CS171 - Introduction to Machine Learning and Data Mining
Spring 2018 - Assignment 2

Instructor: Vagelis Papalexakis, University of California Riverside

In this assignment you will implement basic classifiers and test their performance on real datasets.

Question 0: Getting real data *[5%]*

In this assignment you will focus on a dataset that is suitable for binary classification:
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)
This link contains a collection of datasets gathered over the years. You are going to work on `breast-cancer-wisconsin.data` (and its description `breast-cancer-wisconsin.names`).
Make sure you can download the dataset, load it into your workspace, and make sure that every data point has entries for all features.
Important: If any data point has missing value(s), document how you are going to handle it, according to what we said in class.
Note: the serial number of each data point is stated as a feature but in this task it is not very useful, so remove it from the feature representation of your data.

Question 1: k-Nearest Neighbor Classifier *[50%]*

The second classifier we will see in this assignment is the k-Nearest Neighbor (or k-NN) classifier. k-NN is a lazy classification algorithm, which means that it skips the training step and makes a decision during testing time. For the distance calculations, you will use the Lp norm function you implemented in the previous assignment. You should implement the following function:
y_pred = knn_classifier(x_test, x_train, y_train, k, p)
Where the inputs are:

1. x_test = a 'test data point' x 'feature' matrix containing all the test data points that need to be classified
2. x_train = a 'training data point' x 'feature' matrix containing all the training data points that you will compare each test data point against.
3. y_train = a 'train data point' x 1 vector containing the labels for each training data point in x_train (there is obviously a 1-1 correspondence of rows between x_train and y_train).
4. k = the number of nearest neighbors you select
5. p = the parameter of Lp distance as we saw it in Assignment 1.

The output of the function is:

1. y_pred = a 'test data point' x 1 vector containing the predicted labels for each test data point in x_test (again, there is correspondence between the rows).

Question 2: Evaluation *[45%]*

In this last part of the assignment we will see how we evaluate different classifiers. For this, we are going to compute 1) classification accuracy, 2) sensitivity, and 3) specificity for every classifier configuration we are evaluating. Since we are going to be using 10-fold cross-validation, which systematically splits the data into train and test portions, you should merge the train and test set given to you and combine it to a single dataset.

Important: *We are only merging train and test sets that because cross-validation ensures that the train/test splits within the cross-validation are not overlapping, and therefore our training does not see any test data. If you are given a train/test dataset in any other scenario (e.g., a competition), and you are supposed to test the performance of your classifier in the test data, you should **NEVER** combine train and test data (by doing this you help your classifier "cheat"). In that case, you should conduct cross-validation on the **training data** and then test on the test data. We are only doing it here so that w*e have a big-enough *dataset, and because we are not required to test accuracy on the test data.*

1. **Cross-validation**: Implement 10-fold cross-validation. As described in the lectures, shuffle the entire dataset randomly, and split it in 10 equal-sized portions. Then, select one of the portions and make it the "test" set, and the rest will be the "training set". Train and test your classifier using these two sets and record the following **performance** measures: 1) accuracy, 2) sensitivity, and 3) specificity; for all those metrics, compute the mean and standard deviation (you may use existing functions for mean and stdev). We are going to need the standard deviation so that we report error-bars around the mean value. Repeat this procedure by selecting the next fold as the test set (and the remaining 9 as training), until you iterate over all folds. You can make a single script/source file for the cross-validation, which will wrap around the appropriate calls to the different training/testing procedures with the appropriate inputs, as per the questions below.

2. **Evaluating k-NN**: We are going to evaluate the performance of k-NN for various parameter choices. In particular, within each fold of the cross-validation, record mean and standard deviation of the performance for k = 1:10, for p = 1 and p =2. Organize those results as plots where the y-axis shows the performance and the x-axis shows the number of nearest neighbors (k). You may combine the lines for p = 1 and p = 2 in the same plot and add a legend that shows which lines refers to which parameter. As above, report error-bars. In total, this will be 3 plots (if you combine p = 1 and p = 2 in the same plot) or 2x3 plots if you don't. Which k and which p yield the best performance?

Question 3: Perceptron *[30%] - EXTRA CREDIT*

Doing questions 1 & 2 can give you up to 100%, so you may stop reading here. However, this question can give you extra credit, up to 130%:

Implementing the Perceptron *[20% of total]*

Here you will implement a basic Perceptron classifier with linear combination of inputs and sign() activation function, as we saw it in the lectures.

You should implement two functions:

1. function w = train_perceptron(input_x, output_y, w_init);
   This function takes as inputs:
   a. input_x = a 'data point' x 'feature' matrix, each row of which contains a training point
   b. output_y = a vector of desired outputs. Note that input_x and output_y should match on the 'row' dimension!
   c. w_init = this is an initialization for the Perceptron weights. This can be random (when you start the training) or a previously trained set of weights which you wish to update.

The output of this function is:
w = This is the vector of weights for the Perceptron.

2. function y_pred = classify_perceptron(input_x, w);
   This function takes as inputs:
   a. input_x = a 'data point' x 'feature' matrix, each row of which contains a test point that needs to be classified
   b. w = the trained Perceptron weights from 'train_perceptron'
   The output of this function is:
   y_pred = a 'data point' x 1 vector containing the classification outcome for every point in input_x.

Evaluating Perceptron *[10% of total]*: Within the cross-validation process that you developed in Question 2, you need to compare the performance of the perceptron in two scenarios: 1) the initialization of the weights is all zeros, and 2) the weights are initialized randomly. For initialization #2 because you are introducing randomness in the process, you should run 10 independent train/test instances of the Perceptron and keep the mean performance. We do this so that we smoothen out potential random effects introduced by different initializations of the weights. Report in figures all three performance measures for both configurations (zero weight initialization and random weight initialization), which amounts to 2x3=6 plots, containing error-bars around the average performance measure (hint: you may use functions like Matlab's 'errorbar'). Which initialization works better?

PLEASE READ BELOW - IMPORTANT INFORMATION

Programming language: For this assignment you are going to use *Matlab* or *Python*. Unless noted otherwise, you may ***not*** use library functions that already implement the questions that you have to implement in this assignment. If you are using a non-standard library or function, you must cite it and explain why it is necessary for your implementation.

Deliverables:

1. A report in PDF format describing the approach taken in each question and containing the answers and figures required in each question.
2. **In your PDF you should include**: a) Name, b) UCR ID, c) Late days used for this assignment, d) Total late days used so far.
3. Your code, organized by question, and properly commented.

Both deliverables described above should be uploaded as a single archive on iLearn.

Deadline: Please check the class website for the most current date. The deadline is on 11:59pm on the day specified.

Late policy: Please check the class website for the late policy.

Grade distribution: Unless noted otherwise, the number of points for each question are equally distributed to each sub-question.

Academic Integrity: Each assignment should be done *individually.* You may discuss general approaches with other students in the class, and ask questions to the TAs, but *you must only submit work that is yours*. If you receive help by any external sources (other than the TA and the instructor), you must properly credit those sources, and if the help is significant, the appropriate grade reduction will be applied. If you fail to do so, the instructor and the TAs are obligated to take the appropriate actions outlined at http://conduct.ucr.edu/policies/academicintegrity.html. Please read carefully the UCR academic integrity policies included in the link.