MACHINE LEARNING ASSIGNMENT-2

ANKUR SHARMA
2015CS50278

PART 1 - TEXT CLASSIFICATION USING NAÏVE BAYES MODEL

(using Python 3.6)

1. Preprocessing involved:
    1. HTML Tags Removed : By using the regex module
    2. [OPTIONAL] Numbers Removed : At times, numbers only add noise to the data and carry no extra information and hence can be removed. But, removing numbers only decreases the accuracy, so this has been kept as an optional flag.
    3. Conversion to lower case : All strings have been converted to lower case to remove any disparities.
    4. Punctuations Removed : By using a string transition table for faster replacement of punctuations with spaces
    5. Emoticons Separated : Emoticons are filtered using regex matching only since they carry important information related to the reviewer's sentiment
2. Data Structures used:
    1. Dictionary for each class and the set of corresponding examples.
    2. Vocabulary as a set containing all the words (of length > 1).
    3. BagOfWords Model implemented for each doc and stored as list of such bags to maintain the frequency table and word count for each document, so when the time comes, these things can be accessed in a much faster and efficient way.
    4. Dictionaries to store the parameters for faster lookup at the time of prediction.
3. (PART A) **Naïve Bayes Algorithm:**
    1. Implemented as an event model that follows a multinomial distribution over the words.
    2. However, we assume that the same multinomial distribution has been followed for each word position in the document which greatly reduces the number of parameters.
    3. Naïve Bayes assumption which states that any two distinct words in the text document are conditionally independent of each other given the class.
    4. Feature Vector are implemented as word indices in the vocal set.
    5. Laplacian smoothing has been done with the parameter (C = 1).
    6. Results:
        1. Length of the vocabulary set : 75538
        2. Training Set Accuracy : 68.396%
        3. Test Set Accuracy : 38.46%
        4. Timing Details(in seconds) :
            1. Learn the Parameters : 0.96
            2. Training Set Prediction : 16.21
            3. Test Set Prediction : 15.93
    7. Conclusion: A simple Naïve Bayes classifier makes heavy assumptions regarding the independence between the features given the class, and that the same multinomial event model has been followed for each and every word.
4. (PART B)
    1. Percentage accuracy over the test set (Random Prediction) = 12.692%
    2. Percentage accuracy over the test set (Majority Prediction) = 20.088%
    3. Improvement factor of the classifier over the random baseline = 3.0302x

4. Improvement factor of the classifier over the majority baseline = 1.9145x

5. (PART C) CONFUSION MATRIX:
    1. As we can see from the matrix below, classes 1 and 10 have been distinguished very well with the classifier. So, the classifier is very well able to distinguish extreme reviews (i.e. either rating 1 or rating 10) from other classes. (Class 1 has the highest value in the diagonal entry followed by class 10)
    2. The diagonal values of other classes have a much lower value than those for classes 1 and 10 which shows that the classifier has trouble distinguishing highly polar reviews from the moderate ones.
    3. Other large values in the confusion matrix appear at (1,2), (1,3) and (10, 9) which shows that the classifier hasn't been able to properly distinguish between ratings 1/2, 9/10 and 1/3 which makes sense since they are very similar to each other and the decision boundary for such similar classes isn't very well defined.

			A	**C**	**T**	**U**	**A**	**L**	
P		**1**	**2**	**3**	**4**	**7**	**8**	**9**	**10**
R	**1**	**4267**	1591	1367	1038	400	422	332	796
E	**2**	90	**50**	57	46	9	12	6	12
D	**3**	156	183	**231**	212	79	62	23	44
I	**4**	256	271	485	**673**	259	168	92	98
C	**7**	35	54	129	224	**425**	309	155	178
T	**8**	63	54	116	225	519	**716**	467	567
E	**9**	18	5	13	26	73	131	**121**	172
D	**10**	137	94	143	191	543	1030	1148	**3132**

    4. We can also conclude that classes like 3-4, 7-8, etc. which have some moderate reviews aren't easily distinguished by the classifier (they all have roughly the same values) which can be due to the heavy assumptions that we make or that the features haven't been chosen accurately enough to build a strong classifier.

