CS 170 Algorithms Fall 2014 David Wagner HW12

Due Dec. 5, 6:00pm

Instructions. This homework is due Friday, December 5, at 6:00pm electronically via glookup. This homework assignment is a programming assignment that is based on a machine learning application.

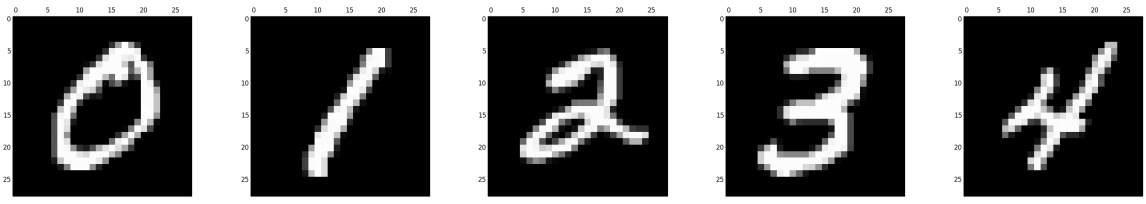
You may work individually or in groups of two for this assignment. You may not work with more than one other person. If you work with a partner, you may only use code you wrote together via pair programming or code you wrote individually. For example, you may choose to do the following: for problem 1, you decide to use pair programming and for problem 2, you decide only to discuss your approaches to one another and use your own implementations. You may not turn in any code for which you were not involved in writing (for instance, "you implement Problem 1, I'll implement Problem 2" is not allowed). In addition, you may not discuss your approaches with anyone other than your partner.

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.

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.

Data sets. The data sets are available for download from the course web page. Detailed information about each data set is present in the `README` file for each data set.

Programming tips. You may use any programming language you like, and any standard libraries of your choice. Python or Matlab might be convenient, but you can use any language you want.

One restriction on this homework: you cannot use any existing machine learning library. For instance, it is easy to find implementations of k-nearest neighbors and random forests for download or in existing libraries. You can't just download or invoke an existing implementation of these algorithms.

We have some advice for you to make implementation easier:

- If you use Python, some of our TAs suggest that using `numpy` or `scipy` to make your implementation faster. This is entirely optional. We won't evaluate or grade you on the speed or asymptotic efficiency of your code. The only benefit is that it might make your code run faster, making it easier to experiment with your algorithm.

- To read in the input files, the easiest approach is probably to split each line on `,` (comma). Alternatively, if you prefer, on Python you can use `csv.reader` (from Python's `csv` module, which is designed for reading CSV files).

As a comparison point, our implementations achieve approximately 10% error rates or less on each problem. Our implementations take under an hour to train and evaluate all of the validation instances.

Submission. You must turn in the following files through glookup, by putting all of the files in a directory on your instructional class account, `cd`ing into that directory, and running the `submit hw12` command from within that directory.

1. `writeup1.txt` — a text file (ASCII file) containing your write-up for Problem 1.
2. `writeup2.txt` — a text file (ASCII file) containing your write-up for Problem 2.
3. `code/` — directory containing all of your source code.
4. `name.txt` — a text file containing the following:
   (a) Line 1: Your first and last name.
   (b) Line 2: Your login.
5. `collaborator.txt` — a text file containing the full name (first and last name) and login of your partner, if you worked in a group of two, or `None` if you worked individually.

   (Note that if you work in a group of two, both of you must turn in all files.)
6. `digitsOutputk.csv` — five text files containing the result of running your program on the validation set of the digits data set using the corresponding k value. You should have 5 of these files: `digitsOutput1.csv`, `digitsOutput2.csv`, `digitsOutput5.csv`, `digitsOutput10.csv`, and `digitsOutput25.csv`. Each file should have exactly 1000 lines, as follows:
   (a) Line 1: Predicted class for the first digit image in the validation set (0-9).
   (b) Line 2: Predicted class for the second digit image in the validation set (0-9).
   (c) ...
   (d) Line 1000: Predicted class for the 1000th digit image in the validation set (0-9).
7. `digitsOutput.csv` — a text file containing exactly 1000 lines, as follows:
   (a) Line 1: Predicted class for the first digit image in the test set (0-9).
   (b) Line 2: Predicted class for the second digit image in the test set (0-9).
   (c) ...
   (d) Line 1000: Predicted class for the 1000th digit image in the test set (0-9).
8. `emailOutputT.csv` — five text files containing the result of running your program on the validation set of the email data set using the corresponding T value. You should have 5 of these files: `emailOutput1.csv`, `emailOutput2.csv`, `emailOutput5.csv`, `emailOutput10.csv`, and `emailOutput25.csv`. Each file should contain exactly 500 lines, as follows:
   (a) Line 1: Predicted class for the first email in the validation set (0 or 1).
   (b) Line 2: Predicted class for the second email in the validation set (0 or 1).
   (c) ...
   (d) Line 500: Predicted class for the 500th email in the validation set (0 or 1).
9. `emailOutput.csv` — a text file containing exactly 500 lines, as follows:
   (a) Line 1: Predicted class for the first email the test set (0 or 1).

(b) Line 2: Predicted class for the second email the test set (0 or 1).

(c) ...

(d) Line 500: Predicted class for the 500th email the test set (0 or 1).

10. `bonusDigitsOutput.csv` — *Optional*. This is for the optional bonus challenge problem. You don't need to include this file, unless you do the optional bonus problem. If you do the optional bonus problem, include this file. This file is the same as `digitsOutput.csv`, except now using your new set of features. Thus, it should be a text file containing 1000 lines, where the *i*th line contains the predicted class of the *i*th digit in the test set, where the classes were computed using your proposed set of features.

Fun fact. Did you know that finding the optimal decision tree is NP-hard? There's a reduction from 3-dimensional matching. The proof was published in a 3-page paper: `http://people.csail.mit.edu/rivest/pubs/HR76.pdf`