Recommender System\_Code

EE 660 Project Type: Individual

Author: Ruchin Patel,

Email: [ruchinpa@usc.edu](mailto:ruchinpa@usc.edu)

USC ID: 9520665364

Phone number: 4322662949

Date: 3rd Dec 2018

CODE

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Tue Dec 4 19:11:28 2018

@author: ruchinpatel

"""

######importing libraries for data manipulation#######

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import gzip

import math

from PIL import Image

import requests

from io import BytesIO

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.model\_selection import ShuffleSplit, train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import fbeta\_score,precision\_score,f1\_score,recall\_score,accuracy\_score

from sklearn.linear\_model import Perceptron

from sklearn.metrics import roc\_curve, auc

from sklearn.model\_selection import StratifiedKFold

from scipy import interp

from sklearn.preprocessing import label\_binarize

from sklearn.multiclass import OneVsRestClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.svm import SVC

from sklearn.linear\_model import Ridge

from sklearn.linear\_model import LogisticRegression

from time import time

import matplotlib.patches as mpatches

import warnings

warnings.filterwarnings('ignore')

import scipy.stats as st

from sklearn.metrics import make\_scorer

from sklearn.model\_selection import GridSearchCV

import sklearn.learning\_curve as curves

import turicreate

from sklearn.tree import DecisionTreeClassifier

###############Function definitions###########

#########Function definitions for separating related feature########

def change\_vals\_new\_col(s,value,new\_cols):

if(value.get(s) != None):

if((type(value[s]) == float) and np.isnan(value[s])):

new\_cols[s] = np.nan

else:

new\_cols.get(s).append(value[s])

else:

new\_cols.get(s).append(np.nan)

def generate\_new\_cols(related):

new\_cols = {'also\_bought':[], 'also\_viewed':[],'bought\_together':[],'buy\_after\_viewing':[]}

for key,value in related.items():

if((type(value) == float) and np.isnan(value)):

new\_cols['also\_bought'].append(np.nan)

new\_cols['also\_viewed'].append(np.nan)

new\_cols['bought\_together'].append(np.nan)

new\_cols['buy\_after\_viewing'].append(np.nan)

else:

change\_vals\_new\_col('also\_bought',value,new\_cols)

change\_vals\_new\_col('also\_viewed',value,new\_cols)

change\_vals\_new\_col('bought\_together',value,new\_cols)

change\_vals\_new\_col('buy\_after\_viewing',value,new\_cols)

return new\_cols

#####Function definitions for separating related feature ends####

def plot\_related\_prods(related,which,final\_metadata):

if(related == None):

print('People who'+str(related)+'this product did not buy any other product:')

return

else:

#print(np.array(related) in final\_metadata.index)

tot = 0

for idx in related:

if(idx in final\_metadata.index):

tot += 1

print(tot)

tot = round(tot/2)

print('final',tot)

f, axes = plt.subplots(tot,tot,figsize=(4,4),dpi=300)

f.suptitle('People also '+str(which))

for i in range(0,tot):

for j in range(0,tot):

curr\_asin = related[i+j]

if((curr\_asin in final\_metadata.index) == True):

curr\_url = final\_metadata.loc[curr\_asin]['imUrl']

curr\_title = final\_metadata.loc[curr\_asin]['title']

curr\_title = curr\_title[0:30]

response = requests.get(curr\_url)

img = Image.open(BytesIO(response.content))

axes[i,j].imshow(img)

axes[i,j].get\_xaxis().set\_ticks([])

axes[i,j].get\_yaxis().set\_ticks([])

plt.axis('off')

axes[i,j].set\_title(curr\_title,size=3)

plt.show()

def Show\_related\_products(meta\_data\_row,final\_metadata):

#print(meta\_data\_row)

curr\_url = meta\_data\_row['imUrl']

#curr\_prod\_id = meta\_data\_row['asin']

title = meta\_data\_row['title']

print('The current product is:',title)

response = requests.get(curr\_url)

img = Image.open(BytesIO(response.content))

plt.savefig('The current product is:'+title, dpi=300)

plt.imshow(img)

plt.show()

####People who bought this product also bought####

also\_bought = meta\_data\_row['also\_bought']

if((type(also\_bought) == float) and np.isnan(also\_bought)):

also\_bought = None

else:

if(len(also\_bought) > 9):

also\_bought = also\_bought[0:9]

plot\_related\_prods(also\_bought,'bought',final\_metadata)

####People who bought this product also viewed####

also\_viewed = meta\_data\_row['also\_viewed']

if((type(also\_viewed) == float) and np.isnan(also\_viewed)):

also\_viewed = None

else:

if(len(also\_viewed) > 9):

also\_viewed = also\_viewed[0:9]

plot\_related\_prods(also\_viewed,'viewed',final\_metadata)

def bootStrap(learner,data,Y,size):

train\_acc = []

test\_acc = []

train\_f1 = []

test\_f1 = []

data\_merged = pd.concat([data,Y],axis=1)

for i in range(0,size):

