FOODIE

RESTAURANT RECOMMENDATION SYSTEM & RESTFUL API

REPORT ON SYSTEM INTEGRATION & EFFICIENCY OF CORE-ALGORITHM

By - Anurag Malik, Lokesh Agrawal (Team5)

MOTIVATION & IDEA

It is generally the case that two different customers/ users for the same product or restaurant provide a personal feedback for their experience with the product or restaurant food/service quality and even if their sentiments while writing a review are same, they are not bound to give the same ratings/stars and thus individual ratings often end up in great contrast. For example, a happy customer who is strict while rating a restaurant gives only 3 stars, while another happy user gives a 4.5 star. This makes the generic stars/ review based recommendation system skewed.

Our idea with Foodie as a better restaurant recommendation system is to analyse the sentiments of a customer/ user providing a review and accordingly decide the overall ratings for a restaurant.

CORE ALGORITHM

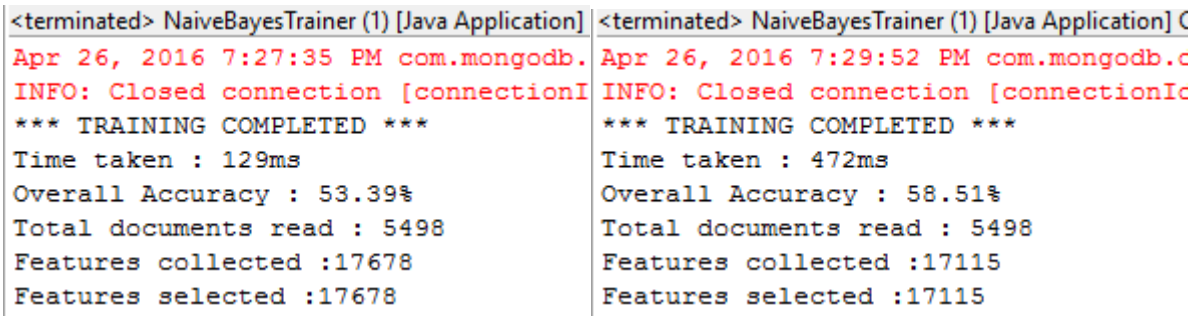
We have implemented Naïve-Bayes Multinomial algorithm for the classification of reviews into following three classes – positive (pos), negative (neg) & neutral (ntr). The core algorithm is trained on crowd sourced reviews and existing ratings for various restaurants available from Yelp dataset.

After the implementation of core-algorithm with Naïve Bayes classifier, we tested the following scenarios and tweaked our original implementation of the classifier to provide better results when trying to predict the sentiments of a review text data. The report below describes how we kept on improving the accuracy:

Stopwords or not? **Accuracy ~ 58%**

The preparation of training data included tokenization of reviews from overall dataset. We removed unnecessary punctuations, white spaces, digit values and other unknown characters from reviews and labelled each review on the basis of their provided star ratings.

We tried removing stopwords from the list of tokens retrieved from the data, using a general stopwords list. A series of few tests showed an improvement in prediction accuracy. Notice the total number of all features collected for the below test cases (Feature selector is OFF):

