Recommender System_Code
EE 660 Project Type: Individual

Author: Ruchin Patel,
Email: 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_Person
al_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,ratin
gs_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_binar
y.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']
```