

**Name: Ashish Singh Tomar**

**Email: ast2124@columbia.edu**

**UNI: ast2124**

**NLP Fall 2010**

**HW Assignment #2 – MOVIE REVIEW CLASSIFICATION**

**Programming Language: Java**

**Tools Used: Weka, Stanford Parser**

### **I. RUN TIME ESTIMATE**

The run time estimate for feature extractor code is dependent on the size of test file. It takes a lot of time because it uses Stanford parser to get the POS tags.

It would take on an average of 7- 7.5 hours for running on a test file consisting of 1000 reviews like those in the training file. A better way to run the experiments if the same test file will be used for all the classification tasks will be to use `mod_run_only_test.sh`. (See details about this script in the scripts section below.)

But if you want to use separate test files for each task you should use the scripts mentioned in the RUN section below.

### **II. FINAL OUTPUT-**

The final classified output can be found in the file that should be given as a parameter to the scripts.

For example –

If you executed

```
./author.sh author.model movie-corpus-test.txt classified-author.txt
```

The final classified output will be in `classified-author.txt`(third argument) file.

### **III. Steps to run the Code—**

#### **1. Make the Scripts Executable**

First of all the scripts need to be made executable.

Type the following on command prompt and from the folder ast2124\_hw2

```
$chmod +x *.sh
```

## 2. Compiling the Code

Type make on the command-

```
$make
```

It will use ant script and compile the Java code for the feature extractor.

## 3. RUN

To run the feature extractor and weka on test file you can use one of the following 5 scripts-

a. For Author classification

```
$/author.sh author.model <TestFile> <OutputFile>
```

**The output will go into <OutputFile>.**

b. For binary rating for same reviewers classification

```
$/binaryRatingSameUsers.sh binaryRatingSameUsers.model <TestFile> <OutputFile>
```

**The output will go into <OutputFile>.**

c. For binary rating for different reviewers classification

```
$/binaryRatingDiffUsers.sh binaryRatingDiffUsers.model <TestFile> <OutputFile>
```

**The output will go into <OutputFile>.**

d. For 4 star rating for same reviewers classification

```
$/starRatingSameUsers.sh starRatingSameUsers.model <TestFile> <OutputFile>
```

**The output will go into <OutputFile>.**

- e. For 4 star rating for different reviewers classification

`./starRatingDiffUsers.sh starRatingDiffUsers.model <TestFile> <OutputFile>`

**The output will go into <OutputFile>.**

#### **4. Models**

The models files can be found in the model folder.

- a. author.model represents the classifier for Reviewer Author classification.
- b. binaryRatingSameUsers.model represents the classifier for positive/negative classification for the same reviewers as in training set.
- c. binaryRatingDiffUsers.model represents the classifier the classifier for positive/negative classification or the different reviewers as in training set.
- d. starRatingSameUsers.model represents the classifier for 4 star classification for the same reviewers as in training set.
- e. starRatingDiffUsers.model represents the classifier the classifier for s star classification for the different reviewers as in training set.

#### **5. Arff Files**

The arff files can be found in the arff folder.

- a. author.arff represents the training arff file for Reviewer Author classification.
- b. binaryRatingSameUsers.arff represents the training arff file for positive/negative classification for the same reviewers as in training set.
- c. binaryRatingDiffUsers.arff represents the training arff file for positive/negative classification or the different reviewers as in training set.
- d. starRatingSameUsers.arff represents the training arff file for 4 star classification for the same reviewers as in training set.
- e. starRatingDiffUsers.arff represents the training arff file for 4 star classification for the different reviewers as in training set.

## 6. Weka ML Algorithm Used

All five of the models were generated using SMO function classifier in Weka and the parameters used were –

```
weka.classifiers.functions.SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K  
"weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
```

## 7. Limitations

It takes a lot of time to extract the features because it uses Stanford parser and it takes approximately 7-7.5 hours for processing a test file consisting of 1000 reviews.

## 8. Success

The classifier for Reviewer classification performs pretty well as its accuracy is around 94%.

The performance for Binary classification was around 79% and 4 star classification accuracy was around 58%.

## 9. Some extra things that I could have could have tried-

I could have used Typed Dependency to improve the accuracy of my classifiers.

I could have used higher n gram features.

## 10. Files Submitted

```
+---src  
    +---code  
        ---FileReader.java  
        ---CountGenerator.java  
        ---Document.java  
        ---ParseFileReader.java  
        ---POSFileReader.java
```

---FinalOutput.java

+---data

- stopwords.txt
- poswords.txt
- neutwords.txt
- negwords.txt
- multiFeatures.txt
- multiFeaturesBigram.txt
- binaryFeatures.txt
- binaryFeaturesBigram.txt
- reviewerfeatures.txt
- reviewerFeaturesBigram.txt
- parsedCorpus.txt

+---model

- starRatingSameUsers.model
- starRatingDiffUsers.model
- binaryRatingSameUsers.model
- binaryRatingDiffUsers.model
- author.model

