SeqAnalyzer

User Guide

Daniel Glickman

Project Overview

The project provides a framework for analyzing sequential data.
It offers modules for processing data and extracting statistically significant features from it, as well as modules for selecting, from these features, those that can differentiate between different sets of sequential data. We show how this is easily extended to perform classification of new data by having the project integrate with a popular machine learning library.
For extraction, It effectively uses a highly optimized java program, which allows working with very large quantities of data, but can also work as a standalone by having a simpler extractor written in python.

Glossary

Sequence - the basic input unit.
e.g a DNA sequence or an english sentence
Set - a group of sequences who belong to the same one class.
e.g one or many files with DNA sequences who share a common trait , a list of sentences from the same text.
Feature(of a sequence) - an interesting subsequence.each feature has a score.
e.g DNA motif or english word.

Setup

Environment

- Python 2.7
- Developed and tested on Linux. Some modules explicitly use linux commands and will not transfer to windows.

Requirements

SeqAnalyzer uses the following dependencies:

Numpy , Scipy and scikit-learn

Installing scikit-learn with pip should take care of all of them:
pip install -U scikit-learn[alldeps]
See http://scikit-learn.org/stable/install.html for more info.

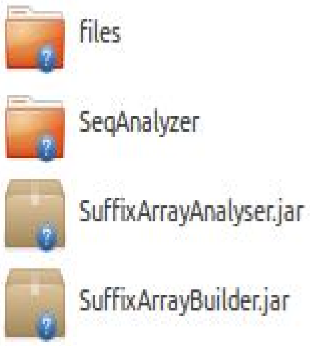
Workspace

SeqAnalyzer is a library containing the python code.
It uses 2 jar files for extraction, SuffixArrayAnalyser.jar and SuffixArrayBuilder.jar.

All examples below assume a working directory containing the following:
SeqAnalyzer, the python library.

SuffixArrayAnalyser.jar and SuffixArrayBuilder.jar.

files is a directory used to store input files and where the output files will be saved to.



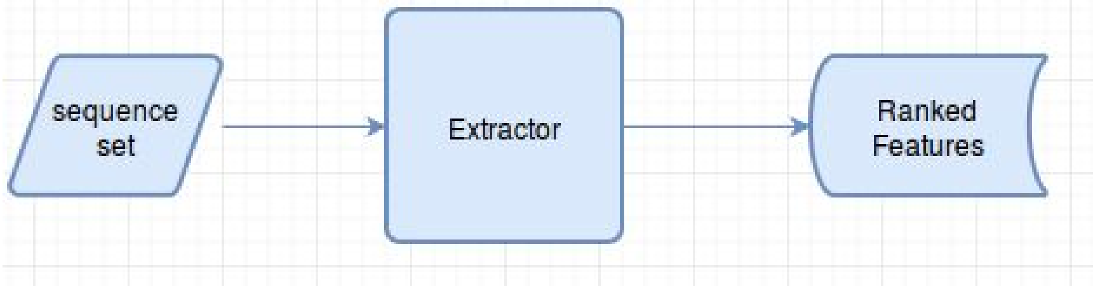
Files
SeqAnalyzer
SuffixArrayAnalyser.jar
SuffixArrayBuilder.jar

The files directory used in all examples contains 2 more directories, input1 and input2 who store different input files.

All parameters and names are configurable from Consts.py under SeqAnalyzer.

