# CS918 NLP: Exercise Two - Twitter Sentiment Classifier

Haiming Zhao (individual submission)

March 2016

## 1 Overview of Implementation

This project built a sentiment classifier for Twitter, which involve extracting and transforming a number of features from text and run a classifier on the weighted combinations of features.

Packages needed to run this project are **nltk**, **sklearn**, **re**, and **numpy**.

A number of sub packages of **nltk** is used and need to be installed by nltk.download() :

- nltk.corpus.stopwords
- nltk.corpus.sentiwordnet
- nltk.corpus. opinion_lexicon
- nltk.tokenize.TweetTokenizer (A custom version of this Class is supplied so the code can be run without installing this package unless TweetTokenizer is changed to the original implementation)

The tokenizer used in this project is a slightly mortified version of `nltk.tokenize.casual.TweetTokenizer`. The package is stored as *custom_tokenize.py* which added Eastern/Japanese Emoticon regex support to `TweetTokenizer`.

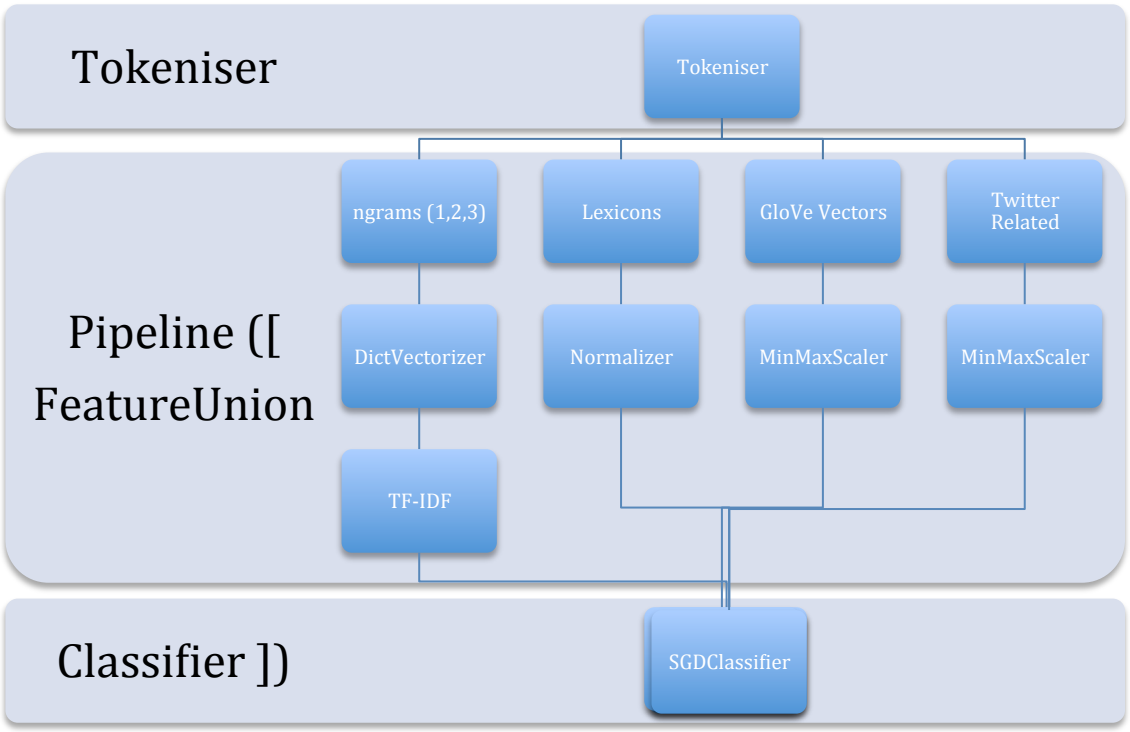
A number of lexicon/word vector data are used in this project, some are installed with **nltk**, some are downloaded, and some are modified to suit the need for this project. All lexicons are included in the submission except the one from **nltk** which can be installed using *nltk.download()* :

