Report

Aarunish Sinha - 2018CS10321

December 2020

0.1 Naive Bayes

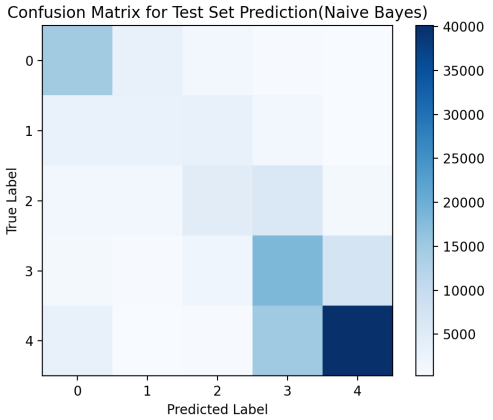
0.1.1 Implementation

First the datasets are read using the `json` library in python. Then the frequency of every word in each class is calculated and stored inside a dictionary. Then Laplace Smoothing is performed on this dictionary and the *log prior* is calculated. Using this dictionary and the log prior, I have implemented the prediction function which returns an array containing the classes predicted for each test set example. The accuracy is calculated by comparing the predicted classes with the the actual class labels for test set.
Accuracy on Test Set: 60.39%
Accuracy on Train Set: 64.59%

0.1.2 Random and Majority Prediction

Accuracy for Random Prediction on Test Set: 20.04%
Accuracy for Majority Prediction on Test Set: 43.98%
The algorithm implemented in the previous part gives an improvement of 40.35% over Random Prediction and an improvement of 16.41% over Majority Prediction.

0.1.3 Confusion Matrix



$$Confusion\ Matrix\ = \begin{bmatrix} 14630 & 3369 & 1213 & 565 & 392 \\ 2947 & 2919 & 3241 & 1321 & 410 \\ 1430 & 1439 & 4729 & 5969 & 964 \\ 1110 & 550 & 2060 & 18382 & 7256 \\ 3091 & 257 & 450 & 14929 & 40095 \end{bmatrix}$$

0.1.4 Stopword Removal and Stemming

For stemming and stopwords removal the functions given in `utils.py` are used.
Accuracy on Test Set: 60.02%
Accuracy after stemming slightly reduces. Stemming reduces all the words to their canonical forms and after removing stopwords the text reviews can become more similar across the different classes. Even though stemming adds a lot in terms of recall, it takes a hit when it comes to precision and in this case of classifying reviews the accuracy decreases.

0.1.5 Feature Engineering

N-grams

Creating bigrams on raw words,
Accuracy on Test Set for Bigrams: 63.97%
Creating bigrams after removing stopwords,
Accuracy on Test Set for Bigrams after removing stopwords: 63.48%

Creating trigrams on raw words,
Accuracy on Test Set for Trigrams: 64.77%
Creating trigrams after removing stopwords,
Accuracy on Test Set for Trigrams after removing stopwords: 50.33%

Creating fourgrams on raw words,
Accuracy on Test Set for Fourgrams: 60.34%

Lemmatization

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.
Performing lemmatization on raw words,
Accuracy on Test Set: 46.79%
Performing lemmatization on tagged words obtained after POS-tagging of raw words,
Accuracy on Test Set: 60.07%

Conclusion

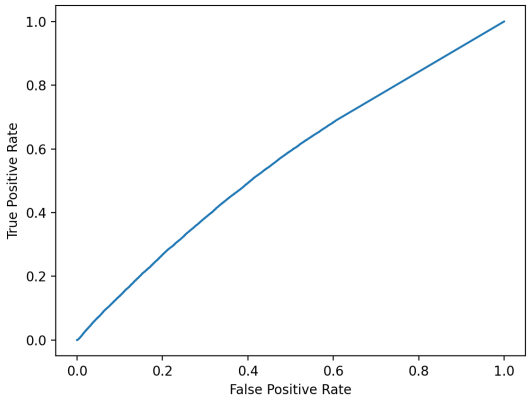
For n-grams, creating bigrams and trigrams improves the accuracy by $3 - 4\%$ over Raw words and Stemming.
Further, increasing n decreases the accuracy as the feartures become more and more distinct and the most features are not present in the test set.
In general, lemmatization performs poorly as compared to stemming and n-grams. By default, the `WordNetLemmatizer()` in creates a lemma assuming it

to be a noun. Hence, performing POS-tagging on the raw words and creating lemmas based on their tags would result in a more meaningful lemmatization and increase the accuracy of the model.

