Assignment 4: Support Vector Machines for MNIST recognition

Harshal Priyadarshi
hp7325

April 8, 2016

1 Overview

Support Vector Machines is an effective linear method for classifying data in a higher dimensional space. It is also called Sparse Kernel Machines because after projection into higher dimensions using kernels, only few vectors are representative of the model, also called Support Vectors. That is in essense sparseness, as only few support vectors are used to classify the entire data. Character recognition is a solved problem and has very high accuracy with deep methods like CNN, however, essentially linear models can also achieve a very high accuracy, as we will see in the case of SVM. I will be varying several parameters to check the trend in test accuracy with the varying parameters.

2 Algorithm

1. Take some portion of training data, and then create a set of top k-eigenvectors using the eigenvector trick discussed in the previous assignments.
2. Project the 784 pixel features into the space of these top k-eigenvectors and use that as the new set of features for the support vector machines.
3. Train the SVM model with different parameters on the transformed training data.
4. Test the SVM model on the transformed test data

The parameters that are varied in the SVM model for the digit classification are shown in the next section.

3 Parameters

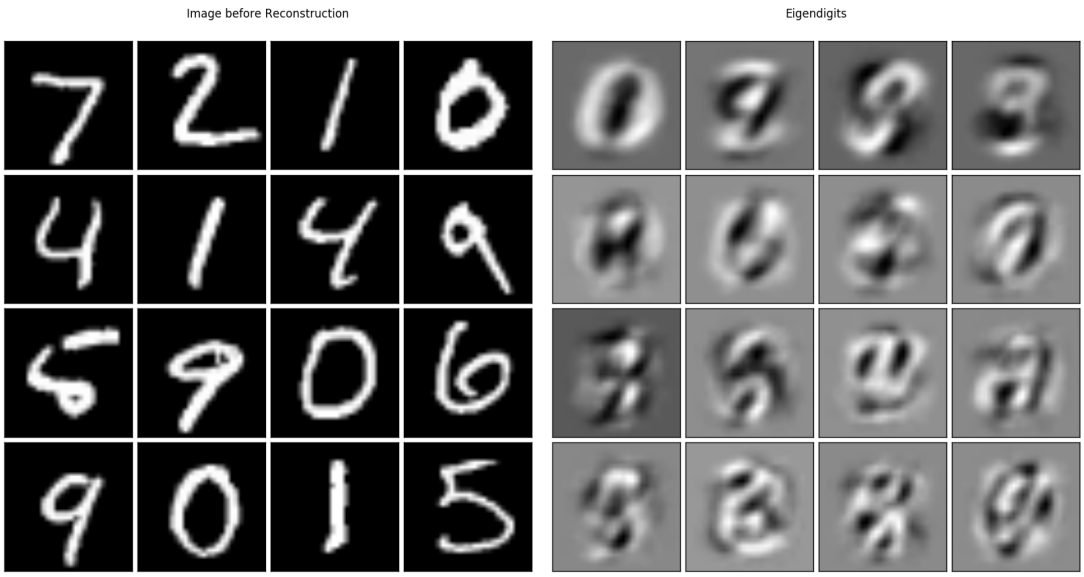
- C = penalty coefficient of the error term (basically the inverse of the regularization coefficient) in Soft-SVM
- gamma = The coefficient of kernels(specifically rbf, polynomial and sigmoid).
- kernel = the kernel used for feature projection into higher dimension space. The kernels trained upon are:
  - Radial Basis Function kernel
  - Sigmoid kernel
  - Polynomial kernel
  - linear kernel
- class_weight = the ratio of the frequency of labels, makes the SVM incorporate the fact that the training data is unbalanced and thus can balance it. I keep this factor constant and set it to balancing mode.

