

Twitter Writing Style Analysis

FAN Yi-Zhe | HSIEH Yung-Kun



Introduction

This project is to apply the machine learning techniques learned in the Master course "Fouille de Données et Aide à la Décision (Data Mining and Decision Support)" at Paris Diderot University. Our ideal final goal is to detect the plagiarism of text on the internet from intellectuals/copywriters. Within the limit of time (in 6 weeks of implementation), we decided to start with the twitter analysis in order to correctly recognize the tweets from different authors.

From Kaggle.com, we found 6 datasets of tweets of different celebrities as: Hillary Clinton (Politician), Donald Trump (President), Barack Obama (Ex-president), Adam Savage (Host of Myth Busters), Kim Kardashian (Actress), and Richard Dawkins (Ethologist). Our objective is to feed these datasets to our program and to predict the new input text of which celebrity-alike.

We implemented all our codes in Python since there are many existing machine learning libraries. The source code is on GitHub:

- https://github.com/QCTW/ParisVII_ProjectMachineLearning

Implementations

In our implementations, we have 2 user flows as in the Figure-1. One flow is simply running train/test by feeding our datasets into seven different classifiers: "Nearest Centroid", "Multinomial Naive Bayes", "Ridge", "Perceptron", "Passive-Aggressive", "K Neighbors", and "Random Forest". It will then generate a pyplot bar-diagram (as in the Figure-3) to show the benchmarks of different classifiers.

The other flow is an interactive mode (-i) which will start right after the first train/test flow and will prompt some message waiting your input for prediction.

