Assignment 2 Report

Pradyumna Meena

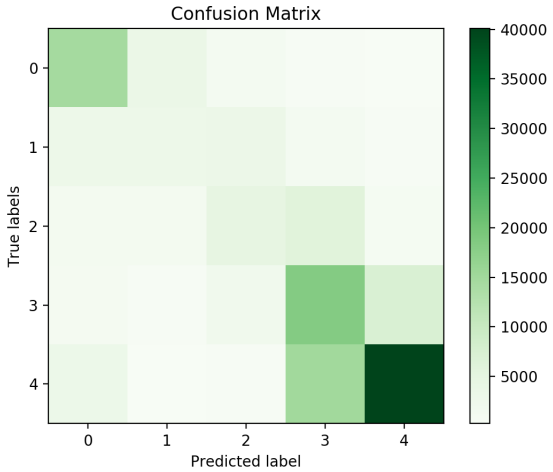
2016CS10375

1 Naive Bayes

1.1 Naive Bayes Vanilla

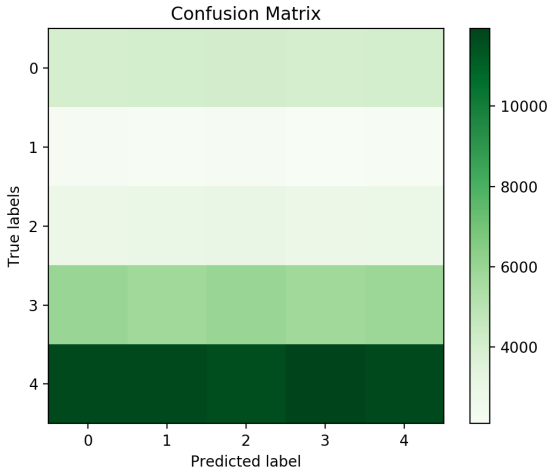
1. Accuracy over test data = 60.3%
2. Accuracy over training data = 64.69%
3. Confusion matrix (predicted over test data) is as followed

14616	3380	1214	565	394
2940	2920	3253	1315	410
1430	1432	4740	5968	961
1108	547	2075	18382	7246
3090	254	452	14949	40077



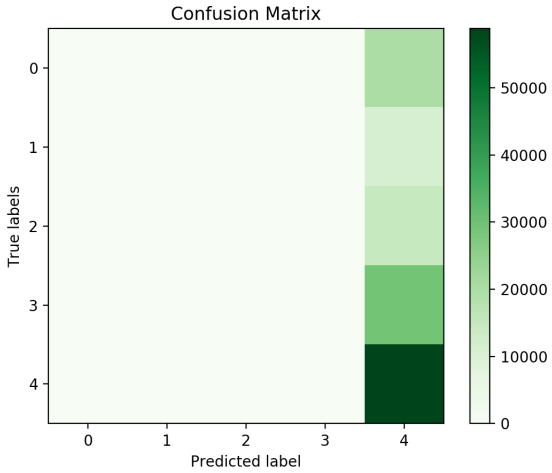
1.2 Random Prediction

1. Accuracy = 19.9%
2. Below is the confusion matrix for the same



1.3 Maximum Prediction

1. Maximum occurrences are of class with 5 stars (label=0)
2. Accuracy = 43.93%

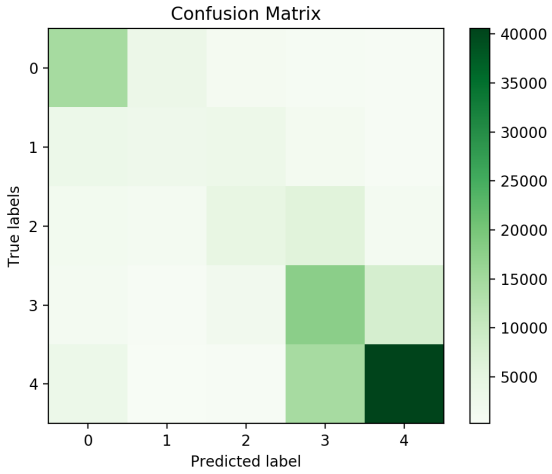


Maximum strategy performs much way better than random prediction. The reason behind this is that the random prediction predicts all of the classes almost uniformly and hence the probability of correct prediction is 0.2 where as in maximum prediction the class which has maximum data-points associated with it is predicted. Since it consists almost half of the data the accuracy is close to 50%.
Main algorithm gives almost 17% improvement over majority prediction and 40% over random prediction.