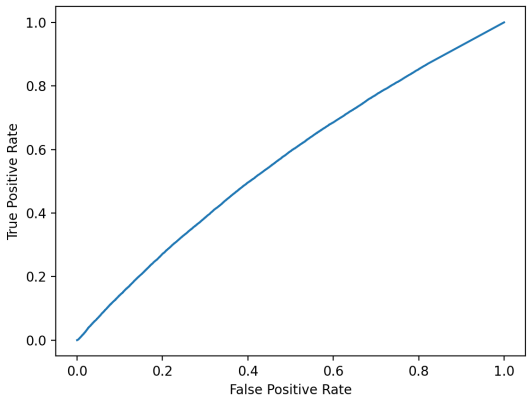
0.1.6 Receiver Operating Characteristic

I have performed micro-averaging for the ROC-plot. The prediction contains the maximum probabilities which is used to make a prediction of a label and the true label array is converted into a binary array with contains '1' if the prediction is correct and '0' if the prediction is incorrect.

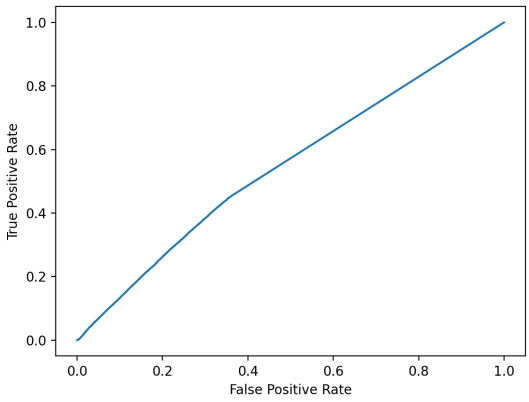
Raw Words Model



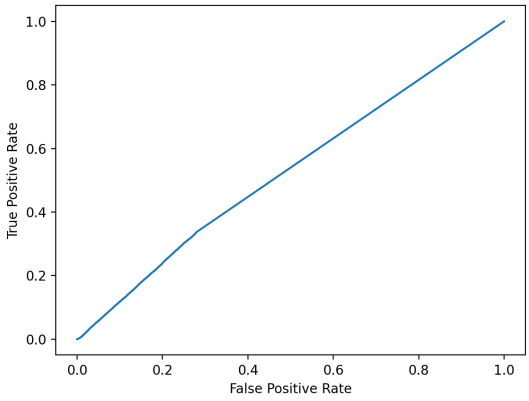
Stemming Model



Bigram Model



Trigram Model



For each of the models the True Positive Rate and the False Positive Rate increase from 0,0 to 1,1 as the value of the threshold is increased.

0.2 Support Vector Machines

0.2.1 Binary Classification

Linear Kernel

Tolerance = 10^{-4}
b = -0.92063451
Number of Support Vectors = 163
Accuracy on Validation Set = 98.4%
Accuaracy on Test Set = 98.4%

Gaussian Kernel

Tolerance = 10^{-4}
b = -0.531842369
Number of Support Vectors = 820
Accuracy on Validation Set = 99.2%
Accuracy on Test Set = 98.9%
The accuracy on the validation set and test set increases when using Gaussian Kernel when compared to Linear Kernel.

0.2.2 Multi-Class Classification

Multi-Class Classification using `cvxopt`

Making $\binom{k}{2}$ classifiers using Gaussian Kernel implemented in Binary Classification
Tolerance = 10^{-6}
Training Time = 2802.9 *seconds*
Accuarcy on Validation Set = 88.00%
Accuracy on Test Set = 87.94%

Multi-Class Classifiaction using `svm.SVC`

Training Time = 233.3 *seconds*
Accuracy on Validation Set = 87.92%
Accuracy on Test Set = 88.08%
The Scikit Learn implementation of Multi-Class classification using Gaussian Kernel is much much faster as compared to the `cvxopt` implemention. Both the implementations of SVM classifiers perform almost identically in terms of accuracy on validation and test set.

Comparison

For `cvxopt`,

$$Confusion\ Matrix\ = \begin{bmatrix} 212 & 0 & 1 & 8 & 0 & 0 & 26 & 0 & 3 & 0 \\ 0 & 239 & 2 & 6 & 0 & 0 & 2 & 0 & 1 & 0 \\ 5 & 0 & 206 & 3 & 18 & 0 & 13 & 0 & 5 & 0 \\ 6 & 0 & 0 & 228 & 6 & 0 & 9 & 0 & 1 & 0 \\ 1 & 1 & 24 & 8 & 200 & 0 & 15 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 241 & 0 & 2 & 1 & 5 \\ 33 & 0 & 28 & 3 & 19 & 0 & 165 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 & 230 & 1 & 11 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 2 & 244 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 & 8 & 1 & 235 \end{bmatrix}$$

