8.2 Module contents

CHAPTER 9

pandas_ml.skaccessors package

9.1 Subpackages

9.1.1 pandas_ml.skaccessors.test package

Submodules

class pandas_ml.skaccessors.test.test_multioutput.**TestMultiOutput**
Bases: pandas_ml.util.testing.TestCase

test_multioutput()

test_objectmapper()

Module contents

9.2 Submodules

class pandas_ml.skaccessors.base.**Bunch**
Bases: object

class pandas_ml.skaccessors.cluster.**ClusterMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: pandas_ml.core.accessor._AccessorMethods

Accessor to sklearn.cluster.

affinity_propagation(**args*, ***kwargs*)
Call sklearn.cluster.affinity_propagation using automatic mapping.

- S: ModelFrame.data

bicluster
Property to access sklearn.cluster.bicluster

dbscan (**args*, ***kwargs*)
Call `sklearn.cluster.dbscan` using automatic mapping.

- `X`: `ModelFrame.data`

k_means (*n_clusters*, **args*, ***kwargs*)
Call `sklearn.cluster.k_means` using automatic mapping.

- `X`: `ModelFrame.data`

mean_shift (**args*, ***kwargs*)
Call `sklearn.cluster.mean_shift` using automatic mapping.

- `X`: `ModelFrame.data`

spectral_clustering (**args*, ***kwargs*)
Call `sklearn.cluster.spectral_clustering` using automatic mapping.

- `affinity`: `ModelFrame.data`

class `pandas_ml.skaccessors.covariance.`**CovarianceMethods** (*df*, *module_name=None*, *attrs=None*)

Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.covariance`.

empirical_covariance (**args*, ***kwargs*)
Call `sklearn.covariance.empirical_covariance` using automatic mapping.

- `X`: `ModelFrame.data`

ledoit_wolf (**args*, ***kwargs*)
Call `sklearn.covariance.ledoit_wolf` using automatic mapping.

- `X`: `ModelFrame.data`

oas (**args*, ***kwargs*)
Call `sklearn.covariance.oas` using automatic mapping.

- `X`: `ModelFrame.data`

class `pandas_ml.skaccessors.cross_decomposition.`**CrossDecompositionMethods** (*df*, *module_name=None*, *attrs=None*)

Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.cross_decomposition`.

class `pandas_ml.skaccessors.decomposition.`**DecompositionMethods** (*df*, *module_name=None*, *attrs=None*)

Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.decomposition`.

dict_learning (*n_components*, *alpha*, **args*, ***kwargs*)
Call `sklearn.decomposition.dict_learning` using automatic mapping.

- `X`: `ModelFrame.data`

dict_learning_online (**args*, ***kwargs*)
Call `sklearn.decomposition.dict_learning_online` using automatic mapping.

- `X`: `ModelFrame.data`

fastica (**args*, ***kwargs*)
Call `sklearn.decomposition.fastica` using automatic mapping.

- X: `ModelFrame.data`

sparse_encode (*dictionary*, **args*, ***kwargs*)
Call `sklearn.decomposition.sparce_encode` using automatic mapping.

- X: `ModelFrame.data`

class `pandas_ml.skaccessors.ensemble.`**EnsembleMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.ensemble`.

partial_dependence
Property to access `sklearn.ensemble.partial_dependence`

class `pandas_ml.skaccessors.ensemble.`**PartialDependenceMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: `pandas_ml.core.accessor._AccessorMethods`

partial_dependence (*gbrt*, *target_variables*, ***kwargs*)
Call `sklearn.ensemble.partial_dependence` using automatic mapping.

- X: `ModelFrame.data`

plot_partial_dependence (*gbrt*, *features*, ***kwargs*)
Call `sklearn.ensemble.plot_partial_dependence` using automatic mapping.

- X: `ModelFrame.data`

class `pandas_ml.skaccessors.feature_extraction.`**FeatureExtractionMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.feature_extraction`.

image
Property to access `sklearn.feature_extraction.image`

text
Property to access `sklearn.feature_extraction.text`

class `pandas_ml.skaccessors.feature_selection.`**FeatureSelectionMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.feature_selection`.

class `pandas_ml.skaccessors.gaussian_process.`**GaussianProcessMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.gaussian_process`.

correlation_models
Property to access sklearn.gaussian_process.correlation_models

regression_models
Property to access sklearn.gaussian_process.regression_models

class pandas_ml.skaccessors.gaussian_process.**RegressionModelsMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: pandas_ml.core.accessor._AccessorMethods

class pandas_ml.skaccessors.isotonic.**IsotonicMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: pandas_ml.core.accessor._AccessorMethods

Accessor to sklearn.isotonic.

IsotonicRegression
sklearn.isotonic.IsotonicRegression

check_increasing(**args*, ***kwargs*)
Call sklearn.isotonic.check_increasing using automatic mapping.