#print(i)

data\_sampled = data\_merged.sample(5000)

X = data\_sampled.iloc[:,0:(data\_sampled.shape[1]-1)]

Y = data\_sampled.iloc[:,(data\_sampled.shape[1]-1):data\_sampled.shape[1]]

#print(X.shape)

#print(Y.shape)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=42,stratify = Y)

#print(X\_train.shape)

#print(y\_train.shape)

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

predictions\_test = learner.predict(X\_test)

predictions\_train = learner.predict(X\_train)

train\_acc.append(accuracy\_score(y\_train,predictions\_train))

test\_acc.append(accuracy\_score(y\_test,predictions\_test))

train\_f1.append(f1\_score(y\_train,predictions\_train))

test\_f1.append(f1\_score(y\_test,predictions\_test))

return train\_acc,test\_acc,train\_f1,test\_f1

def Hypothesis\_test(sampling\_dist,s1):

train\_acc\_samp = sampling\_dist[0]

test\_acc\_samp = sampling\_dist[1]

train\_f1\_samp = sampling\_dist[2]

test\_f1\_samp = sampling\_dist[3]

train\_acc\_mean = np.mean(train\_acc\_samp)

train\_acc\_std = np.std(train\_acc\_samp)

train\_acc\_SE = train\_acc\_std/np.sqrt(100)

print('Train accuracy mean of ',s1,' is',train\_acc\_mean)

print('Train accuracy standard deviation of ',s1,' is',train\_acc\_std)

print('Train accuracy Standard error of ',s1,' is',train\_acc\_SE)

print('-----------------------------------------------------------')

print('-----------------------------------------------------------')

test\_acc\_mean = np.mean(test\_acc\_samp)

test\_acc\_std = np.std(test\_acc\_samp)

test\_acc\_SE = test\_acc\_std/np.sqrt(100)

print('Test accuracy mean of ',s1,' is: ',test\_acc\_mean)

print('Test accuracy standard deviation of ',s1,' is',test\_acc\_std)

print('Test accuracy Standard error of ',s1,' is',test\_acc\_SE)

print('-----------------------------------------------------------')

print('-----------------------------------------------------------')

train\_f1\_mean = np.mean(train\_f1\_samp)

train\_f1\_std = np.std(train\_f1\_samp)

train\_f1\_SE = train\_f1\_std/np.sqrt(100)

print('Train f1 mean of ',s1,' is',train\_f1\_mean)

print('Train f1 standard deviation of ',s1,' is',train\_f1\_std)

print('Train f1 Standard error of ',s1,' is',train\_f1\_SE)

print('-----------------------------------------------------------')

print('-----------------------------------------------------------')

test\_f1\_mean = np.mean(test\_f1\_samp)

test\_f1\_std = np.std(test\_f1\_samp)

test\_f1\_SE = test\_f1\_std/np.sqrt(100)

print('Test f1 mean of ',s1,' is',test\_f1\_mean)

print('Test f1 standard deviation of ',s1,' is',test\_f1\_std)

print('Test f1 Standard error of ',s1,' is',test\_f1\_SE)

print('-----------------------------------------------------------')

print('-----------------------------------------------------------')

dist = np.random.normal(loc=test\_acc\_mean,scale=test\_acc\_SE,size = 10000)

density\_prop = {"color": "green"}

hist\_prop = {"alpha": 0.3, "color": "red"}

s = '95 % confidence interval of test accuracy of '+s1

plot\_densityCurve(dist,density\_prop,hist\_prop,100,5000,test\_acc\_mean,test\_acc\_SE,accuracy\_naive,s)

dist = np.random.normal(loc=test\_f1\_mean,scale=test\_f1\_SE,size = 10000)

density\_prop = {"color": "green"}

hist\_prop = {"alpha": 0.3, "color": "red"}

s = '95 % confidence interval of test f\_beta score '+s1

plot\_densityCurve(dist,density\_prop,hist\_prop,100,5000,test\_f1\_mean,test\_f1\_SE,fscore\_naive,s)

def train\_predict(learner, sample\_size, X\_train, y\_train, X\_test, y\_test):

'''

inputs:

- learner: the learning algorithm to be trained and predicted on

- sample\_size: the size of samples (number) to be drawn from training set

- X\_train: features training set

- y\_train: income training set

- X\_test: features testing set

- y\_test: income testing set

'''

X\_train = X\_train.astype(int)

y\_train = y\_train.astype(int)