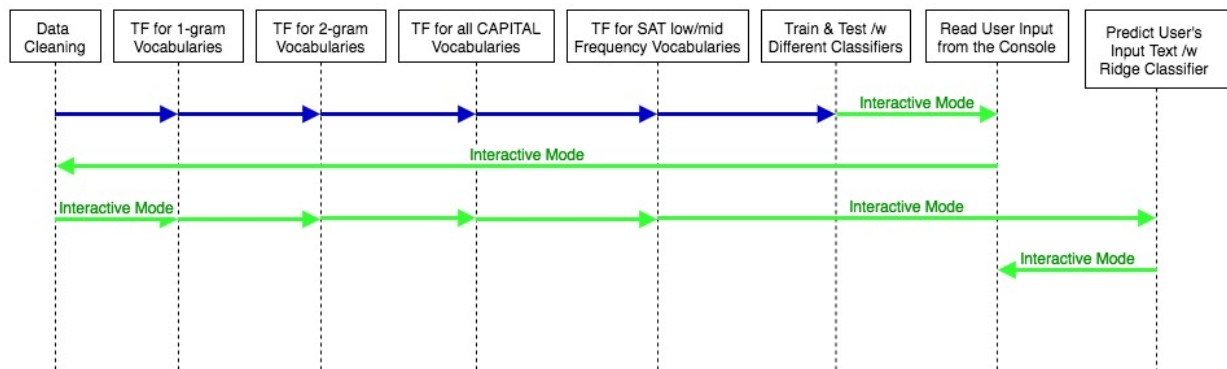


Figure-1: Program User Flow

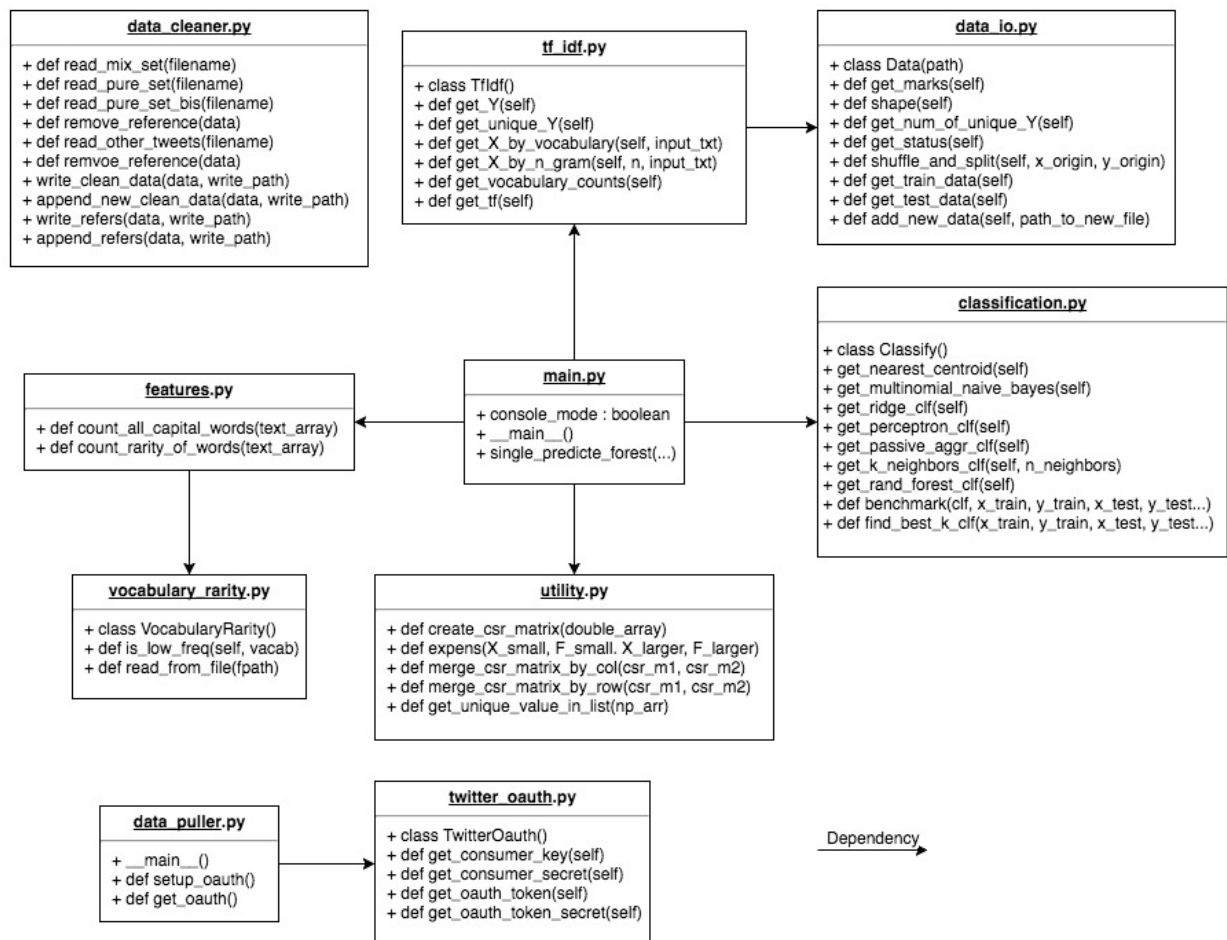


Figure-2: Class Diagrams

We modularize our code so that we can add new functions and new training features for new dataset easily. See Figure-2 for the class diagrams of our implementations.

To run the program with simple train and test, type:

```
python3 main.py
```

To run the program with interactive mode (-i) to predict your input text, type:

```
python3 main.py -i
```

Raw Data Cleaning

To have some proper text for analysis, we pre-processed raw data files from Kaggle and marked it with different authors.

1. Use CSV library to read the raw data files.
2. For each line, replace the quoted(re-tweeted) text between two quotation mark with string "__QUOTE__". This is to make sure that the text is purely from the author.
3. For each line, replace the internet URL which begin with "https:\\\" with string "__URL__".
4. For each line, replace the break-line ("\\n") with blank string.

Features for Classifiers

To train the datasets, we use the following 3003 features:

1. **Term frequency of 1500 1-gram vocabularies.** We use `CountVectorizer` from `sklearn` library with parameters:
 - `min_df` equals to 2: to eliminate the appear only once IDs and low frequency vocabularies. We keep only words who appears in more than 2 documents.
 - `max_df` equals to (1/number of known classes): We considered that if a word appears once in each documents of certain class, such a word appears too frequently and should be eliminated.
 - `max_features` equals to 1500: We keep only the top 1500 frequent words for comparisons. (This setting at the end has been removed and replaced by analysis of variance – `f_classif` filtering)
 - Use default stop words from `sklearn` to eliminate general high frequency words.
2. **Term frequency of 1500 2-gram vocabularies.**
Same parameters settings as 1-gram vocabularies, but we want to have frequency counts with 2 consecutive words.

3. Term frequency of all capital vocabularies.

We implemented all capital words frequency count in features.py.

4. Term frequency of very difficult SAT vocabularies.

We supposed that people who has higher education will write more "sophisticated", as the result in their text is that they tend to use difficult (rare) vocabularies more often. We use the vocabularies for preparing SAT (Scholastic Assessment Test) as a reference of vocabulary's difficulty.

To make sure that words in different forms are treated and count properly. We use `PorterStemmer` from NLTK library to transform both the words of datasets and the vocabularies in the SAT reference to their root forms. For example, the word "pythoning" will transform to "python" both for the words in the datasets and in the SAT reference.

5. Term frequency of medium difficult SAT vocabularies.

Same process as very difficult SAT vocabularies above.

Classifiers & Accuracies

The classifiers and benchmark tests are inspired by the sklearn tutorial for our code:

```
http://scikit-learn.org/stable/auto_examples/text/document_classification_20newsgroup
s.html#sphx-glr-auto-examples-text-document-classification-20newsgroups-py
```

The results of running seven classifiers are in the Figure-3 and Figure-4. We noticed that the Nearest Centroid Classifier has the worst accuracy of prediction (68.3%) and the top 3 most accurate classifiers are: Ridge Classifier (87.9%), Passive Aggressive Classifier (86.9%) and Multinomial Naive Bayes (86.4%). Therefore, we decided to use Ridge Classifier for the prediction of user inputs.