6. (PART D) STEMMING AND STOP-WORD REMOVAL:
    1. Test Set Accuracy (after stemming and stop word removal) = 68.008%
    2. Test Set Accuracy (after stemming and stop word removal) = 38.432%
    3. Observations and Conclusions:
        1. We notice that the test set accuracies decreases slightly after carrying out stemming and stop word removal. The script reduces the words to their root form, and carries out other kinds of preprocessing as well.
        2. This observation can be attributed to the fact that the reason for which stemming is done has in-fact proved to be contiguous to the classifier. For example, words which are really distinct may be wrongly conflated (e.g., "experiment" and "experience")
        3. Stemming tries to cut off the details like the exact form of the word, and tries to produce word bases as features. However, these details in-fact plays a crucial role in itself. Hence, we see that the accuracy doesn't improve at all and remains more or less the same.

7. (PART E) FEATURE ENGINEERING:
    1. **Bigrams**
        1. **TRAINING SET ACCURACY : 99.508**
        2. **TEST SET ACCURACY : 39.352**
    2. Bigrams with word count 2
        1. TRAINING SET ACCURACY : 39.2
        2. TEST SET ACCURACY : 19.296
    3. Bigrams with min length 2
        1. TRAINING SET ACCURACY : 99.508
        2. TEST SET ACCURACY : 39.352
    4. Bigrams with min length 3
        1. TRAINING SET ACCURACY : 99.508
        2. TEST SET ACCURACY : 39.352
    5. Bigrams with min length 4
        1. TRAINING SET ACCURACY : 99.512
        2. TEST SET ACCURACY : 39.32
    6. Bigrams with min length 5
        1. TRAINING SET ACCURACY : 99.492
        2. TEST SET ACCURACY : 38.616
    7. Bigrams with min length 6
        1. TRAINING SET ACCURACY : 99.484
        2. TEST SET ACCURACY : 38.492
    8. Trigrams
        1. TRAINING SET ACCURACY : 99.984
        2. TEST SET ACCURACY : 19.36
    9. Stems + bigrams of stems
        1. TRAINING SET ACCURACY : 92.588
        2. TEST SET ACCURACY : 36.452
    10. Stems + bigrams of stems + Minimum word length
        1. Count 2 :
            1. TRAINING SET ACCURACY : 92.672
            2. TEST SET ACCURACY : 36.5
        2. Count 3:
            1. TRAINING SET ACCURACY : 92.94
            2. TEST SET ACCURACY : 36.584
        3. Count 4:
            1. TRAINING SET ACCURACY : 94.368
            2. TEST SET ACCURACY : 37.072
    11. Stems + bigrams of stems + min word count of 2
        1. TRAINING SET ACCURACY : 53.164
        2. TEST SET ACCURACY : 14.0
    12. **Stems + bigrams of stems + trigram of stems:**
        1. **TRAINING SET ACCURACY : 99.704**
        2. **TEST SET ACCURACY : 38.14**

After implementing bigram and trigram functionality to engineer additional new features into my code for improving accuracy, I tried experimenting with a lot of possible combinations of these features. Some of the accuracies have been reported in the page

above. I implemented minimum word count strategy, minimum word length strategy and auto stemming and k-gram as a result. Increasing the value of k in k-gram features highly overfits the training set and as a result, the test set accuracy drops massively after k greater than 3.

For bigram features, I obtain the highest test set accuracy of 39.352% which is the best of my model. I conducted a lot of experiments by combining stems with bigram of stems, and altering the parameters such as minimum word length, and minimum word count, but I was not able to obtain any good accuracy and most of my accuracies range from 36-40% only.

There are certain other feature combinations that include POS tagging along with stemming, TF-IDF approach which also could've been tried along with my earlier approaches.