+---arffFiles

- authorTrain.arff
- binaryRatingSameUsersTrain.arff
- binaryRatingDiffUsersTrain.arff
- starRatingSameUsersTrain.arff
- starRatingDiffUsersTrain.arff

---run\_weka.sh

---run\_weka\_on\_test.sh

---author.sh

---binaryRatingSameUsers.sh

---binaryRatingDiffUsers.sh

---starRatingSameUsers.sh

---starRatingDiffUsers.sh

---mod\_run\_only\_train.sh

---mod\_run\_only\_test.sh

---mod\_run\_everything.sh

---Makefile

---build.xml

---README.pdf

---WriteupAss2NLP.pdf

## 11. Source Code files

### 1. FileReader.java

This is the main class for feature extraction containing the main method. The file reads the movie-corpus input file line by line and uses CountGenerator class to calculate the values for the features. After that the file creates the final output arff files for classification based on the arraylist of features that it reads from the files from the data folder submitted with the code. It uses `printDocListForReviewer` and `printDocListForClassification` methods to write the output arff files. `readFeaturesFromFile` method is used to read the list of selected predictive features from previously trained data and initialize the data structures. `printDocListForReviewer` method writes the extracted features into the arff files for reviewer classification task. `printDocListForClassification` method writes the extracted features into the arff files for binary and 4 star classification tasks.

### 2. Document.java

This class is used as a data structure to hold the records of feature values for each review from the input file. An arraylist of Document class is used to store all the records.

### 3. CountGenerator.java

This class is used by the main class to calculate feature values for each review. The constructor initializes the arraylists with a set of positive, negative and neutral words from the files in the data folder submitted with the code. The `getCount` method is called from the main class for each review and this method calculates all the features and stores all the values in the Document class object in the form of hashtables and other numeric data types. This function also gets the POS tags with the help of ParseFileReader class and calculates the POS tags features using POSFileReader class. It stores the word counts in various data structures using helper functions. It stores counts for positive, negative, neutral, superlatives words, etc. It also stores the bigram counts and Document frequency for each term. These data structures are used to calculate term frequency, document frequency and tf-idf values later on while creating the arff files. The method also tries to

infer the words like not, isn't, etc that negates the meaning of a word like “not good” when trying to get the count for positive or negative words. The code takes care of such words within a distance of 3 or less from the current word. If it finds such a word it updates the values for the opposite class (For Ex not bad will be positive instead of negative).

#### 4. ParseFileReader.java

This file is called by CountGenerator and main class using `parseTheLine` method to get the POS tags for a given line using Stanford Parser.

#### 5. POSFileReader.java

This class is used to calculate the POS features for each review from the POS tagged lines. This class has several functions like `parseLineforNN`, `parseLineforJJ()`, `parseLineforRB()`, `parseLineforVB()`, `parseLineforDT()`, `parseLineforNNpair()` to get the counts for the nouns, adjectives, adverbs, verbs, determiners, etc. and the values are stored in the respective Document class object.

#### 6. FinalOutput.java

This class is used to convert the final output from weka on test file to the text xml kind of format like the input test file that is required in the assignment. It uses `readAndWrite` method to read the output file from Weka and converts into XML kind of format.

## 12. Data Files

1. stopwords.txt contains a list of stop words.
2. poswords.txt contains a list of positive words got from MPQA specialized lexicon and some form the training data.
3. neutwords.txt contains a list of neutral words got from MPQA specialized lexicon and some form the training data.

4. negwords.txt contains a list of negative words got from MPQA specialized lexicon and some from the training data.
5. multiFeatures.txt contains a list of features for 4 star rating classifier that came out to be predictive during previous experiments with training data.
6. multiFeaturesBigram.txt contains a list of bigram features for 4 star rating classifier that came out to be predictive during previous experiments with training data.
7. binaryFeatures.txt contains a list of features for binary rating classifier that came out to be predictive during previous experiments with training data.
8. binaryFeaturesBigram.txt contains a list of bigram features for binary rating classifier that came out to be predictive during previous experiments with training data.
9. reviewerfeatures.txt contains a list of features for reviewer classifier that came out to be predictive during previous experiments with training data.
10. reviewerFeaturesBigram.txt contains a list of bigram features for reviewer classifier that came out to be predictive during previous experiments with training data.
11. parsedCorpus.txt is POS tagged file from Stanford parser for movie-corpus.txt that was used as our training data. Since it takes a long time to get POS tags for large set of documents. This file was stored so that it will not have to re run POS tags for training data.

### **13. Other Files-**

1. Makefile is used to compile the feature extraction code using build.xml and ant scripting.
2. build.xml uses ant to compile the code.
3. README.pdf – This file explains about how to run and compile the code and some information about the code, models and arff files submitted.
4. WriteupAss2NLP.pdf- this file contains the information about the experiments that were tried for getting the best results possible for all the classification tasks.



## 14. Scripts—

1. `author.sh` -This script is used for running the test on a model for reviewer classification.

```
./author.sh author.model <TestFile> <OutputFile>
```

The final results will be written to <OutputFile>.

