## Contents

1 Introduction ................................................. 1
   1.1 Notes .................................................. 1

2 Hypotheses .................................................. 3
   2.1 What is a hypothesis? ................................. 3
   2.2 Simplification .......................................... 4
   2.3 Explicitation ........................................... 7
   2.4 Normalization .......................................... 8
   2.5 Interference ........................................... 8
   2.6 Miscellaneous ......................................... 10

3 Utility code ................................................. 13
   3.1 `analyze.py` - Runner module ....................... 13
   3.2 Main translationese module ......................... 13
   3.3 Word ranks ............................................. 14
   3.4 Utilities ............................................... 14
   3.5 Memoize ................................................. 15

4 Testing ..................................................... 17
   4.1 Test utility classes .................................... 17

5 Results ..................................................... 19

6 Project description ....................................... 21

7 Getting started ........................................... 23

8 Tools used ................................................. 25

9 Indices and tables ........................................ 27

Python Module Index ......................................... 29
Introduction

The project addresses the difference between translated and original non-translated texts. Those differences, also referred as features or attributes, have been discussed and studied in the last three decades. In this project, we wish to quantify these differences.

For a human, it is not trivial to differentiate between the two groups. In order to successfully distinguish between them, we use machine learning and classification methods. The classification is done based on these features, estimated during texts analysis.

For many years, it has been suggested that there are differences between originally English texts and translated ones. For example, it has been suggested that translated texts have a narrow vocabulary use. Therefore, the number of different words used is smaller among translated text compared to original texts. (See translationese.lexical_variety).

Many tried to find the correct and appropriate attributes that will maximize the classification results. In this work, we try each of these attributes, attempting to analyze which ones provide the best separation.

This project is based on On The Features Of Translationese article, written by V. Volansky, N. Ordan and S. Wintner. Our work implements the hypotheses (sets of attributes) presented in the article and classifies texts according to them. The analysis was conducted on EUROPARL corpus texts.

The hypotheses analyzed are detailed in the Hypotheses page.

1.1 Notes

The following questions remain open when analyzing the original article:

1. Why are some features (e.g. translationese.lexical_variety) magnified by a constant? This should have no effect on any reasonable machine-learning analysis mechanism.

2. Some features are normalized by actual chunk length (in tokens), whereas some features are normalized by corpus-average chunk length (2000 tokens, in this instance and originally). It is not clearly specified when which is done, nor is a reason given.
2.1 What is a hypothesis?

Each hypothesis describes a set of attributes or features of a given text. Specifically, features are chosen in an attempt to distinguish between translated and non-translated texts. Each attribute refers to different characteristics of a text and can be quantified.

For example, it is hypothesized that translated texts contains more repetitions, therefore the amount of different words within the text is smaller when compared to non-translated texts. This is based on the assumption that the author of a translated text would probably have a more narrow vocabulary than a native writer.

The translationese.lexical_variety module quantifies this hypothesis by counting the amount of different words in a given text. The value is expected to be lower for translated texts.

2.1.1 Aspects of Implementation

Each hypothesis is implemented and tested and separately, in its own Python module. Such a module can be written quite easily, extending the functionality of this software. Attributes go in translationese/hypothesis_name.py, and accompanying tests usually go in tests/test_hypothesis_name.py. A simple hypothesis looks like this:

```python
import translationese

def quantify(analysis):
    # Hint for Eclipse PyDev autocompletion
    assert isinstance(analysis, translationese.Analysis)

    # ... compute some attributes of the text

    results = {
        "attribute1": 0.5,
        "attribute2": 0.75,
        # ...
        "attributeN": 0.2
    }

    return results
```

The quantify method receives a translationese.Analysis object. This facilitates pickle-caching performed by translationese.Analysis.
Some hypotheses have multiple possible variants. A module for a hypothesis with variants should look like this:

```python
import translationese

VARIANTS = [0, 1, 2, 3] #: Possible variants

def quantify_variant(analysis, variant):
    #: Hint for Eclipse PyDev autocompletion
    assert isinstance(analysis, translationese.Analysis)

    # ... compute some attributes of the text
    results = {
        "attribute1": 0.5,
        "attribute2": 0.75,
        # ...
        "attributeN": 0.2
    }

    return results
```

A variant is a 0-based integer which can affect the quantification. The VARIANTS variable is used by the analyze module when given the ALL parameter - these are the variants which will be used in the complete analysis, and may be a subset of variants which quantify_variant will really accept.