Since we had learned Random Forest and K-Neighbors in the class, we decided to try to find the optimal parameters (`n_estimators/ neighbors`) for Random Forest Classifier and K-Neighbors Classifier with cross validation, to see if they can have their accuracies improved. Result of the cross validation:

```
Divide the data set into 5 parts for validation cross...
For k 50 Average_score is 0.803410591175
For k 100 Average_score is 0.80605884285
For k 150 Average_score is 0.806097127419
For k 200 Average_score is 0.807172057208
For k 250 Average_score is 0.806596153063
For k 300 Average_score is 0.805789996208
For k 350 Average_score is 0.806327527525
...
```

```

~/git/ParisVII_ProjectMachineLearning — Python main.py -i
RichardDawkins      0.85      0.46      0.60      1499
    avg / total      0.72      0.65      0.65      8685

Confusion matrix:
[[ 579   34    9   11  562   50]
 [   33 1239    4   20  380    9]
 [   14   15  365   41  347   12]
 [   19   44   22  317  371    7]
 [  103   35   15   15 2474   40]
 [   93   23   17   17  661  688]]

=====
Training by: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
    max_depth=None, max_features='auto', max_leaf_nodes=None,
    min_impurity_decrease=0.0, min_impurity_split=None,
    min_samples_leaf=1, min_samples_split=2,
    min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=1,
    oob_score=False, random_state=None, verbose=0,
    warm_start=False)
Training time: 62.709s
Predict time:  0.942s
Accuracy:      0.858
Classification report:
              precision    recall  f1-score   support

HillaryClinton      0.85      0.62      0.72      1245
  DonaldTrump       0.95      0.92      0.93      1685
  BarackObama       0.89      0.77      0.83       794
   AdamSavage       0.86      0.87      0.87       780
  KimKardashian     0.81      0.95      0.87      2682
  RichardDawkins    0.84      0.87      0.86      1499

    avg / total      0.86      0.86      0.86      8685

Confusion matrix:
[[ 772   18   17    7  326  105]
 [   34 1542    3    2    75   29]
 [    5   16  613   91   35   34]
 [    1   16   42  677   19   25]
 [   57   16    7    4 2542   56]
 [   35   10    4    3  138 1309]]

=====
[!] Now it's your turn to input some text and try our prediction!
[?] Please input some text (more than 20 words):
█

```

Figure-3: Running Benchmark on Console

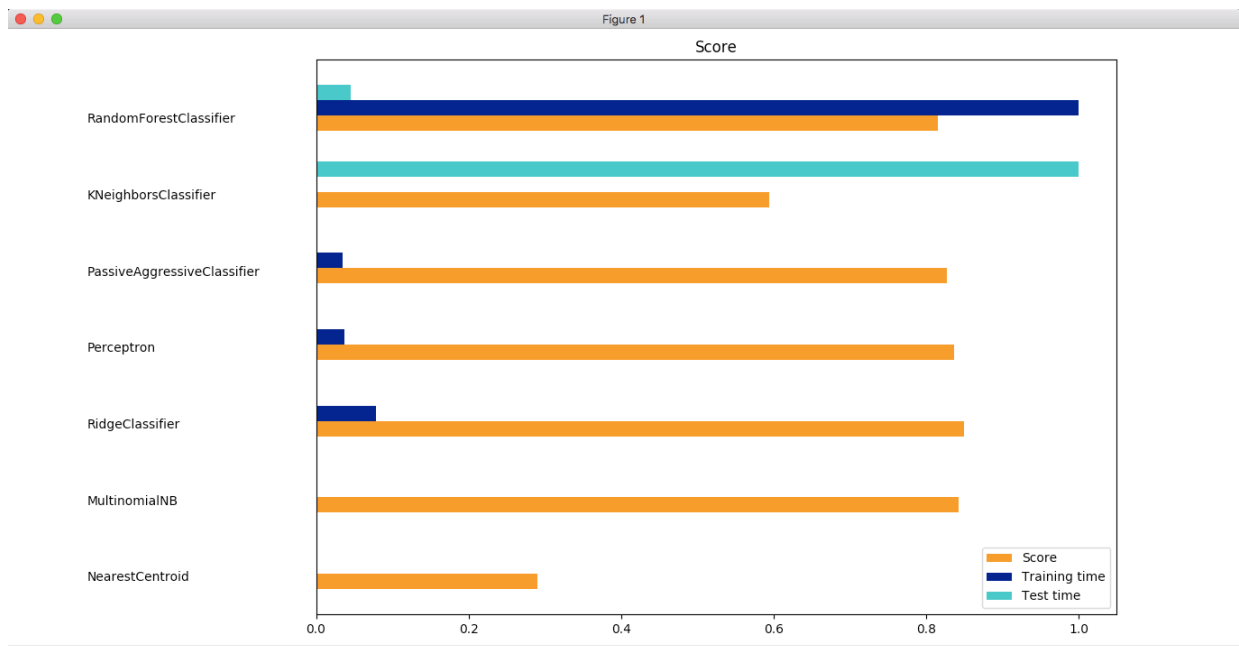


Figure-4: Benchmark of 7 Classifiers

Accuracy Improvements (82% to 88%)