1.4 Stemming

1. Accuracy = 59.95%
2. Running Time = 22mins + 28mins
3. Accuracy is in-fact decreasing (although by 1%). This shows that stemming in this case is not working out in our favour. Possible reasons might be that stemming changes the basic meaning of words while carrying out stemming like removing -ing, -ed etc. Hence the overall meaning of a sentence maight get changed.
4. Below is the confusion matrix obtained after applying stemming (from utils.py)

14656	3251	1112	628	522
3101	2747	2979	1473	538
1548	1308	4487	5941	1247
1213	475	1961	17824	7885
3125	240	486	14421	40550



1.5 Feature Engineering

1.5.1 Bigrams

1. Accuracy = 63.9%
2. Running Time = 487 sec + 609sec
3. F1 score obtained is as followed

[0.74716161 0.1461689 0.21288322 0.51861346 0.8009275]

macro F1 score = 0.485

4. Confusion matrix is as followed

16487	1059	797	1018	448
4221	971	2157	3001	188
1738	286	2163	9122	1222
802	63	476	18410	9607
1319	69	197	10088	47149

1.5.2 Lemmatization

1. Accuracy = 46.78%
2. Running Time = 187 sec + 241 sec
3. F1 score obtained is as followed

[0.37690978 0.10444522 0.11352344 0.21577641 0.64720972]

macro F1 score = 0.291

4. Confusion matrix is as followed

7956	901	544	1343	9425
2843	746	616	1438	5195
2554	668	1051	2666	7592
3370	581	1025	4694	19688
5325	551	749	4009	48188

1.6 F1 score and Best Model

Best model was the Bigram model. Various details related to the model have already been mentioned above. For multi-class datasets or the datasets where one of the classes has very few occurrences choosing test-error as an evaluation metric is not a good idea. This is because test error metric does not considers precision of the algorithm into consideration. If we predict wrong on a few data-points its no issue but if majority of these points are the only points of any class then we have the accuracy according to test error metric but we have essentially lost the output correctness property in the process.

1.7 train_full.json

1. Test set accuracy = 61.77% (decreased from bigrams model)
2. Time taken = 1.5 hours
3. F1 score is as followed

[0.65263644 0.35543266 0.42998469 0.54014992 0.75004370]

macro F1 score = 0.545

4. Confusion matrix is

14294	3987	1167	401	339
2523	3625	3275	933	335
1289	1603	5891	4838	823
1121	697	2307	18519	6597
3120	358	534	14741	40043

1.8 Observations

Highest diagonal entries are for class 4 (5 stars). This means that the model was successful in predicting correct label for the class with maximum occurrences. Moreover there are comparable mis-classification for few classes where we have similar number of occurrence of them. Due to similar occurrences the probability outputs are quite similar though not upto large extent. Bigrams end up performing better than the normal model as well as lematization.

2 Support Vector Machines

2.1 Binary Classification

The dual objective function can be written as

$$\max_{\alpha} \sum_{\alpha} \alpha - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)^T} x^{(j)} \tag{1}$$

Various constraints to be taken care of are as followed

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \tag{2}$$

$$0 \leq \alpha_i \leq C \vee i = 1, 2, 3....m \tag{3}$$

However for CVXOPT we need to convert it to a minimization problem. Hence the new objective is

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)^T} x^{(j)} - \sum_{\alpha} \alpha \tag{4}$$

