SVM classifier for MNIST dataset

Yashad Samant
Master's of Electrical & Computer Engineering
Colorado State University

Abstract

In the last paper, we discussed and compared the performance of CNN and BPNN on MNIST dataset. In this paper, we will take the same dataset and evaluate the performance of SVMs with three different kernels namely: Linear, polynomial and Gaussian Radial Basis Function kernels. Each SVM has two tuning parameters namely the regularization parameter and gamma, which drastically affects the performance of the SVMs, thus using cross validation, we will obtain the best parameters for each kernel and find the result. We will compare the results obtained with the best tuning parameters with the results obtained from CNNs and BPNN in the last paper.

I. Introduction

Text recognition has a rich history in pattern recognition and natural language processing has become one of the booming machine learning fields lately. We can trace back its developments in 1980s when three-layer neural network were used to recognize the text only to achieve accuracy of 40%. After the development of CNN in the late 90s, CNN produced an error rate of 0.7% which was a significant improvement on the neural net. As the years passed by, CNN developed further and in 2012, AlexNet got a breakthrough accuracy of 92% on the MNIST dataset getting a huge improvement on all the previous models.
While we have seen various implementations of neural networks like CNN, K-NN, Maximum Posterior Probability, Back propagation Neural Network, Self-organizing map etc. to evaluate results on MNIST dataset, achieving higher accuracies as good as human eyes is still a myth.
In the last paper, we evaluated the performances of CNNs and BPNNs, found the best tuning parameters using cross validation and obtained accurate classification results.
In this paper, we reference the same problem and try to achieve higher accuracy by modelling different variants of SVM. Support Vector Machines form a kernel in a n-dimensional space and classifies the data in that space. We will learn more about SVM as we go further in this paper. The data used is MNIST dataset which has total of 60,000 training samples and 10,000 testing samples. More than 15% of the data was used for the practical implementation of SVMs in sklearn package in python. We also discuss the tuning parameters and its effect on the performance of various SVMs after validating it on the validation set.

II. Data

DATASET: The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a result of an anti-aliasing technique used by a normalization algorithm. These images were centers using the center mass of the pixels and later translated to form a 28x28 image. Sample image is as follows:

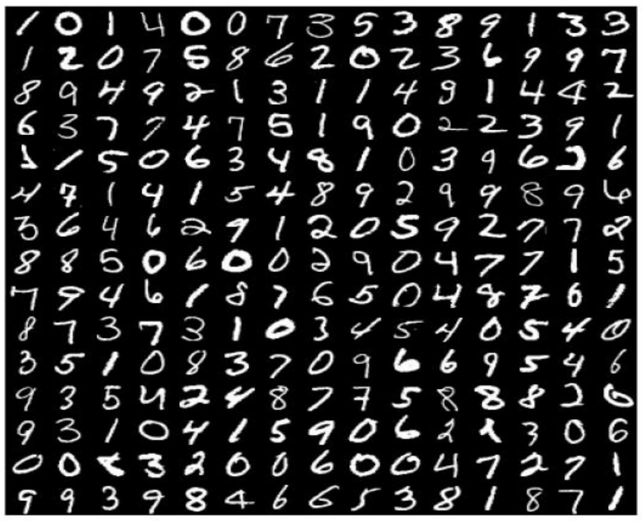


Fig 1: Sample data

To obtain relevant data for this assignment, we used the keras framework to read the whole data. Out of 60,000 samples, first 10,000 samples were selected as the training data and out of 10,000 samples from testing set, first 1000 were selected for testing and the next 1000 were selected for validation set. The number of samples were reduced to take into consideration the time constraint as training a network with 60000 samples will take hours of computations on a normal computer.
Standardization: The resultant data obtained from the preprocessing were sent through a standardization process. We implemented standardization by subtracting each value by the mean of the column and dividing by the standard deviation. Thus, the resultant data after standardization has zero mean and standard deviation equal to 1.
Standardization of data is very important as there's a possibility where the samples which are different range won't contribute equally to the analysis.
Finally, before going to the implementation of our algorithms let's go through the previous implementations and results on the MNIST dataset. The table is as follows:

