EECS 349 Fall 2015
Jason Brown & David Williams - Homework 9

<u>Classification Techniques</u>

1) Exploring the Data

a) Using len(images) or by reading the Yann LeCun mnist website, there are 60,000 images in the training set. There are an average of 6,000 samples of each # 0-9, with "1" having the most samples, 6,742, and "5" having the least samples, 5,421.

b) After looking through 50 "4" and 50 "5", see attached 7 images of "4" and "5" that will be hard to classify because they are scribbles that hardly look like the numbers they are intended to be, and in some cases look closer to "0" or "1".

c) Since we want the training data to create a classifier that can generalize the testing data, we choose a large subset (up to half) of the set of data to serve as the training set. This allowed the classifiers to generalize each digit rather than be set to specific instances. We considered manually selecting what we deemed "good" instances, but viewed this as the *user* doing the learning, rather than the algorithm. We considered cross-validation, but determined that a set size of 60,000 was sufficient without multiple validation iterations.

d) In the classifiers we will be using, there is no correlation between pixels (cell 37 has no correlation to cell 38 though they may be close together). Therefore, try to map so same cell is lit up for all instances of the same number. By making the images more consistent, we create more overlap and allow the classifiers to have the best chance of recognizing the similarities between images.

2) Algorithm Selection

a) <u>Support Vector Machine</u> is a classification algorithm that determines classification hyperplanes that maximize the margin between classes. The support vectors are points on the margin of the hyperplane which are used to classify a test point. Noisy training data is dealt with via the introduction of a slack variable to allow "forgiveness" of irregular training instance. A kernel function is used to map the points to a different dimension to reduce the complexity of the hyperplane separating the classes, allowing for more efficient classification. SVMs were made popular in the '90s.

b) The two most important parameters in an SVM are the kernel function and γ. Our algorithm tried via experiment linear, polynomial (with varying d degree), and sigmoid kernel functions, before determining that a kernel function of $\exp(-\gamma|x - x'|^2)$ ("rbf") performed the best in testing. A second key parameter is γ, which represents the slack variable to allow misclassifications to exist so long as the sum of the distances of misclassifications is less than a given γ. If $\gamma = 0$, no misclassifications are allowed and a "hard" classification line is formed; some slack is better for a soft margin necessary with fuzzy training data; since the mnist data set is "fuzzy" (some 7s look like 3s, some 8's look like 1's), slack is important.

a) K Nearest Neighbors is a classification algorithm which assigns a class based on the distance between a test example and the training examples. In this case, the algorithm simply keeps a record of all the training images as vectors and what class they were. When it comes time to assign a class to a test image, a vector distance formula is used to identify the K nearest training examples that most resemble the test. The mode of these examples is then used to choose a new class. There is also a parameter that allows for training examples that are closer to the current test image to have greater weight in the new class decision. The output of the classifier is simply one of the possible classes, and requires no further interpretation to be compared to the true label.

b) KNN has two main parameters to explore: the K value (how many neighbors to consider) and the weights value (whether a closer neighbor has more weight or not). By increasing K, more training examples are considered, which protects better against noisy data but allows for a more blurred decision boundary. The weights value can be set to 'uniform', in which all neighbors have equal weight, or 'distance' where a closer neighbor affects the classification more.

3) Classification via SVM

a) Model and python files attached. Using an SVM with gamma = $10/n$, where n represents the length of the feature vector, a kernel function of $\exp(-\gamma|x - x'|^2)$ ("rbf"), and a training set and testing set of size $n = 10,000$) our model achieved an error rate of 3.9%.

b) We ran the following experiments (all charts attached):

i) We varied training & set size of $n = 500$, $n = 1,000$, $n = 5,000$, $n = 10,000$, $n = 30,000$. The error rate improved via exponential decay as n increased; the largest training set was best.

ii) The lowest error rate resulted from $\gamma = 10/n$, where n represents the length of the feature vector (length = 784). Smaller and greater γ values hurt classification performance. The kernel function with the best performance was $\exp(-\gamma|x - x'|^2)$ ("rbf"). A polynomial function with a high degree (d = 10) was the worst performing kernel function.