X\_test = X\_test.astype(int)

y\_test = y\_test.astype(int)

results = {}

start = time() # Get start time

learner = learner.fit(X\_train[0:sample\_size],y\_train[0:sample\_size])

end = time() # Get end time

results['train\_time'] = end-start

predictions\_test = learner.predict(X\_test)

predictions\_train = learner.predict(X\_train[0:sample\_size])

end = time() # Get end time

results['pred\_time'] = end - start

results['acc\_train'] = accuracy\_score(y\_train[0:sample\_size],predictions\_train)

results['acc\_test'] = accuracy\_score(y\_test,predictions\_test)

results['f\_train'] = fbeta\_score(y\_train[0:sample\_size],predictions\_train,0.5)

results['f\_test'] = fbeta\_score(y\_test,predictions\_test,0.5)

print("{} trained on {} samples.".format(learner.\_\_class\_\_.\_\_name\_\_, sample\_size))

print("Train accuracy is:", results['acc\_train'])

print("Test accuracy is:", results['acc\_test'])

print("Train F-beta(0.5) score is:", results['f\_train'])

print("Test F-beta(0.5) is:", results['f\_test'])

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

# Return the results

return results

#######Reciever Operating Characteristics definitions###########

def ROC\_AUC(classifier,X,y,which=None,c=None):

X = np.array(X)

y = np.array(y)

y = np.reshape(y,(y.shape[0],1))

title = 'ROC for Binary labels'

if(which == 'multi'):

y = label\_binarize(y, classes=[1,2,3,4,5])

title = 'ROC for multi-class labels'

#f, axes = plt.subplots(3,2,figsize=(8,8),dpi=300)

#f.suptitle('Distribution of drawing cards simulations')

#m = [(0,0),(0,1),(1,0),(1,1),(2,0),(2,1)]

else:

title = 'ROC for Binary labels'

#f, axes = plt.subplots(1,2,figsize=(8,8),dpi=300)

#f.suptitle('Distribution of drawing cards simulations')

#m = [(0,0),(0,1)]

print(y.shape)

random\_state = np.random.RandomState(0)

cv = StratifiedKFold(n\_splits=4)

mean\_fpr = np.linspace(0, 1, 100)

plt.figure(figsize=(9,4), dpi=300)

for t in range(0,y.shape[1]):

i = 0

tprs = []

aucs = []

#a = m[t][0]

#b = m[t][1]

#print(a)

#print(b)

for train, test in cv.split(X, y[:,t]):

#print(y[train].shape)

if(c == 'p'):

probas\_ = classifier.fit(X[train], y[train,t]).score(X[test],y[test,t])

else:

probas\_ = classifier.fit(X[train], y[train,t]).predict\_proba(X[test])

# Compute ROC curve and area the curve

fpr, tpr, thresholds = roc\_curve(y[test,t], probas\_[:, 1])

tprs.append(interp(mean\_fpr, fpr, tpr))

tprs[-1][0] = 0.0

roc\_auc = auc(fpr, tpr)

aucs.append(roc\_auc)

plt.plot(fpr, tpr, lw=1, alpha=0.3,label='ROC fold %d (AUC = %0.2f)' % (i, roc\_auc))

i = i + 1

plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',label='Chance', alpha=.8)

mean\_tpr = np.mean(tprs, axis=0)

mean\_tpr[-1] = 1.0

mean\_auc = auc(mean\_fpr, mean\_tpr)

std\_auc = np.std(aucs)

plt.plot(mean\_fpr, mean\_tpr, color='b',label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean\_auc, std\_auc),lw=2, alpha=.8)

std\_tpr = np.std(tprs, axis=0)

tprs\_upper = np.minimum(mean\_tpr + std\_tpr, 1)

tprs\_lower = np.maximum(mean\_tpr - std\_tpr, 0)

plt.fill\_between(mean\_fpr, tprs\_lower, tprs\_upper, color='grey', alpha=.2,label=r'$\pm$ 1 std. dev.')

plt.xlim([-0.05, 1.05])

plt.ylim([-0.05, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

if(which == 'multi'):

plt.title(title+' for rating = '+str(t+1))

else:

plt.title(title)

plt.legend(loc="lower right")

plt.savefig(title+' with 6000 samples',dpi=300)

plt.show()

def evaluate(results, accuracy, f1):

"""

Visualization code to display results of various learners.

inputs:

- learners: a list of supervised learners

- stats: a list of dictionaries of the statistic results from 'train\_predict()'

- accuracy: The score for the naive predictor

- f1: The score for the naive predictor

"""

# Create figure

fig, ax = plt.subplots(2, 3, figsize = (30,20))

# Constants

bar\_width = 0.15

colors = ['#A00000','#00A0A0','#00A000','orange','purple']

# Super loop to plot four panels of data

for k, learner in enumerate(results.keys()):

for j, metric in enumerate(['train\_time', 'acc\_train', 'f\_train', 'pred\_time', 'acc\_test', 'f\_test']):

for i in np.arange(3):

# Creative plot code

ax[j//3, j%3].bar(i+k\*bar\_width, results[learner][i][metric], width = bar\_width, color = colors[k])

ax[j//3, j%3].set\_xticks([0.45, 1.45, 2.45])

ax[j//3, j%3].set\_xticklabels(["1%", "10%", "100%"])

ax[j//3, j%3].set\_xlabel("Training Set Size",fontsize = 26)

ax[j//3, j%3].set\_xlim((-0.1, 3.0))