TABLE I

PREVIOUS RESULTS ON MNIST DATABASE

Classifier	Preprocessing	Error rate (%)
Pairwise linear classifier	Deskewing	7.6 [12]
Boosted Stumps	Haar features	0.87 [14]
SVM	Deskewing	0.56 [15]
kNN	Shiftable edges	0.52 [13]
Neural Network	None	0.35 [16]
CNN	Width normalizations	0.23 [2]

III. ALGORITHMS

Support Vector Machines:

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. For example, in two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side. Support Vector machines get its name from the adjacent data points which helps the classifier decide the margin and build a hyperplane which can successfully classify all data points. These adjacent data points are called Support Vectors. These points become critical as the distance between the hyperplane and these points is maximized to find the right hyperplane.
Choosing the right hyperplane depends solely on the data. If the data is two dimensional and linearly separable, we can use linear SVC, otherwise, SVMs allows us to take data to a higher dimension and classify data.
If the data is very complex as in the image above, we can say that it's definitely not linearly separable. Thus, we transform this low-dimensional data into high dimensional data to find the right separation. This process is called the kernel trick and we obtain this process by defining a kernel.
There can be many types of complex kernels which can vary according to the data, but to maintain the scope of this assignment, we only discuss the major kernels like the linear kernel, polynomial kernel and Gaussian radial basis function kernel.

Linear kernel: The Linear kernel is the simplest kernel function. It is given by the inner product <x,y> plus an optional constant **c**. Kernel algorithms using a linear kernel are often equivalent to their non-kernel counterparts,
i.e. KPCA with linear kernel is the same as standard PCA.

$$k(x, y) = x^T y + c$$

Linear kernels are used to classify linearly separable data as shown below.

Polynomial kernel: The Polynomial kernel is a non-stationary kernel. Polynomial kernels are well suited for problems where all the training data is normalized.

$$k(x, y) = (\alpha x^T y + c)^d$$

Adjustable parameters are the slope **alpha**, the constant term **c** and the polynomial degree **d**.

RBF kernel: The Gaussian kernel is an example of radial basis function kernel.

$$k(x, y) = \exp\left(-\frac{||x - y||^2}{2\sigma^2}\right)$$

Alternatively, it could also be implemented using

$$k(x, y) = \exp\left(-\gamma ||x - y||^2\right)$$

The adjustable parameter **sigma** plays a major role in the performance of the kernel and should be carefully tuned to the problem at hand. If overestimated, the exponential will behave almost linearly and the higher-dimensional projection will start to lose its non-linear power. In the other hand, if underestimated, the function will lack regularization and the decision boundary will be highly sensitive to noise in training data.

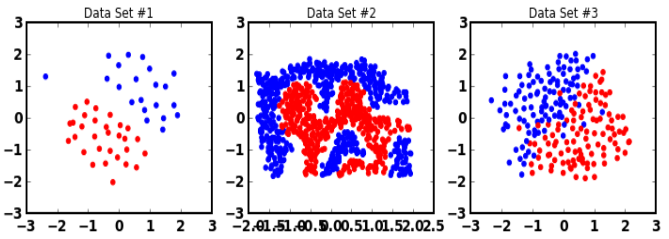


Fig 2: Random Data

In the above figure, there are three types of data, representing three different structures.
First data can be easily separated by a line and hence we can use a linear kernel to separate the data.
Second data is complex and if we use a linear kernel we won't be able to properly classify the data. In this case, we can either use a Gaussian RBF kernel or polynomial kernel of at least third order. Third order represents the value of d=3.
In such examples, RBF kernels are more preferred because of their accurate and robust output.
Polynomial kernels are mostly used in Natural Language Processing, kernel of order =2 is more common nowadays.