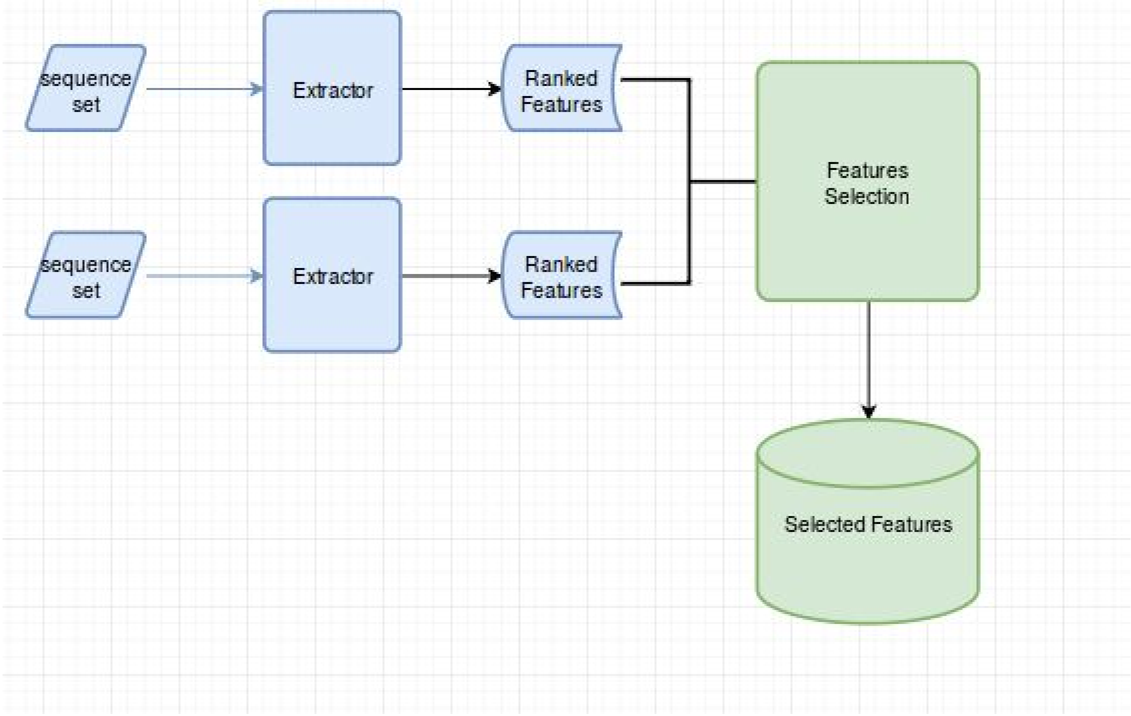
Workflow

Extraction



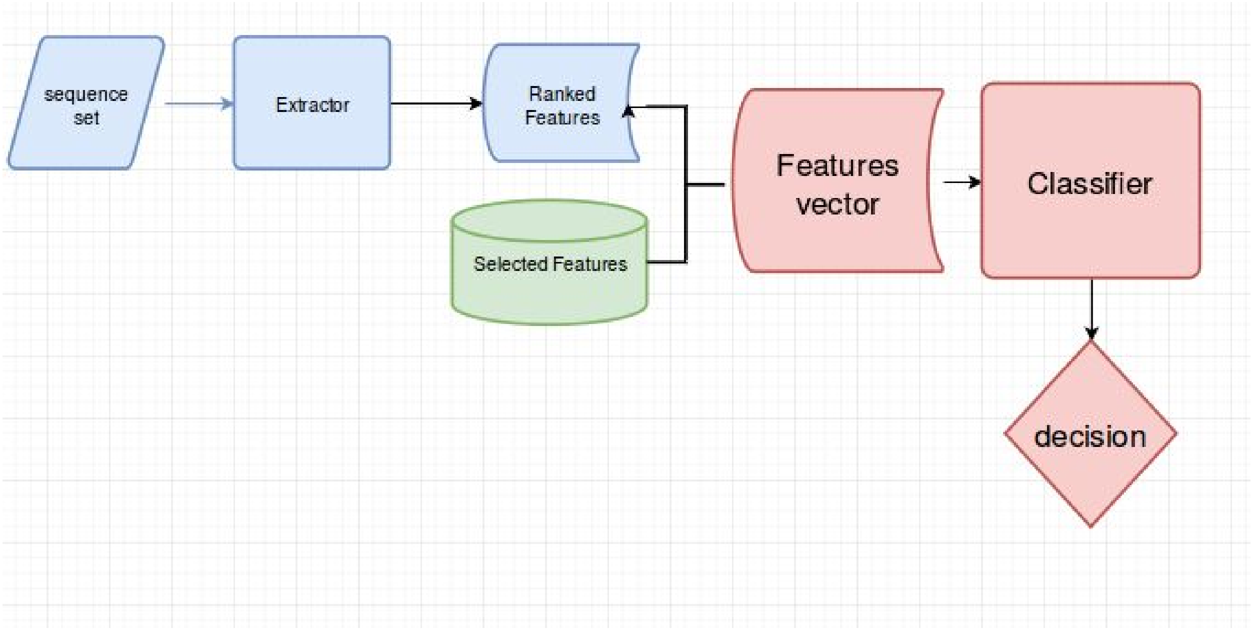
Given a set of sequences, the extractor will extract important subsequences of sequences in the set as features. The features are scored based on their count compared to other features of the same length, using the z-score, and are filtered by it, based on the idea that important features have higher scores.