---

If expensive text-analysis functions (e.g. nltk) are used in the quantification, it is recommended that the be moved to the translationese.Analysis class, using memoize, as this will enable caching.

## 2.2 Simplification

The process of rendering complex linguistic features in the source text into simpler features in the target text.

### 2.2.1 Lexical variety

Lexical variety attempts to capture the hypothesis that original texts are richer in terms of vocabulary. This is done by comparing the number of tokens (words, punctuation, etc.) and types (of different tokens).

```python
class translationese.lexical_variety.LexicalVarietyQuantifier(analysis)
    Class to facilitate analysis of Lexical Variety and its variants.

    log_type_token_ratio()
    Returns 6 times the logarithmic type token ratio (as defined in the article)

    type_token_ratio()
    Returns 6 times the type token ratio (as defined in the article)

    unique_tokens()
    Returns tokens occurring only once in the text.
```
unique_type_token_ratio()
Returns the ratio for unique types. If all tokens are unique, lexical variety is considered to be infinity.

translationese.lexical VarietyVARIANTS = [0, 1, 2]
Possible variants

translationese.lexical Variety.quantify_variant(analysis, variant)
Quantifies lexical variety. Possible variants:

0  Straight type/token ratio
1  Logarithmic type/token ratio
2  Unique type/token ratio

2.2.2 Mean word length (in characters)

We assume that translated texts use simpler words, in particular shorter ones. Punctuation marks are excluded from the tokens in this feature.

translationese.mean_word_length.is_contraction_suffix(token)
Is token a contraction suffix?

>>> is_contraction_suffix("’s") # let’s
True
>>> is_contraction_suffix("n’t") # don’t
True

translationese.mean_word_length.is_hyphenated_word(token)
Is token a hyphenated word?

>>> is_hyphenated_word("ad-hoc")
True

translationese.mean_word_length.quantify(analysis)
Quantify mean word length.

2.2.3 Syllable ratio

We assume that simpler words are used in translated texts, resulting in fewer syllables per word. We approximate this feature by counting the number of vowel-sequences that are delimited by consonants or space in a word, normalized by the number of tokens in the chunk.

translationese.syllable_ratio.quantify(a)
Quantify syllable ratio.

translationese.syllable_ratio.syllables(word)
Approximate the number of syllables in word by counting vowel-sequences surrounded by consonants or space. Sometimes this works well:

>>> syllables("the")
1
>>> syllables("augmented")
3
>>> syllables("may-day")
2

Sometimes it doesn’t:
2.2.4 Lexical density

The frequency of tokens that are not nouns, adjectives, adverbs or verbs.

```
>>> syllables("make")
2
>>> syllables("awesome")
4
>>> syllables("syllables")
2
```

For non-words, 0 is returned:

```
>>> syllables(".")
0
```

2.2.5 Mean sentence length

Splitting sentences is a common strategy in translation, which is also considered a form of simplification. Long and complicated sentences may be simplified and split into short, simple sentences. Hence we assume that translations contain shorter sentences than original texts.

```
>>> mean_sentence_length.quantify(analysis)
Quantify mean sentence length.
```

2.2.6 Mean word rank

We assume that less frequent words are used more often in original texts than in translated ones.

```
>>> mean_word_rank.VARIANTS = [0, 1]
Possible variants for this hypothesis.

  0. Words not in this list are given a unique highest rank of VERY_HIGH_RANK.

  1. Words not in the list are ignored altogether.
```

```
>>> mean_word_rank.VERY_HIGH_RANK = 6000
Very high rank for a word, guessed for unknown words. The highest rank for known words is 5000.
```

```
>>> mean_word_rank.quantify_variant(analysis, variant)
Quantify mean word rank.
```

2.2.7 Most frequent words

The normalized frequencies of the $N$ most frequent words in the corpus. Punctuation marks are excluded.

```
>>> most_frequent_words.VARIANTS = [0, 1, 2]
Possible variants.

  0. $N = 5$

  1. $N = 10$

  2. $N = 50$
```
Quantify most frequent words
N most frequent words in the corpus.

2.3 Explicitation

The tendency to spell out in the target text utterances that are more implicit in the source.

