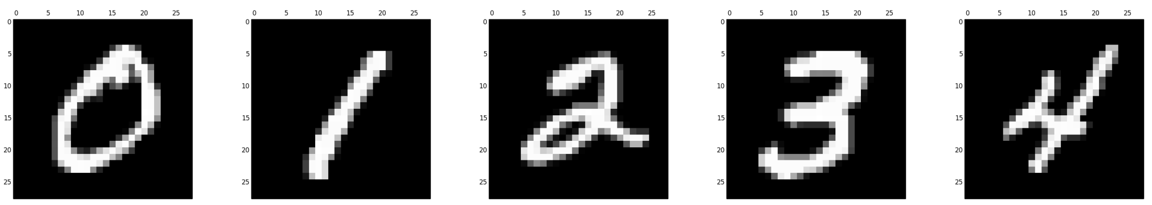
CS 170 Algorithms Soln 12
Fall 2014 David Wagner

1. (50 pts.) K-Nearest Neighbors

Digit classification is a classical problem that has been studied in depth by many researchers and computer scientists over the past few decades. Digit classification has many applications: for instance, postal services like the US Postal Service, UPS, and FedEx use pre-trained classifiers in order to speed up and accurately recognize handwritten addresses. Today, over 95% of all handwritten addresses are correctly classified through a computer rather than a human manually reading the address.

The problem statement is as follows: given an image of a single handwritten digit, build a classifier that correctly predicts what the actual digit value of the image is. Thus, your classifier receives as input an image of a digit, and must output a class in the set $\{0,1,2,\ldots,9\}$. For this homework, you will attack this problem by using a k-nearest neighbors algorithm.

We will give you a data set (a reduced version of the MNIST handwritten digit data set). Each image of a digit is a 28×28 pixel image. We have already extracted features, using a very simple scheme: each pixel is its own feature, so we have $28^2 = 784$ features. The value of a feature is the intensity of that corresponding pixel, normalized to be in the range 0..1. We have preprocessed and vectorized these images into feature vectors for you. We have split the data set into training, validation, and test sets, and we've provided the class of each image in the training and validation sets. Your job is to infer the class of each image in the test set. Here are five examples of images that might appear in this data set, to help you visualize the data:



We want you to do the following steps:

(i) Implement the k-nearest neighbors algorithm. You can implement it in any way you like, in any programming language of your choice. For $k > 1$, decide on a rule for resolving ties (if there is a tie for the majority vote among the k nearest neighbors when trying to classify a new observation, which one do you choose?).

(ii) Using the training set as your training data, compute the class of each digit in the validation set, and compute the error rate on the validation set, for each of the following candidate values of k: $k = 1, 2, 5, 10, 25$.

(iii) Did your algorithm perform significantly better for $k = 2$ than for $k = 1$? Why do you think this is?

(iv) Look at a few examples of digits in the validation set that your k-nearest neighbors algorithm misclassifies. We have provided images of each of the digits, so you might want to look at the corresponding images to see if you can get some intuition for what's going on. Answer the following questions: Which example digits did you look at? Why do you think the algorithm misclassifies these examples? List some additional features we could add that might help classify these examples more accurately.

(v) Using the best k value you achieved on the validation set, compute the class of each image in the test set (0–9) and submit your answer.

(vi) Optional bonus challenge problem: Implement your proposed features from part (iv) and try them out. Did they improve accuracy on the validation set? Try to find a set of features that gives significant reduction in the error rate—you might need to experiment a bit. List the set of features you used, the total number of features, the best value of k, and the error rate you achieved.

Note that item (vi) is entirely optional. Do it only if you would like an extra challenge.

Your write-up should have the following parts:

(a) The error rate on the validation set for each of $k = 1, 2, 5, 10, 25$. State which value of k is best.

(b) Describe what tie-breaking rule you used.

(c) Answer the questions under item (iii) above.

(d) Answer the questions under item (iv) above.

(e) Optional bonus challenge problem: Do item (vi) above and answer the questions under item (vi) above.

Solution:

For implementation, we expected you to use a linear search through the whole training dataset and find the k nearest data points using Euclidean distance as a metric. k-d trees may not have been good for this assignment since the number of features is well above 20. Any rule for resolving ties such as picking a random class out of the ones with majority vote should have been fine.

(a) Our implementation was able to achieve the following error rates for the different k values.

k	1	2	5	10	25
Error Rate	8.9%	10.6%	9.8%	11.5%	13.1%

Why is the error rate *increasing* as k increases? Normally, when we run k-nearest neighbors algorithm on some dataset, the error rate decreases as the value of k increases. However, for the dataset you were given for this problem, you may have noticed that you were getting larger error rates as the value of k grew larger. This is due to the nature of the dataset. For example, a lot of instances such as the following occurred:

The actual digit is 8, but because it is skinny, it looks like a 1. There are a few points in the training set for which the 8 is skinny so the closest neighbors are these values, and if k is small enough, the digit is correctly classified. However, there are only a few of these skinny 8's, and besides these skinny 8's, it is otherwise closer to a 1. Therefore if k is larger, there will be many votes for 1, and the digit might get mis-classified as a 1. For this kind of situation, a smaller k does better on this dataset.

This issue can be solved with a bigger dataset and better features. This is also the reason k-nearest neighbors algorithm works better as the dataset grows larger. If your error rate did not go up for larger k values, you should not worry as this is most likely a result of small implementation details such as tie breaking.

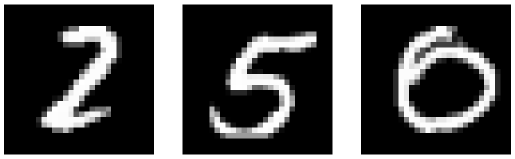
(b) Our tie-breaking rule was to pick randomly between the classes that had the most number of votes.

