COL774 - Assignment 2

Ankit Solanki
2016CS50401

March 13, 2019

1 Part-A: Text Classification Using Naive Bayes

1.1 Part-a

Naive Bayes Implementation: **Accuracy**

```
[>] Number of testing examples: 133718
[*] Accuracy on test set: 0.5572
[-]
[>] Number of training examples: 534872
[*] Accuracy on train set: 0.5621
```

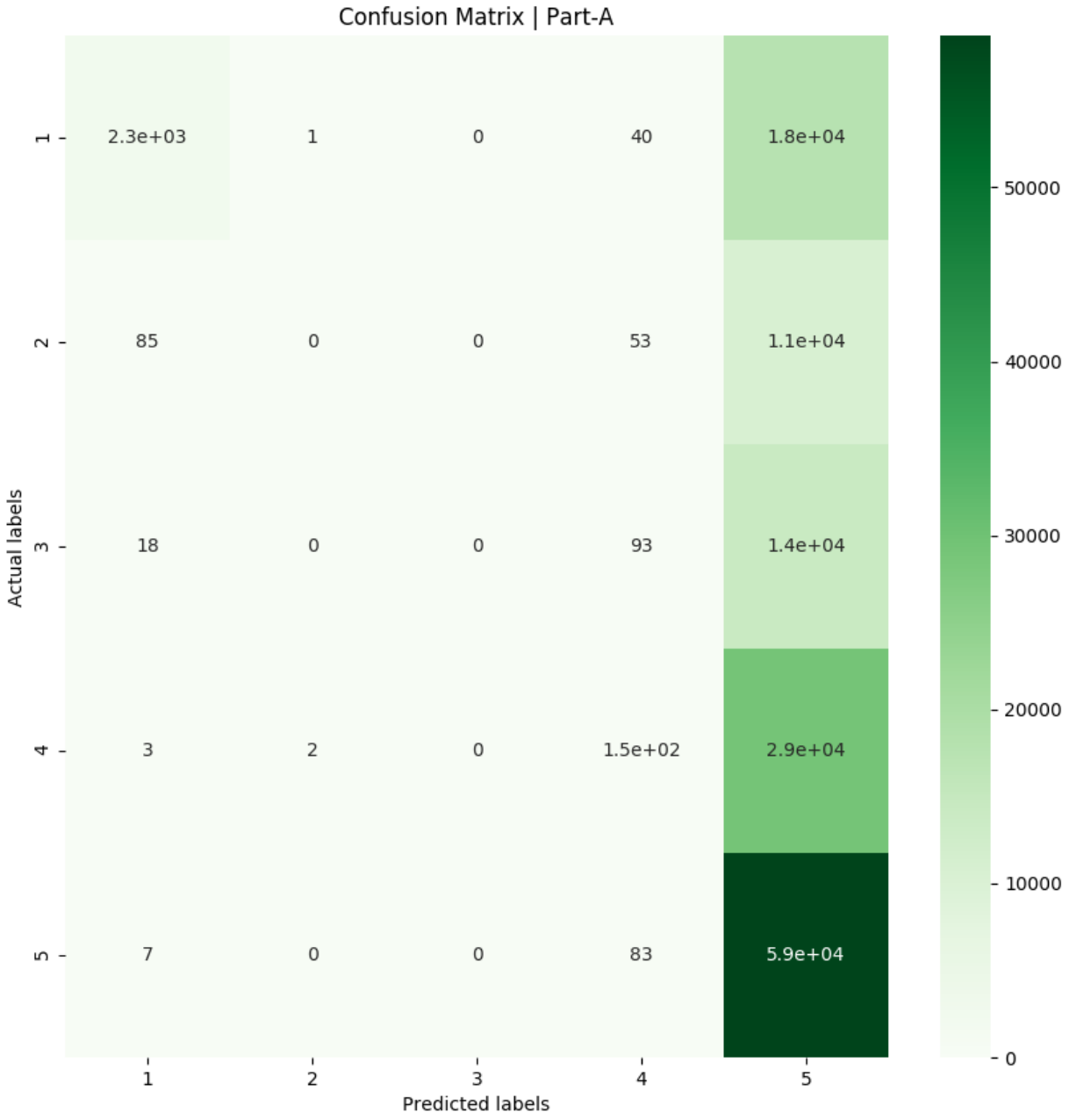
1.2 Part-b

```
[>] Predicting Randomly with 133718 examples!
[*] Accuracy on test set (predicting randomly): 0.1997

[>] Predicting Majority with 133718 examples!
[*] Accuracy on test set (predicting majority): 0.4399
```