# Add unique y-labels

ax[0, 0].set\_ylabel("Time (in seconds)",fontsize = 26)

ax[0, 1].set\_ylabel("Accuracy Score",fontsize = 26)

ax[0, 2].set\_ylabel("F-beta(0.5)",fontsize = 26)

ax[1, 0].set\_ylabel("Time (in seconds)",fontsize = 26)

ax[1, 1].set\_ylabel("Accuracy Score",fontsize = 26)

ax[1, 2].set\_ylabel("F-beta(0.5)",fontsize = 26)

# Add titles

ax[0, 0].set\_title("Model Training",fontsize = 26)

ax[0, 1].set\_title("Accuracy Score on Training Subset",fontsize = 26)

ax[0, 2].set\_title("F-beta(0.5) on Training Subset",fontsize = 26)

ax[1, 0].set\_title("Model Predicting",fontsize = 26)

ax[1, 1].set\_title("Accuracy Score on Testing Set",fontsize = 26)

ax[1, 2].set\_title("F-beta(0.5) on Testing Set",fontsize = 26)

# Add horizontal lines for naive predictors

ax[0, 1].axhline(y = accuracy, xmin = -0.1, xmax = 3.0, linewidth = 1, color = 'k', linestyle = 'dashed')

ax[1, 1].axhline(y = accuracy, xmin = -0.1, xmax = 3.0, linewidth = 1, color = 'k', linestyle = 'dashed')

ax[0, 2].axhline(y = f1, xmin = -0.1, xmax = 3.0, linewidth = 1, color = 'k', linestyle = 'dashed')

ax[1, 2].axhline(y = f1, xmin = -0.1, xmax = 3.0, linewidth = 1, color = 'k', linestyle = 'dashed')

# Set y-limits for score panels

ax[0, 1].set\_ylim((0, 1))

ax[0, 2].set\_ylim((0, 1))

ax[1, 1].set\_ylim((0, 1))

ax[1, 2].set\_ylim((0, 1))

# Create patches for the legend

patches = []

for i, learner in enumerate(results.keys()):

patches.append(mpatches.Patch(color = colors[i], label = learner))

plt.legend(handles = patches,bbox\_to\_anchor=(0.5, 1,0.5,0.5), loc='upper center',ncol = 3, fontsize = 26)

# Aesthetics

#plt.figlegend( 'Ruchin', 'Patel', loc = 'lower center', ncol=5, labelspacing=0. )

plt.suptitle("Performance Metrics for 5 Supervised Learning Models", fontsize = 26, y = 1.10)

plt.tight\_layout()

plt.savefig("Performance Metrics for 5 Supervised Learning Models",dpi=300)

plt.show()

def plot\_densityCurve(\*args):

plt.figure(figsize=(9,4), dpi=300)

sns.distplot(args[0],kde\_kws=args[1])

plt.axvline(args[5], color='yellow', linestyle='-.', linewidth=1,label='sample mean')

plt.axvline(args[5]-args[6], color='black', linestyle=':', linewidth=1,label='1 standard dev')

plt.axvline(args[5]+args[6], color='black', linestyle=':', linewidth=1)

plt.axvline(args[7], color='purple', linestyle='-.', linewidth=2,label='Naive Predictor')

plt.axvline(args[5]-(1.96\*args[6]),color='black',linewidth=2,label='95% confidence line')

plt.axvline(args[5]+(1.96\*args[6]),color='black',linewidth=2)

#plt.xlim(0.72,0.85)

plt.legend()

#plt.title("The sampling distribution with "+str(args[3])+" samples of size n="+str(args[4]))

plt.title(args[8])

plt.savefig(args[8],dpi=300)

plt.show()

print('Mean is:',args[5])

print('95 % confidence range for',args[8],' :(',args[5]-args[6],',',args[5]+args[6],')')

def plot\_norm(sample\_mean,SE,\*args):

plt.figure(figsize=(9,4), dpi=300)

x\_values = np.random.normal(loc = sample\_mean,scale=SE,size=args[0])

x\_values = np.sort(x\_values)

y\_values = st.norm.pdf(x\_values,loc = sample\_mean,scale=SE)

plt.plot(x\_values,y\_values,linewidth=2,color='green')

plt.axvline(sample\_mean, color='yellow', linestyle='-.', linewidth=2,label='sample mean')

plt.axvline(sample\_mean-SE, color='black', linestyle=':', linewidth=1,label='1 standard dev')

plt.axvline(sample\_mean+SE, color='black', linestyle=':', linewidth=1)

plt.axvline(args[1], color='purple', linestyle='-', linewidth=1,label='95% confidence line')

plt.axvline(args[2], color='purple', linestyle='-', linewidth=1)

x\_95 = x\_values[np.logical\_and(x\_values>=args[1],x\_values<=args[2])]

y\_95 = y\_values[np.logical\_and(x\_values>=args[1],x\_values<=args[2])]

plt.fill\_between(x\_95,0,y\_95,color='red',alpha=0.4)

#plt.ylim(0,10)

plt.legend()

plt.show()

def cross\_Val\_model\_selection(learner,X\_train,X\_test,y\_train,y\_test):

#Initialize the classifier

clf = learner

#Create the parameters list you wish to tune, using a dictionary if needed.