2.3.1 Explicit naming

We hypothesize that one form of explicitation in translations is the use of a personal pronoun as a clarification of a proper noun. We calculate the ratio of personal pronouns to proper nouns, both singular and plural, magnified by an order of 3. See also translationese.pronouns.

Quantify explicit naming.

2.3.2 Single naming

The frequency of proper nouns consisting of a single token, not having an additional proper noun as a neighbor. As a contemporary example, it is common to find in German news (as of 2012) the single proper name Westerwelle, but in translating German news into another language, the translator is likely to add the first name of this person (Guido) and probably his role, too (minister of foreign affairs).

Quantify single naming.

2.3.3 Mean multiple naming

The average length (in tokens) of proper nouns (consecutive tokens tagged as Proper Nouns), magnified by an order of 3. The motivation for this feature is the same as translationese.single_naming.

Quantify mean multiple naming.

2.3.4 Cohesive markers

Translations are known to excessively use certain cohesive markers (see list below).

List of cohesive markers

Quantify usage of cohesive markers.
2.4 Normalization

Translators take great efforts to standardize texts. We include in this the tendency to avoid repetitions, the tendency to use a more formal style manifested in refraining from the use of contractions, and the tendency to overuse fixed expressions even when the source text refrains, sometime deliberately, from doing so.

2.4.1 Repetitions

We count the number of content words (words tagged as nouns, verbs, adjectives or adverbs) that occur more than once in a chunk, and normalize by the number of tokens in the chunk. Inflections of the verbs be and have are excluded from the count since these verbs are commonly used as auxiliaries. This feature’s values are magnified by an order of 3.

```
translationese.repetitions.ignored_tokens = set(['be', 'being', 'had', 'is', 'am', 'been', 'are', 'have', 'were', 'has'])
```

```
Ignored tokens
```

```
translationese.repetitions.quantify(analysis)
```

Quantify repetitions.

2.4.2 Contractions

The ratio of contracted forms to their counterpart full form(s). If the full form has zero occurrences, its count is changed to 1.

```
translationese.contractions.CONTRACTIONS = {'i’ll': ['i will'], 'who’ve': ['who have'], 'she’ll': ['she will'], 'you’d': ['you would', 'you had'], ...
```

List of contracted forms used in this hypothesis.

```
translationese.contractions.quantify(analysis)
```

Quantify contractions.

2.4.3 Average PMI

We expect original texts to use more collocations, and in any case to use them differently than translated texts. This hypothesis assumes that translations overuse highly associated words. We therefore use as a feature the average PMI (see `translationese.Analysis.pmi()`) of all bigrams in the chunk.

```
translationese.average_pmi.quantify(analysis)
```

Quantify average PMI

2.4.4 Threshold PMI

We compute the PMI (see `translationese.Analysis.pmi()`) of each bigram in a chunk, and count the (normalized) number of bigrams with PMI above 0.

```
translationese.threshold_pmi.quantify(analysis)
```

Quantify threshold PMI

2.5 Interference

Associated with the output of non-native speakers producing utterances in their second language. Operates on different levels from transcribing source language words, through using loan translations, to exerting structural influence.
2.5.1 POS n-grams

We hypothesize that different grammatical structures used in the different source languages interfere with the translations, and that translations have unique grammatical structure. We model this assumption by defining as features unigrams, bigrams and trigrams of POS tags.

\[ \text{translationese.pos_n_grams.VARIANTS} = [0, 1, 2] \]

Possible variants (unigrams, bigrams, trigrams)

\[ \text{translationese.pos_n_grams.quantify_variant(analysis, variant)} \]

Quantify POS n-grams

2.5.2 Character n-grams

We hypothesize that grammatical structure manifests itself in this feature, and as in POS n-grams, the different grammatical structures used in the different source languages interfere with the translations. We also hypothesize that this feature captures morphological features of the language. These are actually three different features (each tested separately): unigrams, bigrams and trigrams of characters. They are computed similarly to the way POS n-grams are computed: by the frequencies of \( n \)-letter occurrences in a chunk, normalized by the chunk’s size. Two special tokens are added to indicate the beginning and end of each word, in order to properly handle specific word prefixes and suffixes. We do not capture cross-token character n-grams, and we exclude punctuation marks.

\[ \text{class translationese.character_n_grams.CharacterNGramQuantifier(variant)} \]

Class for quantifying character variant-grams

\[ \text{quantify(analysis)} \]

Quantify character n-grams.

\[ \text{translationese.character_n_grams.VARIANTS} = [0, 1, 2] \]

Possible variants: Unigrams, bigrams, trigrams

\[ \text{translationese.character_n_grams.WORD_END} = '>' \]

Special token added to end of word

\[ \text{translationese.character_n_grams.WORD_START} = '<' \]

Special token added to start of word

\[ \text{translationese.character_n_grams.quantify_variant(analysis, variant)} \]

Quantify character n-grams.

