CIS 520, Machine Learning, Fall 2018: Assignment 2
Due: Sunday, September 23rd, 11:59pm (via turnin)

Instructions. This is a MATLAB programming assignment. This assignment consists of multiple parts. For the code part, you should submit your code to autograder via **turnin**, the instruction of which is listed below; for the pdf, you should submit it through **Gradescope**.

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. Also, please DO NOT include data file in your submission.

Collaboration. You are allowed and encouraged to work together. You may discuss the homework to understand the problem and reach a solution in groups up to size **two students.** Please submit **one copy** of your work. Be sure to include **your and your collaborator's** pennkey and name in the **group.txt** file. **We will be using automatic checking software to detect blatant copying of other groups' assignments, so, please, don't do it.** Collaborators:

Type Collaborator Name Here

1 Cross Validation [8 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:
   $$\epsilon_i = \frac{1}{|S_i|} \sum_{(x,y) \in S_i} \mathbf{1}(h(x) \neq y).$$
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 the 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. Please DO NOT use matlab built-in function crossvalind. **See the instructions in `make_xval_partition.m` for the specifications of the function.**

2 K is for Kernel Width [42 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 make sure 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 1) implement two simple classifiers K-NN and kernel regression respectively which is a good practice for you to be familar with coding with matlab and 2) 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 four matlab functions, `k_nearest_neighbours.m`, `kernel_regression.m`, `knn_xval_error.m` and `kernreg_xval_error.m`. Have a careful look at the matlab files for exact specifications of what these functions should do.

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 be given a dataset consisting of 600 data points, along with their labels. You will first have to randomly partition the dataset into a training set, and a testing set. We suggest using 450 data points for training and the remaining 150 for testing. For all experiments please use the training dataset to train your classifier and to perform cross validation, and use the test set for testing its performance.

- For both the original data and the noisy, compute both the N-fold error on the training set, for $N = \{3, 5, 9, 15\}$, and the test error for K-NN, and Kernel Regression with $K = 1$ and $\sigma = 1$ respectively. What trend do you observe? Please plot the cross-validation error and test error in the same figure. **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.**

  Your answer:

  `[Figure on original data with KNN]`

  `[Figure on noisy data with KNN]`

  `[Figure on original data with Kernel Regression]`

  `[Figure on noisy data with Kernel Regression]`

- For both the original data and the noisy, compute both the 10-fold cross validation error on the training set and the test error for K-NN with $K \in \{1, 3, 4, 6, 9, 14, 22, 35\}$ and for Kernel Regression with $\sigma \in \{1, 3, 5, 7, 9, 11\}$. 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, 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) for both original and noisy data? Which are the best values?

  Your answer:

  `[Figure on original data with KNN]`

  `[Figure on noisy data with KNN]`

  `[Figure on original data with Kernel Regression]`

  `[Figure on noisy data with Kernel Regression]`

The best σ is ..., the best K is ....

N-FOLD 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 (trained over the entire training set) on test set. N-fold cross validation is used for choosing the best parameters.*

3 Logistic Regression [50 points]

Description. In this part of the assignment, you will implement a very useful binary classifier Logistic Regression to work with the task in Problem 2.

Your task. Logistic Regression is a classifier used to estimate the probability of a response (label) based on one or more predictors (features). These probabilities are computed by exponentiating the score $\langle \mathbf{w}, \mathbf{x} \rangle$,

$$P(Y = 1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp^{-\mathbf{w}^T \mathbf{x}}}$$

$$P(Y = 0 \mid \mathbf{x}, \mathbf{w}) = \frac{\exp^{-\mathbf{w}^T \mathbf{x}}}{1 + \exp^{-\mathbf{w}^T \mathbf{x}}}$$

, where $\mathbf{x}$ is an observation whose label is to be predicted, and $\mathbf{w}$ is a vector of weights, with each coordinate of the vector corresponding to a feature of $\mathbf{x}$. Thus we can write the likelihood function,

$$\prod_{i=1}^{N} (P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}))^{y_i} (1 - P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}))^{1-y_i}$$

We later need to convert the likelihood function to log-likelihood function.

$$\ell(\mathbf{w}) = \sum_{i=1}^{N} [y_i \log P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) + (1 - y_i) \log(1 - P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}))] \tag{1}$$

$$= \sum_{i=1}^{N} [y_i (\mathbf{w}^T \mathbf{x}_i) - \log(1 + \exp(\mathbf{w}^T \mathbf{x}_i))] \tag{2}$$

Unfortunately, these is no closed form expression for computing this weight vector $\mathbf{w}$ that maximizes the log likelihood over the set of observations and responses (training set). Luckily for us, this loss function is concave, thus allowing us to implement a powerful optimization technique called gradient ascent. We can compute the gradient

$$\nabla_{\mathbf{w}} \ell(\mathbf{w}) = \sum_{i=1}^{N} y_i \mathbf{x}_i - \frac{\mathbf{x}_i}{1 + \exp(-\mathbf{w}^T \mathbf{x}_i)}$$

Your task in this section is to implement the functions `gradient_ascent_fixed.m`, `gradient_ascent_decay.m`, `logistic_regression.m`, `logistic_xval_error.m` (**see each file for exact specifications**):

- `gradient_ascent_fixed.m`. In this function, you shall implement the vanilla gradient ascent algorithm, with a constant step size. In gradient ascent, the choice of step size is crucial, and in this section, you shall explore its effect on the performance of your algorithm. You will have to experiment a little with the choice of the step size to ensure good performance. Set the step size to a value you found to work best empirically.

- `gradient_ascent_decay.m` Implement gradient ascent with step size that decays over time. You will have to experiment a little with the choice of the initial step size to ensure good performance. Did you notice any improvement over your implementation of gradient ascent with a constant step size? What do you think was the cause of the improvement? Plot the evolution of the zero-one loss over training data as the gradient ascent proceeds i.e. plot the training error on the Y axis, and iterations on the X axis for both implementations of gradient ascent (fixed step size and decaying step size) in the same plot. Do this for both the noiseless and noisy data set.

  Your answer:

  `[Figure on original data]`

  `[Figure on noisy data]`

  There is an improvement because ...

- Add an extra feature to your data that is always set to 1, and perform gradient ascent on this new data. This corresponds to adding a constant to the scores, i.e. $\langle \mathbf{w}, \mathbf{x} \rangle + c$. Did you notice any improvement in performance? What do you think happened that caused this improvement? Plot the evolution of the zero-one loss over training data as the gradient ascent proceeds i.e. plot the training error on the Y axis, and iterations on the X axis over the data without the extra feature, and with the extra feature in the same plot. Do this for both the noiseless and noisy dataset.

  Your answer:

  `[Figure on original data]`

  `[Figure on noisy data]`

  There is an improvement because ...

- Implement `logistic_regression.m` and `logistic_xval_error.m`. For both the original data and the noisy, compute both the N-fold error on the training set, for $N = \{3, 5, 9, 15\}$, and the test error. What trend do you observe? Please plot the cross-validation error and test error in the same figure.

  Your answer:

  `[Figure on original data]`

  `[Figure on noisy data]`