Tuning parameters

1. Regularization

The Regularization parameter (often termed as C parameter in python's sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example.
For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.
The images below (same as image 1 and image 2 in section 2) are example of two different regularization parameter. Left one has some misclassification due to lower regularization value. Higher value leads to results like right one.

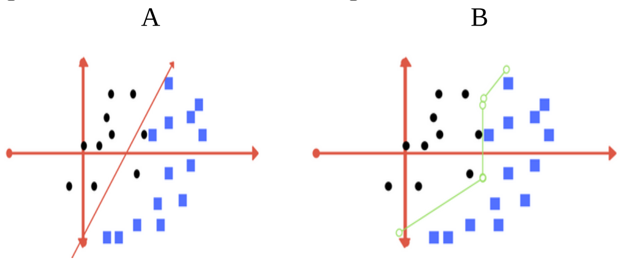


Fig 3: Low (A) & High (B) regularization

2. Gamma

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. In other words, with low gamma, points far away from plausible seperation line are considered in calculation for the seperation line. Where as high gamma means the points close to plausible line are considered in calculation.

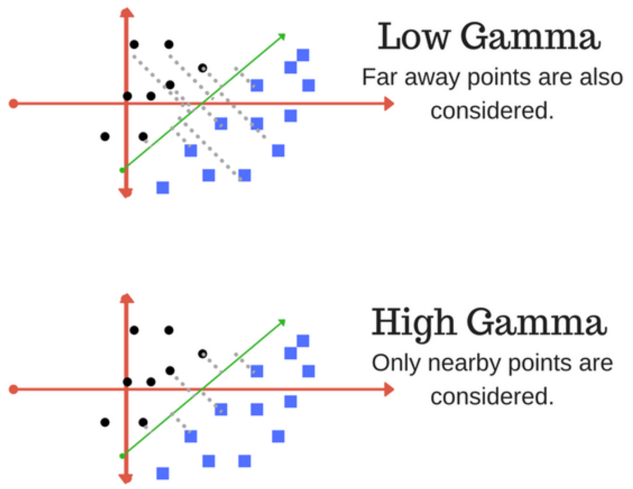

Low Gamma
Far away points are also considered.
High Gamma
Only nearby points are considered.

Fig 4: Gamma effect

IV. Experimental Results

All the three variants were successfully implemented using sklearn package in python.

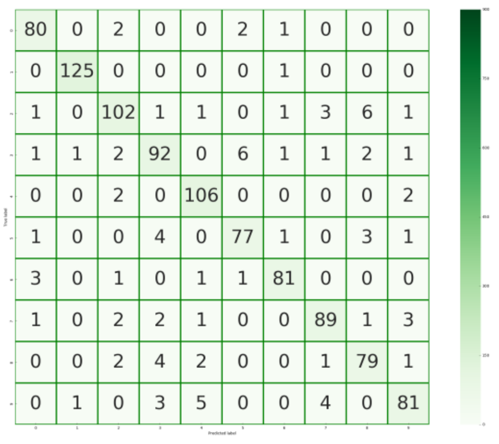
We used the SVM estimator in sklearn package to obtain the results. It gives us the ability to change the kernel name either to Linear, poly or RBF and also has flexibility with the tuning parameters.

We tuned the parameters using grid search, which is a function in sklearn itself. It makes different combinations of regularization parameter (C) and gamma and gives us the best tuning parameters among them. Thus, the parameters chosen for C ranged from -1 to 100 in logspace fashion. We randomly chose 10 values out of the many to get the best parameter. While for gamma we randomly took 10 value between 10^-3 to 10.
We ran the validation set for Linear, polynomial and RBF and found these to be the best parameters for the aforementioned methods respectively

