# Project Report

**Tweet based gender detection**

**Shashank Singh**

**Tushar Sinha**

**Project Description**

In this project, we will classify and predict the gender (male or female) of a particular user based on the linguistic content of the user's tweets. In our attempt to classify to the tweets we assume that there exists a significant difference between the way a male and a female writes a tweet.

We will then use different learning algorithms to predict the gender and then compare their accuracies.

We have used python and MATLAB for the purpose of our project and **our entire project is also available on GIT Hub [1]**

**Training and Test Data Set**

We have written a MATLAB script “**getTweets.m**” which acts as a **wrapper** around “**twitty**” for gathering tweets of public profiles from twitter.

“**twitty**” is an **open-source MATLAB class** available on “**matlabcentral**” for interaction with the twitter platform via its REST API v1.1. We have also used the open-source “JSON parser” (for parsing the data gathered) which is also available on “**matlabcentral**”. **[2][3]**

Once we have collected the tweets (~4000), we divide the data set into train set and test set based on different ratios.

**Data preprocessing and generation of feature set**

This is the most important step in our entire project since it forms the basis of generating the feature sets for our project. Data preprocessing, in our case, basically refers to cleaning up of data by removing unwanted characters and redundancies from the training data set. This step is important because the more clean our data would be, the better would be the feature set for our problem and eventually better accuracy. We used the following methods for cleaning up our data:

1. **Removing retweets** – Once we have all the training and test data, it is important to remove the retweets since they defeat the purpose of identifying the gender of the user based on his original tweets. The code for this is present in the python script “RemoveRetweets.py”
2. Once we get the new file from above after removing the retweets, we execute another python script “**tokenizeAndPruneTweets.py**”. This script performs the following tasks:
   1. Removes other twitter handles referenced in the tweets as they are just names and are insignificant in the task of predicting the gender of a user.
   2. Remove all kinds of urls as these are also insignificant.
   3. **Tokenization and further pruning of the training tweets** – We read all the tweets line by line and then split up these lines into different words. These words are then mapped to a text file which eventually forms our main feature set. But before mapping these words, we performing further cleaning and pruning by stripping unnecessary special characters such as “!$,:\” etc. from each word and then removing unnecessary substrings such as “\n \t” etc. This way, we are only left with proper words for our analysis purpose.

Another importance of removing these unwanted characters is that we could now merge similar words easily. For example “always!” and “always” became one word with a count of 2.

The output at the end of this method is a text file containing different words used in the tweets along with two counts. The first count is the number of distinct female users who used this word in their tweets and second count is the number of distinct male users who used this word in their tweets.

1. This is the step where we generate the final feature sets and the probability table to be used in the Naïve Bayes learning algorithm. We execute the script “**generateFeatureSetsForNgram.py**” to accomplish this task. This script takes an input argument “**threshold**”, which helps us in limiting the feature sets. The main advantage of using a threshold value is to reduce the feature sets and then chose the best feature set based on the accuracy. A “**threshold**” value of 1 would mean that we ignore all the words whose frequency is less than 1 for both male and female user. Once this script is executed, it generates four different feature sets namely:
   1. **“fs\_obsceneJunkWords\_1gm\_trng.txt”** – This contains junk words which are not used very often.
   2. “**fs\_remainingWords\_1gm\_trng.txt**” – This contains all other words (except hashtags) which are above the threshold value discussed above.
   3. **“fs\_wordsBelowThreshold\_1gm\_trng.txt” –** This contains words that are above the threshold value.
   4. **“fs\_wordsWithHashtags\_1gm\_trng.txt”** – This contains words that are hash tags.

All of the above files contain the same data i.e. the word, probability of the word given female (p(Female|word)) , probability of the word given male (p(Male|word)) and the probability of the word itself.

**P(Female|word) = Number distinct female users who used this words in their tweets**

**Total number of tweets by female users**

**P(Male|word) = Number distinct male users who used this words in their tweets**

**Total number of tweets by male users**

**Algorithms**

**Supervised Learning**

Supervised learning involves learning a model based on labelled training data. The algorithm analyzes the training data and produces an inferred function which can predict labels in unseen instances. We used the following supervised learning algorithms in our project:

1**) Naïve Bayes Algorithm**: Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

The MATLAB scripts “**genderDetectionUsingNaiveBayes.m**” and “**getNaiveBayesAccuracy.m**” are used for implementing this algorithm. These scripts internally call another python script “**MergeFiles**.py” which finally create the required training data set matrix where each row represents a tweet and each column represents the probability **p(Female|word)** and **p(Male|word)** (mentioned above) for all the words that were trained and mapped from the training tweets.

