CS 391L Machine Learning **Nidhi Kadkol**
Assignment 3 **UT EID: nk9368**
March 2019 **Email id: nidhik@cs.utexas.edu**

1 Introduction

In this project we train different kinds of Support Vector Machines (SVM) on the MNIST dataset and use the Sequential Minimization Optimization (SMO) Algorithm for training SVMs in their dual form. We use a form of simplified SMO as given by Andrew Ng's Stanford lecture notes. We experiment with polynomial kernels and Radial Basis Function (RBF) kernels, and compare the performance of different polynomial degrees and RBF kernel sizes. We also look at the difference in accuracy between hard and soft margins. We have carried out these experiments on the MNIST dataset, hence we divide out experiments into one vs all (eg. classifying every digit as 0 or not 0) and one vs one (choosing 2 numbers from the dataset and discarding all samples which are neither of the 2 numbers and performing SVM to separate out the 2 numbers). Lastly, we analyze the performance of SVM compared to the performance of PCA.

2 Method

2.1 Primal and Dual Form - Hard Margin

The primal form of the SVM is written as follows:

$$\min_{w,b} \frac{1}{2}||w||^2$$
$$\text{such that } \ y^{(i)}(w^T x^{(i)} + b) \geq 1 \text{ for } i = 1, \cdots, m$$

where there are m training samples, $x^{(i)}$ denotes the i^{th} training sample, and $y^{(i)} \in \{1, -1\}$ denotes its label. From this, we find the best $\hat{w}$ and $\hat{b}$, and then to classify a new feature vector x, we predict

$$\hat{y} = \text{sign}(\hat{w}^T x + \hat{b})$$

Instead of optimizing this, we can optimize the dual form of this problem, which allows us to use kernels and project the feature vectors into higher dimensional spaces. The dual optimization problem also works much better than using a quadratic programming code for the original problem. The dual problem is as follows:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$
$$\text{such that } \alpha_i \geq 0 \text{ for } i = 1, \cdots, m$$
$$\text{and } \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

From this we find the best $\hat{\alpha}_i$s and to classify a new feature vector we predict

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{m} \hat{\alpha}_i y^{(i)} \langle x^{(i)}, x^{(j)} \rangle + \hat{b} \right)$$

.

In this form, we can replace $\langle x^{(i)}, x^{(j)} \rangle$ by a kernel $K(x^{(i)}, x^{(j)})$.

2.2 Primal and Dual Form - Soft Margin

In case the data is not linearly separable, we employ a softer margin allowing some points to lie on the wrong side of the separating hyperplane. We use L1 regularization and get the following primal optimization problem:

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i$$

$$\text{such that } \; y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \text{ for } i = 1, \cdots, m$$

$$\xi_i \geq 0 \text{ for } i = 1, \cdots, m$$

This gives us a dual problem of

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)}, x^{(j)}\rangle$$

$$\text{such that } 0 \leq \alpha_i \leq C \text{ for } i = 1, \cdots, m$$

$$\text{and } \sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$

2.3 Simplified SMO Algorithm

We follow the SMO algorithm as given in the Stanford document. We use coordinate ascent to find the best set of α's which optimize the dual problem. In coordinate ascent, if we have to optimize a set of parameters, we choose one parameter at a time and take a step to optimize that parameter, keeping all other parameters fixed. However, since we have the constraint that $\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$, we have to choose 2 α_i's to optimize at a time, keeping the rest fixed.

We can use heuristics to decide which 2 to pick at a time, however in our simplified version we will pick the first α_i that does not satisfy the KKT conditions to within some numerical tolerance and pick α_j at random from the remaining $m-1$ α's and jointly optimize α_i and α_j. If the values we get are outside of the bounding box conditions of L and H, we clip the values to lie between those conditions.

3 Results

3.1 Polynomial Kernel

The polynomial kernel is

$$K(x, z) = (x \cdot z + 1)^p$$

where x and z are 2 training samples and p is the degree of the polynomial. We keep a soft margin of C = 0.5, numerical tolerance for KKT conditions = 0.1, and vary the degree of the polynomial by training a one vs rest SVM to distinguish 0 from the remaining digits. We use 1000 training samples and all the test samples.