2.5.3 Contextual function words

This feature is a variant of POS n-grams, where the n-grams can be anchored by specific (function) words. This feature is defined as the (normalized) frequency of trigrams of function words in the chunk. In addition, we count also trigrams consisting of two function words (from the same list) and one other word; in such cases, we replace the other word by its POS. In sum, we compute the frequencies in the chunk of triplets \( <w_1, w_2, w_3> \), where at least two of the elements are functions words, and at most one is a POS tag.

See also \text{translationese.function_words}.

\[ \text{translationese.contextual_function_words.function_word_or_POS(token, tag)} \]

Returns the given token if it is a function word, or its POS tag otherwise.

\[ \text{translationese.contextual_function_words.quantify(analysis)} \]

Quantify contextual function words.

\[ \text{translationese.contextual_function_words.trigram_is_functional(trigram)} \]

Returns true iff the given trigram has at least two function words, and three words altogether.
2.5.4 Positional token frequency

Writers have a relatively limited vocabulary from which to choose words to open or close a sentence. We hypothesize that the choices subject to interference. The value of this feature is the normalized frequency of tokens appearing in the first, second, antepenultimate, penultimate and last positions in a sentence. We exclude sentences shorter than five tokens. Punctuation marks are considered as tokens in this feature, and for this reason the three last positions of a sentence are considered, while only the first two of them are interesting for our purposes.

```
translationese.positional_token_frequency.POSITION_NAMES = {'penultimate': -3, 'second': 1, 'antepenultimate': -4, 'last': -2, 'first': 0}
```

Names of the various positions of the sentence, final period excluded.

```
translationese.positional_token_frequency.quantify(analysis)
```

Analyze positional token frequency.

2.6 Miscellaneous

A number of features that cannot be associated with any above hypothesis category.

2.6.1 Function words

We aim to replicate the results of Koppel and Ordan (2011) with this feature. We use the same list of function words (in fact, some of them are content words, but they are all crucial for organizing the text) and implement the same feature. Each function word in the corpus is a feature, whose value is the normalized frequency of its occurrences in the chunk.

```
translationese.function_words.FUNCTION_WORDS = ['a', 'about', 'above', 'according', 'accordingly', 'actual', 'actually', 'after', 'afterward', 'afterwards', 'again', ...
```

Koppel and Ordan’s list of Function Words

```
translationese.function_words.quantify(analysis)
```

Quantify function words.

2.6.2 Pronouns

This hypothesis checks whether pronouns from `translationese.function_words` alone can yield a high classification accuracy. Each pronoun in the corpus is a feature, whose value is the normalized frequency of its occurrences in the chunk.

```
translationese.pronouns.PRONOUNS = ['he', 'her', 'hers', 'herself', 'him', 'himself', 'i', 'it', 'itself', 'me', 'mine', 'myself', 'one', 'oneself', 'ours', 'ourselves', 'she', 'their', 'theirs', 'them', 'themselves', 'they', 'us', 'we', 'you', 'yourself'
```

List of pronouns

```
translationese.pronouns.quantify(analysis)
```

Quantify pronouns.

2.6.3 Punctuation

Punctuation marks organize the information within sentence boundaries and to a great extent reduce ambiguity; according to the explicitation hypothesis, translated texts are less ambiguous, and we assume that this tendency will manifest itself in the (different) way in which translated texts are punctuated.

```
translationese.punctuation.VARIANTS = [0, 1, 2]
```

Possible variants:

0. The normalized frequency of each punctuation mark in the chunk.
1. A non-normalized notion of frequency: n/tokens, where n is the number of occurrences of a punctuation mark; and tokens is the actual (rather than normalized) numbers of tokens in the chunk. This value is magnified by an order of 4.