- **SentiWordnet** Lexicon included in nltk
- **Liu and Hu** Opinion Lexicon included in nltk
- **Emoticon** Sentiment Lexicons by Hogenboom et.al and added additional emoticons
- **Sentiment140** Lexicon and **NRC** Hashtag Lexicon by Saif M. Mohammad et.al, unchanged
- **GloVe** Twitter pre-trained word vectors, use only 25d and cleaned to exclude non English words

Two packages are implemented in this project:

- **PipelineRunner**.py: which wraps the tokeniser, feature extractor and classifier pipeline to perform training and testing on given training and test set specifically for task A and B. So that the pipeline can be run to tokenise, extract features, train, and test in one function, which looks clean and prints nicely formatted output in iPython Notebooks. Functions to calculate f_measure and macro f_measure are included in this module. The runAll functions will first preform 10-fold cross validation on training set, print the accuracy for each fold and the total confusion matrix. F-measures for each class and macro f-measures of positive and negative class are printed. Then training will be performed on the whole training set and test on the dev set. Then the confusion matrix and f-measures are printed.
- **Preprocessor**.py: which contains a wrapper of the custom tokeniser and a number of sklearn Transformer classes for feature extraction:
    - class **extract_ngram**
    - class **extract_kskip_bigram**
    - class **lexicon_liuhu**
    - class **lexicon_sentiwordnet**
    - class **lexicon_emoticon**
    - class **lexicon_NRC_unigram**
    - class **lexicon_NRC_bigram**
    - class **WE_GloVe_Twitter**
    - class **extract_tweeter_related**

The classification process is implemented with sklearn package using Pipeline and Feature Union. The data file is first processed by **PipelineRunner** using the tokeniser to read the files into a list of tokens, then these tokens are feed into the sklearn Pipeline to extract feature, transform, normalised, weighted, and trained. The process can be illustrated as below:




There are 10 features used in this implementation, they are: unigram, kskip-bigram (k=1, for the best result), trigram, SentiWordnet lexicon, Liu and Hu opinion lexicon, NRC (sentiment140 and hashtag) unigram lexicons, NRC (sentiment140 and hashtag) bigram lexicons (applied on 3skip-bigram), GloVe Twitter word vectors, and Twitter related features.

In the submission, 3 IPython notebook is supplied. The most important ones are the TaskA and TaskB notebooks, which runs the described system and present the final testing results using 10 fold cross validation on the training set and test the classifier trained on the whole training set on the dev set.

The Preprocessing_Trial notebook contains some test and trial on the preprocessing methods and classes that is used during development. It can be looked at to try out the implemented tokeniser and feature extractors but it does not present any classification results.

## 2 Pre-processing and Tokenising

The nltk casual tokeniser nltk.tokenize.TweetTokenizer work quite well on separating punctuation, http URLs, and most emoticons. However it cannot work for Eastern/Japanese emoticons like ^.^, *_*. Therefore, a support for Eastern/Japanese Emoticon regex is been added to nltk's TweetTokenizer and the package is stored as *custom_tokenize.py.*

A vector methods called **tokenise** is implemented in *Preprocessor.py* to use the customised TweetTokenizer to tokenised tweets. The method include pre-processing to filter out

punctuation except for '!' and '?', and transform URLs, at users, and numbers to 'LINK', 'AT_USER' and 'NUMBER'. All words except all uppercase words are transformed to lowercase.

The method includes arguments to toggle lemmatization, which can be turned for performance. The method also include arguments to extract the windows of words for Task A, and adding more words to the windows, can improve the performance for Task A. The final implementation for Task A includes one word before and after the windows.

# 3 Feature Extractions

## 3.1 N-gram

Transformer class **extract_ngram** in *Preprocessor.py* is implement to extract a list of n-m grams from the given token list. Class **extract_kskip_bigram** can extract k-skip bigrams. In the pipeline, we use unigram, 1-skip bigram, and trigram separately so that they can be weighted separately. The ngram features need to be vectorise using DictVectorizer in the pipeline to transform into feature vectors, as implemented in the Task A and B notebooks.

## 3.2 Lexicon

### 3.2.1 Hu and Liu opinion lexicon

The Hu and Liu opinion lexicon contains set of words that are positive and set of words that are negative. The data used in this project is from nltk corpus and can be downloaded for nltk using nltk.download(). Transformer class **lexicon_liuhu** is implemented to extract the count for positive words and negative words in a tweet.

