CIS 520, Machine Learning, Fall 2015: Assignment 2
Due: Friday, September 18th, 11:59pm (via turnin)

Instructions. This is a MATLAB programming assignment. This assignment consists of multiple parts. Portions of each part will be graded automatically, and you can submit your code to be automatically checked for correctness to receive feedback ahead of time.

We are providing you with codebase / templates / dataset that you will require for this assignment. Download the file `hw2_kit.zip` from Canvas **before** beginning the assignment. **Please read through the documentation provided in ALL Matlab files before starting the assignment.** The instructions for submitting your homeworks and receiving automatic feedback are online on the wiki:

http://alliance.seas.upenn.edu/~cis520/wiki/index.php?n=Resources.HomeworkSubmission

If you are not familiar with Matlab or how Matlab functions work, you can refer to Matlab online documentation for help:

http://www.mathworks.com/help/matlab/

In addition, please use built-in Matlab functions rather than external library functions. Without proper reference to external library, auto-grader may fail even if your code runs perfectly on your local machine.

Collaboration. For this programming assignment, you must work individually. **The LOWEST grade will be recorded for multiple submissions.** Be sure to include your pennkey and name in the *Group.txt* file. Failure to do so will result in a failure in the grading process. **We will be using automatic checking software to detect blatant copying of other student's assignments, so, please, don't do it.**

1 Cross Validation [15 points]

Description. In this section, we will explore a simple strategy to *estimate* test error from a training set, and then use these estimates to choose the the K and σ parameters for K-NN and kernel regression, respectively.

The simplest way to estimate test error with a training set is **N-Fold Cross Validation.** To compute N-fold cross validation error, we use the following algorithm. Let $\mathcal{D} = (x_1, y_1), \ldots, (x_n, y_n)$ be our training sample.

1. Divide our dataset *at random* into N sets of equal size, $S_1, \ldots, S_N$. Each set is called a *fold.*
2. For $i = 1, \ldots, N$:
   (a) Train our classifier $h(x)$ using the following training set:
   $$\mathcal{D}_i = \bigcup_{j \neq i} S_j.$$
   Note that $\mathcal{D}_i$ contains all folds *except* the i'th fold.
   (b) Compute error on the i'th fold:
   $$\epsilon_i = \frac{1}{|S_i|} \sum_{(x,y) \in S_i} \mathbf{1}\left(h(x) \neq y\right).$$

3. The cross validation error is the average error across all folds:

$$error = \frac{1}{N}\sum_{i=1}^{n}\epsilon_i.$$

Your task. Your first task is to implement step #1 of the above algorithm in the file `make_xval_partition.m`. You will use this function repeatedly in the rest of the assignment, so it is important to make that the partition is random. **See the instructions in `make_xval_partition.m` for the specifications of the function.**

2 K is for Kernel Width [35 points]

Description. A hospital wants to know whether or not you can help them to detect breast cancer in their patients quickly and reliably and have sent you this data that they have collected from patients with confirmed to have/not have breast cancer: ftp://ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.names. We have removed the records with missing values for you. **We have included two versions of the dataset: the original data, and an additional version with noise added.** Obviously, there are patients lives at risk here, so they want you to be certain that your model can accurately detect breast cancer on new patients (they already know the diagnosis for the ones in the data set they've given you).

So how can we build the best model possible and be confident of how accurate it will be on future patients, given the limited data that they have provided us? In general, the answer is to set aside a randomly selected set of patients that we do NOT use for training. Often we do not have enough such patients, and so we use N-fold cross validation on training set to estimate the "true test error". Your task here is to observe the relationship between N-fold cross validation error and true test error.

Your task. The goal here is to compare the N-fold classification estimate and the true test error of the dataset; you will then use your estimates of test error to determine the optimal kernel width σ and # of neighbors K to optimize the performance of kernel regression and K-nearest neighbors for this task. You will need to do the following:

- Implement two Matlab functions, `knn_xval_error.m` and `kernreg_xval_error.m`, that take a dataset, a partition, and a parameter, and return the error of the given algorithm on that partition. See `knn_xval_error.m` and `kernreg_xval_error.m` for exact specifications of what these functions should do.
- Feel free to use and modify the included `knn_test.m` and `kernreg_test.m`.