2. n/p, where p is the total number of punctuations in the chunk; and n as above. This value is magnified by an order of 4.

\[
\text{translationese.punctuation.count_punctuation_marks (analysis)}
\]
Count the amount of punctuation marks in the text of analysis.

\[
\text{translationese.punctuation.punctuation_marks} = [?,?, !, ;, :, ',', '-', '(', ')', '[', ']', '"", "/", ",", ";", \text{etc.}]
\]
Relevant punctuation marks

\[
\text{translationese.punctuation.quantify_variant (analysis, variant)}
\]
Quantify punctuation marks.

### 2.6.4 Ratio of passive forms to all verbs

We assume that English original texts tend to use the passive form more excessively than translated texts, due to the fact that the passive voice is more frequent in English than in some other languages. If an active voice is used in the source language, translators may prefer not to convert it to the passive. Passives are defined as the verb be followed by the POS tag VBN (past participle). We calculate the ratio of passive verbs to all verbs, and magnified it by an order of 6.

\[
\text{translationese.ratio_to_passive_verbs.quantify (analysis)}
\]
Quantify ratio of passive forms to all verbs.
CHAPTER 3

Utility code

3.1 analyze.py - Runner module

This module serves as glue code to facilitate running analysis. For documentation, run:

$ ./analyze.py -h

3.2 Main translationese module

class translationese.Analysis (fulltext=None, stream=None, filename=None)

This class represent and caches an nltk analysis of a given text. The text can be initialized either from a file (stream) or from fulltext.

All analyses performed by this class are cached using memoize, so they can be re-run cheaply. Also, when text is loaded using filename, this object will be saved using pickle to a file with .analysis appended to its name. Consequent analyses will load cached data.

bigrams()

Returns a histogram of bigrams in the text.

>>> Analysis("Hello hello hello world").bigrams()
{('hello', 'world'): 1, ('hello', 'hello'): 2}

case_tokenized_sentences()

List of sentences, tokenized, case-sensitive.

>>> Analysis("Hello. How are you?").case_tokenized_sentences()
[['Hello', '.'], ['How', 'are', 'you', '?']]

case_tokens()

Same as tokens, but case-sensitive.

histogram()

Return a histogram of tokens in the text.

>>> Analysis("Hello, hello world.").histogram()
{'world': 1, ',': 1, 'hello': 2, ',': 1}

histogram_normalized()

Same as histogram, but normalized by number of tokens.
pmi()
Returns a dictionary with the PMI of each bigram. Given a bigram $w_1, w_2$, its PMI is
\[
\log\left(\frac{\text{freq}(w_1w_2)}{\text{freq}(w_1) \cdot \text{freq}(w_2)}\right).
\]

pos_tags()
Return part-of-speech tags, for the entire document.

```python
>>> Analysis("I am fine. How are you?").pos_tags()
...[['I', 'PRP'], ['am', 'VBP'], ['fine', 'NN'], ['.', '.'], ['How', 'WRB'], ['are', 'VBP'], ['you', 'PRP'], ['?', '.']]
```

pos_tags_by_sentence()
Return part-of-speech tags, split by sentence. Case-sensitive, as part-of-speech tagging is case-sensitive by nature (nouns vs. proper nouns).

```python
>>> Analysis("I am fine. How are you?").pos_tags_by_sentence()
...[['I', 'PRP'], ['am', 'VBP'], ['fine', 'NN'], ['.', '.']], [['How', 'WRB'], ['are', 'VBP'], ['you', 'PRP'], ['?', '.']]
```

tokenized_sentences()
List of sentences, tokenized as lowercase.

```python
>>> Analysis("Hello. How are you?").tokenized_sentences()
[['hello', '.'], ['how', 'are', 'you', '?']]
```

tokens()
Tokens are always in lowercase. For tokens with the original case, use case_tokens().

tokens_set()
Same as tokens, but as a set.

exception translationese.MissingVariant
Exception thrown when no variant was specified when quantifying using a module that requires a variant specification.

exception translationese.NoSuchVariant
Exception thrown when an invalid variant was specified when quantifying using a module that supports variant specification.

exception translationese.NoVariants
Exception thrown when a variant was specified when quantifying using a module that does not support variant specification.