### 3.2.2 SentiWordNet Lexicon

The SentiWordNet lexicon used in this project is from nltk. NLTK readers can then be used to look up sentiment scores including positive, negative, and objective scores. SentiWordNet returns a list of definition for each word and only some definition has a sentiment score so we only take the maximum of each score.

A way of calculating a fix score for a Tweet is to get the sum of positive score and minus the sum of negative score. In our implementation in transformer class **lexicon_sentiwordnet** provides option to extract sum of all scores, or sum of positive minus sum of negative scores.

### 3.2.3 Emoticon Sentiment Lexicon

This emoticon sentiment lexicon was created by: Hogenboom et.al. Each emoticon has a sentiment score of "either -1 (negative), 0 (neutral), or 1 (positive)".

Source of the lexicon:
http://people.few.eur.nl/hogenboom/files/EmoticonSentimentLexicon.zip

The paper related to this lexicon can be found at:
http://people.few.eur.nl/frasincar/papers/JWE2015/jwe2015.pdf

This lexicon has been modified to add more variety to the emoticon to capture more emoticon in the dataset. The added lexicons preserved sentiment score of same emoticon of different form.

Here is a list showing the emoticons that has added to the lexicon:

| Contained in original lexicon | Manually Added | Score (preserve the same score in lexicon) |
|---|---|---|
| (-.-) | -.- | -1 |
| (^.^) | ^.^ | +1 |
| _(._.)_ | ._. | 1 |
| -_- | -__- -___- .___. =.= | -1 |
| (~_~) | ~_~ ~.~ | -1 |
| :( :[ :'( :-[ | ): ); ]: )': >:-[ | -1 |
| =) | (= >=) | +1 |
| ;) | (; | 0 |
| :\| | \|: \|= | -1 |
| (>_<) | >_< | -1 |
| :/ | ;/ /: | -1 |
| o.0 | o.o O_o o_o | -1 |
| (*_*) | *_* *__* *___* *.* | -1 |

Transformer class **lexicon_emoticon** is implemented to use the customised data file and extract sum of the emoticon score and count of emoticons.

### 3.2.4 NRC Sentiment140 Lexicons

Source of lexicon: http://www.saifmohammad.com/WebPages/ResearchInterests.html

Both NRC Hashtag Sentiment Lexicon (version 0.1) and Sentiment140 Lexicon (version 0.1) are used and we only use the unigram and bigram lexicons.

Transformer class **lexicon_NRC_unigram** and **lexicon_NRC_bigram** is implemented to use the data file and extract unigram and bigram lexicon and return the sum, min and max of the lexicon values in a Tweet. In the final implementation, we feed a k-skip bigram to the NRC bigram lexicon feature.

### 3.3 Word Embedding: Pre-trained GloVe word vector from Twitter

The Word Embedding feature make use of the pre-train Twitter word vector from the GloVe Project: http://nlp.stanford.edu/projects/glove/

The word vector are trained from "2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors", the file is 1.42 GB download.

This project only take the 25 dimensions and cleaned it to exclude all the non-English words so that the file size is smaller and faster to run. The code used for creating the cleaned file can be viewed in Preprocessing_Trial notebook and can only be run if glove.twitter.27B.25d.txt is downloaded.

Transformer class **WE_GloVe_Twitter** is implemented to use the cleaned data file and extract the average, min, and max value for each dimension of the word vector for each word in Tweets.

### 3.4 Twitter Related Features

Transformer class **extract_tweeter_related** is implemented extract twitter related features including count of words, exclamation mark, question mark, URLs, @users, numbers, all uppercase words, and hashtags.