We use the following formula for predicting the gender based on Naïve Bayes approach:

\hat{y} = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} \ p(C_k) \displaystyle\prod_{i=1}^n p(x_i \vert C_k).

Here k = {1, 2} for **male** and **female** respectively and - word

We have also used **Laplacian smoothing** to smooth our data i.e. we have basically calculated the expected value of our posterior distribution. We have used the smoothing factor ***α = 0.1*** in our case.

We run this algorithm on both the test and train data and observe the accuracies.

**Observations:**

Following are the observations for the Naïve Bayes Classifier when we split the train and test data in **1:1** ratio and apply this algorithm to both data. We also vary the **“threshold”** value mentioned earlier to get the best accuracy and the best feature set.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threshold** | **Obscene junk words** | **Remaining words** | **Words below threshold** | **Words with Hashtags** |
| 0 | ~ | 93.07 | ~ | ~ |
| 1 | 52.06 | 91.29 | ~ | 60.43 |
| 5 | ~ | 66.19 | 93.55 | 51.99 |
| 10 | ~ | 59.95 | 94.31 | ~ |
| 20 | ~ | 57.06 | 94.99 | ~ |
| 50 | ~ | 52.67 | 94.65 | ~ |

**Accuracy % for Naïve Bayes with different thresholds for different feature sets (Train Data)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threshold** | **Obscene junk words** | **Remaining words** | **Words below threshold** | **Words with Hashtags** |
| 0 | ~ | 63.55 | ~ | ~ |
| 1 | 52.68 | 63.14 | ~ | 53.73 |
| 5 | ~ | 59.86 | 60.38 | 52.68 |
| 10 | ~ | 57.00 | 61.74 | ~ |
| 20 | ~ | 57.00 | 62.16 | ~ |
| 50 | ~ | 52.96 | 62.51 | ~ |

**Accuracy % for Naïve Bayes with different thresholds for different feature sets (Test Data)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threshold** | **Obscene junk words** | **Remaining words** | **Words below threshold** | **Words with Hashtags** |
| 0 | 0 | 4343 | 0 | 0 |
| 1 | 5 | 4058 | 0 | 280 |
| 5 | 0 | 339 | 4003 | 1 |
| 10 | 0 | 145 | 4198 | 0 |
| 20 | 0 | 68 | 4275 | 0 |
| 50 | 0 | 18 | 4325 | 0 |

**Number of features in different feature sets**

Following are the observations for the Naïve Bayes Classifier when we split the train and test data in **4:1** ratio and apply this algorithm to both data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threshold** | **Obscene junk words** | **Remaining words** | **Words below threshold** | **Words with Hashtags** |
| 0 | ~ | 92.96 | ~ | ~ |
| 1 | 60.06 | 91.69 | ~ | 66.58 |
| 5 | ~ | 68.77 | 93.35 | 60.06 |
| 10 | ~ | 63.25 | 94.36 | ~ |
| 20 | ~ | 60.80 | 94.09 | ~ |
| 50 | ~ | 59.93 | 93.74 | ~ |

**Accuracy % for Naïve Bayes with different thresholds for different feature sets (Train Data)**

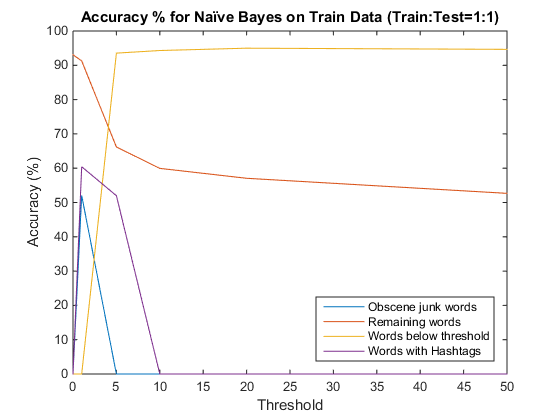
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threshold** | **Obscene junk words** | **Remaining words** | **Words below threshold** | **Words with Hashtags** |
| 0 | ~ | 94.32 | ~ | ~ |
| 1 | 52.68 | 93.04 | ~ | 76.01 |
| 5 | ~ | 75.09 | 94.69 | 72.34 |
| 10 | ~ | 73.99 | 95.05 | ~ |
| 20 | ~ | 71.79 | 95.05 | ~ |
| 50 | ~ | 71.79 | 94.69 | ~ |