3.3 Word ranks
Word ranks obtained from http://www.insightin.com/esl/

3.4 Utilities
Various utilities for translationese analysis

translationese.utils.flatten_list(l)
Returns a flat list given a list of lists.
translationese.utils.flatten_list([[1,2],[3,4],[5,6]])
[1, 2, 3, 4, 5, 6]

translationese.utils.is_proper_noun(token_tag_pair)
Given a pair of a token and a tag, returns True if it represents a proper noun.
>>> import nltk
>>> nltk.pos_tag(nltk.word_tokenize("Impressive! John defeated Jim!"))
(...
(‘Jim’, ‘NNP’), (‘!’ , ‘.’)]
>>> is_proper_noun(('Impressive', 'JJ'))
False
>>> is_proper_noun(('John', 'NNP'))
True

translationese.utils.output_filter_ngram(ngram)
Returns the ngram in a form suitable for ARFF output.
>>> output_filter_ngram(('a','b','c'))
'a b c'

translationese.utils.sparse_dict_increment(d, k)
Increment key k in dictionary d, assuming 0 if missing.
>>> d = {}
>>> sparse_dict_increment(d, "key")
>>> sparse_dict_increment(d, "key")
>>> d
{‘key’: 2}

### 3.5 Memoize

This module provides the @memoize decorator, which is used as a method decorator. Decorating a method with @memoize will cause its result to be cached in the hidden _memoize_cache property, and subsequent calls will return this cache.

This particular implementation of memoize does not support methods with arguments (other than the implicit self argument). However, it does support pickle serialization.

```python
>>> from memoize import memoize
>>> class K:
...     def __init__(self):
...         self.x = 5
...     @memoize
...     def m(self):
...         return self.x * 2
>>> k = K()
>>> k.m()
10
>>> k.x = 6
>>> k.m() # Will not recompute
10
>>> import pickle
>>> s = pickle.dumps(k) # resulting objects can be pickled!
```
memoize.dump(obj, filename)
   Dumps obj's memoized cache as pickle data into filename.

memoize.load(obj, filename)
   Loads pickle data (saved by dump()) from filename and injects it into obj's cache.
This project is comprehensively covered by unit tests. You can run the tests like so:

```
$ sudo pip install nose
$ nosetests -v
```

Furthermore, if you use PyDev, it is recommended to configure it to use `nose` as the test runner, and have it re-run all tests whenever a file is saved.

Two types of tests are present:

- Integration tests, testing the complete quantification of a small sample
  - These are implemented using `unittest`, within the `tests` directory.
- Unit tests, testing the output of a utility function.
  - These are implemented, when convenient, as doctests - examples present in the docstring of the utility function. These are tested by nosetests as well.

### 4.1 Test utility classes

```python
class tests.util.SparseDictEqualMixin (methodName='runTest')
Mixin for testing sparse dictionary equality, where all non-expected keys should be 0.0.

```
To compare against the original article, all hypotheses were run against ~2000-token chunks of EUROPARL data. The chunks were randomly chosen, consisting of 1000 $O$ chunks and 1000 $T$ chunks.

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Variant</th>
<th>Accuracy</th>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Simplification</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>translationese.lexical_variety</td>
<td>0</td>
<td>72.8</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>72.75</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>75.6</td>
<td>76</td>
</tr>
<tr>
<td>translationese.mean_word_length</td>
<td></td>
<td>64.05</td>
<td>66</td>
</tr>
<tr>
<td>translationese.syllable_ratio</td>
<td></td>
<td>59.35</td>
<td>61</td>
</tr>
<tr>
<td>translationese.lexical_density</td>
<td></td>
<td>52</td>
<td>53</td>
</tr>
<tr>
<td>translationese.mean_sentence_length</td>
<td></td>
<td>65.45</td>
<td>65</td>
</tr>
<tr>
<td>translationese.mean_word_rank</td>
<td>0</td>
<td>54.8</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>54.9</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>65.65</td>
<td>64</td>
</tr>
<tr>
<td>translationese.most_frequent_words</td>
<td>1</td>
<td>71.15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>83.3</td>
<td></td>
</tr>
<tr>
<td><strong>Explicitation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>translationese.explicit_naming</td>
<td></td>
<td>59.8</td>
<td>58</td>
</tr>
<tr>
<td>translationese.single_naming</td>
<td></td>
<td>57.95</td>
<td>56</td>
</tr>
<tr>
<td>translationese.mean_multiple_naming</td>
<td></td>
<td>48.05</td>
<td>54</td>
</tr>
<tr>
<td>translationese.cohesive_markers</td>
<td></td>
<td>82.15</td>
<td>81</td>
</tr>
<tr>
<td><strong>Normalization</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>translationese.repetitions</td>
<td></td>
<td>74.5</td>
<td>55</td>
</tr>
<tr>
<td>translationese.contractions</td>
<td></td>
<td>50.35</td>
<td>50</td>
</tr>
<tr>
<td>translationese.average_pmi</td>
<td></td>
<td>59.35</td>
<td>52</td>
</tr>
<tr>
<td>translationese.threshold_pmi</td>
<td></td>
<td>61.7</td>
<td>66</td>
</tr>
<tr>
<td><strong>Interference</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>translationese.pos_n_grams</td>
<td>0</td>
<td>87.5</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>94.1</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>97.1</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>74.65</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>95.2</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>99.45</td>
<td>100</td>
</tr>
<tr>
<td><strong>Explicitation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>translationese.contextual_function_words</td>
<td></td>
<td>95.85</td>
<td>100</td>
</tr>
<tr>
<td>translationese.positional_token_frequency</td>
<td></td>
<td>96.25</td>
<td>97</td>
</tr>
</tbody>
</table>