#parameters = {'parameter\_1': [value1, value2], 'parameter\_2': [value1, value2]}

n\_estimators = [int(x) for x in np.linspace(start = 100, stop = 500, num = 10)]

learning\_rate = list(np.arange(0.5,2,0.2))

parameters = {'n\_estimators': n\_estimators,'learning\_rate': learning\_rate}

#Make an fbeta\_score scoring object using make\_scorer()

scorer = make\_scorer(fbeta\_score,beta = 0.5)

#Perform grid search on the classifier using 'scorer' as the scoring method using GridSearchCV()

grid\_obj = GridSearchCV(clf,parameters,scoring=scorer)

#Fit the grid search object to the training data and find the optimal parameters using fit()

grid\_fit = grid\_obj.fit(X\_train,y\_train)

# Get the estimator

best\_clf = grid\_fit.best\_estimator\_

# Make predictions using the unoptimized and model

predictions = (clf.fit(X\_train, y\_train)).predict(X\_test)

best\_predictions = best\_clf.predict(X\_test)

# Report the before-and-afterscores

#print("Unoptimized model\n------")

#print("Accuracy score on testing data: {:.4f}".format(accuracy\_score(y\_test, predictions)))

#print("F-score on testing data: {:.4f}".format(fbeta\_score(y\_test, predictions, beta = 0.5)))

print("\nOptimized Model\n------")

print("Final accuracy score on the testing data: {:.4f}".format(accuracy\_score(y\_test, best\_predictions)))

print("Final F-score on the testing data: {:.4f}".format(fbeta\_score(y\_test, best\_predictions, beta = 0.5)))

return best\_clf

def finalModel(rev\_txt,X,Y,best\_estimator):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,Y,test\_size=0.2, random\_state=42,stratify = Y)

clf = best\_estimator

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

pred\_train = pd.Series(clf.predict(X\_train),index = y\_train.index)

print('Train accuracy score of final model is: ',accuracy\_score(y\_train, pred\_train, normalize=True))

print('Train F\_beta(0.5) of final model is: ',fbeta\_score(y\_train, pred\_train,0.5))

pred\_test = pd.Series(clf.predict(X\_test),index=y\_test.index)

print('Test accuracy score of final model is: ',accuracy\_score(y\_test, pred\_test, normalize=True))

print('Test F\_beta(0.5) of final model is: ',fbeta\_score(y\_test, pred\_test,0.5))

print()

print()

#print(type(pred\_test))

#print(pred\_test.head(10))

#print(type(y\_test))

#print(y\_test.head(10))

set\_test\_0 = list(y\_test[y\_test==0].index)

set\_test\_0 = set(set\_test\_0)

set\_test\_1 = list(y\_test[y\_test==1].index)

set\_test\_1 = set(set\_test\_1)

set\_pred\_0 = list(pred\_test[pred\_test==0].index)

set\_pred\_0 = set(set\_pred\_0)

set\_pred\_1 = list(pred\_test[pred\_test==1].index)

set\_pred\_1 = set(set\_pred\_1)

set\_1 = set\_test\_1.intersection(set\_pred\_1)

set\_0 = set\_test\_0.intersection(set\_pred\_0)

print('Length of the testing set is: ',y\_test.shape)

print('Total number of ones in the testing set is: ',np.sum(y\_test))

print('Total number of zeros in the testing set is: ',(y\_test.shape[0] - np.sum(y\_test)))

print('Total values predicted 1 from testing set and equal to true values are:',len(set\_1))

print('Total values predicted 0 from testing set and equal to true values are:',len(set\_0))

print()

print()

print("SOME OF THE TRUE CLASSIFIED REVIEWS FROM THE TEST SET ARE AS FOLLOWS: ")

print()

print()

for i in range(0,3):

print('Positive Review: ',rev\_txt[set\_1.pop()])

print()

print('Negative Review: ',rev\_txt[set\_0.pop()])

print()

print()

def check\_polarity(model,review,vocab\_vect):

token = vocab\_vect.transform(review)

polarity = model.predict(token)

if(polarity == 1):

print('Review: ',review[0])

print()

print('The above review is a positive review and hence recommend products similar to the related prodict')

if(polarity == 0):

print('Review: ',review[0])

print()

print('The above review is a negative review and hence recommend products that are the best from its given category')

print('-----------------------------------------------------------------------------------------------------------------')

print('-----------------------------------------------------------------------------------------------------------------')

print()

def ModelComplexity(X, y):

""" Calculates the performance of the model as model complexity increases.