Features that helped me to improve accuracy:

1. Stemmed Bigram Features: Taking two words together as one feature. Acc = 39.352%
2. Stemmed + Stemmed Bigram + Stemmed Trigram Features: Make a new feature vector that include stemmed words, their bigram and trigrams also into one feature vector. Acc = 38.14%

With respect to part(a), and part(d), we see that feature engineering done using the 1st approach gives a better accuracy than either (a) or (d). Moreover, a new feature vector made using stemmed words along with their bigrams and trigrams also gives a resonable accuracy (though less but) compareble to part(a) and (d).

PART 2 - MNIST HANDWRITTEN DIGIT CLASSIFICATION USING SVMs

(a)

Preprocessing includes scaling the feature vectors in the range [0, 1] as mentioned in the assignment specification.

Objective function given in the class:

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{m} \max(0, 1 - t_i)$$

And in the paper given, the objective function given to us is:

$$\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{(\mathbf{x},y) \in S} \ell(\mathbf{w}; (\mathbf{x}, y))$$

$$\ell(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, 1 - y \langle \mathbf{w}, \mathbf{x} \rangle\}$$

In order to map terms, divide the objective function given in the paper by λ, which will make the regularisation term essentially the same. Note that dividing or multiplying a convex optimisation function by a constant doesn't change the optimal values of the parameters.

So, this will give the value of C as $\frac{1}{m\lambda}$. Using this , we can set the appropriate value of $\lambda = \frac{1}{mc}$ in order to use the same update rule given in the paper.

Here, the value of k is the batch size which is given to be 100.

Update Rules:

$$\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \frac{\eta_t}{k} \sum_{i \in A_t^+} y_i \mathbf{x}_i$$

$$b_{t+1} \leftarrow b_t + C.\eta_t.\sum_{i \epsilon A_t^+} y_i$$

Initialisation:
The value of b is initialised to be 0.0
And similarly, all the components of 784-dimensional w-vector were also set to 0.0 before running the mini batch version of Pegasos Algorithm.

Convergence Check: if the maximum absolute difference in any component in the value of w vector in consecutive iterations becomes less than epsilon, then we declare that the algorithm converges. Here, the epsilon was kept to be 10^{-6}.

(b)

C = 1.0
Training Set Accuracy = (19069/20000) -> 95.345%
Test Set Accuracy = (9274/10000) -> 92.74%

TIMING DETAILS :

TIME TAKEN TO LEARN 45 CLASSIFIERS : 59.8s
TIME TAKEN FOR TRAINING SET PREDICTION : 6.93s
TIME TAKEN FOR TEST SET PREDICTION : 3.47s

Accuracies obtained are pretty decent, and hence the classifier is efficient enough for prediction at a larger scale.

(c)

Using a simple python script, the data was converted into the format required by libsvm. After this, the features were scaled using svm-scale() function, and then trained using svm-train() function. After the model has been predicted, finally the prediction was done using svm-predict() which itself reports the accuracies on the terminal screen.