Now that you've finished your implementation, compare the N-fold cross-validation estimate and true test set error on the standard and noisy datasets by answering the following questions. You will need to randomly split the full data into training and testing sets. For all experiments, use 400 examples for training and the remainder for testing. **See `generate_knn_plots.m` for specific instructions on how to submit the plots/analysis electronically.**

1. For both the original data and the noisy, compute both the N-fold error on the training set and the test error for K-NN with $K = 1$, for $N = \{2, 4, 8, 16\}$. What trend do you observe? **Note that any two random partitions of the data will yield somewhat different curves. Therefore, you must repeat all of the above steps 100 times, using different random partitions into training and testing.**
2. For both the original data and the noisy, compute both the 10-fold error on the training set and the test error for K-NN with $K \in \{1, 3, 5, \ldots, 15\}$ and $\sigma \in \{1, 2, 3, \ldots, 8\}$. Generate **four plots**: each plot will show K (for K-NN) or σ (for kernel regression) on the X axis, and error on the Y axis. The plot will have two lines with error bars: one for 10-fold error, and the other for test set error; you will have one plot for each method/dataset combination (e.g., K-NN on standard, K-NN on noisy, etc.). Based

on these charts, can you pick the best σ and K to minimize test set error using cross validation (on average)? Which are the best values?

N-FORD ERROR AND TEST ERROR CLARIFICATION: *N-fold error is the error over the training set, and test error is the error when you test your trained classifier on test set. N-fold cross validation is used for choosing the best parameters.*

3 Decision Trees [50 points]

Description. In this part of the assignment, you will generalize the ID3 algorithm discussed in class to work with *multi-class labels*, e.g. $Y \in \{1, \ldots, K\}$. You will also explore the issue of *overfitting* with decision trees as the depth of the tree increases.

You will be using the MNIST digit dataset, which consists of roughly 54,000 training examples and 9,000 test examples of handwritten digits. Each example is a 28×28 grayscale image, which leads to 784 pixels to use as features.

Note that we have provided you with a working ID3 decision tree implementation for *binary* outputs: $Y \in \{0, 1\}$. Unlike the code from the lecture page, this code is mostly optimized and scales to larger datasets. Portions of the code have been significantly vectorized. Thus, your challenge is to understand and modify the existing decision tree implementation.

Your task. You will need to provide the following: (**see each file for exact specifications**):

- `multi_entropy.m` - This function computes the empirical entropy of samples of Y, where $Y \in \{1, \ldots, K\}$. Note that you may use `binary_entropy.m` as a starting point.
- `dt_choose_feature_multi.m` - This function takes in a multi-class dataset and returns the best split on the features of the dataset to maximize information gain. Note that you may use `dt_choose_feature.m` as a starting point.
- `dt_train_multi.m` - This function takes in a multi-class dataset and a depth limit d and returns an ID3-learned decision tree with depth $\leq d$ as an output. Note that you may use `dt_train.m` as a starting point.

Once you have finished your implementation, answer the following questions (**See `generate_dt_plots.m` for specific instructions on how to submit your plot and analysis**):

1. For depths $d = \{1, \ldots, 6\}$, evaluate your new functions on a subset of the MNIST training for digits 1, 3, and 7. In other words, generate a decision tree of depth d that predicts whether the input is a 1, 3, or 7, for $d = 1, \ldots, 6$; compute the training error and test error for each tree. Now, **generate a plot showing training error and test error (Y axis) as a function of tree depth (X axis).** Do you observe any overfitting? If so, where?

2. Learn a decision tree of depth 6 on the full MNIST dataset, to compare all digits, and compute a **confusion matrix**, defined as follows:

$$M_{i,j} = P(y = i, h(x) = j) \qquad \text{(Classifier confused } i \text{ with } j\text{)}$$

   This confusion matrix tells you where the errors are occuring, e.g. which digits are confused with which other digits. Note that the sum of the diagonal elements is the accuracy of the method. Based on your confusion matrix, which is the hardest comparison between digits? (i.e. which i and j are $\max_{i \neq j} M_{i,j}$)

3. Choose any example from the test set where your depth 6 where $h(x) = j, y = i$ from the previous question. Use the `plot_dt_digit.m` function to plot the pixels that the tree used to make its decision. Why do you think this example caused the tree to fail? Which pixels "confused" the classifier?