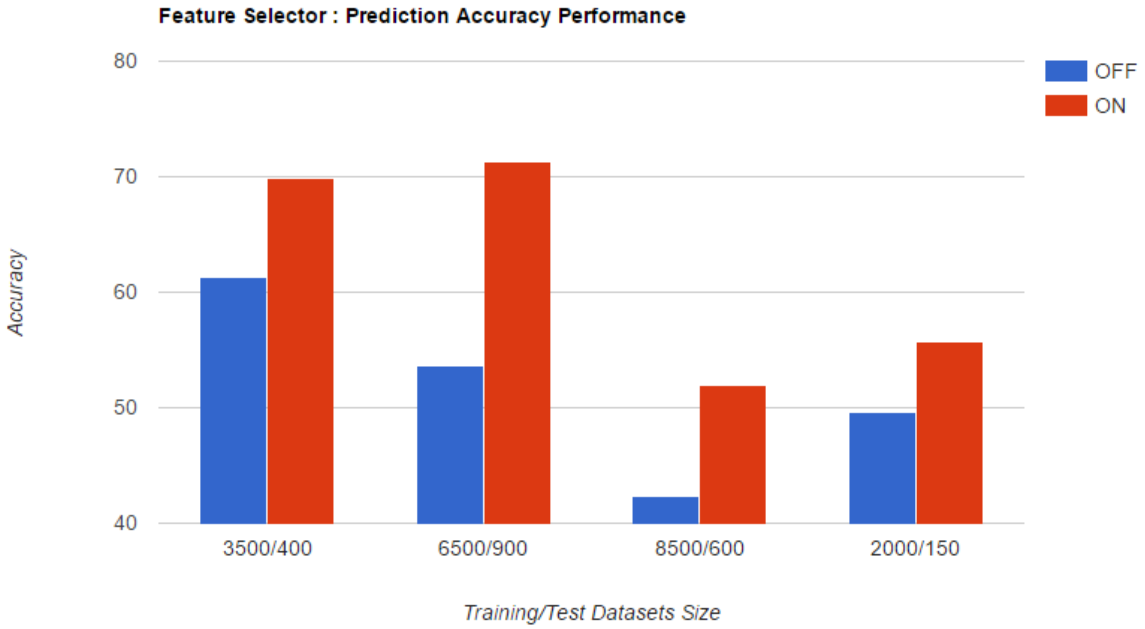


Best Features or not? **Accuracy ~ 71%**

Next step was to test our Chi-Square Feature selector to filter out the best features from the overall features corpus. This resulted in removal of noise and unnecessary features, those which had a chi-square score below a lower-bound. We first noticed a general trend in improvement of prediction accuracy on use of Feature Selection algorithm. The graph below depicts a quick view of our test on training set of various sizes:

```
<terminated> NaiveBayesTrainer (1) [Java Application]
Apr 26, 2016 7:43:12 PM com.mongodb.
INFO: Closed connection [connectionI
*** TRAINING COMPLETED ***
Time taken : 483ms
Overall Accuracy : 53.62%
Total documents read : 6497
Features collected :18811
Features selected :18811
```

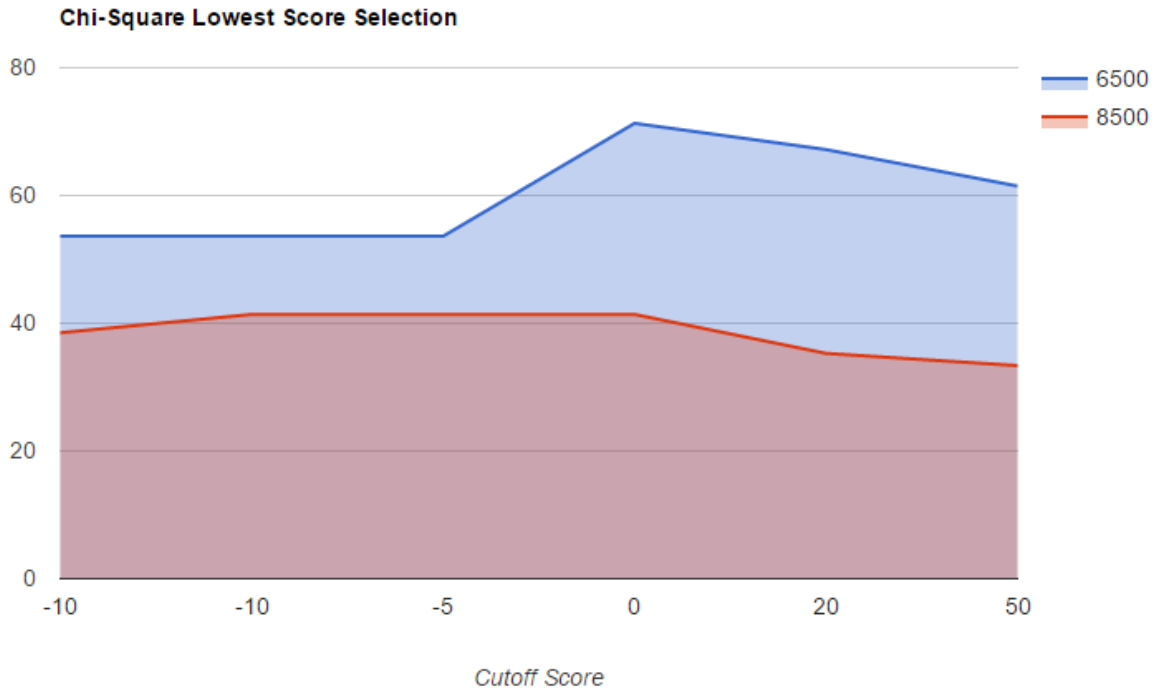
```
<terminated> NaiveBayesTrainer (1) [Java Application]
Apr 26, 2016 7:44:08 PM com.mongodb.
INFO: Closed connection [connectionI
*** TRAINING COMPLETED ***
Time taken : 298ms
Overall Accuracy : 71.30%
Total documents read : 6497
Features collected :18811
Features selected :13791
```