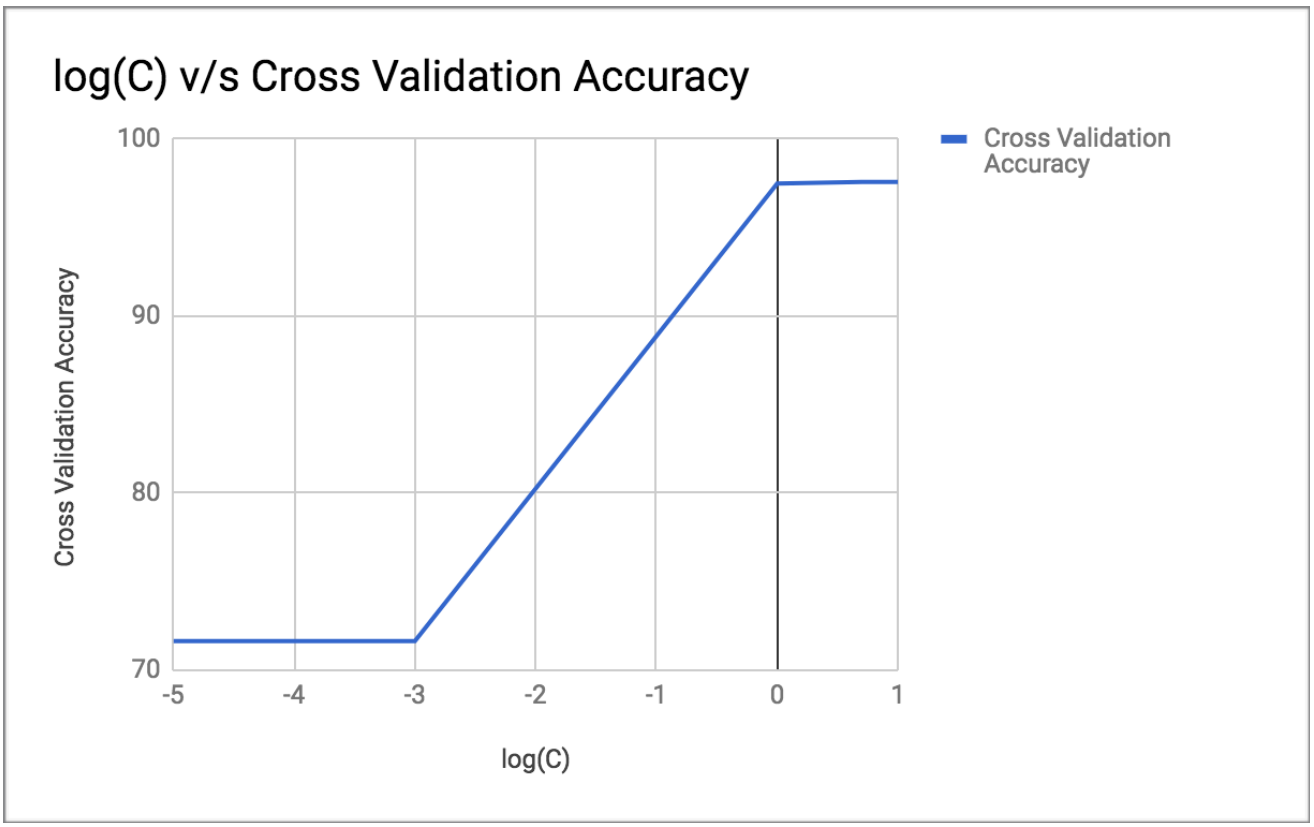
[LINEAR KERNEL MODEL]
Accuracy = 92.78% (9278/10000) (classification)
TRAINING TIME: 43.907s
PREDICTION TIME: 31.823s

[GAUSSIAN KERNEL MODEL]
Accuracy = 97.23% (9723/10000) (classification)
TRAINING TIME: 20.713s
PREDICTION TIME: 82.798s

We can see that the performance(in terms of accuracy) of the linear kernel is almost the same as the performance of the classifier obtained in part (b). This clearly shows that solving the primal problem (pegasos as in part b) is essentially the same thing as solving the dual problem(as in part c).

(d)