(c) Even when the error rate decreases as the value of k increases, the k-nearest neighbors algorithm does not perform significantly better for $k = 2$ than for $k = 1$. This issue arises because of tie-breaking issues. There are two cases:

- Suppose for a testing data point, the two closest neighbors are of the same class. In this case, the k-nearest neighbors algorithm for $k = 1$ and $k = 2$ will return the same class, so there is no difference in this case.

- Alternatively, suppose that the two closest neighbors are of two different classes. In this case, it is more likely that the class associated with the closer neighbor out of the two will represent the correct class. Any tie-breaking procedure for $k = 2$ that picks the second-closest neighbor a non-zero percentage of the time will thus do worse than the k-nearest algorithm with $k = 1$.

(d) The following are some examples of misclassified digits:



The first digit was misclassified as a 7, the second digit was misclassified as a 6, and the last digit was misclassified as a 0. The algorithm misclassified a lot of different points because we are simply computing the Euclidean distances of the pixel vectors. This does not account for any linear transformations to the digits such as rotation or translation. For example, the digit 1 that is centered and a digit 1 that is offset just by a few pixels to the right will have very high Euclidean distance even though semantically the two are essentially exactly the same.

There are many possible ways that might help address this problem. One possibility is to compute features that take number of on pixel values for all possible 4 by 4 or 5 by 5 grid of pixels. A classifier using these features will be more robust to translations or rotations since most of these computed features of a certain digit remain the same even after the translation or rotation of that digit. Orientation histogram features are also good features to add for this problem because they also do provide linear transformation invariance (see, e.g., `https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients`).

Yet another possibility would be to select some additional features that capture information about the shape of the digit. For instance, for each row that is not entirely blank, we could extract the column of the leftmost white pixel, and compute the standard deviation of these values, then do the same for the rightmost white pixel. Thus a vertical stroke like the digit 1 will have low standard deviation on both sides while a digit like 2 or 5 will have a high standard deviation on both sides. The digit 5 might have a higher standard deviation on the left side than the digit 6.

There are many other ideas you could have come up with. Here was a chance to be creative!

2. (50 pts.) Random forests

Spam classification is another problem that has been widely studied and used in many, if not all, email services. Gmail and Yahoo Mail are two examples of services currently using sophisticated spam classifiers to filter spam from non-spam emails.

The problem statement is as follows: build a classifier that, given an email, correctly predicts whether or not this email is spam or not. For this homework, you will attack this problem by using a decision forest algorithm using the email data set we have provided you. We have selected 57 useful features and preprocessed the emails to extract the feature values for you, so you only need to deal with feature vectors. A more detailed description of the feature values we are using can be found in the `README` file inside the `emailDataset` directory. We have split the data set into training, validation, and test sets, and we've provided the class of each email in the training and validation set (spam or non-spam). There are two classes: 0 (non-spam) and 1 (spam).

Do the following steps:

(i) Implement a random forests classifier with T trees.

Use the approach presented in lecture. For each of the T trees, use bagging to select a subsample of the training data set (a different subsample for each tree). Use the greedy algorithm to infer a decision tree. At each node, choose a random subset of the features as candidates, and choose the best feature (among those candidates) and best threshold. We suggest you use a subset of size $\sqrt{57} = 8$. Use a different random subset for each tree and each node. Use information gain (i.e., the entropy-based measure) as your impurity measure for greedily selecting the splitting rule at each node of each decision tree. In your greedy decision tree inference algorithm, use any stopping rule of your choice to choose when to stop splitting (you might want to impose a limit on the depth of the tree, or stop splitting once the impurity is below some fixed value, or both). Your classifier will classify a new email by taking a majority vote of the T classes output by each of the T trees. When $T > 1$, you'll need to choose a rule for resolving ties.

(ii) For each of the following candidate values of T, $T = 1, 2, 5, 10, 25$: run your code on the training set provided to learn a random forest of T trees, compute the class of each digit in the validation set using the resulting classifier, and compute the error rate on the validation set for this value of T. (T is the number of decision trees. Each value of T will give you a different classifier and thus potentially a different error rate on the validation set.)

(iii) Using the best T value you achieved on your validation set, compute the class of each email in the data set (0 = non-spam, 1 = spam) and submit your answer.

Your write-up should have the following parts:

(a) The error rate on the validation set for each of $T = 1, 2, 5, 10, 25$. State which value of T is best.

(b) Describe what rule you used to resolve ties.

(c) Describe what you used as the stopping rule.

(d) If you deviated from the suggested approach above in some detail, describe how your implementation differs.

Solution:

For implementation, we expected you to use the pseudocode provided to you in the random forest lecture notes with the addition of bagging and feature sampling. We expected you to fix the size of the training set for each tree to anywhere between $0.7n$ and n, where n is the size of the original training set.

(a) For this problem, your error rate should decrease as T increases. Our implementation was able to achieve the following error rates for the different T values.

T	1	2	5	10	25
Error Rate	13.35%	12.38%	8.7%	6.19%	6.19%

As a side note, for our implementation, a simple decision tree algorithm on this dataset was able to get around 10 percent error rate.

(b) You could use any rule of your choice to resolve ties, such as picking a random class out of the classes that are tied.

(c) You could use any stopping rule of your choice. Reasonable stopping rules might include a combination of any of the following (or others you have defined): all remaining points are of the same class, the depth of the current node is greater than some fixed depth such as 10, the entropy impurity does not change by more than some fixed threshold such as 0.001, and/or the number of data points left is less than some number such as 50.