Tuning parameters

1. Linear:

While we ran the grid search algorithm on the validation dataset made of 100 samples, the search iterated for 100 different combinations and found C=0.1 and gamma=0.0001 as the best values for Linear kernel. With C=0.1, the regularization of the model is not that high while gamma is low too with a value of 0.0001. Theoretically, we might assume not a good model overall but we observe that it has worked better than many by giving a validation accuracy of **86.38%.** It's a remarkable accuracy while considering very low C and gamma.
After achieving the parameters, we trained the Linear model with it and achieved a whooping accuracy of **91%.**
It worked well with the test data as well, giving better accuracy than the training dataset. We achieved an accuracy of **91.43%**. We also calculated the loss using Mean squared error and we obtained loss of 1.436. The confusion matrix is depicted below:

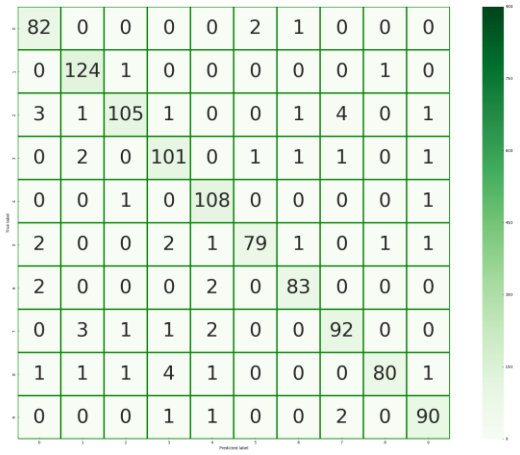


80	0	2	0	0	2	1	0	0	0
0	125	0	0	0	0	1	0	0	0
1	0	102	1	1	0	1	3	6	1
1	1	2	92	0	6	1	1	2	1
0	0	2	0	106	0	0	0	0	2
1	0	0	4	0	77	1	0	3	1
3	0	1	0	1	1	81	0	0	0
1	0	2	2	1	0	0	89	1	3
0	0	2	4	2	0	0	1	79	1
0	1	0	3	5	0	0	4	0	81

Fig 5: Linear confusion matrix

2. *Polynomial:*

Polynomial kernel with order 3 has worked the best among the three kernels. We expected RBF to perform the best but we didn't really achieve expected results when it comes to RBF paving way for polynomial kernel to win the competition. While we ran the grid search algorithm on the validation dataset made of 100 samples, the search iterated for 100 different combinations and found C=2.5 and gamma=0.1 as the best values for Polynomial kernel (d=3). With C=2.5, the regularization of the model is high enough while gamma is also much higher than both RBF and Linear kernels. We achieved the lowest validation accuracy for this kernel with 83.7%.
But once we trained the model with updated parameters, it gave a training accuracy of **94%** and testing accuracy of **94.39%.** MSE for this kernel was 1.084. The confusion matrix is depicted below:

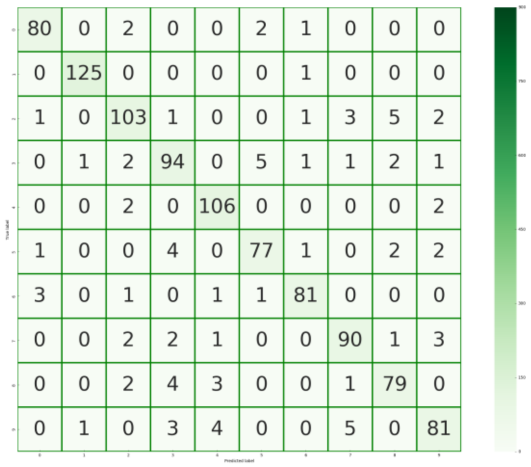


82	0	0	0	0	2	1	0	0	0
0	124	1	0	0	0	0	0	1	0
3	1	105	1	0	0	1	4	0	1
0	2	0	101	0	1	1	1	0	1
0	0	1	0	108	0	0	0	0	1
2	0	0	2	1	79	1	0	1	1
2	0	0	0	2	0	83	0	0	0
0	3	1	1	2	0	0	92	0	0
1	1	1	4	1	0	0	0	80	1
0	0	0	1	1	0	0	2	0	90