## 4 Classifier

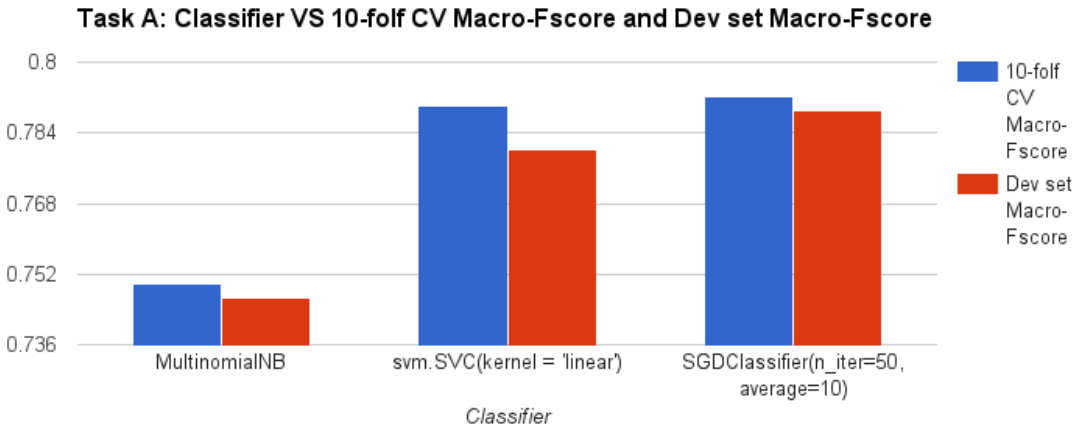
The final classifier used it is SGDClassifier from sklearn set to (n_iter=50,average=10).

In the baseline attempt, Multinomial Naïve Bayes is used to obtain a score to compare with other classifiers.

Also tried are nltk.NaiveBayseClassifier, Sklearn.svm.SVC(kernel='linear') also with *rbf, poly* kernels. The linear kernel is faster than other kernels and perform better while the *rbf, poly* kernels may require further tweaking to reach better performance. The performance Naïve Bayes is fastest but result not as good as the svm classes.

Other attempts are RandomForest and knn, which perform somewhat similar to Naïve Bayes. OneVsRestClassifier from sklearn is also tried to fit one classifier per class but it does not show advantages.

SGDClassifier performs very similar to SVC and it is a lot faster than SVC, as shown in the below charts. Therefore, the final implementation uses SGDClassifier. However, because SGD uses gradient decent, the test result varies slightly between each runs. The n_iter and average parameter has been increased to tried to overcome this but because of the nature of gradient decent the result can still change for about 0.001~0.003.