The learning and testing errors rates are then plotted. """

# Create 10 cross-validation sets for training and testing

#cv = StratifiedKFold(n\_splits=4, shuffle=True, random\_state=42)

# Vary the n\_estimatorsmax\_depth parameter from 1 to 10

n\_estimators = np.arange(50,1000,50)

# Calculate the training and testing scores

train\_scores, test\_scores = curves.validation\_curve(AdaBoostClassifier(learning\_rate=1.5), X, y, \

param\_name = "n\_estimators", param\_range = n\_estimators, scoring = 'f1')

# Find the mean and standard deviation for smoothing

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

# Plot the validation curve

plt.figure(figsize=(9,4), dpi=300)

plt.title('AdaBoost Complexity Performance')

plt.plot(n\_estimators, train\_mean, 'o-', color = 'r', label = 'Training Score')

plt.plot(n\_estimators, test\_mean, 'o-', color = 'g', label = 'Validation Score')

plt.fill\_between(n\_estimators, train\_mean - train\_std, \

train\_mean + train\_std, alpha = 0.15, color = 'r')

plt.fill\_between(n\_estimators, test\_mean - test\_std, \

test\_mean + test\_std, alpha = 0.15, color = 'g')

# Visual aesthetics

plt.legend(loc = 'lower right')

plt.xlabel('Maximum Depth')

plt.ylabel('Score')

plt.ylim([-0.05,1.05])

plt.savefig('AdaBoost Complexity Performance',dpi=300)

plt.show()

def most\_popular\_products(data,rec\_data,recommend,s1 = None):

train\_data = turicreate.SFrame(rec\_data)

rev\_id = list(data.sample(1,random\_state=3)['reviewerID'])

if(s1 == None):

popularity\_model = turicreate.popularity\_recommender.create(train\_data, user\_id='reviewerID', item\_id='asin', target='overall\_binary')

else:

popularity\_model = turicreate.popularity\_recommender.create(train\_data, user\_id='reviewerID', item\_id='asin', target='predicted\_binary')

popularity\_recomm = popularity\_model.recommend(users=rev\_id,k=recommend)

popularity\_recomm.print\_rows(num\_rows=25)

if(s1 == None):

path\_list = ['Popularity\_real/']

else:

path\_list = ['Popularity\_pred/']

l = list(popularity\_recomm[popularity\_recomm['reviewerID'] == rev\_id[0]]['asin'])

for i in range(len(l)):

all\_ratings = data[data['asin'] == l[i]]['overall']

mean\_rating = np.mean(all\_ratings)

title = final\_metadata.loc[l[i],'title']+'| mean rating: '+str(np.round(mean\_rating,decimals=2))+'| rank:'+str(i+1)

curr\_url = final\_metadata.loc[l[i],'imUrl']

response = requests.get(curr\_url)

img = Image.open(BytesIO(response.content))

plt.title(title)

plt.imshow(img)

s = title.split('/')

final\_title = ''

for i in range(len(s)):

final\_title = final\_title + ' '+s[i]

heading = final\_title +'.png'

final\_path = path\_list[0]+heading

plt.savefig(final\_path,dpi=300)

plt.show()

def Collaborative\_filtering(data,rec\_data,recommend,s1=None):

train\_data = turicreate.SFrame(rec\_data)

rev\_id = list(data.sample(3,random\_state=1234)['reviewerID'])

if(s1 == None):

item\_sim\_model = turicreate.item\_similarity\_recommender.create(train\_data, user\_id='reviewerID', item\_id='asin', target='overall\_binary', similarity\_type='cosine')

else:

item\_sim\_model = turicreate.item\_similarity\_recommender.create(train\_data, user\_id='reviewerID', item\_id='asin', target='predicted\_binary', similarity\_type='cosine')

item\_sim\_recomm = item\_sim\_model.recommend(users=rev\_id,k=recommend)

item\_sim\_recomm.print\_rows(num\_rows=25)

if(s1 == None):

path\_list = ['user\_1\_photos/','user\_2\_photos/','user\_3\_photos/','user\_4\_photos/','user\_5\_photos/']

path\_list\_rec = ['user\_1\_rec/','user\_2\_rec/','user\_3\_rec/','user\_4\_rec/','user\_5\_rec/']

else:

path\_list = ['user\_11\_photos/','user\_22\_photos/','user\_33\_photos/','user\_44\_photos/','user\_55\_photos/']

path\_list\_rec = ['user\_11\_rec/','user\_22\_rec/','user\_33\_rec/','user\_44\_rec/','user\_55\_rec/']

for j in range(len(rev\_id)):

rev\_data = data[data['reviewerID'] == rev\_id[j]]

l2 = list(rev\_data['asin'])

for i in range(len(l2)):

title = final\_metadata.loc[l2[i],'title']+' rating: '+str(rev\_data.iloc[i,5])

curr\_url = final\_metadata.loc[l2[i],'imUrl']

response = requests.get(curr\_url)

img = Image.open(BytesIO(response.content))

plt.title(title)

plt.imshow(img)

s = title.split('/')

final\_title = ''

for i in range(len(s)):

final\_title = final\_title + ' '+s[i]

heading = final\_title+' purchased by '+rev\_id[j]+'.png'

final\_path = path\_list[j]+heading

print(final\_path)

plt.savefig(final\_path,dpi=300)

plt.show()

l = list(item\_sim\_recomm[item\_sim\_recomm['reviewerID'] == rev\_id[j]]['asin'])

for i in range(len(l)):

all\_ratings = data[data['asin'] == l[i]]['overall']

mean\_rating = np.mean(all\_ratings)

title = final\_metadata.loc[l[i],'title']+'| mean rating: '+str(np.round(mean\_rating,decimals=2))+'| rank:'+str(i+1)

curr\_url = final\_metadata.loc[l[i],'imUrl']

response = requests.get(curr\_url)

img = Image.open(BytesIO(response.content))

plt.title(title)

plt.imshow(img)

s = title.split('/')

final\_title = ''

for i in range(len(s)):

final\_title = final\_title + ' '+s[i]

heading = final\_title +' recommended to'+rev\_id[j]+'.png'

final\_path = path\_list\_rec[j]+heading

plt.savefig(final\_path,dpi=300)

plt.show()

def error\_bound(X\_train, X\_test, y\_train, y\_test):

f\_train = []

f\_test = []

for i in range(1,20):

base\_est = DecisionTreeClassifier(max\_depth=i)

clf = AdaBoostClassifier(algorithm='SAMME.R', base\_estimator=base\_est, \

learning\_rate=1.4999999999999998, n\_estimators=50, \

random\_state=42)

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

y\_pred\_train = clf.predict(X\_train)

y\_pred\_test = clf.predict(X\_test)

f\_train.append(fbeta\_score(y\_train,y\_pred\_train,0.5))

f\_test.append(fbeta\_score(y\_test,y\_pred\_test,0.5))

print(f\_train)

print(f\_test)

print()

y = list(range(100,2000,100))

plt.plot(f\_train,y)

plt.plot(f\_test,y)

plt.show()

#####Reading Data#######

#####This is a smaller data for initial data exploration and model testing#####

######The data is about Heal and Personal care Products on Amazon##########

def parse(path):

g = open(path, 'rb')

for l in g:

yield eval(l)

def getDF(path):

i = 0

df = {}

for d in parse(path):

df[i] = d

i += 1

return pd.DataFrame.from\_dict(df, orient='index')

data\_path = '/Users/ruchinpatel/Desktop/USC\_EVERYTHING/SPRINGBOARD/CAPSTONE/Health\_and\_Personal\_Care\_5.json'

metadata\_path = '/Users/ruchinpatel/Desktop/USC\_EVERYTHING/SPRINGBOARD/CAPSTONE/meta\_Health\_and\_Personal\_Care.json'

data = getDF(data\_path)

metadata = getDF(metadata\_path)

########## Generating seperate columns for related feature######

related = metadata['related'].to\_dict()

newly\_created\_columns = pd.DataFrame(generate\_new\_cols(related))

#########Final Metadata dataframe###########

final\_metadata = pd.concat([metadata,newly\_created\_columns],axis = 1)

final\_metadata = final\_metadata.drop('related',axis=1)

final\_metadata = final\_metadata.set\_index('asin')

######converting the dates to date time format#####

data['unixReviewTime'] = pd.to\_datetime(data['unixReviewTime'],unit='s')

data['reviewTime'] = pd.to\_datetime(data['reviewTime'])

reviews\_data = data[['reviewText','overall']]

reviews\_data\_binary = data[['reviewText','overall']]

replace = {1:0,2:0,3:0,4:1,5:1}

reviews\_data['overall\_binary'] = reviews\_data[['overall']].replace(to\_replace=replace,value=None)

#print(reviews\_data.shape)

#print(reviews\_data\_binary.shape)

#print(np.unique(reviews\_data\_binary['overall']))

#reviews\_data[['overall','overall\_binary']].head(10)

print('Total ratings of all classes')

print()

print()

ratings\_count = reviews\_data.groupby(by='overall').count()

ratings\_count['reviewTextPercent'] = ratings\_count['reviewText']\*100/np.sum(ratings\_count['reviewText'])

ratings\_count['classWeights'] = 20/ratings\_count['reviewTextPercent']

print(ratings\_count[['reviewText','reviewTextPercent','classWeights']])

#####Tokenizing our text data####

count\_vect = CountVectorizer(analyzer = 'word',stop\_words = 'english',min\_df=0.01,binary=False)

vocab\_vect = count\_vect.fit(reviews\_data['reviewText'])

review\_text\_tokenized = vocab\_vect.transform(reviews\_data['reviewText'])

print(review\_text\_tokenized.shape)

review\_text\_tokenized = pd.DataFrame(review\_text\_tokenized.toarray())

ratings\_multi = reviews\_data['overall']

ratings\_binary = reviews\_data['overall\_binary']

X\_red\_tr, X\_red\_tt, y\_red\_tr, y\_red\_tt = train\_test\_split(review\_text\_tokenized,ratings\_binary,train\_size=0.02, random\_state=42,stratify=ratings\_binary)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_red\_tr,y\_red\_tr,test\_size=0.2, random\_state=42,stratify=y\_red\_tr)