Fig 6: Polynomial confusion matrix

3. *RBF*

RBF didn't perform up to it's expectations, with second highest test accuracy. Surprisingly, while validating the model, we achieved C to be 1.7782*10^24 and gamma=0.0001. While gamma is not going to affect the model much, C is going to change the model drastically. Also, without regularization, we got an accuracy of 26% for training data while implementing RBF kernel. This proves that regularization is very important for proper modelling of SVM-RBF kernel
We achieved a validation accuracy of 86.38% and a training accuracy of **92%.**
On the test data, we received an accuracy of **92.6%** and MSE error of 1.327. The confusion matrix is depicted below:



80	0	2	0	0	2	1	0	0	0
0	125	0	0	0	0	1	0	0	0
1	0	103	1	0	0	1	3	5	2
0	1	2	94	0	5	1	1	2	1
0	0	2	0	106	0	0	0	0	2
1	0	0	4	0	77	1	0	2	2
3	0	1	0	1	1	81	0	0	0
0	0	2	2	1	0	0	90	1	3
0	0	2	4	3	0	0	1	79	0
0	1	0	3	4	0	0	5	0	81

Fig 7: RBF confusion matrix

Thus, when we compare the models of SVM, we observe that polynomial kernel SVM works best with MNIST data.

While we have compared the performance of the SVM kernels, with the best parameters, now we have to compare it with performances of CNN and BPNN. In the previous assignment, we obtained the computation times for CNN and BPNN and plotting the time taken by each method, we get the following plot.

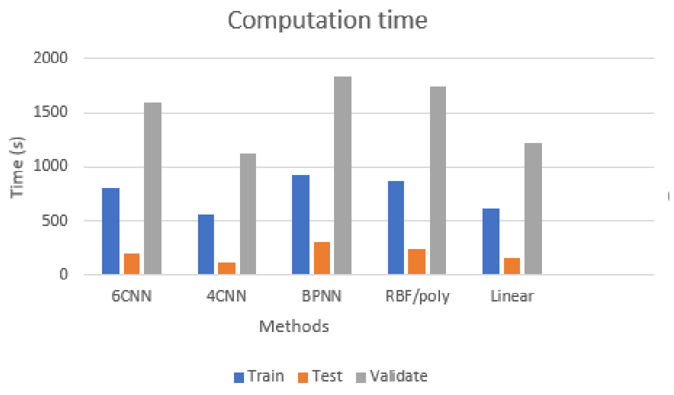


Fig 8: Computation time

In the above plot the computation time of each algorithm is plotted with standards as mentioned in the implementation. The y-axis is the time in seconds. As BPNN contains fully-connected layers, it still takes the highest time among all other methods. BPNN also had a large number of tuning parameters unlike other methods, taking a lot of time to validate the set. Being a fully-connected layer, it takes ample of time to converge. Second highest convergence goes to RBF and poly kernel SVMs while next in line is the six-layer CNN. Four-layer CNN and Linear kernel takes the lowest time among the others. Thus, when it comes to time four-layer CNN gave us the best result as it gave a very high accuracy for the time.

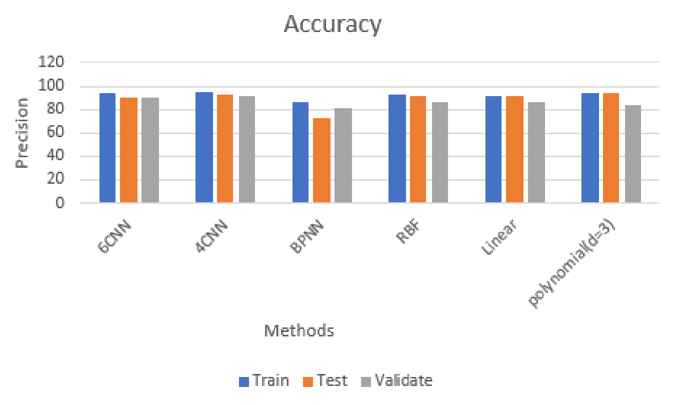


Fig 9: Computation performance

In the above graph, we have plotted the train, test and validation accuracy. SVMs tend to have less validation score but it works well on training and test data-set. BPNN is the worst performer of all as the number of fully-connected layers were not enough to train the system properly for images. All other techniques have an accuracy over 90% which can be considered a decent count.

