MNIST Performance Measure Using Dual Perceptron Learning and SVM

Saba Kanwal (18L-1860),Maham(18L-1886)

December 2, 2018

Abstract

In this document, I measure Performance of MNIST dataset, by using Kernel version Dual Perceptron learning algorithm by feature expansion. For feature expansion data mapped to two-dimensional space. It approximately fits all training data with 225 misclassification in all ten perceptron. When accuracy tested on testing data, it about 95.6%. While accuracy of MNIST dataset using simple perceptron learning algorithm was 88%. Feature expansion results in performance improvement. MNIST performance also tested for Support Vector Machine using built in classifier of sk-learn python library, and accuracy on testing data was 97.25%.

Chapter 1

Introduction

In this report, Two classification algorithms are discussed. One was kernel version of dual perceptron learning and the other was support vector Machine(SVM).

1.1 Kernel Perceptron Leaning

When an input instance comes, we can map that instance in some high dimensional space by simply adding more features in that instance. For example, if we have instance with two features x1 and x2. We can map it to two dimensional space by adding more features. If we map above instance to two dimensional space then features of above instance will be

$$[x_1, x_2, x_1x_2, x_1^2, x_2^2] \tag{1.1}$$

Kernel version of perceptron learning is that, dot product of vectors in higher dimensional space can be written as dot product of vectors in lower dimensional space. For example, dot product of two 2-dimensional vectors can be written as square of their corresponding vectors in 1-dimensional space.

$$\mathbf{K(x_1, x_2) = (x_1 \cdot x_2)^2 = \varphi(x_1) \cdot \varphi(x_2)}$$

In simple perceptron learning, we update weights after misclassification of an example simply by adding or subtracting that example. If we know, how many times an example added or subtracted, then we can find weight vector from it. Weight vector can now be represented as

$$\vec{w} = \sum_n \alpha_n y_n \vec{x}_n \tag{1.2}$$

In above equation, α_n represent number of times n^{th} example misclassified. y_n represents label of n^{th} example. The above equation gives same effect of adding or subtracting positive or negative examples respectively on misclassification. And function used for classification of an example is given below

$$f(x) = \vec{w} \cdot \vec{x} = \sum_n \alpha_n y_n \vec{x}_n \cdot \vec{x} \tag{1.3}$$

If $f(x) > 0$, example correctly classified else its misclassified.

1.2 Support Vector Machine(SVM)

Support vector machine is a supervised classifier that tries to find the best separation point between the classes. For example, if our data is linearly separable and only two classes, then many of the lines possible that can completely fit the training data. SVM find the best line that has maximum margin for the support vectors(data points at minimum distance from the defined line). This line can be defined only with the help of support vectors. The above discussion is about a Linear SVM. But if data not linearly separable then our algorithm never terminate. We can solve this problem by limit the number iterations performed. And we can also define some penalty when an error occur in classification. And also SVM can be applied in high dimensional space. In this assignment, we not apply SVM classifier on original images. First important features of images are extract using HOG(histogram of gradients).

1.2.1 HOG

Histogram of Oriented Gradients are used for improving performance of many Machine learning algorithms e.g. SVM. It is used to extract useful information from images and remove extra information that is not of our interest. It actually used for object or edge detection from images using gradients computations. The idea is to divide the whole image in to fixed size windows. In each window the change in gradient and its direction is computed. Then a histogram is plotted with b bins that shows which intensity value appears in how many times.

Chapter 2

Experiments and Results

2.1 Dual Perceptron Learning

Perceptron gives binary output. But we have to classify data into 10 classes. To use perceptron learning, we perform some preprocessing on our training data.We train 10 perceptron, one for each digit. I trained my algorithm by using 50,000 examples. And test it for 10,000 testing examples. Before training perceptron for each digit, first i computed a 50,000*50,000 matrix that store dot product of all training examples. I only computed diagonal and above entries.

Termination condition for my wight update was error less than 0.001. The results for each individual perceptron on training data are given in table below.

Perceptron	Size of training	Misclassification on training
0	50,000	40
1	50,000	15
2	50,000	28
3	50,000	23
4	50,000	20
5	50,000	12
6	50,000	13
7	50,000	15
8	50,000	25
9	50,000	35

The total misclassification of all perceptron values is about 225. The correctly classified instances of training at last step 49,775. Means it fits about 99% of training dataset. I think, if i use error 0 to stop weight update rule. It may be possible it will give 100% accuracy on training examples. The accuracy on testing data set is given below

Size of training	Size of testing	Accuracy	Time to Predict and train
50,000	10,000	95.6%	more than 1 day

2.2 SVM Using HOG

For SVM, i take code from Internet that is for MNIST digit recognition dataset using HOG feature extraction from input image. For HOG feature extraction, 14*14 window size used. It means total 4 blocks of 14*14 in 28*28 image. And 9 dimensional orientation vector is used. So total HOG features size is 9*4=36. Then this vector used as input for linear SVM. And for training 60,000 images are used and testing 10,000 test images are used. The accuracy for of SVM on test data is given in below table.

Size of training	Size of testing	Accuracy	Time to Predict and train
60,000	10,000	97.25%	within one hour

Chapter 3

Discussion and Conclusion

From the above results, we conclude that, dual perceptron learning perform better than simple perceptron learning. Because MNIST dataset is not linearly separable. When we expand features of MNIST dataset in dual space its accuracy on testing data is about 95.6%. That is about 8% improvement on perceptron learning in dual space. From above, we conclude that if data is not linearly separable then it may be linear separable in some high dimensional space. Linear SVM accuracy is more than dual perceptron learning and it's about 97%. One reason for this, instead of original images we use feature vector that is extracted from original images and it better represent the distinguish features of images. Second if we use dual perceptron learning without kernel trick then its computational time will be very high than SVM. In SVM, we reduce the feature vector size that result in faster computations.