Selection



To better understand what subsequences can characterize and differentiate between 2 sets of sequences, features selection iterates the numerous initial features extracted from both groups and selects a smaller subset who may help that goal.

Classification



When a new set arrives, it is classified as belonging to one of the reference sets by extracting features from it and using the selected ones as an input for a classifier.

Usage

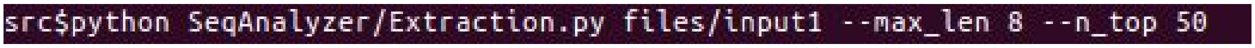
Extraction

Input

Path to a directory containing files with sequences.
Files should just contain sequences of characters, each sequence in a separate line.

Usage

Run Extraction.py with the input files directory path as first parameter.



```
src$python SeqAnalyzer/Extraction.py files/input1 --max_len 8 --n_top 50
```

Parameters:

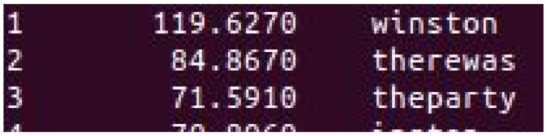
    --max_len maximum length of extracted subsequences.(default 10)
    --top_n the amount the script will output.(default 100)

* max_len has great effect on the run time so be aware

Output

The n_top best features and their score, in decreasing order.

The score of a feature is the z-score $z_i = (x_i - \bar{x})/s.$ of the feature frequency using the mean and standard deviation of subsequence of the same length.



```
1       119.6270    winston
2        84.8670    therewas
3        71.5910    theparty
```

The script is meant to enable easy user interaction so the output is printed directly to the screen. The output can be saved to a file by just piping from the command line, but for working with full features(as opposed to a list of top ones), which can be many millions, see the java user documentation or python maintenance manual.

Changing extractors

By default the extraction is done by the SuffixArray program, which is optimized to handle very large files.

It is also possible to perform extraction using pure python implementation. The implementation is not as memory efficient and scalable, however it is still able to work with reasonably sized data and may be useful when having many different small sets of sequences, since using the python extractor can save the overhead of calling another program and handling its I/O.

This makes the python extractor a good choice for classification, when there are many small individual sets.

A good rule of thumb would be that if the python extractor can effectively handle a single set it will handle many sets of the same size, since after extracting features from a single set, the number of features saved for that set will be greatly reduced by filtering out features that were not selected during feature selection.

To change extractors, change the variable EXTRACTOR_TYPE found in Params.py, under SeqAnalyzer, to python/java.

Selection

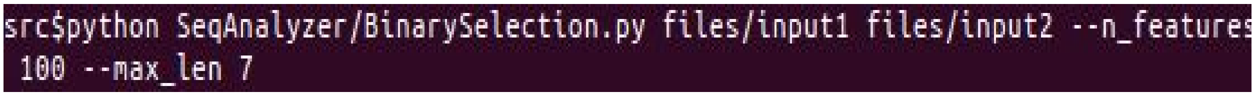
Input

Paths to 2 directories or files, containing sequences of the first and second class.
There are 2 options:

1) Using the “raw” input files, the script will do the extraction itself, so the input paths should follow same rules as in extraction.
2) Using the extraction output files directly, assuming it was already done earlier. The files lines format must be:
   *   score   feature(name)
   Where * is any non-whitespace string.

Usage

run the BinarySelection.py script with the appropriate paths and desired parameters.