4 Experiment

The following experiments were conducted:

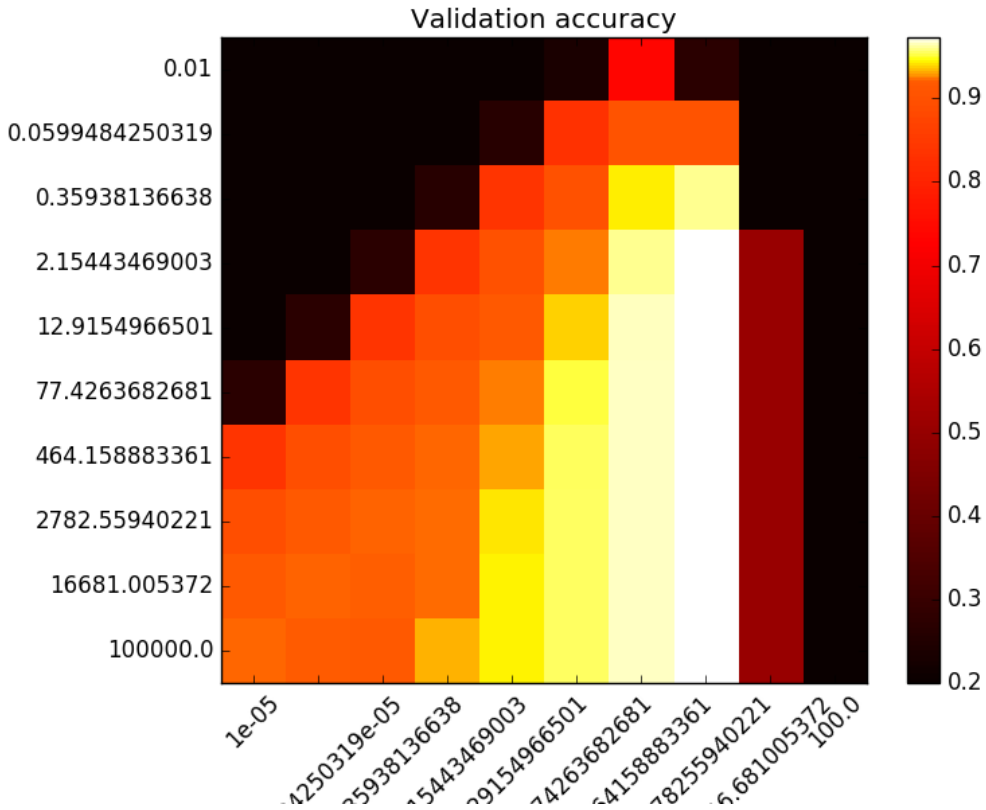
- Varying C and gamma simultaneously to see there cumulative effect
- Varying the kernel and see their effect on the best C and gamma obtained from the previous experiment
- Effect of increasing size of eigenvector space on the test accuracy for the best C, gamma and kernel obtained in the previous 2 experiments
- Effect of One vs One, One vs All training methodology effect on the testing accuracy.
- Final table of the best SVM for the data

The original images from the MNIST data, and the data after the projection are shown below:



4.1 Varying C and Gamma

I varied the C and gamma to see the effect of performance on the SVM accuracy, and wanted to see their combined effect. Thus I plotted the heat map of accuracy for the two parameters on the cross-validation part of the training data. Throughout the model selection process of Experiment 1 - 4, I will not touch the test data, as ideally we dont have labels for it. The plot looks like this.



The best results were obtained for $C = 2.154435, gamma = 2.782559$.

4.2 Varying Kernel

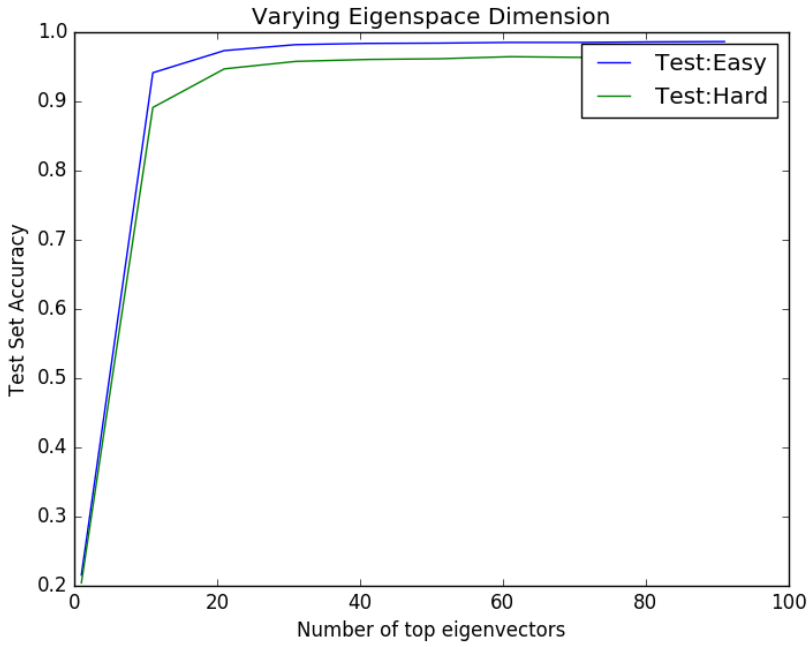
Now given the best values from the previous experiments, I wanted to see as to which kernel is the best. The general notion is that RBF kernel performs better than linear kernels. The results obtained after doing 5 fold cross-validation, on the cross-validated data is, as follows:

Kernel	Accuracy(%)
Linear	91.94
Polynomial	97.05
RBF	97.00
Sigmoid	57.2

Thus the best performance is obtained by the polynomial kernel.

4.3 Varying Eigenspace size for projection

Now varying the projection space, i.e. the number of top k-eigenvectors to project to, I checked the performance on the actual test data. I wanted to see that how much compression in feature space can be achieved without effecting my accuracy. The results are shown below, as the performance for both easy and hard test data.



4.4 OVO vs OVR

Now once the model is obtained, i.e kernel is fixed to polynomial kernel, the $C = 2.154435$, $gamma = 2.782559$, the only parameter left to tune is the decision function. There are basically 2 types of decision functions, the One vs One decision function and the One vs All. The One vs One decision function basically compares each class to other, and then does the final voting, to get the final class with the most votes. The One vs All decision function compares each class with the rest, and then assigns the sample to the class with the maximum probability among all the N classes. OVO is very computationally expensive, O(n^2). OVR is linear in complexity, O(n). Thus choosing decision function makes a difference.
The resulting accuracies obtained was:

Decision Function	Accuracy(%)
One vs Rest (OVR)	97.25
One vs One(OVO)	97.25

Surprisingly both the models performed similarly. This shows that the actual data is not very noisy, and both the models either determine the two classes or fails to do so. Thus the best choise would be to choose the OVO policy.

4.5 Best Results

Now finally to see as to which algorithm did the very best, we decided to vary all the 5 parameters simultaneously and see as to what is the best we can achieve. The best results obtained were:

Train Accuracy	100%
Test Accuracy (Hard Data)	96.36%
Test Accuracy (Easy Data)	98.62%

5 Observation & Justification

The following are the interesting obervations followed by the justifications that I feel is the best in my opinion:

- The RBF kernel beats the linear kernel by a large margin. This is essentially justified, and almost guaranteed if the hyperparameters for RBF kernel are chosen properly, because, linear kernel is a degenerate type of the RBF kernel, and thus can't perform better than RBF kernel, as under certain conditions RBF kernel becomes linear kernel.

- Polynomial kernel just beats the RBF kernel by a very small amount, thus most generally RBF should be the default choice when the feature space is large, or when there are too many observations.

- Increasing the effective feature size(= top k-eigenvectors), improves the performance till say 20 eigenvectors and then saturates. This shows that we can reduce the feature space from 784 dimensions to only 20 dimensions and still get equivalent accuracy. Thus the PCA is an essential component in reducing the compute time.

- It is visible from the heatmap that increasing C for a given gamma, improves performance upto a certain value, after which the performance drops. Thus the best C obtained was 2.15 instead of 10^7. This is justified, as higher C ensures that all the training data is nicely classified, while lower C ensures that the decision boundary is smooth. We want good classification as well as smooth boundary to prevent overfitting. Thus this performance by the algorithm is justified, to have a good performance somewhere in the middle.

- Increasing gamma improved performance, and then very high gamma lead to the loss in performance. Intuitively, the gamma parameter defines how far the influence of a single training example reaches, with low values meaning far and high values meaning close. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. Thus the observation is justified as now, we need gamma to be high, to ensure that the support vectors cover most of the region, but we dont want them to high enough that each support vector covers the entire decision space, and thus leads to poor accuracy.

- We see that the OVO and OVR performs equally well. This shows that the classification algorithm is not confused much. It either confidently classifies it wrong, or confidently classified it right. Thus we should choose OVO as it gives better accuracy.

- our model is not overfitting. Though train accuracy is 100%, test accuracy is close to 97%

6 Results

Thus we have achieved a very high accuracy close to gold standard for this problem by essentially tuning the hyperparameters of CVM sequentially using cross-validation and then fitting and prediciting using the resultant model.