The CVXOPT specifications of objective function and constraints are as followed. Here x is column vector consisting of the variables of the objective function

$$\min_{x} \frac{1}{2} x^T P x + q^T x \tag{5}$$

$$Gx \leq h \tag{6}$$

$$Ax = b \tag{7}$$

2.1.1 Linear Kernel with CVXOPT

Here our variable vector is the set of α_i for each data-point present in training set. The various matrices required by the package are calculated as followed

$$\text{A= } \begin{bmatrix} 1 & 1 & 1 & ..... & 1 & 1 \end{bmatrix} \qquad \text{b} = \begin{bmatrix} 0 \end{bmatrix}$$

$$\text{G} = \begin{bmatrix} -1 & 0 & 0 & 0 & ... & 0 \\ 0 & -1 & 0 & 0 & ... & 0 \\ 0 & 0 & -1 & 0 & ... & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & ... & 0 & -1 \\ 1 & 0 & 0 & 0 & ... & 1 \\ 0 & 1 & 0 & 0 & ... & 1 \\ 0 & 0 & 1 & 0 & ... & 1 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{h} = \begin{bmatrix} 0 \\ .. \\ .. \\ 0 \\ C \\ .. \\ .. \\ C \end{bmatrix} \qquad \text{q} = \begin{bmatrix} 1 \\ 1 \\ .. \\ .. \\ 1 \\ 1 \end{bmatrix}$$

In the vector G the first m rows ensure the left side of constraint specified by equation 4 and the lower half takes care of right side of the same constraint. The matrix **P** is a square matrix of dimensions *(m,m)*. Any element P_{ij} can be described by the following equation

$$P_{ij} = \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)^T} x^{(j)} \tag{8}$$

Various statistics about the execution (for 5,6)

1. Training time = 3 mins
2. Threshold used (for selecting support vectors) = 0.0001
3. Number of support vectors = 233
4. Value of b = 1.635
5. Average test set accuracy = 97.29%

weight matrix and support vector indices are stored in *weight_matrix.txt* and *linear_support_vector_indices.txt* respectively both of which are enclosed with the submission

2.1.2 Gaussian Kernel with CVXOPT

All of the variable matrices of CVXOPT still have the same value except for matrix P which will get changed in the following way (K is the kernel used)

$$P_{ij} = \alpha_i \alpha_j y^{(i)} y^{(j)} K\left(x^{(i)}, x^{(j)}\right) \tag{9}$$

For gaussian kernel the weight matrix is not fixed. It is different for different data-points. For any particular data-point say x it is calculated as

$$w_x = \sum_{i=1}^{m} \alpha_i y^i K\left(x^i, x\right) \tag{10}$$

$$b = \frac{1}{m}\sum_{i=1}^{m}\left(y^i - \sum_{j=1}^{m}\alpha_j y^j K\left(x^{(i)}, x^{(j)}\right)\right) \tag{11}$$

Prediction on any data-point is done in the same way as in linear kernel. Various statistics obtained are

1. Training time = 4 mins
2. Threshold used (for selecting support vectors) = 0.0001
3. Number of support vectors = 1496
4. Average test set accuracy = 99.98%
5. Value of b = 0.1398

support vectors obtained are stored in *gaussian_support_vector_indices.txt* which is enclosed with the submission. We can see that using gaussian kernel adds to our advantage but that is not the general case. If we already have data which have high number of features or we can say high-dimensional data then using gaussian kernel to map it higher dimensions is not going to help. Linear kernels end up performing better in cases where the feature space is huge where as gaussian kernels are used when we have fewer features compared to the number of observations. Here we see that both end up with almost similar accuracies.

2.1.3 LIBSVM package with Linear and Gaussian Kernel

1. Number of support vectors
   (a) Linear Kernel = 233
   (b) Gaussian Kernel = 1477
2. Bias
   (a) Linear Kernel = -1.624401
   (b) Gaussian Kernel = -0.140654
3. Training Time
   (a) Linear Kernel = 2.3 sec
   (b) Gaussian Kernel = 6.3 sec
4. Accuracy on test set
   (a) Linear Kernel = 97.3%
   (b) Gaussian Kernel = 99.2%