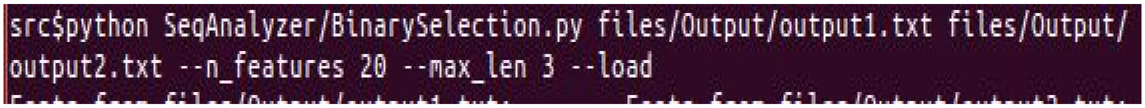


```
src$python SeqAnalyzer/BinarySelection.py files/input1 files/input2 --n_features
 100 --max_len 7
```

Parameters

  -- max_len : max length of features(default 10)
  --n_features : number of features that will selected, half from each class. (default 100)

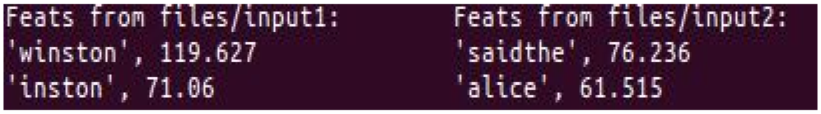
[optional] --load : use the extractor's output instead of original input files.(not used by default)



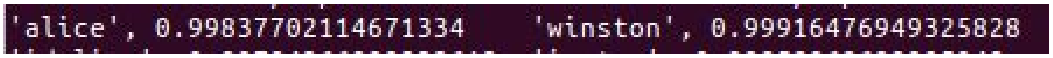
```
src$python SeqAnalyzer/BinarySelection.py files/Output/output1.txt files/Output/
output2.txt --n_features 20 --max_len 3 --load
```

Output

The selected features presented side by side in decreasing order by their feature selection function score. This score is not the same as the one presented in extraction.



```
Feats from files/input1:       Feats from files/input2:
'winston', 119.627             'saidthe', 76.236
'inston', 71.06                'alice', 61.515
```



```
'alice', 0.99837702114671334    'winston', 0.99916476949325828
```

Selection functions

Selection functions are used to evaluate how good a feature is at differentiating one class from another. Selection functions get the score of a feature according the first sequence set and the score of the same feature in the other set, and return a number indicating how good that feature is and what set it may characterize.

There are a few different selection functions, that each give different features as a result.Generally speaking, there are 2 factors that indicate that a feature is a good one:

1) Absolute strength - The feature has a high score with that set, indicating strong statistical significance.
2) Relative strength - The feature is stronger in one set relative to the other and is more likely to be found in that set.

To illustrate the idea, top absolute strength features, for a text about technology, can include the word “computer”, but it can also be common terms such as “of” and “the”, while more rare term like “macintosh” may not show. Relative ones will not include “the” as it is strong in all texts, not only technology ones, but can include rare words that show in the text by chance.

Changing functions

The different functions varies in what factors they emphasize and can give different results.

To change selection functions, change the variable SELECTION_TYPE in Params.py . Possible values codes are highlighted in yellow below.

Defining new functions in selection_functions.py will automatically make their name useable from Params.py.

Gini impiurity - ‘gini’

https://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity

This function optimizes directly for relative strength, looking at the impurity between the 2 scores.

** the function is implemented so that if 2 different features gini impurity is equal, the feature with the higher scores is prefered.

Difference / Variance - ‘diff’

This function measures the absolute difference between the 2 scores. While still considering relative strength, the function emphasises absolute strength more.

It will rank the features in the exact same way as using the variance between 2 scores would, assuming equal probability.

Combined - ‘combined’

Combines gini impurity and difference by multiplying the two.

Chi square test - ‘chi’ : https://en.wikipedia.org/wiki/Chi-squared_test

Unlike difference, which only considers the scale of difference, chi square also factors the difference in proportion to the score of the features.

Note that due to practical considerations the functions are implementations do not exactly match their mathematical definitions, but the results should be similar.

Classification

With basic python programming abilities, the program can easily be extended to classify your data. Depending on your requirements, it can be as simple as writing a function to read your data and plug to existing code.

