Homework Assignment 4: Regularization, KNN, and one-vs-all classification

Due Monday, February 13th, 2023 at 11:59pm

Description

In class, we learned about regularization to avoid over-fitting, KNN, and one-vs-all classification. 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 one-vs-all algorithm using logistic regression.

What to submit

Create a folder ps4_xxxx_LastName_FirstName (i.e. ps4_matlab_LastName_FirstName, or ps4_python_LastName_FirstName). The structure of your folder should be as follows:

ps4_xxxx_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 or ps4.py - your Matlab/Python 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 or *.py - Any other supporting files, including Matlab function files, python modules, etc.

Zip it as `ps4_xxxx_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.

[Important] Loading ".mat" data

Introduction: This assignment features the use of ".mat" data format. Files with the .mat extension are files that are in the binary data container format that the MATLAB program uses. Mathworks developed the extension and MAT files are categorized as data files that include variables, functions, arrays and other information. In this assignment, MAT files are used to hold the datasets for our questions.

MATLAB users: to read a mat file, you simple use the load function; e.g., load('hw_data1.mat'). After executing the load command the variables in the mat files are now loaded into your MATLAB workspace.

Python users: to read a mat file, you will need to use the scipy.io.loadmat function; e.g.
data2 = scipy.io.loadmat('hw4_data2.mat')
The variable data2 is a dictionary that holds all variable data from the MAT file, where each variable is a key in that dictionary. For example, listing the keys of data2 (py command: print(data2.keys())) yields the following:

```
dict_keys(['__header__', '__version__', '__globals__', 'X1', 'X2', 'X3', 'X4', 'X5', 'y1', 'y2', 'y3', 'y4', 'y5'])
```

You can see that this dictionary has 10 variables (X1 through X5 and y1 through y5). In your code, you have to read each of these variables into a numpy array. For example, to read the feature matrix X2, you would need to execute the following command:

```
X2 = np.array(data2["X2"])
```

To read the corresponding output vector, i.e., y2, you would need to execute the following command:

```
y2 = np.array(data2["y2"])
```

Repeat and do the same to read all other variables in the dictionary/MAT file.

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)

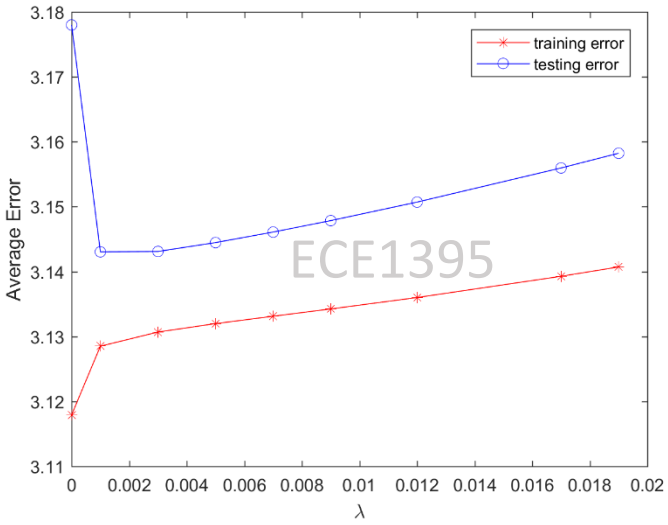
b. Load 'hw4_data1.mat' into your workspace. This MAT file has two variables: X_data is the feature matrix and y is the output vector. The features are already normalized, but you need to add the offset feature x_0 to your feature matrix, i.e., add a column of ones to your matrix.
Text output: what is 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` (python: `sklearn.neighbors.KneighborsClassifier`) 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 your workspace. 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 large 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?

3- **One-vs-all**: In this part, you will implement and apply the one-vs-all approach for multiclass classification. One-vs-all is an approach that can be used with any binary classifier. In this assignment, we are going to use it with a logistic regression classifier, using the equation in lecture slides, and apply it to some testing examples.

For this problem, 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. The dataset has points from three different classes, i.e., $y \in \{1, 2, 3\}$. Thus you need to train three different classifiers, 1 vs not 1, 2 vs not 2, and 3 vs not 3. For a given testing vector, the logistic regression output for each classifier is the probability that this vector belongs to that class. You classify that vector to the classifier that gives the highest probability.

While MATLAB/Python can automatically train a multi-class model, you will need to use only binary classifiers and implement the one-vs-all approach. For example, when creating the 2-vs-not 2 model, every training example that belongs to class 2 should get a label of 1 (positive), and any other class label should be made 0 (negative). You will need to the same for all other classifiers; positive class take a label of 1 and all others takes a label of 0.

a. Write a function, `y_predict = logReg_multi(X_train,y_train,X_test)` to implement the one-vs-all approach using logitic regression.
   - `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,
   - `y_predict` should be a $d \times 1$ vector that contains the predicted labels for the test instances.

   Ideally, your functions should be able to detect the number of different classes (C) from the y_train vector, **then train C different classifiers based on the one-vs-all approach** using a loop. However, it is ok if you would like to hard code three different classifiers for this very particular dataset.

   Once the C classifiers are trained, you are ready to make a prediction. Use every trained classifier to obtain the corresponding class probability for a given testing sample. You can use an array to record the output of each classifier for a given testing vector. Then, this testing sample get a label that corresponds to the classifier ID that yields the max probability among the other classifiers.

   MATLAB programmers, you can use `mdl = fitclinear(X_train,y_train,'learner','logistic');` to train one model. To get the probability output of that model use `[~,proba] = predict(mdl,X_test)`

   Python programmers, you can use `mdl = LogisticRegression(random_state=0).fit(X_train, y_train)`, after importing `from sklearn.linear_model import LogisticRegression`, to train one model. To get the probability output of that model use `mdl.predict_proba(X_test)`

All programmers, since you are training binary classifiers, in all of your models the values of y_train has to be either 0 or 1.

Also, the output of the probability predict function **is a 1x2 vector for each testing sample** [prob of negative class, prob of positive class]. You only need to keep the probability of the positive class for each training sample from each classifier.

Important: don't use the built-in mechanism for one-vs-all in any of the previous functions/methods, you **must implement your own one-vs-all procedure.**

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

b. Load 'hw4_data3.mat' into your workspace, and use your function to make predictions on the training and testing feature sets. Compute the training and testing accuracy of your model.
Output: a table that list the training accuracy and testing accuracy
Text output: comment on your results.