Kernel	Accuracy (%)	True Positives (%)	Number of Passes to Converge
dot	98.73	93.88	148
degree 1	98.78	93.27	234
degree 2	99.25	94.49	214
degree 3	89.44	98.47	4
degree 4	82.00	99.59	2
degree 5	80.99	99.18	2
degree 6	82.79	99.18	2

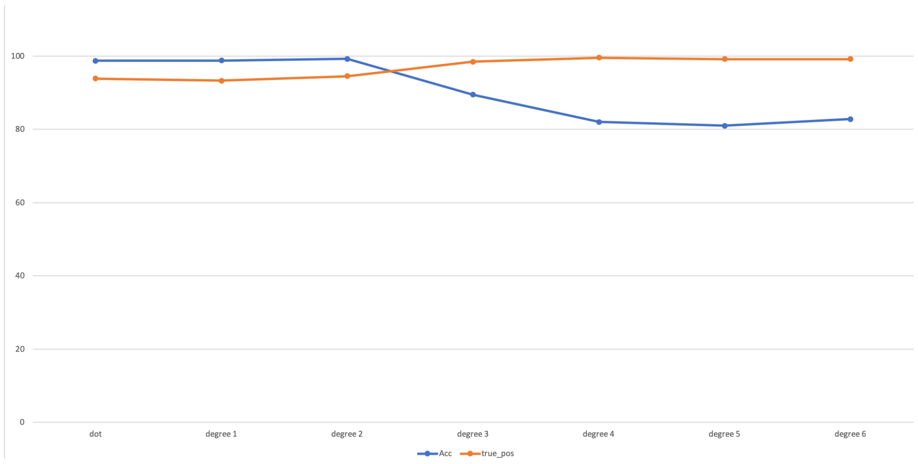

Figure 1: Plot of change in accuracy and percentage of true positives vs degree of polynomial kernel. Blue line shows accuracy, orange line shows percentage of true positives.

The first data point is simple dot product, and after that we vary the degree from 1 to 6. Since we are doing one vs rest, the dataset is quite imbalanced. So, in addition to tracking accuracy we also keep track of percentage of true positives. We see that while the accuracy decreases with an increase in polynomial degree, the percentage of true positives increases, indicating a better performance.

3.2 Radial Basis Function kernel

The form of the RBF kernel is

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

where where x and z are 2 training samples and σ is called the RBF size of the kernel. We vary the value of σ to see the change in accuracy. We keep a soft margin of C = 0.5, numerical tolerance for KKT conditions = 0.1, and vary the degree of the polynomial by training a one vs rest SVM to distinguish 0 from the remaining digits. We use 1000 training samples and all the test samples.

Sigma	Accuracy (%)	True Positives (%)	Number of Passes to Converge
1	90.20	0.00	17
2	90.22	0.20	15
3	95.60	55.10	18
4	98.77	87.86	29
5	99.29	93.67	46
10	99.21	94.80	85
15	99.07	93.57	73
20	98.73	89.49	88
25	98.72	89.39	84
30	98.47	86.43	91