My algorithm gives an improvement of 35.75% on *predicting randomly* baseline, and an improvement of 11.73% on *predicting majority* baseline.

1.3 Part-c

Confusion matrix (for *test data* only) in *part-a*:



Observations:

- Category 5 has the highest number of value in diagonal entry.
- It means that most of the labels that were predicted to be 5 are actually 5.

- 2 and 3 labels are predicted significantly lower than others, showing the model has not learned how to classify well between those.

1.4 Part-d

```
[*] Stopwords Removal = True and Stemming = True:
[>] Testing with 133718 examples!

[*] Accuracy on test set: 0.4772
```

Surprisingly, accuracy decreases. This may be a result of many factors, but as I see that, the model predicted good with more number of tokens to a string, and perhaps the stemming didn't help.

1.5 Part-e

Two alternative feature techniques I thought are of **bigram**, and with **essential linguistic words (using adverb, adjective, and verb only).**

```
[*] Feature Technique: Bigrams + Unigrams
[*] Stopwords Removal = True and Stemming = True:
[>] Testing with 133718 examples!

[*] Accuracy on test set: 0.5426

[*] Feature Technique: Extracting Essential Linguistic Features + Unigrams
[*] Stopwords Removal = True and Stemming = True:
[>] Testing with 133718 examples!

[*] Accuracy on test set: 0.4956
```

Comments: Which would help me in improving my overall accuracy: From the test accuracies, it looks like bigrams would help in improving overall accuracy of the model. //

Even though both feature extraction techniques work better than the one with just stemming and stopwords removal, but *bigram* seems to works very well with an accuracy of 54.26%.

1.6 Part-f

For my best performing model, which is bigrams + unigram: The scores are:

```
[.] Score per Class: [0.70108634 0.0009214  0.00219464 0.10909906 0.68081332]
[.] Macro Average Score: 0.2988229510920756
```

Metric best suited: Accuracy is more intuitive than F1 score, accuracy tells us correctly predicted by total observations, but this might not be that accurate, if we have an uneven class distribution in data, and False Positives and False Negatives are differently distributed. We have to see Precision (Correctly predicted positive / total predicted positive) and Recall (Correctly predicted positive / total observations). **tl:dr;** In an unevenly distributed data, F1 score is a good approximation.

So, I think F1 Score metric is best suited for over data. As the distribution is uneven, this is to say false positives are not distributed as false negatives.

1.7 Part-g

```
[*] Attention! Training over full YELP dataset!
[>] Training with 5348720 examples!

[>] Testing with 133718 examples!
[*] Accuracy on test set:  0.5456
[.] Macro Average Score: 0.268972
```

2 Part-B: MNIST Hand Written dataset Using SVM

2.1 Binary Classification

2.1.1 Part-a

Number of support vectors obtained after optimization (in-feasible and quite space filling, to write the set of support vectors) and accuracy on test set are:

```
[!] SVM -> Kernel: linear | C: 1.0
[*] 158 support vectors!
[*] Bias: 2.0839791

[*] Average accuracy on test set: 0.98985
```

2.1.2 Part-b

Number of support vectors and accuracy on test set are:

```
[!] Kernel: gaussian | Gamma: 0.05 | C: 1.0
[*] 954 number of support vectors!
[*] Bias: -0.890968

[*] Accuracy on test set: 0.99677
```

Comparison Entity	Linear Kernel	Gaussian Kernel
Accuracies (test set)	98.985 %	99.677 %

Gaussian kernel performs better than Linear kernel. If no issue of speed, use Gaussian. Also, it has been observed that if the number of features are more than the number of observations Linear performs well, and the other case tends to favor Gaussian kernel.

