Please include your Name and USC ID in the first cell of the Jupyter Notebook file for each assignment. Name your jupyter file as Lastname\_Firstname\_HW(i).ipynb, for example, John Doe’s HW2 should look like Doe\_John\_HW2.ipynb. Similar to HW1, a single ipynb file is the deliverable for this homework.

# Preliminaries and Setup

This homework assumes that you have completed hw-1 and are familiar enough with Python and Jupyter by now. Our main goal in this homework is to play with **sklearn**, which is the primary toolkit in Python that is used for basic machine learning like the model we studied in class (Naive Bayes).

As a first step, make sure that you are able to import the following into your Jupyter notebook. You may have to ‘pip install’ the right packages to make this work, but in environments like Anaconda they are already included.

import sklearn # make sure this is installed in your environment.

from sklearn.datasets import \*

from sklearn import tree

For our experiments, we will use a dataset publicly available in the UCI repository called the **Bank Marketing Data Set.** While this is not a 'text' dataset, as we studied in class, the classification models/supervised learning tend to involve the same kinds of workflow. This dataset is easier to work with than a text dataset because of the presence of numeric attributes (not requiring you to really know 'vector space models' in detail). Think of it as a review of running machine learning experiments, and conducting the appropriate statistical analyses.

Go to the website to read more about the dataset: <https://archive.ics.uci.edu/ml/datasets/Bank+Marketing>

**All files mentioned/referenced below can be found at this link.**

We will be using the **bank-additional-full.csv** file for our experiments. This is the most complete

dataset (i.e. has all 41,000+ examples and 20 attributes). It is a binary classification task (the very last column,

which is either a yes or a no). However, as you'll see, even before we can get to all the good model-fitting

stuff, there's some data cleaning/processing that we'll need to do first.

To help you complete this assignment, I will be providing function signatures and specifications that you have to fill in with actual code/logic in your Jupyter notebook. In Python, we often use 'pass' as a placeholder. Wherever you see 'pass', it's a sign that you should replace with your own code (which could span multiple lines). Once again, everything must be implemented in a single Jupyter notebook!

***Some other things to note:***

* If it makes you comfortable to define additional functions or data structures (including using pandas) to help you, go for it.
* It is always good to 'execute' the functions as necessary, as you proceed with the assignment. This will give you confidence you are implementing things correctly! Remember that *an early bird catches the worm, but an early error becomes the worm.*

**DISCLAIMER:** The first part of this exercise (in my opinion) is the most time-consuming, despite technically being the easiest, just like in actual analytics projects (in practice, about 80-90% time in machine learning/data science projects in the real-world are taken up by 'data cleaning/wrangling').

Perhaps a shorter way to state the hint above is, read the entire assignment before jumping on to it, and be strategic about how you allocate time to the various problems. Don't get frustrated!

Total possible points are 100, although sum of individual question totals may add to more than 100 (so you have opportunity for extra credit).

Good luck!

# HW-2

"""

[30 points] Complete encode\_record\_into\_vector and parse\_file\_into\_matrix

"""

def encode\_record\_into\_vector(record):

"""

the goal of this function is to take a record (informally, a 'row' in your file)

and 'encode' it numerically so that we can actually do

machine learning with it. How you do the encoding is up to you (e.g., you can use pandas), but one recommended way is to:

--leave the numeric variables (like age) as is

--assign a 'one hot encoding' to each possible value of a `categorical' variable. For example, for the categorical

variable 'loan', we have three possible values ('no', 'yes', 'unknown'). Since there are three possible values,

you can do a 'one hot encoding' by using three binary 'variables'. We may want to assign 'no' to 0 0 1,

'yes' to 0 1 0 and 'unknown' to 1 0 0 (you can see now why it's called one hot encoding).

:param record: I highly recommend that record be a 'dict', but you can use other data structures (like 'list') if

you want. For example,

you could even just pass in a string! However, the output MUST be a numeric vector.

Also, we will not do 'type' checking on this function, so you don't need to waste time checking for exceptions,

or worrying that we will try to trip you up by running your code using weird inputs.

The only requirement is that code you write must work for the dataset used in this assignment.

:return:

x: the vector representation of the record

"""

x = None

pass # Hint: use helper functions to encode categorical variables as vectors that can be appended to the (bigger) x vector

# rather than write a lot of messy code here. It will also help you to try more things.

return x

def parse\_file\_into\_matrix(file\_name):

"""

Hint 1: Even though the file is a .csv, be careful about using Python's csv package for reading in the file.

The single biggest source of error is not reading in the file correctly. It's good to test that you're

doing the initial data processing correctly on the first few records.

Hint 2: This function will make a call to encode\_record\_ for each record (not including header) in your input file

:param file\_name: The path to the bank-additional-full.csv file

:return:

X: A D X p matrix, where D is the number of 'instances' or records in the file and p is the dimensionality of

the x vector that is output by the previous function. I won't give you the value for p; you have to know what it is

based on your code in encode\_record\_into\_vector

y: A D X 1 matrix containing only 1's (for 'yes' in the output variable) or 0's (for 'no')

"""

X=None

y=None

pass

return (X, y)

"""

We will define 'positive' instances as those with label 1, and negative instances as those with label 0.

Q1 [5 points]. Write some code to count the number of 1s and 0s in y. How many positive and negative instances each

are in your dataset?

ANS:

"""

"""

If you want to do the next part before the previous part, I recommend calling X\_y\_for\_running\_tests below,

which reads in a sample dataset from sklearn. I've included a short snippet of code below for your reference. Remember, however, that

to score points, you must work with the banking dataset in the output you submit.

"""

def X\_y\_for\_running\_tests():

# just call this, and it will return X and y.

X, y = load\_digits(2, True) # return only two classes, although there are ten total

return X, y

# X, y=X\_y\_for\_running\_tests()

# print X

# print 'now printing y'

# print y

def training\_testing\_split(X, y, training\_ratio=0.8):

"""

Here's an opportunity to test your sampling skills using numpy. We want to do stratified sampling on X, y. Basically,

this means that we want to 'split' the original X, y (the 'complete' dataset) into a training dataset (X\_train, y\_train)

that will be used for training the model, and a testing dataset (X\_test, y\_test) that will be used for evaluating

the model. Here are the requirements:

--Since the sampling is stratified, we want to make sure that the proportion of positive instances (to total instances)

is equal in both training and testing data. For example:

Imagine that you had 100 positive instances and 50 negative instances in your full dataset. Suppose the training\_Ratio

is 80%, as specified by default in the signature. Then, we want to randomly sample 0.8\*100 positive instances (or 80

instances) and 0.8\*50 negative instances (or 40 instances) and place all 120 instances in X\_train (correspondingly, y\_train

is filled with 1s and 0s based on the label). The remaining 30 instances (20 positive and 10 negative) are

placed in X\_test (with y\_test populated with 1s and 0s correspondingly). Notice that the ratio of positive

to positive+negative is equal in both training and testing datasets (compute it for yourself), and by extension,

so is the ratio of negative to positive+negative. There is a very good reason why this is so important (recall

the definition of learning in our very first slide! Would the test data still be from the same population as the

training data if we did not decide to 'stratify' the sample in this way?)

--Your method should work for any numeric choice of X, y and training ratio (that is between 0 and 1, including the extreme

cases of 0 and 1). We may test this function with X, y and training\_ratio values of our own!

"""

X\_train, y\_train, X\_test, y\_test = None

pass

return (X\_train, y\_train, X\_test, y\_test)

"""

Q2 (20 points). Run the code above with the X and y that you got from parse\_file\_into\_matrix, with training ratios

of 0.8, 0.5, 0.3 and 0.1. For each of these four cases, what is the 'ratio' of positive instances in the training

dataset to the total number of instances in the training dataset? Verify that this same ratio is achieved in the test

dataset. Write additional code to run these verifications if necessary (5 points per case).

"""

def train\_models(X\_train,y\_train,model='decision\_tree'):

"""

In the code below, I have trained a model specifically for decision tree. You must expand the code to accommodate

the other two models. To learn more about sklearn's decision trees, see https://scikit-learn.org/stable/modules/tree.html

:param X\_train: self-explanatory

:param y\_train:

:param model: we will allow three values for model namely 'decision\_tree', 'naive\_bayes' and 'linear\_SGD\_classifier'

(Hint: you must change the loss function in https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html

to squared\_loss to correctly implement the linear classifier. For the naive bayes, the appropriate model to use

is the Bernoulli naive Bayes.)

:return:

"""

if model == 'decision\_tree':

clf = tree.DecisionTreeClassifier()

clf = clf.fit(X\_train, y\_train)

return clf

pass

"""

[10 points] Expand/replace 'pass' above to return the other two models based on the value of the model parameter 'model' [10 points]

[5 points] Expand to return one other model that I have not taught in class.

"""

"""

Code for this function has already been written

"""

def evaluate\_model(X\_test, y\_test, model):

"""

Note that the model here is different from model in train\_models. Here, we will be passing in the 'actual' trained

model and using it to evaluate X\_test and y\_test

We will be using the F\_measure metric to evaluate the model: I have already written code in compute\_f\_measure

that takes your predicted y, produced in this function, as well as the 'true' y, which is y\_test, and will

return a number to you within 0 and 1. The higher the f-measure, the better. I will briefly explain in class

why we prefer f-measure over accuracy in many ML tasks.

"""

y\_predict = model.predict(X\_test)

return sklearn.metrics.f1\_score(y\_test, y\_predict)

"""

[20 points] This part of the assignment involves statistical analysis of results you’ve generated. As a first step, we would like you to generate 5 tables with 10 rows per table. Each table will correspond to a value of training percent, specifically 10%,30%,50%,70%,90%

Each table will contain four columns (trial number ranging from 1-10, decision\_tree, naive\_bayes and linear\_classifier). In each cell, you will be reporting the f-measure achieved.

[15 points] Now we will return to some statistics. You should

ignore the linear classifier for this question. Our research (hint: is this null or alternative?) hypothesis is that the

naive bayes is better than decision tree using the f-measure metric. For **which** (there may be zero, one or more answers) of the five training percentages, using 95% as the confidence level, can you reject the null hypothesis? State your p-values here for all five training percentages. Which test did you use?