- x: ModelFrame.index
- y: ModelFrame.target

isotonic_regression(**args*, ***kwargs*)
Call sklearn.isotonic.isotonic_regression using automatic mapping.

- y: ModelFrame.target

class pandas_ml.skaccessors.linear_model.**LinearModelMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: pandas_ml.core.accessor._AccessorMethods

Accessor to sklearn.linear_model.

enet_path(**args*, ***kwargs*)
Call sklearn.linear_model.enet_path using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

lars_path(**args*, ***kwargs*)
Call sklearn.linear_model.lars_path using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

lasso_path(**args*, ***kwargs*)
Call sklearn.linear_model.lasso_path using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

lasso_stability_path(**args*, ***kwargs*)
Call sklearn.linear_model.lasso_stability_path using automatic mapping.

- X: ModelFrame.data

- y: ModelFrame.target

orthogonal_mp_gram (**args*, ***kwargs*)
Call sklearn.linear_model.orthogonal_mp_gram using automatic mapping.

- Gram: ModelFrame.data.T.dot(ModelFrame.data)
- Xy: ModelFrame.data.T.dot(ModelFrame.target)

class pandas_ml.skaccessors.manifold.**ManifoldMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: pandas_ml.core.accessor._AccessorMethods

Accessor to sklearn.manifold.

locally_linear_embedding (*n_neighbors*, *n_components*, **args*, ***kwargs*)
Call sklearn.manifold.locally_linear_embedding using automatic mapping.

- X: ModelFrame.data

spectral_embedding (**args*, ***kwargs*)
Call sklearn.manifold.spectral_embedding using automatic mapping.

- adjacency: ModelFrame.data

class pandas_ml.skaccessors.metrics.**MetricsMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: pandas_ml.core.accessor._AccessorMethods

Accessor to sklearn.metrics.

auc (*kind='roc'*, *reorder=False*, ***kwargs*)
Calcurate AUC of ROC curve or precision recall curve

Parameters

kind [{'roc', 'precision_recall_curve'}]

Returns

float [AUC]

average_precision_score (**args*, ***kwargs*)
Call sklearn.metrics.average_precision_score using automatic mapping.

- y_true: ModelFrame.target
- y_score: ModelFrame.decision

confusion_matrix (**args*, ***kwargs*)
Call sklearn.metrics.confusion_matrix using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.predicted

consensus_score (**args*, ***kwargs*)
Not implemented

f1_score (**args*, ***kwargs*)
Call sklearn.metrics.f1_score using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.predicted

fbeta_score (*beta*, **args*, ***kwargs*)
Call sklearn.metrics.fbeta_score using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.predicted

hinge_loss (*args, **kwargs)
Call sklearn.metrics.hinge_loss using automatic mapping.

- y_true: ModelFrame.target
- y_pred_decision: ModelFrame.decision

log_loss (*args, **kwargs)
Call sklearn.metrics.log_loss using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.proba

pairwise
Not implemented

precision_recall_curve (*args, **kwargs)
Call sklearn.metrics.precision_recall_curve using automatic mapping.

- y_true: ModelFrame.target
- y_probas_pred: ModelFrame.decision

precision_recall_fscore_support (*args, **kwargs)
Call sklearn.metrics.precision_recall_fscore_support using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.predicted

precision_score (*args, **kwargs)
Call sklearn.metrics.precision_score using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.predicted

recall_score (*args, **kwargs)
Call sklearn.metrics.recall_score using automatic mapping.

- y_true: ModelFrame.target
- y_true: ModelFrame.predicted

roc_auc_score (*args, **kwargs)
Call sklearn.metrics.roc_auc_score using automatic mapping.

- y_true: ModelFrame.target
- y_score: ModelFrame.decision

roc_curve (*args, **kwargs)
Call sklearn.metrics.roc_curve using automatic mapping.

- y_true: ModelFrame.target
- y_score: ModelFrame.decision

silhouette_samples (*args, **kwargs)
Call sklearn.metrics.silhouette_samples using automatic mapping.

- X: ModelFrame.data

- labels: ModelFrame.predicted

silhouette_score (**args*, ***kwargs*)
Call sklearn.metrics.silhouette_score using automatic mapping.

- X: ModelFrame.data
- labels: ModelFrame.predicted

class pandas_ml.skaccessors.model_selection.**ModelSelectionMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: pandas_ml.core.accessor._AccessorMethods

Accessor to sklearn.model_selection.

StratifiedShuffleSplit (**args*, ***kwargs*)
Instanciate sklearn.cross_validation.StratifiedShuffleSplit using automatic mapping.