Fixing the lower threshold for feature selection:

Since the above described chi-square feature selection algorithm works on a lower cut-off score. It selects only those features which have a score higher than this score. For different size of test cases, the algorithm had to be able to calculate this value and discard only the lower $1/6^{th}$ of average score for all features.

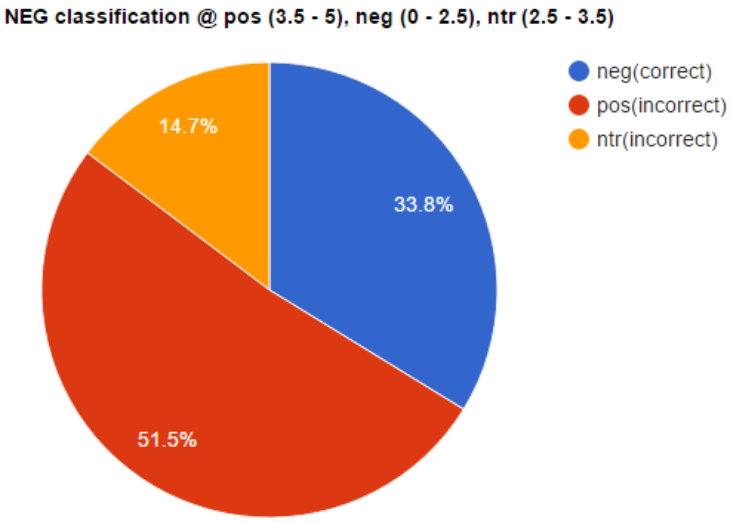
Below is a graph depicting the accuracy in prediction for training datasets of size – 6500 & 8500 records:



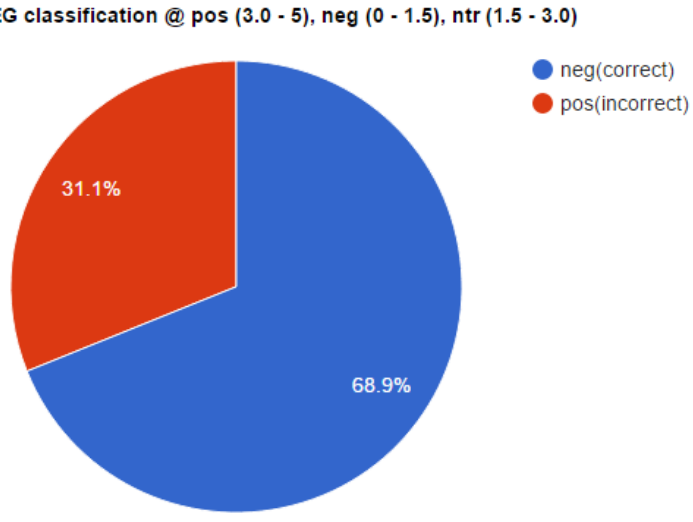
Positive, Neutral or Negative? **Accuracy ~ 73%**

