Capstone_F74112035

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.

```
import gzip
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
import pandas as pd
```

Task 1: Data Processing

TODO 1: Read the data and Fill your dataset

由於課程附件內容有包括 Submission 跟 example 內容，所以在後面 Task 中會盡量兩個檔案的內容都包括到 example file expression: Take care of int casting the votes and rating. Also **add this bit of code** to your for loop, taking off the outer " ":

" d['verified_purchase'] = d['verified_purchase'] == 'Y' "

This simple makes the verified purchase column be strictly true/false values rather than Y/N strings.

```
# 匯入資料集
file_path = 'amazon_reviews_us_Musical_Instruments_v1_00.tsv'

# 使用sep='\t' 明確指定分隔符
data = pd.read_csv(
    file_path,
    sep='\t',  # 指定制表符為分隔符
    on_bad_lines='skip',  # 跳過解析錯誤的部分
    low_memory=False  # 增加內存使用來處理大文件(not nessary)
)

print(data.head())
print(data.info())
```

```
  marketplace  customer_id       review_id  product_id  product_parent  \
0          US     45610553  RMDCHWD0Y5OZ9  B00HH62VB6       618218723   

1          US     14640079  RZSL0BALIYUNU  B003LRN53I       986692292   

2          US      6111003  RIZR67JKUDBI0  B0006VMBHI       603261968   

3          US      1546619 R27HL570VNL85F  B002B55TRG       575084461   

4          US     12222213 R34EBU9QDWJ1GD  B00N1YPXW2       165236328   


                                       product_title  
product_category  \
0  AGPtek® 10 Isolated Output 9V 12V 18V Guitar P...  Musical
Instruments
1         Sennheiser HD203 Closed-Back DJ Headphones  Musical
Instruments
2                   AudioQuest LP record clean brush  Musical
Instruments
3      Hohner Inc. 560BX-BF Special Twenty Harmonica  Musical
Instruments
4         Blue Yeti USB Microphone - Blackout Edition  Musical
Instruments

   star_rating  helpful_votes  total_votes vine verified_purchase  \
0            3              0            1    N                 N
1            5              0            0    N                 Y
2            3              0            1    N                 Y
3            5              0            0    N                 Y
4            5              0            0    N                 Y

                                     review_headline  \
```

```
0                                        Three Stars
1                                         Five Stars
2                                        Three Stars
3  I purchase these for a friend in return for pl...
4                                         Five Stars

                                        review_body review_date
0        Works very good, but induces ALOT of noise.  2015-08-31
1             Nice headphones at a reasonable price.  2015-08-31
2                       removes dust. does not clean  2015-08-31
3  I purchase these for a friend in return for pl...  2015-08-31
4                            This is an awesome mic!  2015-08-31
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 904004 entries, 0 to 904003
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   marketplace        904004 non-null  object
 1   customer_id        904004 non-null  int64
 2   review_id          904004 non-null  object
 3   product_id         904004 non-null  object
 4   product_parent     904004 non-null  int64
 5   product_title      904003 non-null  object
 6   product_category   904004 non-null  object
 7   star_rating        904004 non-null  int64
 8   helpful_votes      904004 non-null  int64
 9   total_votes        904004 non-null  int64
 10  vine               904004 non-null  object
 11  verified_purchase  904004 non-null  object
 12  review_headline    903998 non-null  object
 13  review_body        903941 non-null  object
 14  review_date        903996 non-null  object
dtypes: int64(5), object(10)
memory usage: 103.5+ MB
None
```

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%.

```
#YOUR CODE HERE
# Shuffle the data
shuffled_data = data.sample(frac=1,
random_state=42).reset_index(drop=True)

# Split the data (80% training, 20% testing)
split_index = int(len(shuffled_data) * 0.8)
train_data = shuffled_data[:split_index]
```

```
test_data = shuffled_data[split_index:]

print(len(train_data), len(test_data))
```

```
723203 180801
```

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.

```
#YOUR CODE HERE
# Delete the original dataset to avoid referencing it
del data
```

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?

question from example file:

1. What is the **average rating**?
2. What fraction of reviews are from **verified purchases**?
3. How many **total users** are there?
4. How many **total items** are there?
5. What fraction of reviews have **5-star ratings**?

```
# 確認訓練集大小
total_entries = len(train_data)
print("Total entries in the training set:", total_entries)

# 計算已驗證購買的比例
verified_percentage = (train_data['verified_purchase'] == 'Y').mean()
* 100
print(f"Percentage of verified purchases: {verified_percentage:.2f}%")

# 計算高評分且已驗證購買的比例
high_rating_verified = ((train_data['verified_purchase'] == 'Y') &
(train_data['star_rating'] >= 4)).mean() * 100
print(f"Percentage of high ratings (>=4) and verified purchases:
{high_rating_verified:.2f}%")
```

```python
# 計算平均評分
average_rating = train_data['star_rating'].mean()
print(f"Average rating: {average_rating:.2f}")

# 計算已驗證購買評論的比例
verified_fraction = (train_data['verified_purchase'] == 'Y').mean()
print(f"Fraction of reviews from verified purchases:
{verified_fraction:.2%}")

# 計算唯一用戶數
total_users = train_data['customer_id'].nunique()
print(f"Total users: {total_users}")

# 計算唯一商品數
total_items = train_data['product_id'].nunique()
print(f"Total items: {total_items}")

# 計算5 星級評分評論的比例
five_star_fraction = (train_data['star_rating'] == 5).mean()
print(f"Fraction of reviews with 5-star ratings:
{five_star_fraction:.2%}")
```

```
Total entries in the training set: 723203
Percentage of verified purchases: 86.37%
Percentage of high ratings (>=4) and verified purchases: 70.53%
Average rating: 4.25
Fraction of reviews from verified purchases: 86.37%
Total users: 481940
Total items: 111046
Fraction of reviews with 5-star ratings: 63.34%
```

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!

```python
# Define the feature function
def feature(d):
    """
    Extract features from a single row of data.
    Attributes used:
```

```
    - star_rating (numerical)
    - review_body (length of the text)
    """
    feat = [1, d['star_rating'], len(d['review_body'])]
    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.

```
#YOUR CODE HERE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# 定義特徵提取函數
def feature(d):
    """
    Extract features from a single row of data.
    Attributes used:
    - star_rating (numerical)
    - review_body (length of the text)
    """
    review_body_length = len(d['review_body']) if
pd.notna(d['review_body']) else 0
    feat = [1, d['star_rating'], review_body_length]
    return feat

# 提取特徵和標籤向量(訓練集)
X_train = train_data.apply(lambda d: feature(d), axis=1).tolist()
y_train = (train_data['verified_purchase'] ==
'Y').astype(int).tolist()

# 訓練邏輯回歸模型
model = LogisticRegression()
model.fit(X_train, y_train)
print("Model trained successfully.")
#print(model.head())

Model trained successfully.
```

TODO 3: Compute Accuracy of Your Model

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

```
#YOUR CODE HERE
from sklearn.metrics import accuracy_score
```

```python
# 提取特徵和標籤向量(測試集)
X_test = test_data.apply(lambda d: feature(d), axis=1).tolist()
y_test = (test_data['verified_purchase'] == 'Y').astype(int).tolist()

# 預測
y_pred = model.predict(X_test)

# 計算準確率
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the Logistic Regression Model:", accuracy)
```

```
Accuracy of the Logistic Regression Model: 0.864840349334351
```

question from example file: Finding the Balanced Error Rate

1. Compute **True** and **False Positives**
2. Compute **True** and **False Negatives**
3. Compute **Balanced Error Rate** based on your above defined variables.

```python
# 計算混淆矩陣
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

# True Positives, False Positives, True Negatives, False Negatives
print(f"True Positives (TP): {tp}")
print(f"False Positives (FP): {fp}")
print(f"True Negatives (TN): {tn}")
print(f"False Negatives (FN): {fn}")

# 計算Balanced Error Rate (BER)
false_positive_rate = fp / (fp + tn) if (fp + tn) > 0 else 0
false_negative_rate = fn / (fn + tp) if (fn + tp) > 0 else 0
balanced_error_rate = (false_positive_rate + false_negative_rate) / 2
print(f"Balanced Error Rate (BER): {balanced_error_rate:.4f}")
```

```
True Positives (TP): 155202
False Positives (FP): 23345
True Negatives (TN): 1162
False Negatives (FN): 1092
Balanced Error Rate (BER): 0.4798
```

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.

```python
import os
import gzip
```

```python
# 使用普通文件讀取方式
path = "amazon_reviews_us_Musical_Instruments_v1_00.tsv"

if not os.path.exists(path):
    raise FileNotFoundError(f"The file at {path} was not found.")

# 打開文件並讀取內容
with open(path, 'r', encoding="utf8") as f:
    header = f.readline().strip().split('\t')  # 讀取表頭
    reg_dataset = []
    for line in f:
        fields = line.strip().split('\t')
        d = dict(zip(header, fields))
        d['star_rating'] = int(d['star_rating'])  # 確保評分為整數
        reg_dataset.append(d)

#print("Data loaded successfully. Sample:", reg_dataset[:2])
```

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**.

