import string  
import numpy as np  
import matplotlib  
import matplotlib.pyplot as plt  
import sklearn.metrics  
matplotlib.rc('xtick', labelsize=14)  
matplotlib.rc('ytick', labelsize=14)  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.linear\_model import SGDClassifier  
  
import warnings  
warnings.filterwarnings("ignore")  
  
with open("/Users/Ramesh/Desktop/Py/full\_set.txt") as f:  
 content = f.readlines()  
  
 # Remove leading and trailing white space  
 content = [x.strip() for x in content]  
# print(content)  
  
 # Separate the sentences from the labels:  
 sentences = [x.split("\t")[0] for x in content]  
 labels = [x.split("\t")[1] for x in content]  
 print(sentences)  
print(labels)  
  
y = np.array(labels, dtype='int8')  
y = 2 \* y - 1  
print("Numpy Array in python :", y)  
  
  
def full\_remove(x, removal\_list):  
 for w in removal\_list:  
 x = x.replace(w,' ')  
 return x  
  
digits = [str(x) for x in range(10)] # Remove digits  
digit\_less = [full\_remove(x, digits) for x in sentences]  
punc\_less = [full\_remove(x, list(string.punctuation)) for x in  
digit\_less] # Remove punctuation  
sents\_lower = [x.lower() for x in punc\_less]  
  
stop\_set = set(['the', 'a', 'an', 'i', 'he', 'she', 'they', 'to',  
'of', 'it', 'from'])  
# Remove stop words  
sents\_split = [x.split() for x in sents\_lower]  
sents\_processed = [" ".join(list(filter(lambda a: a not in stop\_set,x))) for x in sents\_split]  
  
sents\_processed[0:20]  
  
  
  
  
# Transform to bag of words representation.  
vectorizer = CountVectorizer(analyzer = "word", tokenizer = None,preprocessor = None, stop\_words = None, max\_features = 4500)  
data\_features = vectorizer.fit\_transform(sents\_processed)  
# Append '1' to the end of each vector.  
data\_mat = data\_features.toarray()  
print(data\_mat)  
  
np.random.seed(0)  
test\_inds = np.append(np.random.choice((np.where(y==-1))[0], 250,replace=False), np.random.choice((np.where(y==1))[0], 250,replace=False))  
train\_inds = list(set(range(len(labels))) - set(test\_inds))  
train\_data = data\_mat[train\_inds,]  
train\_labels = y[train\_inds]  
test\_data = data\_mat[test\_inds,]  
test\_labels = y[test\_inds]  
print("train data: ", train\_data.shape)  
print("test data: ", test\_data.shape)  
  
clf = SGDClassifier(loss="log", penalty="none")  
clf.fit(train\_data, train\_labels)  
  
  
  
# Pull out the parameters (w,b) of the logistic regression model  
w = clf.coef\_[0,:]  
b = clf.intercept\_  
# Get predictions on training and test data  
preds\_train = clf.predict(train\_data)  
preds\_test = clf.predict(test\_data)  
#print("Accuracy: \n", accuracy\_score(preds\_train, preds\_test))  
#accuracy = sklearn.metrics.accuracy\_score(preds\_train, preds\_test)  
# Compute errors  
errs\_train = np.sum((preds\_train > 0.0) != (train\_labels > 0.0))  
errs\_test = np.sum((preds\_test > 0.0) != (test\_labels > 0.0))  
print ("Training error: ", float(errs\_train)/len(train\_labels))  
print ("Test error: ", float(errs\_test)/len(test\_labels))  
  
  
print ("train is as under:", train\_data)  
print ("test is as under:", test\_data)  
  
def margin\_counts(clf, test\_data, gamma):  
# Compute probability on each test point  
 preds = clf.predict\_proba(test\_data)[:,1]  
# Find data points for which prediction is at least gamma away from 0.5  
 margin\_inds = np.where((preds > (0.5+gamma)) | (preds < (0.5-gamma)))[0]  
 return float(len(margin\_inds))  
#Let us visualize the test set's distribution of margin values.  
gammas = np.arange(0, 0.5, 0.01)  
f = np.vectorize(lambda g: margin\_counts(clf, test\_data, g))  
plt.plot(gammas, f(gammas) / 500.0, linewidth=2, color='red')  
plt.xlabel('Margin', fontsize=14)  
plt.ylabel('Fraction of points above margin', fontsize=14)  
plt.show( )  
  
  
def margin\_errors(clf, test\_data, test\_labels, gamma):  
 # Compute probability on each test point  
 preds = clf.predict\_proba(test\_data)[:,1]  
 # Find data points for which prediction is at least gamma away from 0.5  
 margin\_inds = np.where((preds > (0.5+gamma)) | (preds < (0.5-gamma)))[0]  
 # Compute error on those data points.  
 num\_errors = np.sum((preds[margin\_inds] > 0.5) != (test\_labels[margin\_inds] > 0.0))  
 return float(num\_errors)/len(margin\_inds)  
#Let us visualize the relationship between margin and error rate.  
# Create grid of gamma values  
gammas = np.arange(0, 0.5, 0.01)  
# Compute margin\_errors on test data for each value of g  
f = np.vectorize(lambda g: margin\_errors(clf, test\_data, test\_labels, g))  
# Plot the result  
plt.plot(gammas, f(gammas), linewidth=2)  
plt.ylabel('Error rate', fontsize=14)  
plt.xlabel('Margin', fontsize=14)  
plt.show()