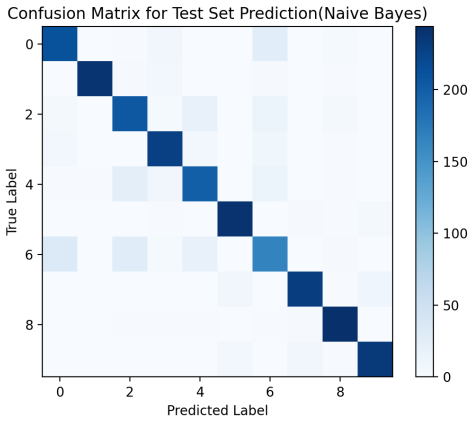


Figure 1: Confusion Matrix for Classification on Validation Set

$$Confusion\ Matrix\ = \begin{bmatrix} 426 & 0 & 5 & 11 & 3 & 0 & 45 & 0 & 10 & 0 \\ 0 & 483 & 4 & 8 & 0 & 0 & 5 & 0 & 0 & 0 \\ 4 & 0 & 407 & 7 & 37 & 0 & 37 & 0 & 8 & 0 \\ 12 & 1 & 2 & 455 & 8 & 0 & 17 & 0 & 5 & 0 \\ 3 & 1 & 43 & 13 & 397 & 0 & 38 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 473 & 0 & 16 & 5 & 6 \\ 73 & 0 & 57 & 9 & 30 & 0 & 324 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 14 & 0 & 471 & 1 & 14 \\ 1 & 0 & 1 & 1 & 1 & 2 & 3 & 2 & 489 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 & 14 & 1 & 475 \end{bmatrix}$$

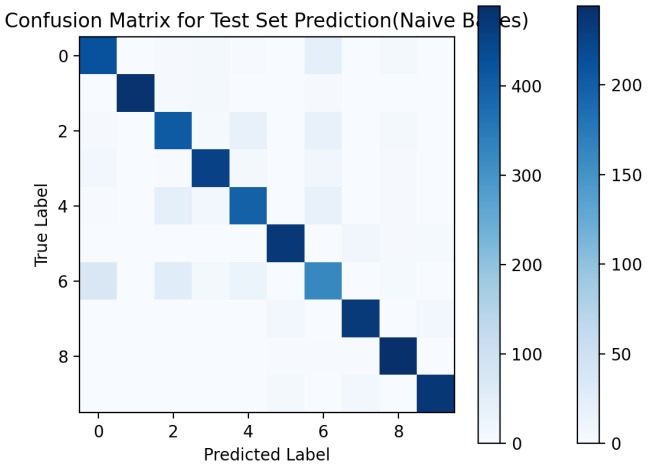


Figure 2: Confusion Matrix for Classification on Test Set

The article with $label = 6$ is the most mis-classified. It is frequently classified as the articles with $label = 0/2/4$.

For `svm.SVC`,

$$Confusion\ Matrix = \begin{bmatrix} 212 & 0 & 1 & 8 & 0 & 0 & 26 & 0 & 3 & 0 \\ 0 & 237 & 3 & 7 & 0 & 0 & 2 & 0 & 1 & 0 \\ 5 & 0 & 206 & 3 & 18 & 0 & 13 & 0 & 5 & 0 \\ 6 & 0 & 0 & 228 & 6 & 0 & 9 & 0 & 1 & 0 \\ 1 & 1 & 24 & 8 & 200 & 0 & 15 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 241 & 0 & 2 & 1 & 5 \\ 34 & 0 & 28 & 3 & 19 & 0 & 165 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 & 230 & 1 & 11 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 2 & 244 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 & 8 & 1 & 235 \end{bmatrix}$$

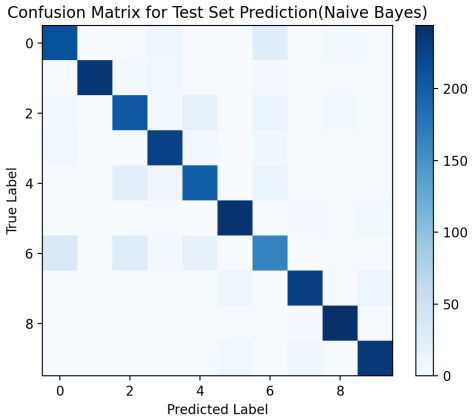


Figure 3: Confusion Matrix for Classification on Validation Set

$$Confusion\ Matrix\ = \begin{bmatrix} 433 & 0 & 5 & 11 & 3 & 0 & 38 & 0 & 10 & 0 \\ 1 & 482 & 4 & 9 & 0 & 0 & 4 & 0 & 0 & 0 \\ 5 & 0 & 411 & 7 & 37 & 0 & 32 & 0 & 8 & 0 \\ 12 & 0 & 3 & 457 & 9 & 0 & 14 & 0 & 5 & 0 \\ 3 & 1 & 41 & 13 & 399 & 0 & 38 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 473 & 0 & 16 & 5 & 6 \\ 80 & 0 & 55 & 9 & 34 & 0 & 315 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 14 & 0 & 471 & 1 & 14 \\ 1 & 0 & 1 & 1 & 2 & 2 & 2 & 2 & 489 & 0 \\ 0 & 0 & 0 & 0 & 0 & 11 & 0 & 14 & 1 & 474 \end{bmatrix}$$