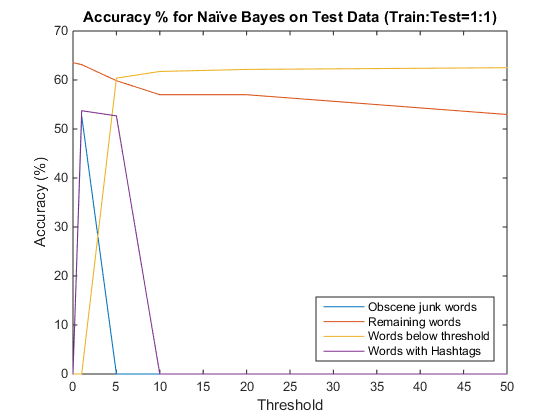
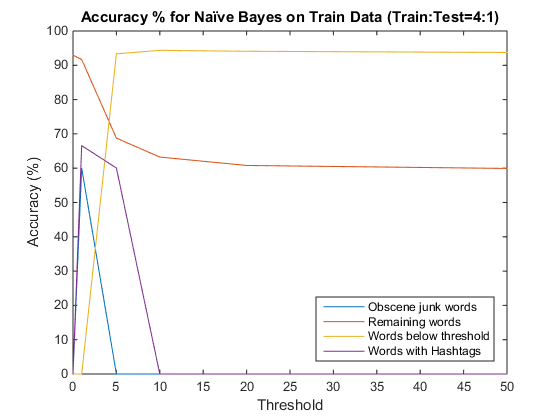
**Accuracy % for Naïve Bayes with different thresholds for different feature sets (Test Data)**

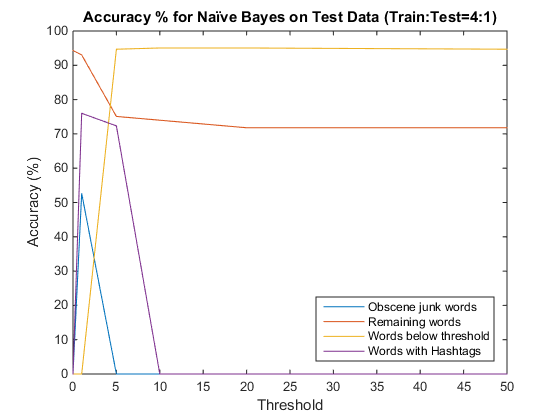
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threshold** | **Obscene junk words** | **Remaining words** | **Words below threshold** | **Words with Hashtags** |
| 0 | 0 | 5829 | 0 | 0 |
| 1 | 6 | 5421 | 0 | 402 |
| 5 | 0 | 495 | 5330 | 4 |
| 10 | 0 | 212 | 5617 | 0 |
| 20 | 0 | 97 | 5732 | 0 |
| 50 | 0 | 30 | 5799 | 0 |

**Number of features in different feature sets**

Here is the visual representation of the various accuracies based on the “threshold” value and the type of data (train or test).





We can make the following deductions from the above graphs:

1. The accuracy of Naïve Bayes Classifier **increases significantly from 63% to 94%** for the test data **when we increase the ratio of train to test data from 1:1 to 4:1**, which means, the more we train, the better is the accuracy for this classifier.
2. **The hash tags and the obscene junk words do not form the best of the feature sets** to determining the gender of a user, which means that even if we remove them we will still get a very good accuracy in our case (i.e. for “Remaining words” and “Words below threshold”)
3. **The train data accuracy is almost the same (~93%)** whichever way we split the train and test data.
4. We get the best **accuracy** of **60.38%** if we run the **Naïve Bayes on the test data** and if we use the words below threshold of 5 in the case of the **1:1** train and test data split up. This gives us the best feature set and hence helps us in reducing the original feature to include only those words that contribute the most. Even though a threshold of 0 gives a slightly better accuracy of 63% but this is a trade-off we are willing to do considering we are able to reduce our feature set by approximately 340 words.
5. We get the best **accuracy** of **94.69%** if we run the **Naïve Bayes on the test data** and if we use the words below threshold of 5 in the case of the **4:1** train and test data split up. This gives us the best feature set and hence helps us in reducing the original feature to include only those words that contribute the most. Even though a threshold of 10 gives a slightly better accuracy of 95% but this is a trade-off we are willing to do considering we are able to reduce our feature set by approximately 400 words.