iii) Confusion matrix attached. "9" was the most misclassified digit, often misclassified as a 3,4,7, or 8. Interestingly, the confusion matrix was not symmetric: a "2" was misclassified as an "8" at twice the rate of an "8" being misclassified as a "2".

c) Misclassified images attached. In all images, it is fairly clear to see why they were misclassified - they were often sloppily written and even a human may have had trouble determining whether the digit was a poorly-written "4" or "9". This tells us that the boundaries that the SVM learned from training dataset are fragile. Since a human generally could with effort have classified the digits correctly using reasoning on what that the writer meant, this SVM should not be used for important tasks like mail sorting or check-cashing; a human, or a better algorithm, should be used with a lower error rate than 3.87%.

4) Classification via KNN

a) Model and python files attached. Using $k = 3$, $n = 10,000$, and distance = 'weighted' yielded an error rate of 5.1%.

b) We ran the following experiments (graphs for 1 and 2 can be seen on the second page of the Graphs Appendix):

   i) As shown in the first graph, a higher training set size increased the performance (lowered the error rate). This lowered exponentially, so the greatest gains were made by the first increases in size.

   ii) We varied the K value using both uniform weights and distance-based weights. Overall, distance is better than uniform, and a K value of 3 is optimal. This tells us that the boundaries between classes are pretty fragile, and a misclassification is quite easy due to the great variety in the image set.

   iii) This is the confusion matrix for the best set of parameters: $K = 3$ and 'distance' run on $10,000$ training examples and $10,000$ separate test examples. The columns correspond to what value was predicted $(0 - 9)$ and the row corresponds to what the actual label was $(0 - 9)$.

c) As can be seen in the images attached at the end of this report, the images which our KNN misclassified are similar to those that were misclassified in problem 3. Sloppily written numbers, or those that are written so wide they could overlap a different solution, are the main source of error in our second classifier. The most interesting of these however was a 0 that was misclassified as a 5. It is clearly a 0, and is neatly written, albeit likely a bit wider and thinner than would be typical. This shows the limitations of our classifiers, as the need for pixel overlap as a distance measure cannot be as accurate at telling digits are the same as the human eye.

5) Model Comparison
Initially, the KNN classifier seemed to outperform the SVM. However, as γ, training set size n, and kernel function parameters were optimized, the SVM "took the lead" in outperforming the KNN. Using an SVM with gamma = $10/n$, where n represents the length of the feature vector, a kernel function of $\exp(-\gamma|x - x'|^2)$ ("rbf"), and a training set and testing set of size $n = 10,000$) our model achieved an error rate of 3.9%. Using a KNN with $K = 3$, distance = 'weighted', and n = 10,000 resulted in an error rate of 5.1%. Both classifiers yielded asymmetric confusion matrices. In examining the the misclassified images, KNN had more odd results. In one instance, a clear '0' was misclassified as a '5'. In the SVM, the misclassifications were more justified; even a human may have had the same issue.

6) Boosting (Extra Credit)

a) A weak classifier, in this case a short decision tree, yielded better-than-random results upon boosting. However, the performance was much worse than the KNN or SVM, indicating that indeed "You *do* need that fancy classifier". The weak classifier, using a long (default) decision tree, and n = 1000, and n_estimators = 1000, both yielded an error rate of 34%. Please see the attached confusion matrix. As n_estimator was decreased to 50, the error rate increased to 65%. Using a short decision tree and n_estimators = 50 yielded an error rate of 56%. This indicates that surprisingly, a longer decision tree (stronger) outperformed a short decision tree.

b) The strong classifier, using the parameters attainted in problem 3 ($\gamma = 10/784$ (length of feature vector)), and default kernel function, did not work. It yielded an error rate of 89% and classified every # as an '8', as shown in the confusion matrix.

c) The weak classifier in A outperformed the strong classifier in B handily. The weak classifier seemed to make significant gains, especially when allowed to perform the boosting cycle over many iterations (n_estimators = 1000 yielded an error rate of 34%). The strong classifier SVM that we had optimized in problem 3 seemed to be unable to correctly identify numbers when boosted, maintaining an error rate of 89%, even when given the same amount of iterations. This shows that boosting is a method of improving a weak classifier, and is not a suitable method for working with strong classifiers.