- y: ModelFrame.target

check_cv (*cv*, **args*, ***kwargs*)
Call sklearn.cross_validation.check_cv using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

cross_val_score (*estimator*, **args*, ***kwargs*)
Call sklearn.cross_validation.cross_val_score using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

describe (*estimator*)
Describe grid search results

Parameters

estimator [fitted grid search estimator]

Returns

described [ModelFrame]

iterate (*cv*, *reset_index=False*)
deprecated. Use .split

learning_curve (*estimator*, **args*, ***kwargs*)
Call sklearn.lerning_curve.learning_curve using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

permutation_test_score (*estimator*, **args*, ***kwargs*)
Call sklearn.cross_validation.permutation_test_score using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

split (*cv*, *reset_index=False*)
Generate ModelFrame using iterators for cross validation

Parameters

cv [cross validation iterator]

reset_index [bool] logical value whether to reset index, default False

Returns

generated [generator of `ModelFrame`]

train_test_split (*reset_index=False*, **args*, ***kwargs*)
Call `sklearn.cross_validation.train_test_split` using automatic mapping.

Parameters

reset_index [bool] logical value whether to reset index, default False

kwargs [keywords passed to `cross_validation.train_test_split`]

Returns

train, test [tuple of `ModelFrame`]

validation_curve (*estimator*, *param_name*, *param_range*, **args*, ***kwargs*)
Call `sklearn.learning_curve.validation_curve` using automatic mapping.

- `X`: `ModelFrame.data`
- `y`: `ModelFrame.target`

class `pandas_ml.skaccessors.neighbors.`**NeighborsMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.neighbors`.

class `pandas_ml.skaccessors.pipeline.`**PipelineMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.pipeline`.

make_pipeline
`sklearn.pipeline.make_pipeline`

make_union
`sklearn.pipeline.make_union`

class `pandas_ml.skaccessors.preprocessing.`**PreprocessingMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.preprocessing`.

add_dummy_feature (*value=1.0*)
Call `sklearn.preprocessing.add_dummy_feature` using automatic mapping.

- `X`: `ModelFrame.data`

class `pandas_ml.skaccessors.svm.`**SVMMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `sklearn.svm`.

l1_min_c (**args*, ***kwargs*)
Call `sklearn.svm.l1_min_c` using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

liblinear
Not implemented

libsvm
Not implemented

libsvm_sparse
Not implemented

9.3 Module contents

CHAPTER 10

pandas_ml.xgboost package

10.1 Subpackages

10.1.1 pandas_ml.xgboost.test package

Submodules

Module contents

10.2 Submodules

class `pandas_ml.xgboost.base.`**`XGBoostMethods`**(*df*, *module_name=None*, *attrs=None*)

Bases: `pandas_ml.core.accessor._AccessorMethods`

Accessor to `xgboost`.

`XGBClassifier`

`XGBRegressor`

`plot_importance`(*ax=None*, *height=0.2*, *xlim=None*, *title='Feature importance'*, *xlabel='F score'*, *ylabel='Features'*, *grid=True*, ***kwargs*)

Plot importance based on fitted trees.

Parameters

ax [matplotlib Axes, default None] Target axes instance. If None, new figure and axes will be created.

height [float, default 0.2] Bar height, passed to ax.barh()

xlim [tuple, default None] Tuple passed to axes.xlim()

title [str, default "Feature importance"] Axes title. To disable, pass None.

xlabel [str, default "F score"] X axis title label. To disable, pass None.

ylabel [str, default "Features"] Y axis title label. To disable, pass None.

kwargs : Other keywords passed to ax.barh()

Returns

ax [matplotlib Axes]

plot_tree (*num_trees=0*, *rankdir='UT'*, *ax=None*, ***kwargs*)
Plot specified tree.

Parameters

booster [Booster, XGBModel] Booster or XGBModel instance

num_trees [int, default 0] Specify the ordinal number of target tree

rankdir [str, default "UT"] Passed to graphiz via graph_attr

ax [matplotlib Axes, default None] Target axes instance. If None, new figure and axes will be created.

kwargs : Other keywords passed to to_graphviz

Returns

ax [matplotlib Axes]

to_graphviz (*num_trees=0*, *rankdir='UT'*, *yes_color='#0000FF'*, *no_color='#FF0000'*, ***kwargs*)
Convert specified tree to graphviz instance. IPython can automatically plot the returned graphiz instance. Otherwise, you shoud call .render() method of the returned graphiz instance.

Parameters

num_trees [int, default 0] Specify the ordinal number of target tree

rankdir [str, default "UT"] Passed to graphiz via graph_attr

yes_color [str, default '#0000FF'] Edge color when meets the node condigion.

no_color [str, default '#FF0000'] Edge color when doesn't meet the node condigion.

kwargs : Other keywords passed to graphviz graph_attr

Returns

ax [matplotlib Axes]

10.3 Module contents

Python Module Index

p

Index

A

B

C

D

Q

R