# Tweet User Classification

**Kien Nguyen**

# 1. Data downloading and cleaning

- Some users do not have tweets or are inaccessible. Finally, a total of 510 users are processed
- Use ASCII to encode the messages and ignore all unicode characters. This is to reduce the feature space in such a small training set, to reduce overfitting
- Use preprocessor package to remove urls
- Remove punctuations
- Use english dictionary (PyEnchant) to remove words that are not English
- Remove stop words in English
- Split the dataset into training and test set. The training set contains 70% of the users (357 users), and the test set contains the remaining 30% (153 users).

# 2. Feature Representation

Use TF-IDF score as feature representation for this user classification problem.
From the training set, collect all tweets of all training users, and use them to build the vocabulary of terms as well as their IDF scores. This vocabulary (with IDF scores) is then applied to both the training set and test set to create training feature vector and test feature vector for each user.

Two different methods to compute IDF score for the term are considered here:

**Method 1 (IDF is computed based on user)**: Compute IDF by creating each document as the collection of all tweets of a user (i.e., each document represents a user). This is a common method in literature. In this method, the IDF of a term indicates the uniqueness of this term with respect to users. A low value of IDF means the term is used by many users (regardless of categories) and vise versa.

**Method 2 (IDF is computed based on category)**: Compute IDF by creating each document as the collection of all tweets of a category (i.e., each document represents a category). Hence, the number of documents is the number of categories. In this method, the IDF of a term indicates the uniqueness of this term with respect to categories. A low value of IDF means the term is used by many different categories and vise versa. So, intuitively, the IDF computed this way is more suitable to a user categorization (classification) task.

In the model selection section discussed below, both of these methods are evaluated to find the best one for the test phase.

Recall that the feature vector of a user is computed based on the vocabulary learned from the training set. Let D denote the vocabulary of all terms. The feature vector of each user is of |D| dimensions, where the $i^{th}$ feature corresponds to the TF-IDF score of the term Di. TF is computed based on the user tweets only, while IDF is computed by either method 1 or 2 above.
For both methods, the size of the vocabulary (feature dimension) learned from this dataset (from all users) is ~ 33,500 (thanks to the cleaning step). However, the feature matrix is very sparse, with only around 1200 (4%) non-zero features for each user, i.e., each user only uses a small set of terms in the vocabulary.

# 3. Feature selection and Model selection:

Model and feature selection are conducted by performing cross-validation (5 folds) in the training set.

For model selection, four different machine learning algorithms are evaluated: SVM (linear), Naive Bayes, Decision Tree and Random Forest (100 trees). Due to the high dimension of the features, only linear kernel is used for SVM because more complex kernel requires larger training set to avoid overfitting.

To perform feature selection, I compute feature scores (term scores) by using the **TF-IDF scores of the terms computed from Method 2**, i.e., ranking the terms based on their frequency in the category and IDF scores based on the categories. Hence, feature selection can be done by ranking the feature scores from high to low, and select only the top k% of the features.

For each of the feature representation method mentioned in section 2, I run the model selection independently and compare their results as shown in the two tables below.

| Top features selected (%) | Naive Bayes | Random Forest | SVM | Decision Tree |
|---|---|---|---|---|
| 100 | 0.54 | 0.78 | **0.82** | 0.67 |
| 90 | 0.56 | 0.77 | **0.82** | 0.68 |
| 50 | 0.61 | 0.78 | **0.82** | 0.69 |
| 30 | 0.64 | 0.77 | **0.82** | 0.68 |
| 10 | 0.67 | 0.81 | 0.81 | 0.68 |

**Table 1**: **Using feature representation method 1**. Cross-validation accuracy of different classifiers, with different portions of top features to be selected

