Homework Assignment 4: Regularization and Nearest Neighbors

Due Monday, February 14th, 2022 at 11:59pm

Description

In class we learned about regularization to avoid over-fitting, KNN, and weighted nearest neighbor. In this problem set, you will explore the role of λ in the regularized cost function. You will study the effect of choosing K on the classification accuracy, and implement the weighted voting algorithm.

What to submit

Create a folder ps4_LastName_FirstName. The structure of your folder should be as follows:

ps4_LastName_FirstName /

- input/ - input data, images, videos or other data supplied with the problem set
- output/ - directory containing output images and other generated files
- ps4.m - your Matlab code for this problem set
- ps4_report.pdf - A PDF file that shows all your output for the problem set, including images labeled appropriately (by filename, e.g. ps0-1-a-1.png) so it is clear which section they are for and the small number of written responses necessary to answer some of the questions (as indicated). Also, for each main section, if it is not obvious how to run your code please provide brief but clear instructions (no need to include your entire code in the report).
- *.m - Any other supporting files, including Matlab function files, etc. It's a good practice to add the keyword 'end' at the end of each function file

Zip it as `ps4`_LastName_FirstName`.zip`, and submit on canvas.

Guidelines

1. Include all the required images in the report to avoid penalty.
2. Include all the textual responses, outputs and data structure values (if asked) in the report.
3. Make sure you submit the correct (and working) version of the code.
4. Include your name and ID on the report.
5. Comment your code appropriately.
6. Please avoid late submission. Late submission is not acceptable.
7. Plagiarism is prohibited as outlined in the Pitt Guidelines on Academic Integrity.

Questions

1- **Regularization**: In this problem we study overfitting and regularization. We start by a linear regression problem on 500 features using 1000 data samples. With this large number of features, your model may over-fit the training data and may not generalize to unforeseen testing data. To avoid overfitting, we can use regularization, but we still need to choose the value of the regularization parameter λ. We are going to compute the average training and testing error for different values of λ to detrmine which value is the best candidate to our problem.

a. Write a function, `[theta] = Reg_normalEqn(X_train, y_train, lambda)` that computes the closed-form solution to linear regression using normal equation with regularization.

inputs:

- `X_train` is an mx(n+1) feature matrix with m samples and n feature dimensions. m is the number of samples in the training set.
- `y_train` is an mx1 vector containing the output for the training set. The i-th element in `y_train` should correspond to the i-th row (training sample) in `X_train`
- `lambda`, the value of the regularization parameter λ.

outputs:

- `theta` is a (n+1)x1 vector of weights (one per feature dimension).

Function file: Reg_normalEqn.m containing the function `Reg_normalEqn` (identical names)

b. Load 'hw4_data1.mat' into your MATLAB workspace. X_data is the feature matrix and y is the output vector. The features are already normalized, but you need to add the offset feature x_0 to your feature matrix, i.e., add a column of ones to your matrix.
Text output: what is the size of the feature matrix?

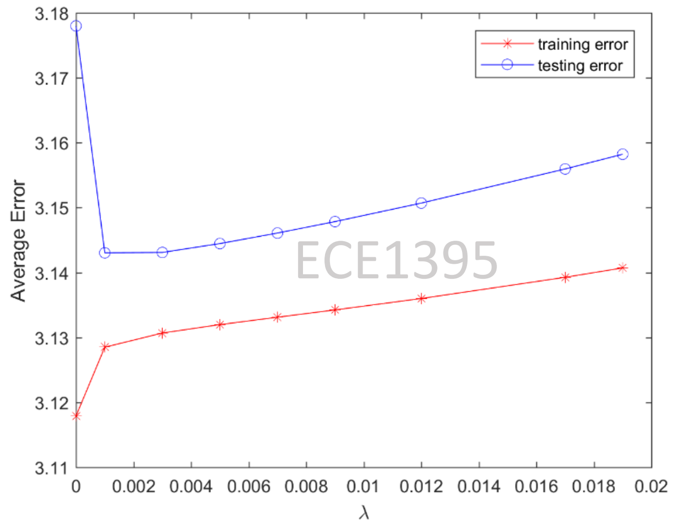
c. Compute the average training and testing error from 20 different model trained on this data. To do so, you will need to write a 'for loop' that runs for 20 iterations. For each iteration, you should do the following:
   i. **Randomly** split your data into training and testing sets using 88% for training. After this you should have these variables in your workspace: X_train and y_train as your training dataset, and X_test and y_test as your testing dataset. Note that these variables contents will not be identical in each iteration, since you're randomly splitting the original in each iteration.
   ii. Use the function you developed in (a), train eight linear regression models, one each of the following values of $\lambda = [0\ 0.001\ \ 0.003\ 0.005\ 0.007\ 0.009\ 0.012\ 0.017]$. Hint: you may need an inner loop to loop over the different values of λ.