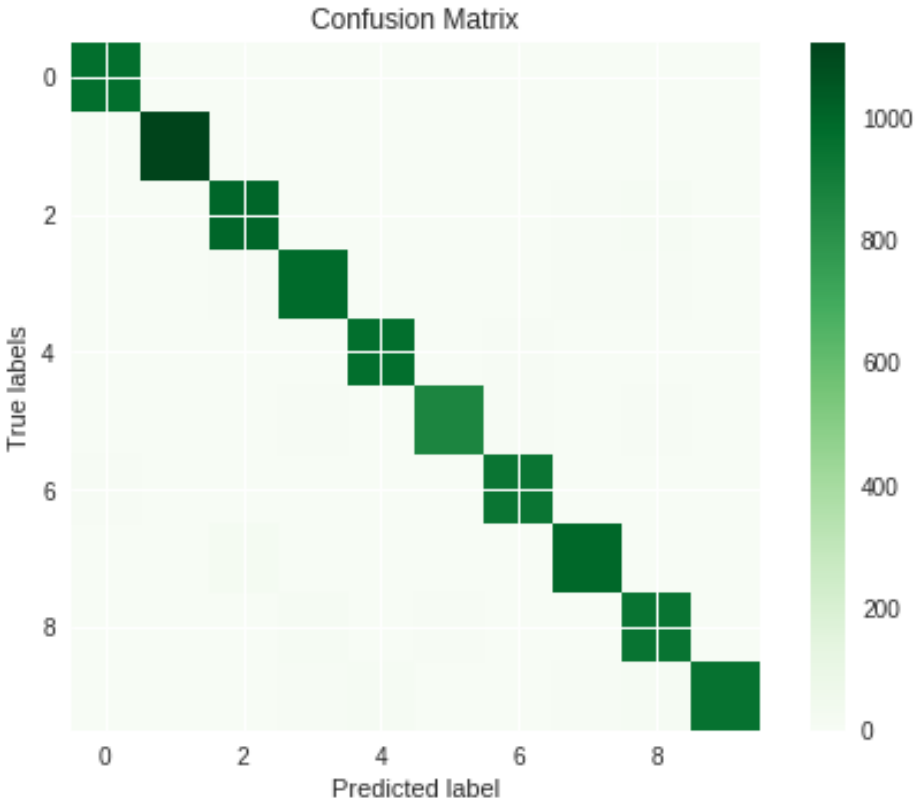
As we can see from training times LIBSVM is much faster than CVXOPT. However both of the packages have obtained same number of support vectors for both the kernels. If we see the bias value they are exactly negative of each other. This is because of internal functioning of LIBSVM package.

2.2 Multi-class Classification

2.2.1 CVXOPT Package

1. Confusion Matrix

969	0	1	0	0	3	4	1	2	0
0	1122	3	2	0	2	2	1	3	0
4	0	1000	4	2	0	1	6	15	0
0	0	8	986	0	4	0	7	5	0
0	0	4	0	970	0	6	0	2	0
2	0	3	6	1	867	7	1	5	0
6	3	0	0	4	4	939	0	2	0
1	4	19	2	4	0	0	996	2	0
4	0	3	10	3	5	1	4	944	0
4	4	4	8	12	4	0	8	12	953