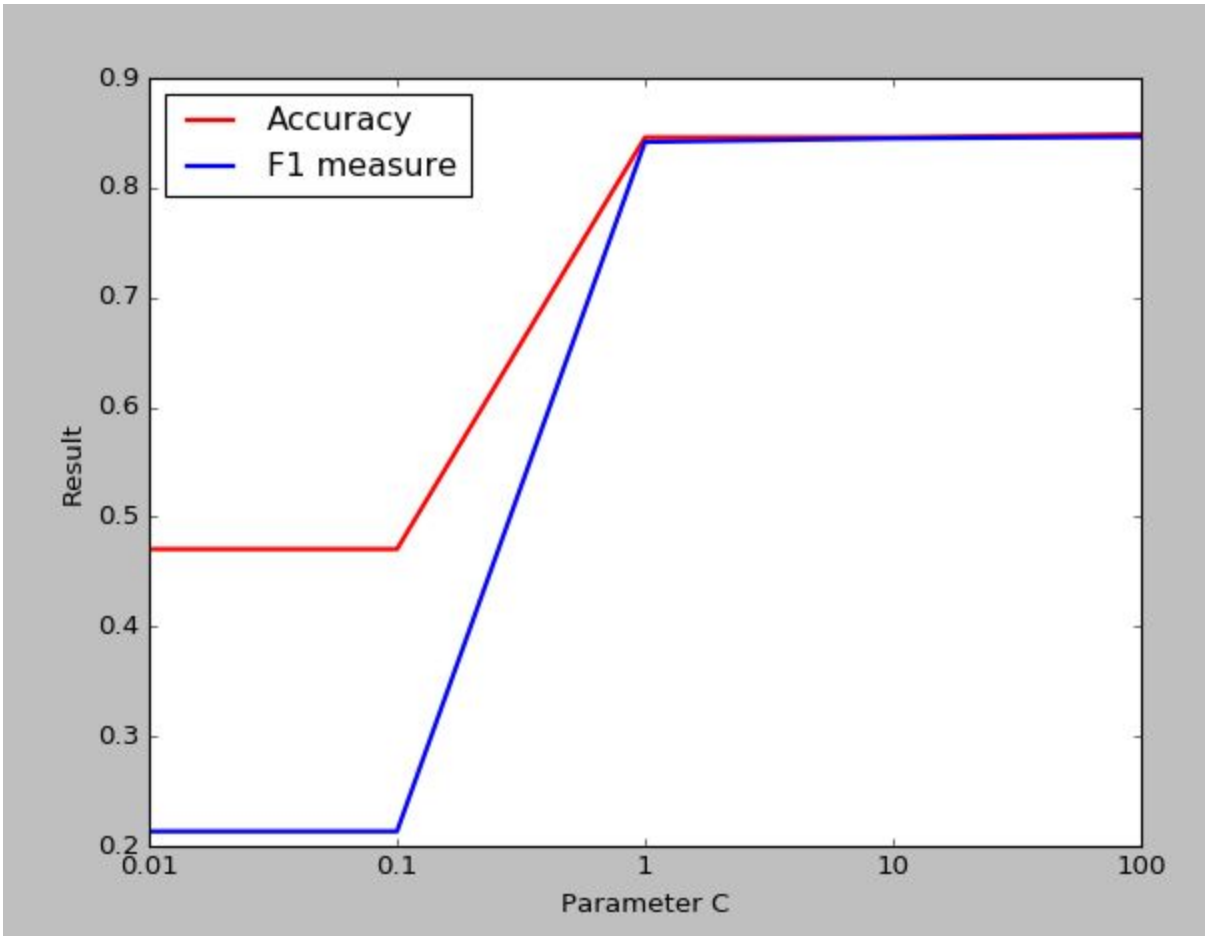
| Top features selected (%) | Naive Bayes | Decision Tree | Random Forest | SVM |
|---|---|---|---|---|
| 100 | 0.50 | 0.67 | 0.78 | **0.85** |
| 90 | 0.50 | 0.68 | 0.78 | **0.85** |
| 50 | 0.52 | 0.67 | 0.77 | **0.85** |
| 30 | 0.53 | 0.66 | 0.78 | **0.85** |
| 10 | 0.61 | 0.68 | 0.79 | **0.85** |