At the very first run, the best accuracy we have is around 82%. We had tried many ways to improve the accuracy of our predictions.

1. We tried to add another 1000 3-gram features to see if the accuracy improved. But it only took longer time to compute without increasing the accuracies.
2. We tried to increase 1-gram and 2-gram features from 1000 to 2000 each and we found that 1500 each is a good average value. The accuracy increased 1-2% when TF features increased from 1000 to 1500 each.
3. At the very beginning, when calculating all capital words and difficult SAT vocabularies, their values are just sum of appearance counts. We change this value from “sum of counts” to “(sum of counts)/(total words count)”, the accuracy increased 2-3%, while the accuracy of Nearest Centroid jump from 45% to 67%! This taught us that the value of each feature must be correctly designed and within the same range as the other features.
4. At the beginning, we limit our max 1-gram and 2-gram features with parameter `max_features=n` of `CountVectorizer` which takes the top n largest value. We decided to remove this limit and filter out the top n variant values with `sklearn.feature_selection.f_classif`. It greatly increased the calculation time but also increased the accuracy 2%.

Difficulties

The difficulties that we had encountered:

1. How to feed classifier one tweet input from user? To predict one tweet input from user, we have to expend much fewer features calculated from the input text to the 3003 features used by training data.
2. How to manipulate the sparse matrices correctly? To add new feature to the Tf-Idf sparse matrices, we need to merge our new matrices into Td-Idf matrices.
3. What's the best parameters for each classifier? Most of time, we just use the default one since each classifier has its mathematical theory behind it...

Generalization?

Last, we asked ourselves: Could our algorithm of 3003 features work (with 88% of accuracy) for only datasets of Twitters or we can apply it on other datasets while having the same accuracy?

We applied our code on a Kaggle competition called “Spooky Author Identification” (<https://www.kaggle.com/c/spooky-author-identification>) and we obtained our best accuracy as 82.7% with the same Ridge Classifier with 3003 features. Its code is in `spooky_author_identification.py`.

```

[ 203 1082 153]
[ 217 259 992]]
=====
Training by: MultinomialNB(alpha=0.01, class_prior=None, fit_prior=True)
Training time: 0.020s
Predict time: 0.002s
Accuracy: 0.825
Classification report:
      precision    recall  f1-score   support

     EAP      0.82      0.81      0.82     1989
     HPL      0.84      0.83      0.84     1438
     MWS      0.81      0.84      0.82     1468

 avg / total      0.83      0.83      0.83     4895

Confusion matrix:
[[1616 155 218]
 [ 173 1197  68]
 [ 175  67 1226]]
Dimensionality: 3004
Density: 1.000000
=====
Training by: RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None, solver='auto',
                             tol=0.01)
Training time: 0.679s
Predict time: 0.002s
Accuracy: 0.827
Classification report:
      precision    recall  f1-score   support

     EAP      0.80      0.86      0.83     1989
     HPL      0.88      0.80      0.84     1438
     MWS      0.83      0.81      0.82     1468

 avg / total      0.83      0.83      0.83     4895

Confusion matrix:
[[1707 101 181]
 [ 219 1150  69]
 [ 217  60 1191]]
Dimensionality: 3004
Density: 1.000000

```

Figure-5: Benchmark of 3 Spooky Authors













Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Late Submission
835	▼ 70	Neural Nut					 0.52629	6 19d
836	▼ 23	Fabia					 0.52763	2 14d
837	▼ 3	BoBdec					 0.52990	1 12d
838	▼ 7	Fisher					 0.53290	7 1mo
839	▲ 107	Miles Hill					 0.53354	1 2mo
840	▲ 5	Suresh					 0.53672	1 1mo
841	▼ 13	EnriqueSantos					 0.54228	8 2mo
842	▲ 11	Aleksandr Shatilov					 0.54325	7 19d
843	▲ 20	Quincy					 0.54499	2 4d
844	▲ 20	haoeric					 0.54559	1 5d
845	▼ 2	ShahMuzaffarBashir					 0.54620	1 4d
846	▼ 5	Robert Sobolewski					 0.54784	2 2mo

Figure-5: Final Ranking of Kaggle Competition