We calculated confusion matrix to understand the details of what could be the possible reasons for wrong predictions. This led us to another interesting observation. We noticed that the accuracy in prediction of reviews belonging to various classes had a similar pattern for almost all our training/test datasets. Prediction accuracy for reviews with 'positive' sentiment analysis was very high. But reviews with negative sentiments were being confused with neutral as well as positive reviews. Below are the pie chart descriptions for various training and test data cases:

```
<terminated> NaiveBayesTrainer (1) [Java Application] C:\Prog
Apr 26, 2016 9:12:18 PM com.mongodb.diagr
INFO: Closed connection [connectionId{loc
*** TRAINING COMPLETED ***
Time taken : 316ms
Overall Accuracy : 57.73%
Total documents read : 5999
Features collected :17944
Features selected :17944
Number of classes for classification :3
--> Class Description <--
        neg : 33.78%
        pos : 100.00%
        ntr : 39.33%
--> Confusion Matrix <--
neg(299) : neg(101), 154(pos), 44(ntr)
pos(300) : pos(300)
ntr(300) : ntr(118), 182(pos)
```




```
<terminated> NaiveBayesTrainer (1) [Java Application] C:\Progra
Apr 26, 2016 9:14:34 PM com.mongodb.diagno
INFO: Closed connection [connectionId{loca
*** TRAINING COMPLETED ***
Time taken : 503ms
Overall Accuracy : 73.08%
Total documents read : 5999
Features collected :17852
Features selected :17852
Number of classes for classification :3
--> Class Description <--
        neg : 68.90%
        pos : 100.00%
        ntr : 50.33%
--> Confusion Matrix <--
neg(299) : neg(206), 93(pos)
pos(300) : pos(300)
ntr(300) : ntr(151), 16(neg), 133(pos)
```



This was related to the preparation of our training datasets from the database. Since the reviews were spread over a range of star ratings – 0 to 5 with steps of size 0.5. We initially started with the simple idea of categorizing the reviews using the same scale and partition the three classes over the range 0 to 5. But the results showed that changing this range of class labelling effected the feature selection algorithm as well as the final predictions. We ended up setting the following range for our input training data:

- 3.5 to 5 – positive (pos)
- 0 to 1.5 – negative(neg)
- 1.5 to 3.5 – neutral (ntr)

FOODIE V/S WEKA **Accuracy ~ 79.5**

After implementing all the above parameters and improvising our core-algorithm we achieved a prediction accuracy of about 79.5% for the following configuration:

- Training Data Records: 6500
- Test Data Records: 900
- Core-Algorithm: Naïve Bayes Multinomial with Chi-Square Feature Extraction Algorithm
- Number of classification classes: 3 {pos, neg, ntr}
- Total features collected for learning: 13752 out of 18530
- Class-wise accuracy: pos (100%), neg(78.26%), ntr(60%)

Below are the screenshots from Foodie Recommendation System and Weka:

```
<terminated> NaiveBayesTrainer (1) [Java Application] C:\Prograr
Apr 26, 2016 9:17:01 PM com.mongodb.diagnos
INFO: Closed connection [connectionId{local
*** TRAINING COMPLETED ***
Time taken : 271ms
Overall Accuracy : 79.42%
Total documents read : 6497
Features collected :18530
Features selected :13752
Number of classes for classification :3
--> Class Description <--
        neg : 78.26%
        pos : 100.00%
        ntr : 60.00%
--> Confusion Matrix <--
neg(299) : neg(234), 65(pos)
pos(300) : pos(300)
ntr(300) : ntr(180), 24(neg), 96(pos)
```