**Table 2**: **Using feature representation method 2**. Cross-validation accuracy of different classifiers, with different portions of top features to be selected

The result shows that

1. SVM obtained the best accuracy among the four classifiers.
2. Reducing feature space to 10% of original feature dimension does significantly reduce the cross-validation accuracy.
3. Method 2 of feature representation (IDF computed based on category) is better than method 1 (IDF computed based on user)

Another experiment is to select the most important parameter for SVM (C value). The following chart shows the cross validation results (accuracy and F1 score) with C = [0.01, 0.1, 1, 10, 100] on the training set. It is concluded that the parameter C=1 is among the best options.




Based on the observations above, I decided to use SVM (C=1), with the reduced feature space (90% of the original) to train the model on the training set and use the model on the test set. Using this reduced feature space helps to reduce the complexity of SVM model given the small training data. However, I decided not to reduce a large portion of features, because using too small feature set corresponds to using a very small set of terms in the dictionary when representing features. Given a new user, it is likely that this small set of terms do not appear in the user tweets, hence, creating a feature vector with all 0 values. Hence, it may not be a good idea to significantly reduce the feature dimension.

# 4. Testing Results

The vocabulary learned from the training set is applied on the tweets of the test users to generate feature vectors for these users. Next, SVM model trained on the training set is applied on the test set to evaluate the model performance (feature vectors of both sets are reduced to 90% as mentioned above).

The result is
Accuracy, average f1_score (among all classes) on the test set, including 153 users is: 0.87, 0.85

# 5. Finding the best terms for each category

To gain more insights to the problem, it is interesting to see the terms that appear often in one category but not in any other categories. This is done using the feature representation of method 2, i.e., creating one feature vector for each category (by aggregating all tweets in that category), and finding the features with highest values in each category  and zero values in other categories.

**Results:**
Top 10 terms in each category
Journalist
['gophers' 'grog' 'slayer' 'gopher' 'mascara' 'supremacists' 'skincare'
 'timeout' 'palette' 'sighs']
Politician
['unionist' 'childcare' 'unison' 'charing' 'seminars' 'hustings'
 'goldsmith' 'canvass' 'leafleting' 'sturgeons']
Trader
['bearish' 'adapter' 'molt' 'currencies' 'premarket' 'lithium' 'divergence'
 'equities' 'pullback' 'rectangular']

This result does make sense, for example, in Politician tweets, we found that unionist, childcare, are best representative terms,  while in Trader tweets, premarket or equities are the best terms

# 6. Source code

The code can be run as follow (use python3)
python3 user_analysis.py -m [mode] -u [user_id]
- Possible values for **mode** are:
    + **download** (to download tweet data from user dataset)
    + **feature** (to extract features by both method 1 and 2 above and save to pickle files)
    + **model** (for model and feature selection, using method 2 by default)
    + **test** (train on the training set and evaluate on the test set)
    + **predict** (to predict the probability of each class for a user)
- user_id: the id of the user to be predicted

**Examples:**

| | |
|---|---|
| To download tweet datas: | Python3 user_analysis.py -m download |
| To extract features: | Python3 user_analysis.py -m feature |
| To perform model selection: | Python3 user_analysis.py -m model |
| To evaluate the model on the test dataset: | Python3 user_analysis.py -m test |
| To predict the class of a user with id: | Python3 user_analysis.py -m predict -u 4872354465 |

# 7. Potential Improvements

If larger dataset is available, the following improvements should be considered:

1. Extract bi-grams or n-grams from the tweets. Large dataset are essential to extract meaningful n-grams because this method will significantly increase the feature dimension.
2. Consider re-current neural network for language modeling (which is available in Tensorflow)
3. Use unicode encoding. We may obtain more useful information with more words extracted from tweets.

**Reference**

http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction