Week 7 - Assessed Exercises

Data Programming with Python

This week we learnt how to load and save `Series` and `DataFrames` from and to data files. We then looked into combining, cutting and binning datasets. In this assignment you will learn about classification problems and how what we learnt this week can be used to solve them.

Each question asks you to write a function with a specific set of input arguments. The *.py* template defines the function name and inputs for each question, **do not** change these. Be sure you test your functions before you submit your code to make sure that they are outputting the correct answer. Unless otherwise stated, all functions must have a return value.

Include the import statements for all packages used within your code. Additionally, please include the package prefixes (`pd`, `np`, etc.) for functions/methods from these packages, even if the command runs in Canopy without the prefix.

This week we will use the `ebola_test` dataset to test the code. It contains the results for a new medical test for detecting Ebola that is much faster, though less accurate, than that currently available. The new test has been applied to a number of patients who are known (from the old test) to have the disease or not. The first column of the data set (called *prob*) is the probability under the new test that the patient has Ebola. The second column (*ebola*) is the result from the older test which definitively says whether the patient has ebola (*ebola*=1) or not (*ebola*=0). Download the dataset, load it in and have a look at the first few entries to see what it looks like.

The *.py* file contains a suggested tests for each function. You don't need to include the output of your tests in your PDF.

In a classification problem we have an indicator variable and a set of observation variable. For these exercises, we will assume that there is only one observation variable. Our classifier will use the observation variable to predict the indicator variable and then test how good the classification was based on how many time its got it correct.

1. Write a function that returns a `dict` with some information about the `DataFrame` *df*. The keys of the `dict` should be *Percentage* and *Quartiles*. The value for *Percentage* should be a single number (not a list with a number in it), specifying the percentage of entries with positive results for the given criteria, i.e. indicator variable is 1. This value should be rounded to 1 decimal place. The value for *Quartiles* should be a list (not a Series or array) with the number of observations in the 1st quartile (0-25%), 2nd quartile (25-50%), 3rd quartile (50-75%) and 4th quartile (75-100%) for a observation variable. The names of indicator and observation variables should be given to the function as strings. Be sure that your keys are exactly as specified above and that the values have the data type specified above.

In a classification problem one must define a *cutoff*, a value for which observations above that value are classified as positive results (and assigned the value 1), and those less than or equal to the value are classified as negative results (and assigned the value 0). If we had a perfect classifier, the predicted values (0s or 1s) would match the indicator variable. A false positive is defined as the classifier predicting a positive test result (1), when the actual result was negative (0). A false negative is the opposite, the classifier predicts a negative test result (0), when the true result was positive (1).

2. Write a function that takes a `DataFrame` *df*, two strings specifying the names of indicator column and the observation column, and a *cutoff* value, and returns the rate of false positive and rate of the false negative as a `dict`. The keys of the `dict` should be *False Pos* and *False Neg* and the values must be rounded to 3 decimal places.

We do not know a priori what the best *cutoff* value is, as such, we should loop over a range of *cutoff* values to decide on the best one.

3. Write a function that takes the same inputs as Q2, but *cutoff* will now be replaced with *cutoff_list* (an array of different *cutoff* values to test). The function should run the classification for each value in *cutoff_list* and determine which is the best *cutoff* value. We will define the best classifier as having the lowest false results (false positives plus false negatives). The function should return a `dict` with the keys *Cutoff value*, *False Pos* and *False Neg*, and the values of the false positive rate and false negative rate should be rounded to 3 decimal places

A related concept to the choice of cut-offs is the Receiver Operator Characteristic (ROC) curve. The ROC curve aims to quantify how well a classifier beats a random classifier for any level of probability cut-off. An introduction can be found here: `http://en.wikipedia.org/wiki/Receiver_operating_characteristic` The idea is to plot the false positive rate against the true positive rate for every possible cut-off

4. Write a function which calculates the ROC curve. The function should have three arguments, the `DataFrame` *df*, the name of the indicator variable *ind_col* and the name of the observation variable *obs_col*. Your ROC curve function should perform the following steps:
   (a) Find the unique values in the observation column
   (b) Use each of these unique values as a *cutoff* value and
      i. Classify all the observations as either positive or negative based on the current *cutoff* value
      ii. Calculate the number of true positives (tp), true negatives (tn), false positives (fp) and false negatives (fn)

      Note 1: a tp is when the classification value and actual value are both 1, a tn is when they are both 0
      Note 2: tp, fn, etc must all be vectors/Series of the same length as the vector/Series of *cutoff* values
      Note 3: be careful when you're at the maximum *cutoff* value that you can still calculate these values correctly
   (c) Create the true positive rate (tpr) as tp/(tp+fn)
   (d) Create the false positive rate (fpr) as fp/(fp+tn)
   (e) Create a `DataFrame`, indexed by the *cutoff* values (unique values of observation column), with columns *True_Pos* and *False_Pos*, containing tpr and fpr, respectively.
   (f) Return the `DataFrame` sorted by index (lowest to highest)

All of your code should be written into the *.py* template. Save your filled *.py* file with the following name structure *SurnameFirstname_Week7.py* (where *Surname* and *Firstname* should be replaced with your name) and upload it to Brightspace. Additionally, you must upload a PDF of your code. Create a PDF from Canopy by selecting *File* → *Print*, and print to PDF.