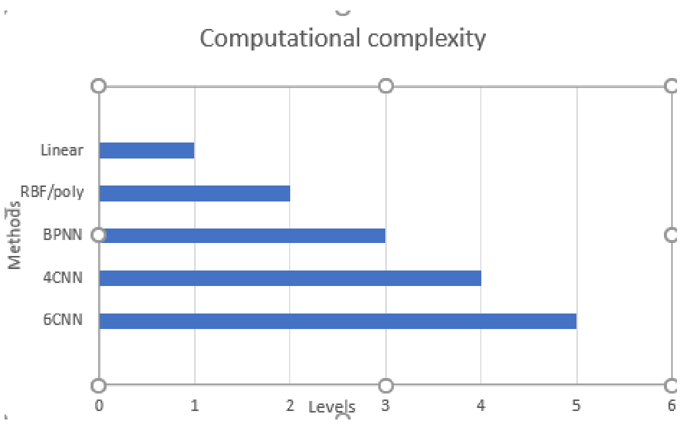


Fig 10: Computational complexity

Computational complexity gives us the measure of how complex the multiplications are going to be and how much pressure it's going to put on the processor.
CNNs has the highest complexity due to the convolution layers and iterated for loops.
BPNN is more complex than SVMs because of the extra hidden layer. It's complexity will be around x*K +K*y, x is the number of inputs, y is the number of outputs and K is the number of hidden neurons. Complexity is also dependent on the number of hidden layers for BPNN.
RBF SVM has a complexity dependent onto dimension but a square of dimension but linear SVM is dependent onto data dimension as well but not the square.
Usually in machine learning, it becomes complex to determine the complexity and it changes for every algorithm and for every data. Also, to determine complexity, model has to converge.

Advantages & Disadvantages:

1. *SVM*

Pros:

- It works really well with clear margin of separation
- It is effective in high dimensional spaces.
- It is effective in cases where number of dimensions is greater than the number of samples.
- It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Cons:

- It doesn't perform well, when we have large data set because the required training time is higher
- It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
- SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python scikit-learn library.

2. *CNN*

Pros
- Very high accuracy
- Easy implementation
- Best recognition for images
- Very flexible in terms of tuning parameters and changing structure because of the cascading

Cons
- Computationally complex
- Convolution is slow process so takes more time to train
- Huge amount of data required

3. *BPNN*

Pros
- Very complex model can be implemented
- Works well with both linear and non-linear model

Cons
- Overfitting
- More computational time
- Huge data required to train
- Computationally complex

V. Conclusion

Thus, in this assignment we have successfully implemented various kernels of SVM in python using sklearn module. We successfully discussed the advantages and disadvantages of each of the kernels and also observed the effects of tuning parameter C and gamma on the kernels. We successfully compared all the kernels with CNNs and BPNNs and compared the complexity, time and accuracy of all the methods. We found out that four-layer CNN works best among all the methods. We also discussed the pros and cons of each algorithm.

VI. References

1. https://plon.io/explore/svm-mnist-handwritten-digit/USpQjoNcO8QHlmG6T
2. https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
3. http://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score
4. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
5. http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex8/ex8.html
6. http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
7. http://scikit-learn.org/stable/modules/cross_validation.html
8. http://sdsawtelle.github.io/blog/output/week7-andrew-ng-machine-learning-with-python.html