*No variants were specified for Most frequent words in the original article.*
This project analyzes the various features attributed to a language known as *translationese*. The project was developed in the Natural Language Processing Laboratory, under the supervision of Prof. Shuly Wintner.

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Variant</th>
<th>Accuracy (^3)</th>
<th>Original (^4)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>translationese.function_words</em></td>
<td>3</td>
<td>96.3</td>
<td>96</td>
</tr>
<tr>
<td><em>translationese.pronouns</em></td>
<td>4</td>
<td>76.4</td>
<td>77</td>
</tr>
<tr>
<td><em>translationese.punctuation</em></td>
<td>0</td>
<td>81.75</td>
<td>81</td>
</tr>
<tr>
<td><em>translationese.pronouns</em></td>
<td>1</td>
<td>81.6</td>
<td>85</td>
</tr>
<tr>
<td><em>translationese.ratio_to_passive_verbs</em></td>
<td>2</td>
<td>81.35</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>65.25</td>
<td>65</td>
</tr>
</tbody>
</table>

\(^3\) Accuracy \(^4\) Original
The project’s goal is distinguishing between texts that were originally written in English (“O” texts) and texts written in foreign languages and were translated into English (“T” texts). This is done based on features and characteristics found within the texts.

This project was designed to be easily extensible by adding new hypotheses. See *Aspects of Implementation*. 

---

**Project description**

---
For basic usage, place several originally-english (O) files in /home/user/original, and several translated-to-english (T) files in /home/user/translated. Now, run the following:

$ ./analyze.py -o ~/original -t ~/translated lexical_variety:0
$ weka lexical_variety:0.arff
# Or, for immediate textual results:
$ weka -c weka.classifiers.functions.SMO -- -t lexical_variety:0.arff

This will run the lexical_variety quantifier, using variant 0, and run WEKA's SMO classifier-builder on the output.
Tools used

**Python 2.7**  Code for processing, analyzing and self-testing. http://python.org

**NLTK Library**  Natural Language ToolKit for python. Text processing and analysis was performed using this library. http://nltk.org

**Weka 3.7**  Classification tool. Uses ARFF files as input. http://www.cs.waikato.ac.nz/ml/weka/

**Sphinx**  Documentation generator. http://sphinx-doc.org

We would like to thank our guide, Prof. Shuly Wintner.
CHAPTER 9

Indices and tables

- genindex
- modindex
- search
m

memoize, 15

t

tests.util, 17
translationese, 13
translationese.average_pmi, 8
translationese.character_n_grams, 9
translationese.cohesive_markers, 7
translationese.contextual_function_words, 9
translationese.contractions, 8
translationese.explicit_naming, 7
translationese.function_words, 10
translationese.lexical_density, 6
translationese.lexical_variety, 4
translationese.mean_multiple_naming, 7
translationese.mean_sentence_length, 6
translationese.mean_word_length, 5
translationese.mean_word_rank, 6
translationese.most_frequent_words, 6
translationese.pos_n_grams, 9
translationese.positional_token_frequency, 10
translationese.pronouns, 10
translationese.punctuation, 10
translationese.ratio_to_passive_verbs, 11
translationese.repetitions, 8
translationese.single_naming, 7
translationese.syllable_ratio, 5
translationese.threshold_pmi, 8
translationese.utils, 14
translationese.word_ranks, 14