CS 178: Machine Learning: Winter 2018

Homework 1

Due Date: **Thursday, January 18, 2018**

This homework (and many subsequent ones) involves data analysis, and discussion of methods and results, using Python. You must submit ***a single PDF file*** that contains all answers, including any text needed to describe your results, the complete code snippets used to answer each problem, any figures that were generated, and scans of any (clearly readable) work on paper that you want the graders to consider. It is important that you include enough detail that we know how you solved each problem, since otherwise we will be unable to grade it.

We recommend that you use Jupyter/iPython notebooks to write your report. It will help you not only ensure all of the code for the solutions is included, but also provide an easy way to export your results to a PDF file.[1] We recommend liberal use of Markdown cells to create headers for each problem and sub-problem, explaining your implementation/answers, and including any mathematical equations. For parts of the homework you do on paper, scan it in such that it is legible (there are a number of free Android/iOS scanning apps, if you do not have access to a scanner), and include it as an image in the iPython notebook. If you have any questions about using iPython, ask us on Piazza. If you decide not to use iPython notebooks, and instead create your PDF file with Word or LaTeX, make sure *all* of the answers can be generated from the code snippets included in the document.

Problem 0: Get Connected (0 points, but it will make the course easier!)

Please visit our class forum on Piazza: `http://piazza.com/uci/winter2018/cs178/home`. Piazza will be the place to post your questions and discussions, rather than by email to the instructor or TAs. Often, other students have the same or similar questions, and will be helped by seeing the online discussion.

Problem 1: Python & Data Exploration (20 points)

In this problem, we will compute some basic statistics and create visualizations of an example data set. First, download the zip file for Homework 1, which contains some course code (the `mltools` directory) and the "Fisher iris" data set introduced in lecture. Load the data into Python:

```
import numpy as np
import matplotlib.pyplot as plt

iris = np.genfromtxt("data/iris.txt",delimiter=None)  # load the text file
Y = iris[:,-1]              # target value (iris species) is the last column
X = iris[:,0:-1]            # features are the other columns
```

The Iris data consist of four real-valued features used to predict which of three types of iris flower was measured (a three-class classification problem).

1. Use `X.shape` to get the number of features and the number of data points. Report both numbers, mentioning which number is which. *(5 points)*
2. For each feature, plot a histogram (`plt.hist`) of the data values. *(5 points)*
3. Compute the mean & standard deviation of the data points for each feature (`np.mean`, `np.std`). *(5 points)*
4. For each pair of features (1,2), (1,3), and (1,4), plot a scatterplot (see `plt.plot` or `plt.scatter`) of the feature values, colored according to their target value (class). (For example, plot all data points with $y = 0$ as blue, $y = 1$ as green, and $y = 2$ as red.) *(5 points)*

[1] For example, by doing a *Print Preview* in Chrome and *printing* it to a PDF.

Problem 2: k-nearest-neighbor predictions (25 points)

In this problem, you will continue to use the Iris data and create a k-nearest-neighbor (kNN) classifier using the provided `knnClassify` python class. While completing this problem, please explore the implementation to become familiar with how it works.

First, we will shuffle and split the data into training and validation subsets:

```
iris = np.genfromtxt("data/iris.txt",delimiter=None)  # load the data
Y = iris[:,-1]
X = iris[:,0:-1]
# Note: indexing with ":" indicates all values (in this case, all rows);
#  indexing with a value ("0", "1", "-1", etc.) extracts only that value (here, columns);
#  indexing rows/columns with a range ("1:-1") extracts any row/column in that range.

import mltools as ml
# We'll use some data manipulation routines in the provided class code
# Make sure the "mltools" directory is in a directory on your Python path, e.g.,
#   export PYTHONPATH=$\$${PYTHONPATH}:/path/to/parent/dir
# or add it to your path inside Python:
#   import sys
#   sys.path.append('/path/to/parent/dir/');

np.random.seed(0)           # set the random number seed
X,Y = ml.shuffleData(X,Y);  # shuffle data randomly
# (This is a good idea in case your data are ordered in some systematic way.)

Xtr,Xva,Ytr,Yva = ml.splitData(X,Y, 0.75);  # split data into 75/25 train/validation
```

Make sure to set the random number seed to 0 before calling shuffleData as in the example above (and in general, for every assignment). This ensures consistent behavior each time the code is run.

Learner Objects Our learners (the parameterized functions that do the prediction) will be defined as python objects, derived from either an abstract classifier or abstract regressor class. The abstract base classes have a few useful functions, such as computing error rates or other measures of quality. More importantly, the learners will all follow a generic behavioral pattern, allowing us to train the function on one data set (i.e., set the parameters of the model to perform well on those data), and then make predictions on another data set.