####### We will first construct a Naive predictor#####

##### Predicts everything as 5 stars as it is the maximum######

X\_train\_n, X\_test\_n, y\_train\_n, y\_test\_n = train\_test\_split(review\_text\_tokenized,ratings\_binary,test\_size=0.2, random\_state=42,stratify=ratings\_binary)

naive\_pred = np.ones(shape=y\_test\_n.shape)

accuracy\_naive = accuracy\_score(y\_test\_n,naive\_pred)

precision\_naive = precision\_score(y\_test\_n,naive\_pred)

recall\_naive = recall\_score(y\_test\_n,naive\_pred)

# TODO: Calculate F-score using the formula above for beta = 0.5 and correct values for precision and recall.

fscore\_naive = fbeta\_score(y\_test\_n,naive\_pred,0.5)

# Print the results

print("Naive Predictor: [Accuracy score: {:.4f}, Precesion: {:.4f}, Recall: {:.4f}, F-score: {:.4f}]".format(accuracy\_naive,precision\_naive,recall\_naive,fscore\_naive))

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import AdaBoostClassifier

clf\_A = MultinomialNB()

clf\_B = RandomForestClassifier(n\_estimators=200, max\_depth=2,random\_state=0)

clf\_C = AdaBoostClassifier(n\_estimators=200,random\_state=0)

clf\_D = SVC(kernel='rbf',C=2,gamma=20)

clf\_E = LogisticRegression()

# TODO: Calculate the number of samples for 1%, 10%, and 100% of the training data

# HINT: samples\_100 is the entire training set i.e. len(y\_ train)

# HINT: samples\_10 is 10% of samples\_100 (ensure to set the count of the values to be `int` and not `float`)

# HINT: samples\_1 is 1% of samples\_100 (ensure to set the count of the values to be `int` and not `float`)

samples\_100 = int(len(y\_train))

samples\_10 = int(0.1 \* len(y\_train))

samples\_1 = int(0.01 \* len(y\_train))

# Collect results on the learners

results = {}

for clf in [clf\_A, clf\_B, clf\_C,clf\_D,clf\_E]:

clf\_name = clf.\_\_class\_\_.\_\_name\_\_

results[clf\_name] = {}

for i, samples in enumerate([samples\_1, samples\_10, samples\_100]):

print

results[clf\_name][i] = \

train\_predict(clf, samples, X\_train, y\_train, X\_test, y\_test)

# Run metrics visualization for the three supervised learning models chosen

evaluate(results, accuracy\_naive, fscore\_naive)

sampling\_dist = bootStrap(LogisticRegression(),review\_text\_tokenized,ratings\_binary,100)

Hypothesis\_test(sampling\_dist,'Logistic Regression')

sampling\_dist = bootStrap(AdaBoostClassifier(n\_estimators=200,random\_state=0),review\_text\_tokenized,ratings\_binary,100)

Hypothesis\_test(sampling\_dist,'AdaBoost')

best\_estimator = cross\_Val\_model\_selection(AdaBoostClassifier(random\_state=42),X\_train,X\_test,y\_train,y\_test)

best\_estimator

X\_train, X\_test, y\_train, y\_test = train\_test\_split(review\_text\_tokenized,ratings\_binary,train\_size=0.02, random\_state=42,stratify=ratings\_binary)

print(X\_train.shape)

ModelComplexity(X\_train, y\_train)

ROC\_AUC(best\_estimator,review\_text\_tokenized.sample(6000,random\_state=42),ratings\_binary.sample(6000,random\_state=42))

finalModel(reviews\_data['reviewText'],review\_text\_tokenized,ratings\_binary,best\_estimator)

a = ["The screen of the magnifier is small. If you're looking to read text this is not going to work. Though I have not attempted to replace the battery, battery container seems to be contained by a very small screw-A Phillips screwdriver-of which would have to be incredibly small. I dread having to replace his battery."]

b = ["I am disappointed in this product. I should have worked better."]

c = ["There isn't a product which can be worse than this. I have no idea why I even bought this"]

d = ["I find this product comfortable. This product can never be worse"]

revs = [a,b,c,d]

for r in revs:

check\_polarity(best\_estimator,r,vocab\_vect)

replace = {1:0,2:0,3:0,4:1,5:1}

data['overall\_binary'] = data[['overall']].replace(to\_replace=replace,value=None)

data['predicted\_binary'] = best\_estimator.predict(review\_text\_tokenized)

rec\_data = data[['reviewerID','asin','overall','overall\_binary','predicted\_binary']]

Collaborative\_filtering(data,rec\_data,5)

Collaborative\_filtering(data,rec\_data,5,s1='pred')

most\_popular\_products(data,rec\_data,10)

most\_popular\_products(data,rec\_data,10,s1 = 'pred')

data[data['reviewerID'] == 'A3D0HMC6RQT0N0']