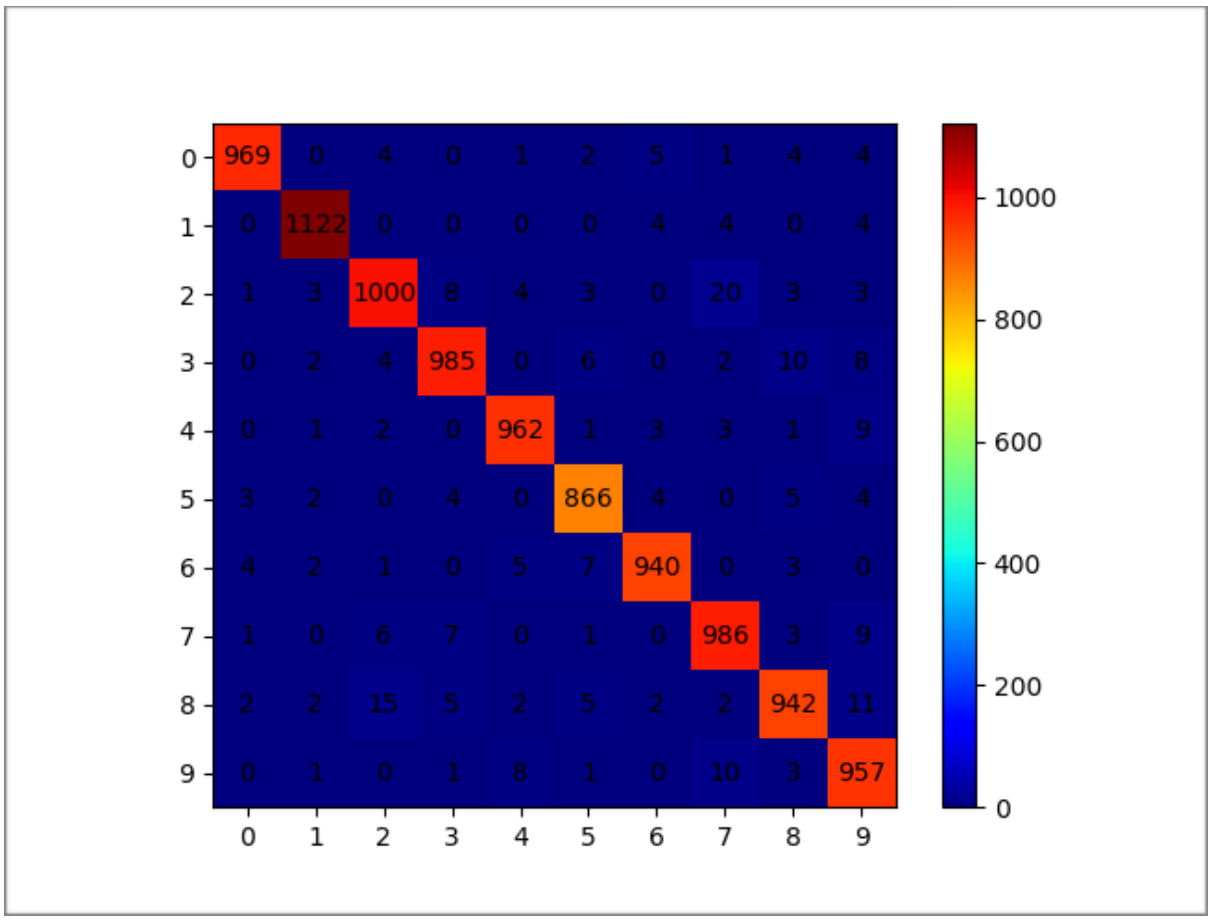

log(C) v/s Test Set Accuracy
Test Set Accuracy
100
90
80
70
-5
-4
-3
-2
-1
0
1
log(C)
Test Set Accuracy


log(C) v/s Cross Validation Accuracy
Cross Validation Accuracy
100
90
80
70
-5
-4
-3
-2
-1
0
1
log(C)
Cross Validation Accuracy

Looking at the graphical trends for the cross validation and test accuracy v/s the value of the hyper-parameter (C), it is evident that where we obtain a higher cross validation accuracy, we get a higher test accuracy as well which signifies the fact that cross validation is a good metric for optimising the value of the model parameters which are unknown to us since it means we are essentially testing on the unknown data.
Here, we see that C = 5, and C = 10 are the best values of hyper-parameters at which we obtain the maximum test and cross validation accuracy.

