Project Submission

This notebook will be your project submission. All tasks will be listed in the order of the Courses that they appear in. The tasks will be the same as in the Capstone Example Notebook, but in this submission you ***MUST*** use another dataset. Failure to do so will result in a large penalty to your grade in this course.

Finding your dataset

Take some time to find an interesting dataset! There is a reading discussing various places where datasets can be found, but if you are able to process it, go ahead and use it! Do note, for some tasks in this project, each entry will need 3+ attributes, so keep that in mind when finding datasets. After you have found your dataset, the tasks will continue as in the Example Notebook. You will be graded based on the tasks and your results. Best of luck!

As Reviewer:

Your job will be to verify the calculations made at each "TODO" labeled throughout the notebook.

First Step: Imports

In the next cell we will give you all of the imports you should need to do your project. Feel free to add more if you would like, but these should be sufficient.

In [1]:

```
import gzip
import pandas as pd
from collections import defaultdict
import random
import numpy
import scipy.optimize
import string
from sklearn import linear_model
from nltk.stem.porter import PorterStemmer # Stemming
```

Task 1: Data Processing

TODO 1: Read the data and Fill your dataset

In [2]:

```
#YOUR CODE HERE
df = pd.read_csv("winequality-red.csv")
df.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9

TODO 2: Split the data into a Training and Testing set

First shuffle your data, then split your data. Have Training be the first 80%, and testing be the remaining 20%.

In [3]:

```
#YOUR CODE HERE
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.drop('quality', 1), df['quality'], t
```

Now delete your dataset

You don't want any of your answers to come from your original dataset any longer, but rather your Training Set, this will help you to not make any mistakes later on, especialy when referencing the checkpoint solutions.

In [4]:

```
#YOUR CODE HERE
del df
```

TODO 3: Extracting Basic Statistics

Next you need to answer some questions through any means (i.e. write a function or just find the answer) all based on the **Training Set:**

1. How many entries are in your dataset?
2. Pick a non-trivial attribute (i.e. verified purchases in example), what percentage of your data has this atttribute?
3. Pick another different non-trivial attribute, what percentage of your data share both attributes?

In [5]:

```
print("No of entries in my training set : {}".format(len(X_train)))
print("Value Counts")
print(y_train.value_counts())

print("% of {} in our dataset is : {}".format(5,(547*100)/len(y_train)))
print("% of {} in our dataset is : {}".format(6,(520*100)/len(y_train)))
print("% of {} in our dataset is : {}".format(7,(152*100)/len(y_train)))
```

```
No of entries in my training set : 1279
Value Counts
5    539
6    507
7    161
4     47
8     17
3      8
Name: quality, dtype: int64
% of 5 in our dataset is : 42.76778733385457
% of 6 in our dataset is : 40.65676309616888
% of 7 in our dataset is : 11.884284597341674
```

Task 2: Classification

Next you will use our knowledge of classification to extract features and make predictions based on them. Here you will be using a Logistic Regression Model, keep this in mind so you know where to get help from.

TODO 1: Define the feature function

This implementation will be based on ***any two*** attributes from your dataset. You will be using these two attributes to predict a third. Hint: Remember the offset!

In [6]:

```
#FIX THIS

def feature(d):
    feat = [1, d['pH'], d['alcohol']]
    return feat
```

TODO 2: Fit your model

1. Create your **Feature Vector** based on your feature function defined above.
2. Create your **Label Vector** based on the "verified purchase" column of your training set.
3. Define your model as a **Logistic Regression** model.
4. Fit your model.

In [7]:

```
#YOUR CODE HERE
X = [feature(row) for index,row in X_train.iterrows()]
Y = y_train.values.tolist()

lr = linear_model.LogisticRegression(solver='lbfgs',multi_class='auto',max_iter=1000)
lr.fit(X,Y)
```

Out[7]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

TODO 3: Compute Accuracy of Your Model

1. Make **Predictions** based on your model.
2. Compute the **Accuracy** of your model.

In [8]:

```
#YOUR CODE HERE
x_test = [feature(row) for index,row in X_test.iterrows()]
predictions = lr.predict(x_test)
from sklearn import metrics
metrics.accuracy_score(predictions,y_test)
```

Out[8]:

```
0.5875
```

Task 3: Regression

In this section you will start by working though two examples of altering features to further differentiate. Then you will work through how to evaluate a Regularaized model.

In [9]:

```
#CHANGE PATH
path = "amazon_reviews_us_Digital_Video_Games_v1_00.tsv.gz"

#GIVEN
f = gzip.open(path, 'rt', encoding="utf8")
header = f.readline()
header = header.strip().split('\t')
reg_dataset = []
for line in f:
    fields = line.strip().split('\t')
    d = dict(zip(header, fields))
    d['star_rating'] = int(d['star_rating'])
    reg_dataset.append(d)
```

TODO 1: Unique Words in a Sample Set

We are going to work with a new dataset here, as such we are going to take a smaller portion of the set and call it a Sample Set. This is because stemming on the normal training set will take a very long time. (Feel free to change sampleSet -> reg_dataset if you would like to see the difference for yourself)

1. Count the number of unique words found within the 'review body' portion of the sample set defined below, making sure to **Ignore Punctuation and Capitalization**.
2. Count the number of unique words found within the 'review body' portion of the sample set defined below, this time with use of **Stemming, Ignoring Puctuation, *and* Capitalization**.

In [10]:

```
#GIVEN for 1.
wordCount = defaultdict(int)
punctuation = set(string.punctuation)

#GIVEN for 2.
wordCountStem = defaultdict(int)
stemmer = PorterStemmer() #use stemmer.stem(stuff)

#SampleSet and y vector given
sampleSet = reg_dataset[:2*len(reg_dataset)//10]
y_reg = [d['star_rating'] for d in sampleSet]
```

In [11]:

```
#YOUR CODE HERE
for i in sampleSet:
    r_body = i['review_body']
    r_body = r_body.lower()
    r_body = "".join([j for j in r_body if j not in punctuation])
    words = r_body.split(" ")
    for w in words:
        wordCount[w] += 1
len(wordCount)
```

Out[11]:

```
30022
```

In [12]:

```
for i in sampleSet:
    r_body = i['review_body']
    r_body = r_body.lower()
    r_body = "".join([j for j in r_body if j not in punctuation])
    words = r_body.split(" ")
    for w in words:
        wordCountStem[stemmer.stem(w)] += 1
len(wordCountStem)
```

Out[12]:

```
22818
```

TODO 2: Evaluating Classifiers

1. Given the feature function and your counts vector, **Define** your X_reg vector. (This being the X vector, simply labeled for the Regression model)

2. **Fit** your model using a **Ridge Model** with (alpha = 1.0, fit_intercept = True).
3. Using your model, **Make your Predictions**.
4. Find the **MSE** between your predictions and your y_reg vector.

In [13]:

```
#GIVEN FUNCTIONS
def feature_reg(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review_body'].lower() if not c in punctuation])
    for w in r.split():
        if w in wordSet:
            feat[wordId[w]] += 1
    return feat

def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)

#GIVEN COUNTS AND SETS
counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()

#Note: increasing the size of the dictionary may require a lot of memory
words = [x[1] for x in counts[:100]]

wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
```

In [14]:

```
#YOUR CODE HERE
X_reg = [feature_reg(x) for x in sampleSet]
model = linear_model.Ridge(1.0,fit_intercept = True)
model.fit(X_reg,y_reg)
pred = model.predict(X_reg)
```

In [15]:

```
print(MSE(pred,y_reg))
```

```
1.678402484529701
```

Task 4: Recommendation Systems

For your final task, you will use your knowledge of simple similarity-based recommender systems to make calculate the most similar items.

The next cell contains some starter code that you will need for your tasks in this section. Notice you should be back to using your **trainingSet**.

In [16]:

```
#GIVEN
usersPerItem = defaultdict(set)
itemsPerUser = defaultdict(set)
```

TODO 1: Fill your Dictionaries

1. For each entry in your training set, fill your default dictionaries (defined above).

In [17]:

```
#YOUR CODE HERE
itemNames = {}

for d in reg_dataset:
    user,item = d['customer_id'], d['product_id']
    usersPerItem[item].add(user)
    itemsPerUser[user].add(item)
    itemNames[item] = d['product_title']
```

In [18]:

```
#GIVEN
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom

def mostSimilar(n, m): #n is the entry index
    similarities = []  #m is the number of entries
    users = usersPerItem[n]
    for i2 in usersPerItem:
        if i2 == n: continue
        sim = Jaccard(users, usersPerItem[n])
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:m]
```

TODO 1: Fill your Dictionaries

1. Calculate the **10** most similar entries to the **first** entry in your dataset, using the functions defined above.

In [19]:

```
#YOUR CODE HERE
query = reg_dataset[10]['product_id']
mostSimilar(query, 10)
```

Out[19]:

```
[(1.0, 'B0149HT55K'),
 (1.0, 'B014592BBC'),
 (1.0, 'B0143EZYCW'),
 (1.0, 'B0143EZUO4'),
 (1.0, 'B013XJ2M8M'),
 (1.0, 'B013TVXHYC'),
 (1.0, 'B013TOCFP6'),
 (1.0, 'B013PZFS58'),
 (1.0, 'B013PZFMR2'),
 (1.0, 'B013PZF862')]
```

Finished!

Congratulations! You are now ready to submit your work. Once you have submitted make sure to get started on your peer reviews!