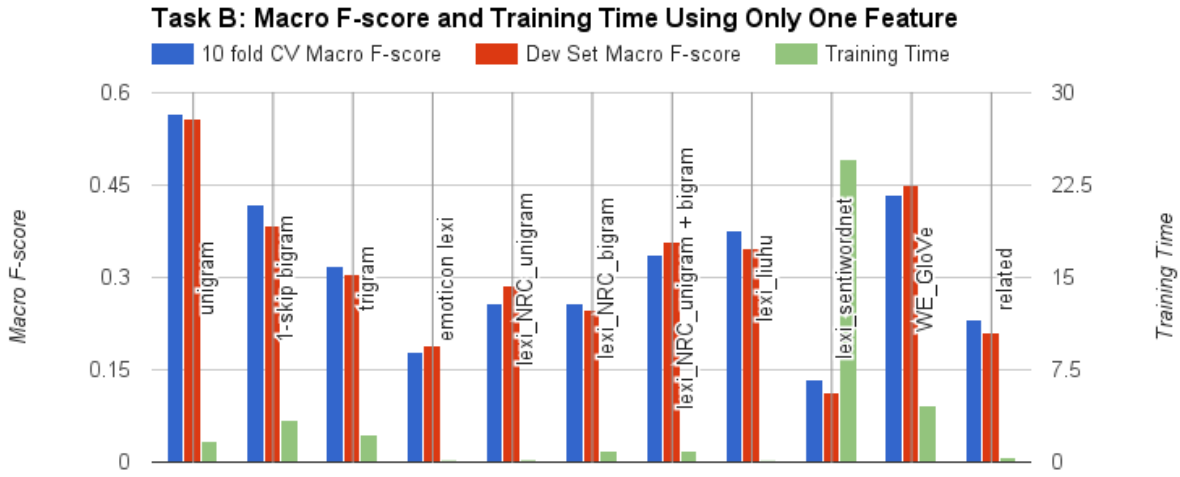


## 5 Performance Turning

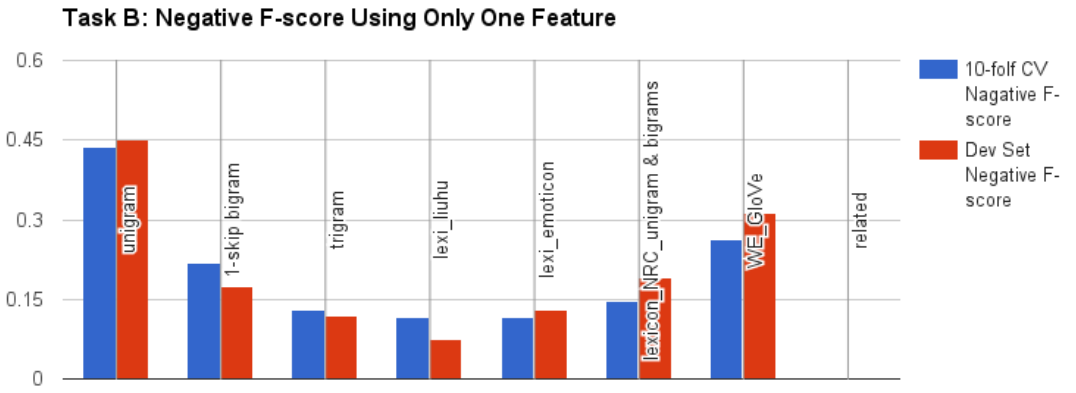
To test and turn the classifier we perform 2 test: 10-fold cross validation on training set, and then train on the whole training set and test on dev set. The performance is tweaked for better performance on dev set. After applying the classifier trained on the whole training set on dev set, the classifier can then be used to classify the instances in the test set and write to result file in the ***result/*** folder.

## 5.1 Contribution of Different Features

We implemented a number of features extractors in the project. A test can be perform to evaluate how useful is each feature by running the classifier with only one feature at a time. The below chart observes the Macro F-score from testing with 10–fold cross validation and dev set. We also observed the time spent to change the classifier using each feature extractor individually. We found that using the SentiWordNet lexicon takes very long for Task B while it did not seem to contribute much to the classification. Therefore, in the final classifier we excluded the SentiWordNet lexicon feature. Overall, N-gram contributes the most to classification and GloVe Twitter Word Vector performs well as well. Among the lexicons, LiuHu lexicon and the NRC lexicon seems to be the most useful.




The observed performance for negative class in Task B is low (also see Section 6 Evaluation). We here examine the F-score for Negative class using only one feature at a time. The following chart shows that unigram still contribute the most to classifying Negative. LiuHu lexicon performs worst than emoticon lexicons, contradict to the result above for Macro F-score. Word Embedding still provide a lot of information while Twitter related features does not provide any input for classifying negative Tweets. However, Using only Twitter related features give on dev set gives Positive f-score of 0.421182 and Neutral f-measure: 0.628537.



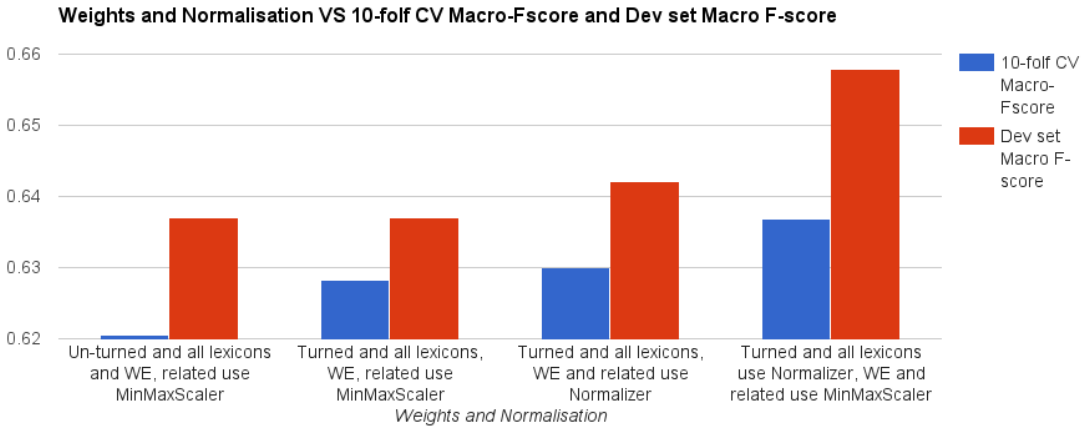

The above experiments give us an idea about how much information each feature can pick up and also help us to turn the weights accordingly.