C	log(C)	Test Set Accuracy(%)	Cross Validation Accuracy(%)
0.00001	-5	72.11	71.645
0.001	-3	72.11	71.645
1	0	97.23	97.485
5	0.6989700043	97.29	97.575
10	1	97.29	97.575

(e)

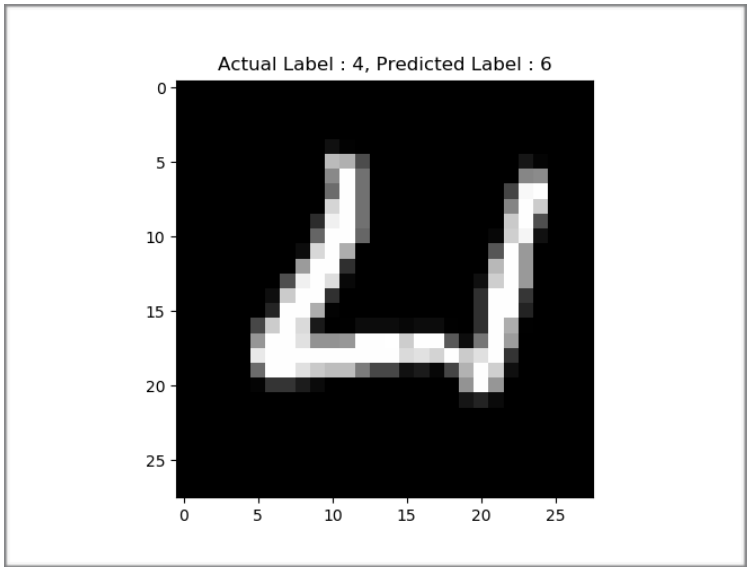


In the confusion matrix, we can see that almost all the diagonal entries range from 900-1100, which shows that our classifier is robust enough to correctly classify majority of the examples. Now looking at the off diagonal elements, we can say that the one with the highest value is the one with row = 3, and column = 8. Now it makes sense to conclude that our classifier is having most difficulty in classifying examples of 7, and often misclassifies it as class 2. 20 such examples of class 7 were misclassified into class 2. One can think of such misclassification based on the fact that the shape structure of letter 7 and 2 has the highest similarity amongst all the numbers from 0 to 9. All the other values in the matrix are quite less compared to the corresponding diagonal elements.

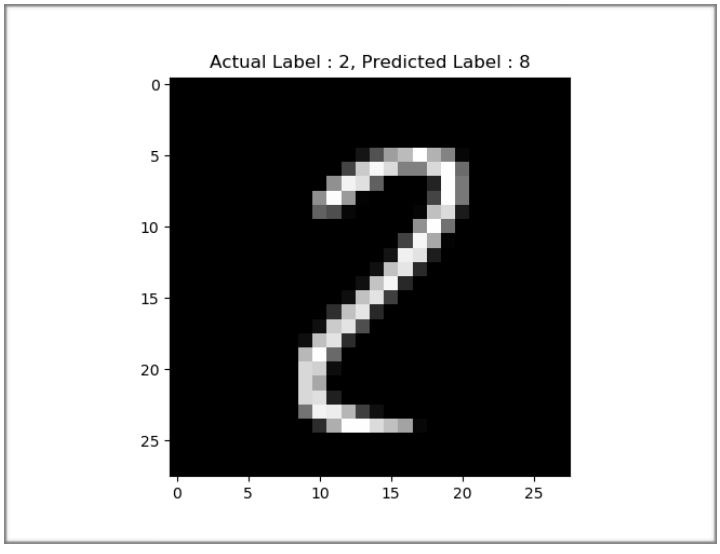
In the mnist test files given, consider the following three examples:

(Indexing is from 1)

<u>Example #34</u>
Predicted Label = 6
Actual Label = 4



<u>Example #614</u>
Predicted Label = 8
Actual Label = 2



Example # 4991

Predicted Label = 2

Actual Label = 3