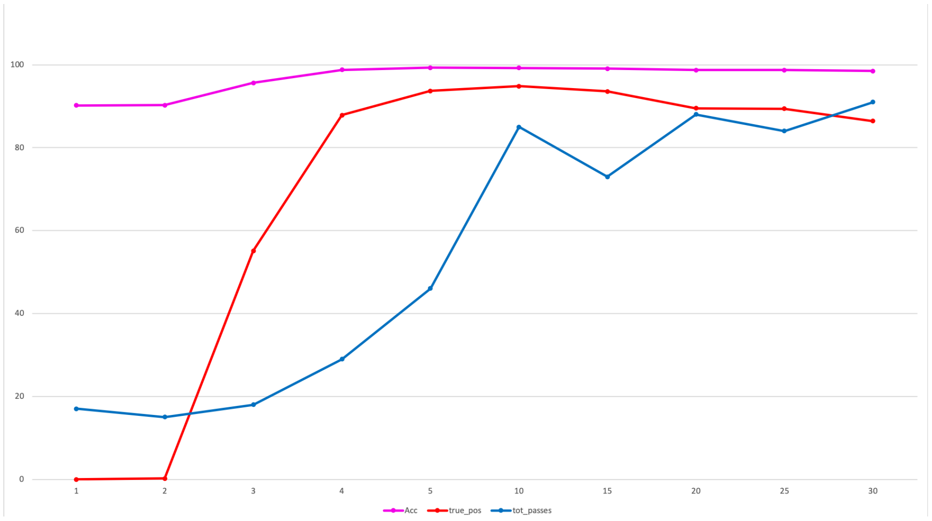

Figure 2: Plot of change in accuracy and time to converge vs size of RBF kernel. Pink line shows accuracy, red line shows percentage of true positives, and blue line shows number of passes required to converge.

We run experiments for sigma values = [1, 2, 3, 4, 5, 10, 15, 20, 25, 30]. Since we are doing one vs rest, the dataset is quite imbalanced. So, in addition to tracking accuracy we also keep track of percentage of true positives. We also note the total number of passes required for convergence. We see that the accuracy and true positive percentage drastically increase until $\sigma = 5$, after which it stabilizes. The number of passes required for convergence increases until $\sigma = 10$ after which the value fluctuates.

3.3 Hard vs Soft Margin

We now run a series of experiments and compare hard and soft margins. We use the RBF kernel with $\sigma = 5$, 1000 training samples and the entire test set, and use a numerical tolerance of 0.1. For hard margins, we set C to infinity, and for soft margins we set C=5.

3.3.1 0 vs rest

Metric	**Soft**	**Hard**
Accuracy (%)	99.30	99.32
True Positives (%)	93.88	93.88
Passes to converge	55	43

We see that 0 vs rest hard margin performs marginally better than soft margin, indicating that the data is separable for this hyperplane division.

We also chart which digits were predicted as the positive class (0) incorrectly. From the table below we see that 0 is confused with 5, 2, 6, and 8. This intuitively makes sense, as all these digits have curves in their shapes.

Confused 0 with	Soft (%)	Hard (%)
1	0.00	0.00
2	0.29	0.29
3	0.00	0.00
4	0.00	0.00
5	0.34	0.34
6	0.21	0.10
7	0.00	0.00
8	0.10	0.10
9	0.00	0.00

3.3.2 7 vs rest

Metric	**Soft**	**Hard**
Accuracy (%)	98.43	98.42
True Positives (%)	88.04	87.84
Passes to converge	74	59

The accuracy is marginally better with a soft margin than a hard margin for 7 vs rest, and the percentage of true positives is much more with a soft margin. This shows that a perfectly separating hyperplane does not exist for separating out 7 from the rest of the digits.

We also chart which digits were predicted as the positive class (7) incorrectly. From the table below we see that 7 is confused with 9, 2 and 5 the most. 9 and 2 are likely because of the slanting line from top right to bottom left, and 5 because it also has a horizontal line at the top like 7.

Confused 7 with	Soft (%)	Hard (%)
0	0.10	0.10
1	0.00	0.00
2	0.78	0.68
3	1.09	1.09
4	0.00	0.00
5	0.45	0.45
6	0.00	0.00
8	0.10	0.10
9	0.89	0.89

3.3.3 4 vs 9

Taking 1000 training examples after shuffling the training data, there were 499 4s and 501 9s in the train set, and 982 4s and 1009 9s in the test set.

Metric	**Soft**	**Hard**
Accuracy (%)	97.84	97.74
True Positives (%)	97.86	98.07
Passes to converge	54	44

We see that although it takes longer to converge, the performance with a soft margin is significantly better than that with a hard margin. This shows that 4 and 9 are not perfectly separable, and using a softer margin gives us better results.

3.3.4 0 vs 8

Taking 1000 training examples after shuffling the training data, there were 483 0s and 517 8s in the train set, and 980 0s and 974 8s in the test set.

Metric	**Soft**	**Hard**
Accuracy (%)	99.38	99.12
True Positives (%)	99.39	99.39
Passes to converge	42	49

For 0 vs 8, we see that a soft margin gives us marginally better results than a hard margin. A soft margin also converges faster than a hard margin.