2. `binaryRatingSameUsers.sh` -This script is used for running the test on a model for binary classification for same reviewers as in training set.

```
./binaryRatingSameUsers.sh      binaryRatingSameUsers.model      <TestFile>  
<OutputFile>
```

The final results will be written to <OutputFile>.

3. `binaryRatingDiffUsers.sh` -This script is used for running the test on a model for binary classification for different reviewers as in training set.

```
./binaryRatingDiffUsers.sh binaryRatingDiffUsers.model <TestFile> <OutputFile>
```

The final results will be written to <OutputFile>.

4. `starRatingSameUsers.sh` -This script is used for running the test on a model for 4 star classification for same reviewers as in training set.

```
./starRatingSameUsers.sh starRatingSameUsers.model <TestFile> <OutputFile>
```

The final results will be written to <OutputFile>.

5. `starRatingDiffUsers.sh` -This script is used for running the test on a model for 4 star classification for different reviewers as in training set.

```
./starRatingDiffUsers.sh starRatingDiffUsers.model <TestFile> <OutputFile>
```

The final results will be written to <OutputFile>.

6. `mod_run_only_train.sh` -This script can be used to train and create new models for all types of classifiers. It will take training data as input.

```
./run_only_train.sh movie-corpus.txt
```

It creates the models in the current directory and names them as –

```
---starRatingSameUsers.model  
---starRatingDiffUsers.model  
---binaryRatingSameUsers.model  
---binaryRatingDiffUsers.model  
---author.model
```

7. `mod_run_only_test.sh` -This script can be used to test the models for all types of classifiers. It will take test data file as input. It assumes the models are named

---starRatingSameUsers.model  
---starRatingDiffUsers.model  
---binaryRatingSameUsers.model  
---binaryRatingDiffUsers.model  
---author.model  
and they are in the current directory.

`./run_only_test.sh movie-corpus-test.txt`

It generates the final classified files for the entire 5 classifiers task. The names for the final output files generated will be –

classified-starRatingSameUsersTest.txt  
classified-starRatingDiffUsersTest.txt  
classified-binaryRatingSameUsersTest.txt  
classified-binaryRatingDiffUsersTest.txt  
classified-authorTest.txt

8. `mod_run_everything.sh` -This script takes the training and test corpus as input and creates the models and tests them on test file and outputs the classified files for all types of classifiers.

`./run_everything.sh movie-corpus.txt movie-corpus-test.txt`

It generates the final classified files for all 5 classifiers task. The names for the final output files generated will be –

classified-starRatingSameUsersTest.txt  
classified-starRatingDiffUsersTest.txt  
classified-binaryRatingSameUsersTest.txt  
classified-binaryRatingDiffUsersTest.txt  
classified-authorTest.txt

9. `run_weka.sh` -This script is used by other scripts internally to create the model using training arff file and Weka's SMO ML algorithm. It can be run like this -

`./run_weka.sh starRatingSameUsersTrain.arff starRatingSameUsers.model`  
`outt_starRatingSameUsers.txt`

where `starRatingSameUsersTrain.arff` is the input arff file

`starRatingSameUsers.model` will be the name of model that it will generate

and outt\_starRatingSameUsers.txt will get the output of the accuracy and other parameters that Weka will print.

10. run\_weka\_on\_test.sh -This script is also used internally to run Weka on already created model and a test file and it generates the final output in XML type format. It can be executed like this-

```
./run_weka_on_test.sh starRatingSameUsers.model starRatingSameUsersTest.arff  
out_starRatingSameUsersTest.arff multi <Input Test File> <Output File>
```

where <Input Test File> here is the name of input text test file. (It is like movie-corpus.txt without star and reviewer tags) and <Output File> is the output file that will be generated.

## 15. To run just the feature extractor without the Weka code

You can type --

```
java -cp bin/./home/cs4705/stanford-parser-2010-08-20/stanford-parser-2010-08-20.jar -  
Xmx2048m code/FileReader Train movie-corpus.txt data/ 6
```

This will generate training arff files for all the classification tasks.

To understand how to run it you need to understand the use of parameters here.

It takes 4 parameters-

The first one is Train or Test.

The second one is Input file.

The third one is the path for the directory that contains all the input files for positive, negative words, feature lists for star rating, feature lists for binary rating etc. It should be the data/ folder that is submitted along with the code.

The last parameter is integer varying from 1 to 6.

If the value is 6 it generates the arff files for all the classification types.

If the value is 1 it generates the arff file only for the binary rating for same reviewers.

If the value is 2 it generates the arff file only for the binary rating for different reviewers.

If the value is 3 it generates the arff file only for the 4 star rating for same reviewers.

If the value is 4 it generates the arff file only for the 4 star rating for different reviewers.

If the value is 5 it generates the arff file only for the reviewer classification.

For example –

```
java -cp bin/./home/cs4705/stanford-parser-2010-08-20/stanford-parser-2010-08-20.jar -Xmx2048m code/FileReader Test movie-corpus-test.txt data/ 2
```

This will generate test arff file only for the binary rating for different reviewers.