2.1.3 Part-c

Comparison between **Linear** and **Gaussian** kernel:

Comparison Term	Linear Kernel	Gaussian Kernel
# of SV	158	847
Computational Cost	0.9067064 *sec*	4.0570992 *sec*
Accuracies (test set)	99.031%	0.99.585%
Bias	2.0839791	-0.890968

As I reported above, **Gaussian** kernel performs better than **Linear** kernel but takes more computational time.

2.2 Multi-Class Classification

2.2.1 Part-a

```
[!] MultiSVM -> Kernel: gaussian | Gamma: 0.05 | C: 1.0
[!] Solving by CSXOPT!

[*] Accuracy on test set: 0.96940
[*] Accuracy on train set: 0.99495

[!] Computational time (of training): 1069.5375081830134
```

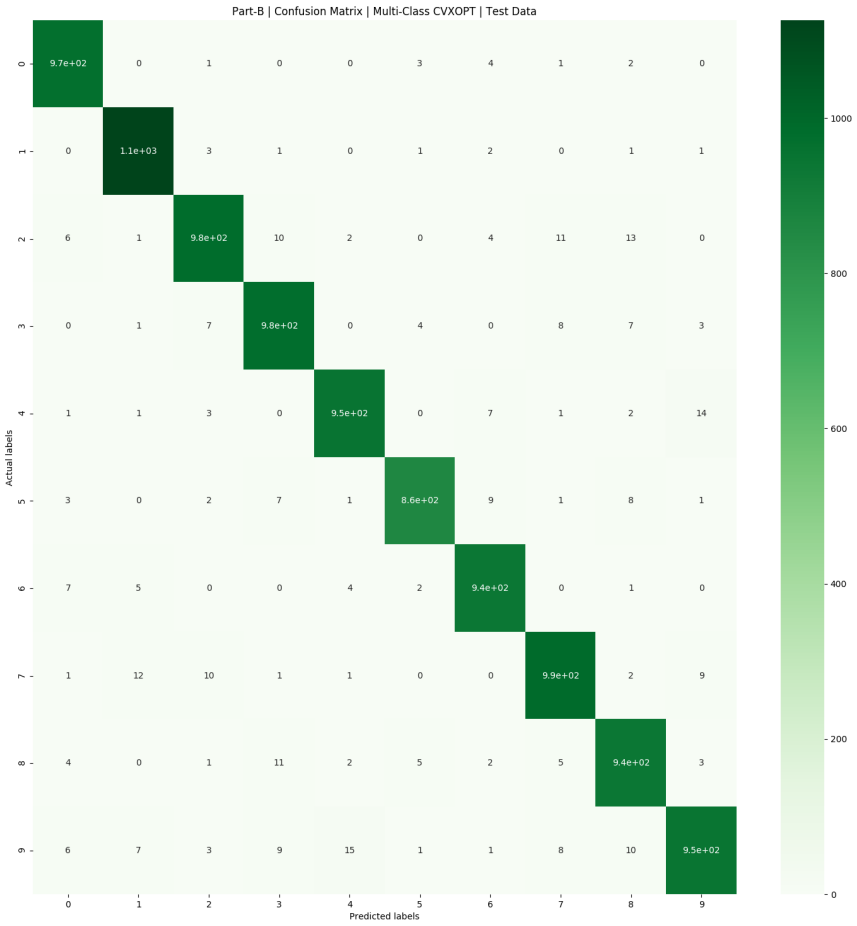