3.3.5 0 8 3 vs 1 7 9

Taking set 1 as [0,8,3] and set 2 as [1,7,9], we take 1000 training examples after shuffling the training data and get 500 samples belonging to each set. In the test data, we have 2964 samples from set 1 and 3172 samples from set 2.

Metric	**Soft**	**Hard**
Accuracy (%)	97.82	97.72
True Positives (%)	97.44	97.37
Passes to converge	43	49

In all 3 respects as seen in the table above, a soft margin performs better than a hard margin.

3.3.6 Variation of accuracy with number of training samples

We look at the change in accuracy and percentage of true positives for an RBF Kernel SVM with C=5, tol=0.1, σ = 5 on 1000 training samples and 2000 test examples.

Number of training samples	Accuracy (%)	True Positives (%)
10	91.35	1.14
50	91.85	6.86
100	95.19	45.71
500	98.75	86.29
1000	99.00	90.29
2000	99.50	95.43

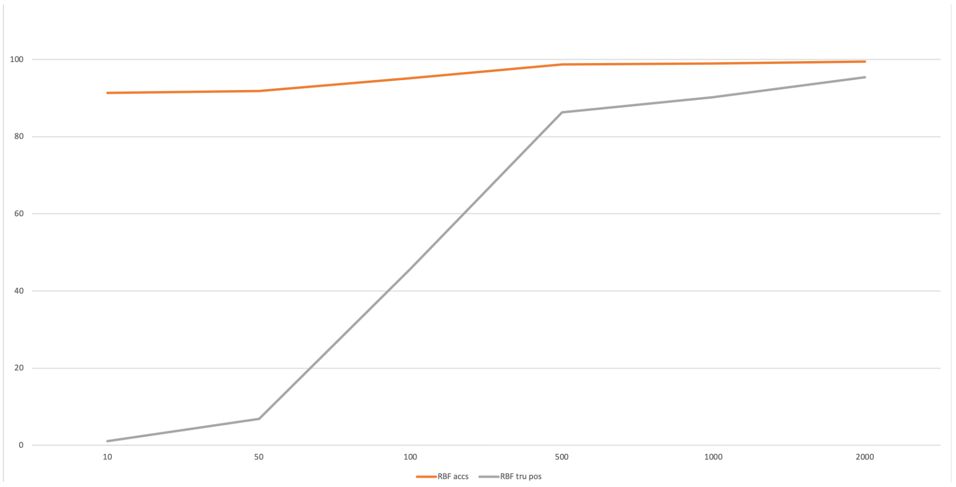

Figure 3: Plot of accuracy and percentage of true positives vs number of training examples. Orange line shows accuracy and grey line shows percentage of true positives.

As expected, with more training data, the model performs better - both in terms of accuracy and percentage of true positives.

3.3.7 Comparison with PCA

In assignment 1, we used Eigen Vector decomposition into Principal components to project the images into a lower-dimensional space. Here, we use PCA to project the 784 dimensional images onto a 200 dimensional vector and we train an SVM on these new images. We use an RBF Kernel SVM with soft margin (C = 5, tol = 0.1, σ = 5) trained on 1000 training samples and tested on the entire test dataset. We run the experiment on 0 vs rest.

Metric	**No PCA**	**PCA**
Accuracy (%)	99.34	99.31
True Positives (%)	94.18	93.88
Passes to converge	39	59

We see that doing SVM on the original data gives us slightly better results than after doing PCA. This is possibly because there is some loss in information when projecting from 784 dimensions to 200 dimensions.

4 Summary

We thoroughly investigated the performance of different kinds of SVMs, both polynomial and RBF kernels. For polynomial kernels, we found that higher degrees give better percentages of true positives predicted. For RBF kernels, we found that σ = 5 gives good performance and relatively fast convergence compared to other values. We also saw that in general, softer margin works the same as or better than hard margins for the MNIST dataset. We also varied the number of training samples and saw that the accuracy increases with an increase in the size of the training dataset. Comparing with PCA, we find that projecting the data into a lower-dimensional subspace gives us marginally lower accuracies (although there is not much difference) than the original data, since there is some loss of information. Overall, we implemented the simplified SMO algorithm for optimizing the dual problem of SVMs, and conclude that a highly effective SVM will use an RBF kernel and a large number of training samples.