iii. Compute the training error and testing error for each model (i.e., estimate of theta) for the corresponding value of λ. Use data structures to keep track of all the errors you compute. Hint: you can use your cost function from HW2.

Once your iterations are over, the dimension of training error and testing error matrices should be 20 (#iterations) x 8 (# of different values of λ). Take the average error over each column so that you have the average error for each value of λ. Plot the average training error and the average testing error vs λ. You figure should look similar, but not identical, to the figure below.

Output: A figure showing the average error vs λ as ps4-1-a.png

Text response: what value of lambda do you suggest for this particular problem/application? Comments on your figure.



2- **KNN - Effect of K**: In this part, you will use MATLAB built-in functions `fitknn` and `predict` to study the effect of K on the prediction accuracy and determine the optimal value of K for the given multiclass dataset. You will also learn about cross-validation. Load the data file 'hw4_data2.mat' into MATLAB. The data contains 5 equally sized folds (small dataset) obtained from a large training set. The folding was done, to enable us to test the algorithm using a proportion from the original training set. For each fold, you will find a training matrix (e.g., X1) and the corresponding labels vector (e.g., y1). Your results reported below should be an average of the results when you **train** on four folds and **test** on the remaining one. Thus, you will need to train **5 different KNN classifiers** and test them. For example, the first classifier is trained using the first 4 folds and tested using the fifth, where the second classifier is trained using the first three folds in addition to the fifth and then tested using the fourth fold (see the code snippet below); and so on for the remaining classifiers.

```
% second classifier data
X_train2 = [X1;X2;X3;X5];
y_train2 = [y1;y2,y3;y5];
X_test2 = X4;
y_test2 = y4;
```

a. Compute the average accuracy (over the five folds) for the following values of K = 1:2:15.
   Hint: To compute accuracy for one fold, check the ratio of test samples whose predicted labels are the same as the ground-truth labels, out of all test samples.
   Output: A figure showing average accuracy vs K as ps4-2-a.png
   Text output: what value of K do you suggest for this particular problem? Is this value robust to any other problem? Why?

2- **<u>Weighted NN</u>**: In this part you will implement and apply the Gaussian weighted neighbors classifier, using the equation in lecture slides, and apply it to some testing examples. For this example, you will need to use the training and testing samples in 'hw4_data3.mat': X_train and y_train are the training example and the corresponding class labels. X_test are the testing features that you are required to predict a class for, and y_test are the ground truth labels that you can use to compute the accuracy.

a. Write a function, `y_predict = weightedKNN(X_train,y_train,X_test,sigma)` that uses all neighbors to make a prediction on the test set, but weighs them according to their distance to the test sample. Use the Euclidian distance as your distance metric. Hint: you may find the MATLAB function `pdist2` useful.
   - `X_train` is an $m \times (n+1)$ features matix, where m is the number of training instances and n is the feature dimension,
   - `y_train` is an $m \times 1$ labels vector for the training instances,
   - `X_test` is an $d \times (n+1)$ feature matrix, where d is the number of test instances,
   - `sigma` is a scaler denoting the bandwidth of the Gaussian weighing function,
   - `y_predict` should be a $d \times 1$ vector that contains the predicted labels for the test instances.

   Function file: `weightedKNN.m` containing function `weightedKNN`

b. Test your function using the provided training matrix (X_train), training labels (y_train), and testing features matrix (X_test). Use the following values for sigma: 0.01, 0.1, 0.5, 1, 3, and 5. Compute the classification accuracy for each value of sigma.
   Output: a table that list the accuracy vs sigma
   Text output: comment on your results and the effect of sigma.