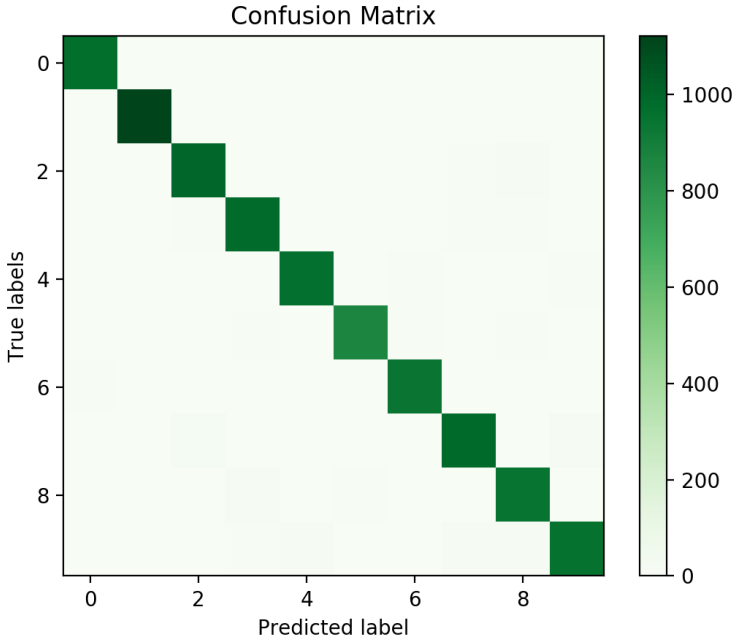


2. Training set accuracy = 99.94%
3. Test set accuracy = 97.46%
4. Training Time = 4 hours
5. Threshold = 0.0001

2.2.2 LIBSVM Package

1. Confusion Matrix

969	0	1	0	0	3	4	1	2	0
0	1121	3	2	1	2	2	0	3	1
4	0	1000	4	2	0	1	6	15	0
0	0	8	985	0	4	0	6	5	2
0	0	4	0	962	0	6	0	2	8
2	0	3	6	1	866	7	1	5	1
6	3	0	0	4	4	939	0	2	0
1	4	19	2	4	0	0	987	2	9
4	0	3	10	1	5	3	3	942	3
4	4	3	8	13	4	0	9	12	952



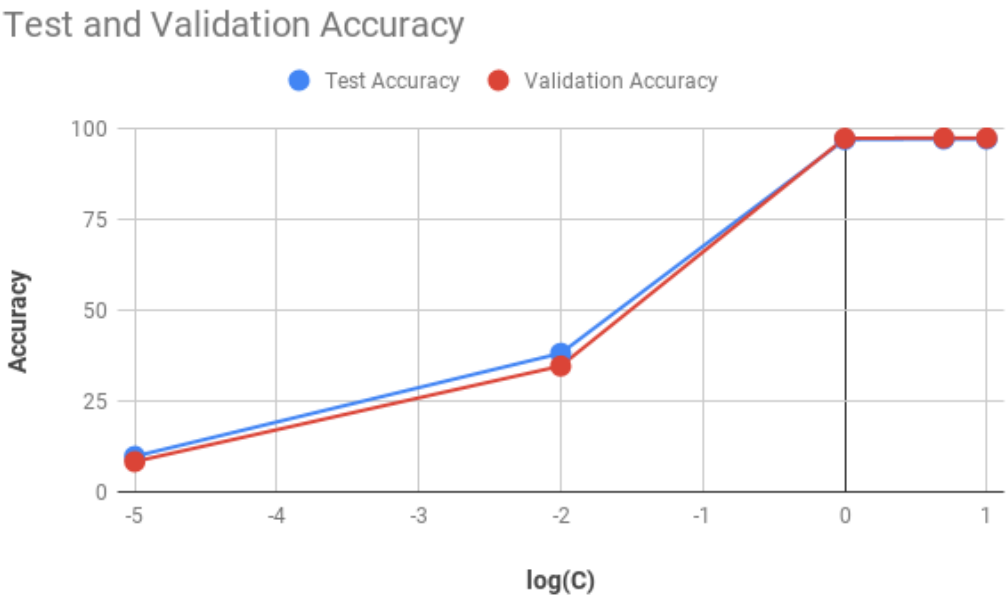
2. Training set accuracy = 99.92%
3. Test set accuracy = 97.23%
4. Training Time = 234.64 sec

Most mis-classified digit is 8. It has been mistaken for 3. Since 3 and 8 are atleast half-alike there are scenarios of misclassification amongst the two. Similar cases are observed for (4,9), (2,7), (8,9) and (2,8). In some cases both the digits in the above specified pair are quite alike depending on the handwriting and hence there have been cases of misclassification.

As we can see there is a tremendous amount of difference amongst the training times. This is because we are training 45 SVMs in the prior part and only a single SVM in the latter.

Moreover LIBSVM was much faster than CVXOPT package and therefore LIBSVM turns out to have lower training time than CVXOPT.

2.2.3 Validation Set



Value of C which gives best validation accuracy is 1 and test set accuracy is also highest at this point. Moreover at 5 and 10 also the results are a lot same. Having large value of C is not going to help since we have to minimize the sum over all the lagrange multipliers all of which are constrained to be less than C. While increasing C we are allowing them to take large values but at the same time we also want to minimize the objective function which itself depends on them (both the weight matrix term and the summation over all lagrange multipliers).