2.2.2 Part-b

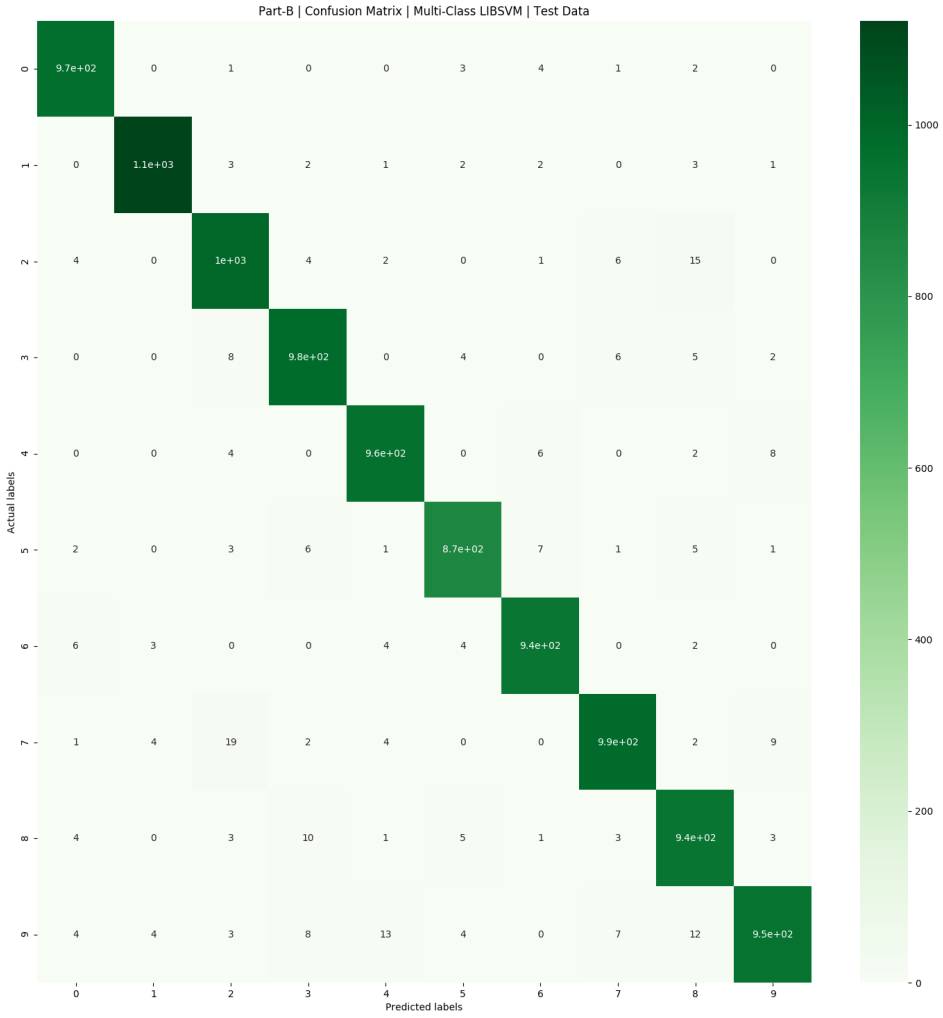
```
[!] MultiSVM -> Kernel: gaussian | Gamma: 0.05 | C: 1.0
[!] Solving using LIBSVM!

[*] Accuracy on test set: 0.97270
[*] Accuracy on train set: 0.99920

[!] Computational time (of training): 393.13890766199984
```

Comparison Term	Part-a	Part-b
Test Accuracy	96.940%	97.270%
Train Accuracy	99.495%	99.920%
Computational time (training)	1069.53750 *sec*	393.13890 *sec*

LIBSVM test and training accuracies are more than those by CVXOPT, which shows that LIBSVM optimises better than CVXOPT. The computational time is significantly lower than of CVXOPT. In conclusion, LIBSVM is efficient and performs better.