```python
#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]

#YOUR CODE HERE
# TODO 1: Unique Words in a Sample Set
from collections import defaultdict
import string
from nltk.stem import PorterStemmer

# Count unique words (no stemming)
```

```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for datum in sampleSet:
    text = datum['review_body']
    text = ''.join([c for c in text if c not in punctuation]).lower()
    for word in text.split():
        wordCount[word] += 1
unique_words_count = len(wordCount)
print("Number of unique words (no stemming):", unique_words_count)

# Count unique stemmed words
wordCountStem = defaultdict(int)
stemmer = PorterStemmer()
for datum in sampleSet:
    text = datum['review_body']
    text = ''.join([c for c in text if c not in punctuation]).lower()
    for word in text.split():
        stemmed_word = stemmer.stem(word)
        wordCountStem[stemmed_word] += 1
unique_words_count_stemmed = len(wordCountStem)
print("Number of unique words (with stemming):",
unique_words_count_stemmed)

Number of unique words (no stemming): 101381
Number of unique words (with stemming): 83875
```

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.

```
#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)

#YOUR CODE HERE
# TODO 2: Evaluating Classifiers
from sklearn.linear_model import Ridge

# Define feature vector
X_reg = [feature_reg(d) for d in sampleSet]

# Fit Ridge regression model
model = Ridge(alpha=1.0, fit_intercept=True)
model.fit(X_reg, y_reg)

# Make predictions
predictions = model.predict(X_reg)

# Compute MSE
mse = MSE(predictions, y_reg)
print("Mean Squared Error (MSE):", mse)

Mean Squared Error (MSE): 1.2041184392177153
```

Task 4: Recommendation Systems

For your final task, you will use your knowledge of simple latent factor-based recommender systems to make predictions. Then evaluating the performance of your predictions.

Starting up

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

TODO 1: Calculate the ratingMean

1. Find the **average rating** of your training set.
2. Calculate a **baseline MSE value** from the actual ratings to the average ratings.

```
#YOUR CODE HERE
# 計算平均評分
ratingMean = train_data['star_rating'].mean()
print(f"Average rating (ratingMean): {ratingMean:.2f}")
```

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

labels = train_data['star_rating'].tolist()
baseline_predictions = [ratingMean] * len(labels)
baseline_mse = MSE(baseline_predictions, labels)
print(f"Baseline MSE: {baseline_mse:.4f}")
```

```
Average rating (ratingMean): 4.25
Baseline MSE: 1.4769
```

Here we are defining the functions you will need to optimize your MSE value.

```
#GIVEN
alpha = ratingMean

def prediction(user, item):
    return alpha + userBiases[user] + itemBiases[item]

def unpack(theta):
    global alpha
    global userBiases
    global itemBiases
    alpha = theta[0]
    userBiases = dict(zip(users, theta[1:nUsers+1]))
    itemBiases = dict(zip(items, theta[1+nUsers:]))

def cost(theta, labels, lamb):
    unpack(theta)
    predictions = [prediction(d['customer_id'], d['product_id']) for d
in trainingSet]
    cost = MSE(predictions, labels)
    print("MSE = " + str(cost))
    for u in userBiases:
        cost += lamb*userBiases[u]**2
    for i in itemBiases:
        cost += lamb*itemBiases[i]**2
    return cost

def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(trainingSet)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dItemBiases = defaultdict(float)
    for d in trainingSet:
        u,i = d['customer_id'], d['product_id']
```

```
        pred = prediction(u, i)
        diff = pred - d['star_rating']
        dalpha += 2/N*diff
        dUserBiases[u] += 2/N*diff
        dItemBiases[i] += 2/N*diff
    for u in userBiases:
        dUserBiases[u] += 2*lamb*userBiases[u]
    for i in itemBiases:
        dItemBiases[i] += 2*lamb*itemBiases[i]
    dtheta = [dalpha] + [dUserBiases[u] for u in users] +
[dItemBiases[i] for i in items]
    return numpy.array(dtheta)
```

TODO 2: Optimize

1. **Optimize** your MSE using the scipy.optimize.fmin_1_bfgs_b("arguments") functions.

```
#YOUR CODE HERE
users = train_data['customer_id'].unique()
items = train_data['product_id'].unique()
nUsers = len(users)
nItems = len(items)

# 初始化偏差
userBiases = {u: 0 for u in users}
itemBiases = {i: 0 for i in items}
theta = [ratingMean] + [0] * (nUsers + nItems)

from scipy.optimize import fmin_l_bfgs_b
from collections import defaultdict

# 訓練集格式轉換
trainingSet = train_data.to_dict(orient='records')

# 設定正則化參數
lamb = 0.1

# 優化
optimized_theta, final_cost, info = fmin_l_bfgs_b(
    func=cost,
    x0=theta,
    fprime=derivative,
    args=(labels, lamb),
    maxiter=100
)

print(f"Optimized MSE: {final_cost:.4f}")

MSE = 1.476904941435598
MSE = 1.4658348300504576
MSE = 1.476190268173242
```

```
MSE = 1.4761932521290224
MSE = 1.476190086408308
MSE = 1.4761900730469077
MSE = 1.4761899647366807
Optimized MSE: 1.4765
```

Finished!

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

```
```