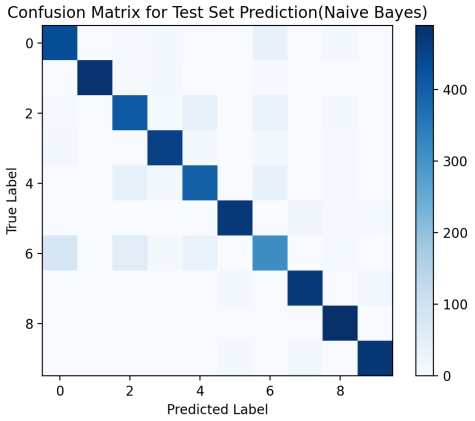
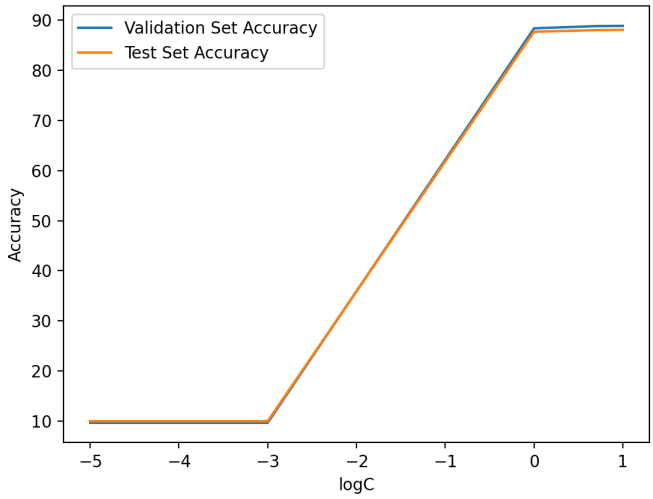


Figure 4: Confusion Matrix for Classification on Test Set

The article with $label = 6$ is the most mis-classified. It is frequently classified as the articles with $label = 0/2/4$.

K-Fold Cross Validation



The accuracy for both cross-validation set and the test is very low for smaller values of C and C increases the accuracy rapidly increases. For $C = 10$, highest accuracy for the cross-validation set is obtained, i.e. 88.84%. The highest accuracy on test set (88.06%) is also obtained for $C = 10$.
For $C > 1$ the value of accuracy for cross-validation set and test set does not increase much.

0.3 Large Scale Text Classification

0.3.1 Naive Bayes and SVM(with LIBLINEAR)

Training Time of Naive Bayes(Using Trigrams) = 161.5 *seconds*
Accuracy of Naive Bayes = 64.77%

Training Time of SVM = 45.9 *seconds*
Accuracy of SVM = 67.77%

$$Confusion\ Matrix\ = \begin{bmatrix} 17594 & 946 & 442 & 315 & 872 \\ 4270 & 2364 & 2196 & 1171 & 837 \\ 1477 & 1205 & 4406 & 5498 & 1945 \\ 550 & 221 & 1563 & 13363 & 13661 \\ 492 & 62 & 261 & 5109 & 52898 \end{bmatrix}$$

For SVM, hyperparameter tuning is done on the regualisation parameter C.
10% training set is taken as the validation set for hyperparameter tuning.
The model is evaluated on the validation for all the values of C in $\{10, 5, 1, 0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}\}$ and highest accuracy is observed for $C = 0.1$.

C	Accuracy on Validation Set
10	64.21%
5	65.13%
1	66.96%
0.1	67.85%
0.01	66.51%
10^{-3}	60.93%
10^{-4}	46.05%
10^{-5}	43.80%

The SVM algorithm performs better than the Naive Bayes algorithm by 3% in terms of accuracy.

0.3.2 SVM using SGD algorithm

Training Time = 37.4 *seconds*
Accuracy = 67.63%
For this part, I have used the `SGDClassifier` in the `sklearn` library.
The SGD classifier performs very similar to the LIBLINEAR implementation above in terms of the accuracy on the test set.
For hyperparameter tuning, I chose *alpha* for tuning. It is a parameter passed to the `SGDClassifier` which multiplies the regularization term. The higher the value, the stronger the regularization. So it has a similar effect as C in the above algorithm.

$$Confusion\ Matrix = \begin{bmatrix} 17196 & 1008 & 481 & 365 & 1119 \\ 4003 & 2201 & 2314 & 1340 & 980 \\ 1363 & 954 & 4480 & 5710 & 2024 \\ 512 & 143 & 1402 & 13946 & 13355 \\ 501 & 36 & 200 & 5466 & 52619 \end{bmatrix}$$

The confusion matrix shows that the classification into each label is also similar to the LIBLINEAR implementation.