Digit Recognition Using KNN and SVM

Harnoor Singh
Department of Computer Science & Engineering
University of Washington
hsingh@cs.washington.edu

Brian Walker
Department of Computer Science & Engineering
University of Washington
bdwalker@cs.washington.edu

1 Objective

As described in our project proposal, we are using the Kaggle digit dataset to classify written numbers.

Our goal is to implement digit recognition using K Nearest Neighbors (KNN) and a Support Vector Machine (SVM) and compare the two approaches. A stretch objective is to enable data that isn't a part of the Kaggle dataset to work with our algorithms. This would require some form of normalization in the input image for consistency.

2 Progress

2.1 KNN

We have made significant progress in both the KNN algorithm and the SVM algorithm. Our KNN algorithm is completely implemented as is cross validation for it.

To compute the K nearest neighbors, we take the euclidean difference between the feature we are classifying, X_1, and every feature in our training set.

$$\sum_{X_2 \exists N} (X1 - X2)^2$$

Once we have calculated the K closest features, a simple voting method is used to classify X_1.

In order to select a value of K we do cross validation on a subset of the training set. We begin by passing a list of potential K values we are considering. For each item, K_i in this list, we hold out K_i samples from our training set and construct our classifier. We then attempt to classify the K_i samples and record the error rate.

The K value that led to the lowest error rate is used to build our final classifier. The result of our cross validation can be seen in Table 1 below.

Table 1: Cross Validation Selection of K

K	Error
1	135
2	164
3	137
4	148
5	144
10	163
15	176
20	191

2.2 SVM

The SVM implementations we have worked with previously have involved only two classes. Because we have to classify the digits 0-9, we need to create an SVM that can classify from a broader range.

We are in the process of implementing a one-vs-one classifier. We train $nchoose2$ SVMs which compare two numbers. For example, one SVM would classify a digit as either a 1 or a 2. Another would classify a digit as a 1 or a 3, and so on. We get a prediction from all of these classifiers and take a vote to see which digit was most likely based on our SVMs.

In our training step, every time we make a mistake we log it. Future predictions are made based on past mistakes.

$$Classification = sign(C * \sum_{m \exists Mistakes} (Y_m * k(X_m, X)))$$

The biggest challenge thus far has been deciding how to regularize our SVM without relying on the weights. Because we use the kernel trick, the weights are never explicitly stored.

3 Future Work

Our primary focus moving forward is to get our SVM implementation working. Most of the structure is in place, however we need to decide on our method of regularization. Once we have the regularization done we can analyze the effectiveness of our SVM.

For KNN it might be beneficial to attemp to increase performance. Classifying each sample takes about 1 second, and with the testing dataset holding about 50,000 samples it takes several hours to classify all the points. Possible areas of speedup are a better algorithm for finding the nearest neighbors or parallelizing the nearest neighbor search.