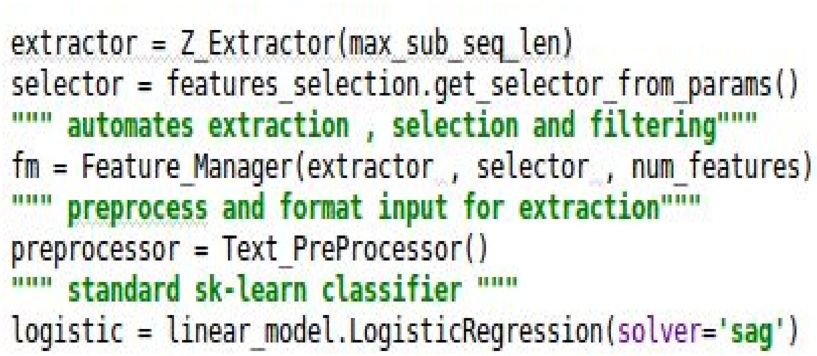
In addition the project integrates with scikit-learn, a popular machine learning library.

Integrating with scit-learn

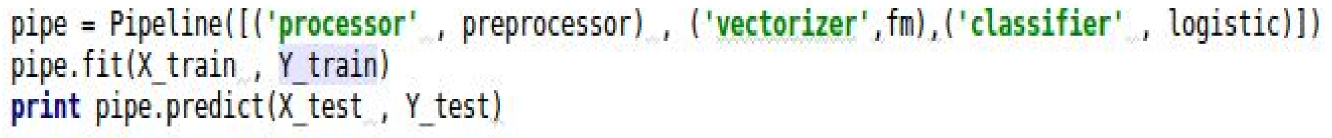
The project deals with data in a non standard way. Processing sequences, extracting massive amounts of subsequences and making sense from those using efficient feature selection all require special treatment. However, by the end of the process, we are left with a system that transforms sequential data to a vector of selected features in a standard representation.

Integrating that with scikit-learn instantly opens new ways to play with our data using known implementations.

We will show a small example to illustrate the idea.



```
extractor = Z_Extractor(max_sub_seq_len)
selector = features_selection.get_selector_from_params()
""" automates extraction , selection and filtering"""
fm = Feature_Manager(extractor , selector , num_features)
""" preprocess and format input for extraction"""
preprocessor = Text_PreProcessor()
""" standard sk-learn classifier """
logistic = linear_model.LogisticRegression(solver='sag')
```



```
pipe = Pipeline([('processor' , preprocessor) , ('vectorizer',fm),('classifier' , logistic)])
pipe.fit(X_train , Y_train)
print pipe.predict(X_test , Y_test)
```

Once handling sequences is taken care of its easy to plug a ready-made classifier.In addition to extraction and selection, we provide tools to process, format and create data. For full information see the maintenance manual.

Example result

The example uses 2 different texts corpuses, one is about technology and the other is about politics. White spaces and punctuation were removed.

First we extract many initial features. Among the 20 top ranking ones are popular words such as : “the” , “of” , “the” , along with combinations of words such as “theyare” and parts of words.

From this initial features, which are many millions , we select only a small portion using feature selection. The highest ranking feature that is characterized as belonging to the technology corpus is the word “computer”. Other features in the top 20 are “program” , “software” , “function” and “graphics” while the political selected features contain words such as “president” , “israel” and “middleeast”.

With these features we can efficiently train a classifier using standard machine learning libraries. We extracted a vector of 30000 selected features for each item in the corpus, that was used in training or testing.
a simple logistic regression classifier that was able to correctly predict over 92% of unseen data.

Other

Data Generation

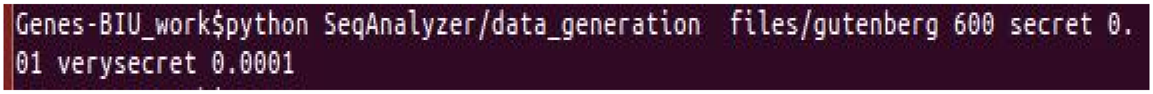
Data generation helps the process of testing with english text files.

Input

Path to directory with all text files
Max line/sequence length(for formating)
Pairs of target word , probability to insert word

Usage

Call SeqAnalyzer/data_generation.py with the parameters in order.



```
Genes-BIU_work$python SeqAnalyzer/data_generation  files/gutenberg 600 secret 0.
01 verysecret 0.0001
```

Output

Will create 2 directories , one with a formatted version of the original files and the other is with formated files with target words inserted to them at random with the given probability.