You can now build and *train* a kNN classifier on `Xtr,Ytr` and make predictions on some data `Xva` with it:

```
knn = ml.knn.knnClassify()  # create the object and train it
knn.train(Xtr, Ytr, K)      # where K is an integer, e.g. 1 for nearest neighbor prediction
YvaHat = knn.predict(Xva)   # get estimates of y for each data point in Xva

# Alternatively, the constructor provides a shortcut to "train":
knn = ml.knn.knnClassify( Xtr, Ytr, K );
YvaHat = predict( knn, Xva );
```

If your data are 2D, you can visualize the data set and a classifier's decision regions using the function

```
ml.plotClassify2D( knn, Xtr, Ytr );  # make 2D classification plot with data (Xtr,Ytr)
```

This function plots the training data and colored points as per their labels, then calls `knn`'s `predict` function on a densely spaced grid of points in the 2D space, and uses this to produce the background color. Calling the function with `knn=None` will plot only the data.

1. Modify the code listed above to use only the first two features of X (e.g., let X be only the first two columns of `iris`, instead of the first four), and visualize (plot) the classification boundary for varying values of $K = [1, 5, 10, 50]$ using `plotClassify2D`. *(10 points)*

2. Again using only the first two features, compute the error rate (number of misclassifications) on both the training and validation data as a function of $K = [1, 2, 5, 10, 50, 100, 200]$. You can do this most easily with a for-loop:

```
K=[1,2,5,10,50,100,200];
for i,k in enumerate(K):
  learner = ml.knn.knnClassify(...  # TODO: complete code to train model
  Yhat = learner.predict(...    # TODO: predict results on training data
  errTrain[i] = ...             # TODO: count what fraction of predictions are wrong
  #TODO: repeat prediction / error evaluation for validation data

plt.semilogx(...                #TODO: average and plot results on semi-log scale
```

Plot the resulting error rate functions using a semi-log plot (`semilogx`), with training error in red and validation error in green. Based on these plots, what value of K would you recommend? *(10 points)*

3. Create the same error rate plots as the previous part, but with all the features in the dataset. Are the plots very different? Is your recommendation for the best K different? *(5 points)*

Problem 3: Naïve Bayes Classifiers (50 points)

In order to reduce my email load, I decide to implement a machine learning algorithm to decide whether or not I should read an email, or simply file it away instead. To train my model, I obtain the following data set of binary-valued features about each email, including whether I know the author or not, whether the email is long or short, and whether it has any of several key words, along with my final decision about whether to read it ($y = +1$ for "read", $y = -1$ for "discard").

x_1 know author?	x_2 is long?	x_3 has 'research'	x_4 has 'grade'	x_5 has 'lottery'	y ⇒ read?
0	0	1	1	0	-1
1	1	0	1	0	-1
0	1	1	1	1	-1
1	1	1	1	0	-1
0	1	0	0	0	-1
1	0	1	1	1	1
0	0	1	0	0	1
1	0	0	0	0	1
1	0	1	1	0	1
1	1	1	1	1	-1

I decide to try a naïve Bayes classifier to make my decisions and compute my uncertainty. In the case of any ties where both classes have equal probability, we will prefer to predict class +1.

1. Compute all the probabilities necessary for a naïve Bayes classifier, i.e., the class probability $p(y)$ and all the individual feature probabilities $p(x_i|y)$, for each class y and feature x_i. *(10 points)*

2. Which class would be predicted for $\underline{x} = (0\ 0\ 0\ 0\ 0)$? What about for $\underline{x} = (1\ 1\ 0\ 1\ 0)$? *(10 points)*

3. Compute the posterior probability that $y = +1$ given the observation $\underline{x} = (0\ 0\ 0\ 0\ 0)$. Also compute the posterior probability that $y = +1$ given the observation $\underline{x} = (1\ 1\ 0\ 1\ 0)$. *(10 points)*

4. Why should we probably not use a "joint" Bayes classifier (using the joint probability of the features x, as opposed to the conditional independencies assumed by naïve Bayes) for these data? *(10 points)*

5. Suppose that before we make our predictions, we lose access to my address book, so that we cannot tell whether the email author is known. Do we need to re-train the model to classify based solely on the other four features? If so, how? Hint: How do the parameters of a naïve Bayes model over only features $x_2, \ldots, x_5$ differ? *(10 points)*

Problem 4: Statement of Collaboration (5 points)

It is **mandatory** to include a *Statement of Collaboration* in each submission, that follows the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments in particular, I encourage students to organize (perhaps using Piazza) to discuss the task descriptions, requirements, possible bugs in the support code, and the relevant technical content *before* they start working on it. However, you should not discuss the specific solutions, and as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (no photographs of the blackboard, written notes, referring to Piazza, etc.). Especially *after* you have started working on the assignment, try to restrict the discussion to Piazza as much as possible, so that there is no doubt as to the extent of your collaboration.