## 5.2 Normalisation

The ngram features are transformed using sklean's DictVectorizer into feature vectors and then calculate tf-idf using sklean's TfidfTransformer. An attempt has been made to implement tf-idf using the normalisation by maximum raw frequency. However, the performance was not as good as using tf-df transformer from sklearn, which uses l2 normalisation. In addition, the implementation was not as neat so the final implementation did not include tf-idf calculation from scratch.

The lexicon features are normalised using sklearn.preprocessing.Normalizer(), which proves to perform better than using MinMaxScaler(). The word vector and Twitter related feature use MinMaxScaler() to normalise because provide similar or better performance and slightly faster.

All the features are then weighted. The weights are passed to the *transformer_weights* argument of FeatureUnion and have been turned for the best performance. Interestingly, changing the weights to values that are somewhat similar to the value of their individual contributed f-score shown in 5.1 gives the best performance for Task B. Using the same weights in Task A proved to achieve the best performance as well.




The final weights used for both Task A and Task B shown below: (`'lexi_sentiwordnet'` feature is excluded in TaskB )

```
weights = {
   'unigram_tfidf':       1.0,
   'kskip_bigram_tfidf':  0.4,
   'trigram_tfidf':       0.5,
   'lexi_sentiwordnet':    0.3,
   'lexi_liuhu' :         0.5,
   'lexi_emoticon':       0.2,
   'lexicon_NRC_unigram': 0.2,
   'lexicon_NRC_bigram' : 0.2,
   'WE_GloVe'   :         0.4,
   'related' :            0.2
}
```

# 6 Evaluation

The evaluation results of the final setting are shown below:

## Task A:

**10 fold validation on training set:**

```
test range: [0, 765] accuracy: 0.825065274151
test range: [766, 1531] accuracy: 0.828981723238
test range: [1532, 2297] accuracy: 0.860313315927
test range: [2298, 3063] accuracy: 0.83681462141
test range: [3064, 3828] accuracy: 0.870588235294
test range: [3829, 4593] accuracy: 0.806535947712
test range: [4594, 5358] accuracy: 0.786928104575
test range: [5359, 6123] accuracy: 0.822222222222
```

```
test range: [6124, 6888] accuracy: 0.833986928105
test range: [6889, 7653] accuracy: 0.864052287582
          |    n         p |
          |    e    n    o |
          |    g    e    s |
          |    a    u    i |
          |    t    t    t |
          |    i    r    i |
          |    v    a    v |
          |    e    l    e |
----------+----------------+
negative |<2009>  16  472 |
 neutral |  107  <51> 224 |
positive |  421   34<4320>|
----------+----------------+
(row = reference; col = test)

positive f-measure: 0.882443
neutral f-measure: 0.211180
negative f-measure: 0.798172
Macro f-measure: 0.840308
```

**classifier on whole training set and test on dev set:**

```
~~ Training classifier took (sec):
9.104012
          |   n       p |
          |   e   n   o |
          |   g   e   s |
          |   a   u   i |
          |   t   t   t |
          |   i   r   i |
          |   v   a   v |
          |   e   l   e |
----------+-------------+
negative |<269>  .  73 |
 neutral |  14  <.> 31 |
positive |  45   2<494>|
----------+-------------+
(row = reference; col = test)

positive f-measure: 0.867428
neutral f-measure: 0.000000
negative f-measure: 0.802985
Macro f-measure: 0.835206
```

**Comments:**

The accuracy for fold 6-7 is a bit lower. The f-score of positive and negative class are both high with negative being a bit lower than positive. The f-score for neutral is low but there are not many neutral instances in the training and dev set. The macro f-score obtained from both test are very similar. The macro f-score obtain from both tests are 0.8403 and 0.8352, respectively, with an average of 0.8378. (or 83.78 in percentage)