2.2.3 Part-c

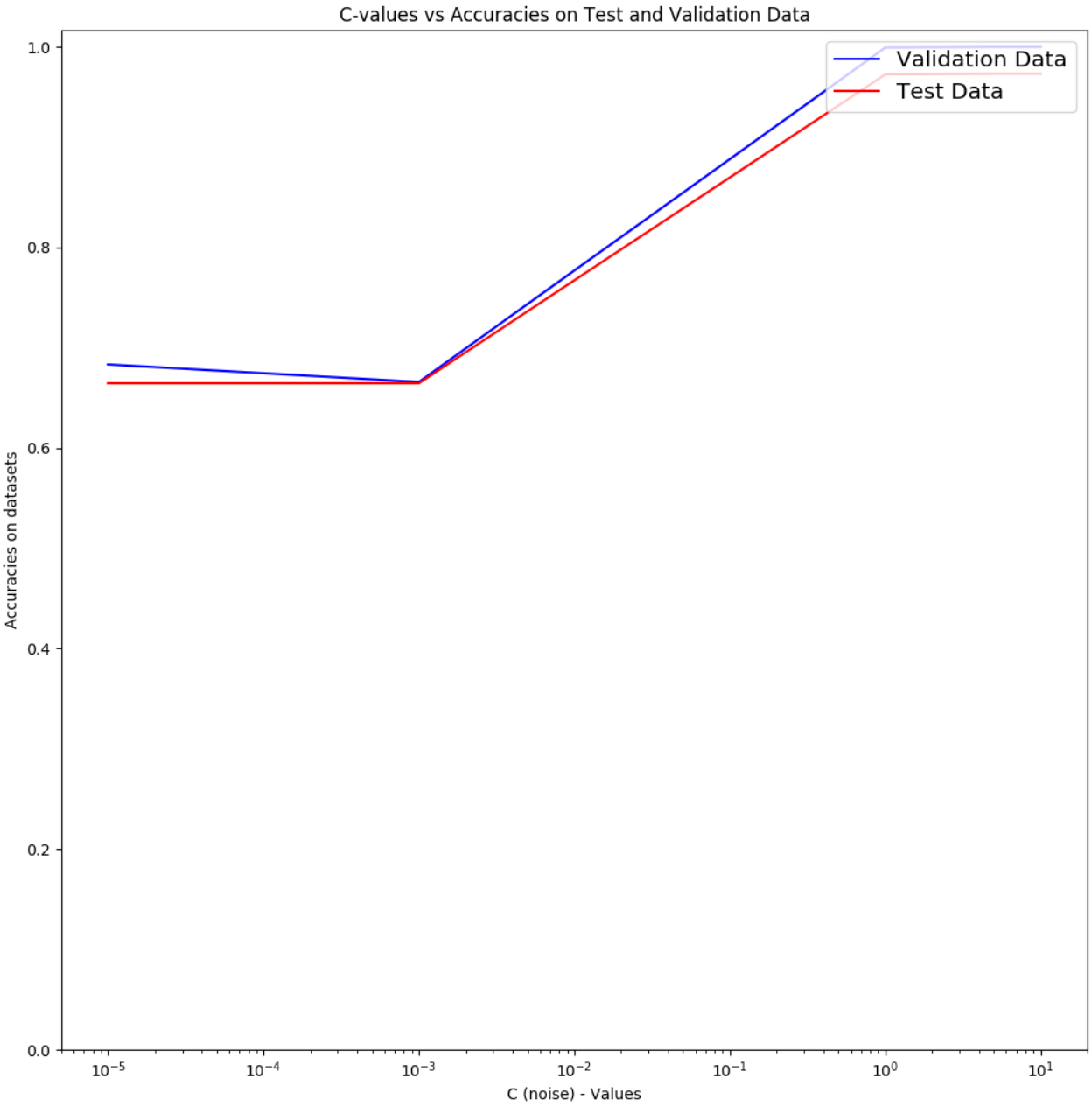




Observations about **Confusion Matrix**:

- The pairs **2-7 and 4-9** are mis-classified the most.
- Yes, the results makes sense. As 4 and 9 are written almost same with a free hand writing. The difference between 4 and 9 is the small joining of the free end of upper forks of 4. Similary, 2 - 7 are very look alike when written in a flow. The machine mis-classifies digits that even a human is likely to make the same mistake.

2.2.4 Part-d



Observations:

- Test and validation set accuracies for C = 0.001 and C = 0.00001 are significantly lower. Because of the decreased interval of *alpha* constraints.
- For C>=1, the value of Test and Validation accuacies saturates, and moves higher.
- For C=10 gives the hightest accuracy. But for even more higher values of C, the difference is so small its statistically insignificant to comment about it. As because the more we keep the interval of noise, after a certain point, it will not effect the value of *alphas*.