## Task B

**10 fold validation on training set:**

```
test range: [0, 798] accuracy: 0.670838548185
test range: [799, 1597] accuracy: 0.730913642053
test range: [1598, 2396] accuracy: 0.742177722153
test range: [2397, 3195] accuracy: 0.73717146433
test range: [3196, 3994] accuracy: 0.743429286608
test range: [3995, 4793] accuracy: 0.734668335419
test range: [4794, 5591] accuracy: 0.689223057644
test range: [5592, 6389] accuracy: 0.74686716792
test range: [6390, 7187] accuracy: 0.709273182957
test range: [7188, 7985] accuracy: 0.713032581454
          |    n         p |
          |    e    n    o |
          |    g    e    s |
          |    a    u    i |
          |    t    t    t |
          |    i    r    i |
          |    v    a    v |
          |    e    l    e |
----------+-----------------+
negative | <552> 434  165 |
 neutral |  198<3101> 563 |
positive |  116  746<2111>|
----------+-----------------+
(row = reference; col = test)

positive f-measure: 0.726428
neutral f-measure: 0.761636
negative f-measure: 0.547348
Macro f-measure: 0.636888
```

**classifier on whole training set and test on dev set:**

```
~~ Training classifier took (sec):
10.037573
```

```
          |   n       p |
          |   e   n   o |
          |   g   e   s |
          |   a   u   i |
          |   t   t   t |
          |   i   r   i |
          |   v   a   v |
          |   e   l   e |
---------+-------------+
negative |<147> 84  44 |
 neutral |  52<461>105 |
positive |  17 112<352>|
---------+-------------+
(row = reference; col = test)

positive f-measure: 0.716904
neutral f-measure: 0.723137
negative f-measure: 0.598778
Macro f-measure: 0.657841
```

**Comments:**

The accuracy for fold 1 and 7 is lower than other folds. The f-score of positive and neutral class are both high and negative class has a low f-score, which make the macro f-score lower. Many positive and negative Tweets are classified as neutral.

The macro f-score obtained from both test are similar with dev set perform better than 10-fold cross validation. This is probably because there are more data used for training the classifier. The macro f-score obtain from both tests are 0.6369 and 0.6578, respectively, with an average of 0.6474 (or 64.74 in percentage)

# 7 Discussion

Observe the performance for Task B, it shows that our classifier had a hard time classifying negative tweets. The F-score for negative are below 0.6, which is a lot lower compare to that of positive. There are s lot of false negatives and false positives.

We inspect the false negatives and false positives for negative class. We found that the classifier is confused does not work well on picking up the semantic meaning of the overall sentences. Here are some examples; the tuples are in format (line_number, actual label, classified label, tweet)

Example false negative:

```
(14,   'negative',   'neutral',   '@prodnose is this one of your little
jokes like Elvis playing at the Marquee  next Tuesday?\n')
```

The real class is neutral. However, the words are not very negative, from reading the sentence it sounds a bit sarcastic and insulting but it cannot imply from the words itself, but from the semantic meaning.

Example false positives:

```
(617,    'neutral',    'negative',    'My Pain may be the reason for
somebody,s laugh But My laugh must never be a reason for somebody,s
pain #Charlie_Chaplin')
```

The real class is neutral but if we read the sentences, there are a some strong negative words, “pain”, “never”, “but”. But the sentiment is neutral. This type of text is hard to classifier without implying the actual meaning of the sentences.

# 8 Notes on running the code

All the notebook code in the IPython notebooks are runnable. The notebook should provide enough code to test the project. The features and classifier can be commented and uncommented to change around the feature can classifier used. The arguments can be changed as well. Data files are included except the one from **nltk**, which are mentioned in the beginning of the report.

All the code are changed to read-only to prevent further changing. However, the file permission can be changed if really need to.

The test results are save in txt files in the ***result/*** folder with the classification result written out along with Tweet id, user id etc.

# 9 Conclusion

The project developed Twitter classification technique using both trained and pre-trained features. The pre-trained features improves the performance and saves development time, they are do not over fit the training set because they are trained on much larger datasets. Although there are a lot of feature used, this project did not cover some of the possible features like POS tagging. The results produced, based solely on 10-fold cross validation and development are close to the top scores of SemEval 2015. The modules implemented are designed to be extendable and easy to use has potential in further application.