SI 630: Homework 1

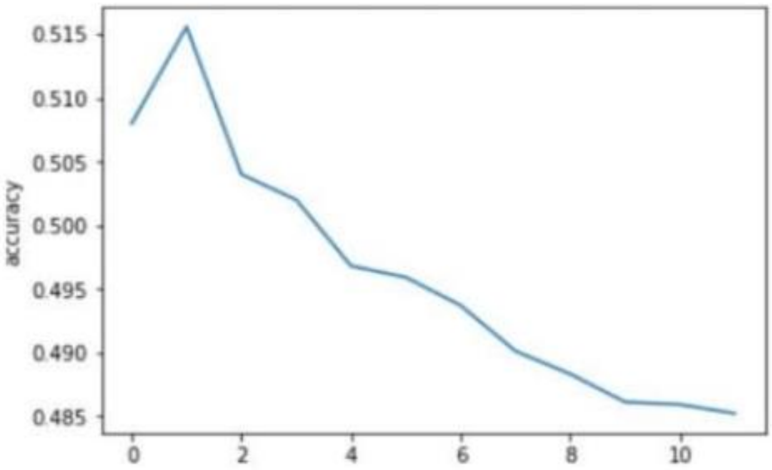
Yashaswini Joshi (Unique Name : yjoshi) Kaggle Name: Yashaswini Joshi

TASK 1:

1. What happens as you change the value of smoothing alpha? Include a plot of your classifier's performance on the development data where (i) your model's performance is on the y-axis and (ii) the choice in smoothing alpha is on the x-axis. Note that most people use α = 1; does this value give good performance for you?

Ans:

When alpha value is changed, I observed a decrease in accuracy with increase in alpha value.



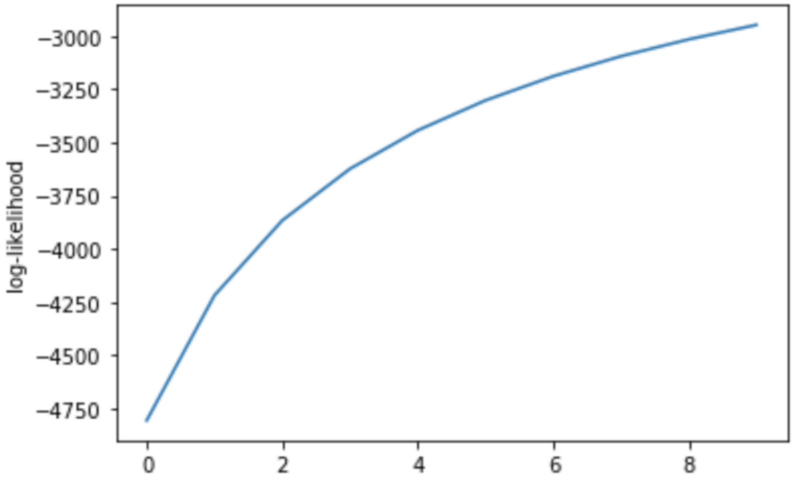
Code for Naïve Bayes:

```
import numpy as np
import matplotlib.pyplot as plt
import re
def tokenize(text):
    temp = text.split(" ")
    return temp
def better_tokenize(text):
    temp = text.split(" ")
    temp = [(word.lower()) for word in temp]
    temp = [(re.sub('[^A-Za-z]+', '', word)) for word in temp]


    words_to_be_removed=["having",'didnt','doesnt','n','u','ur','i','I',"youre","im","a","an",
"about", "above", "across", "after", "afterwards", "again", "against", "all", "almost", "alone", "along",
"already", "also","although","always","am","among", "amongst", "amoungst", "amount", "an", "and",
"another", "any","anyhow","anyone","anything","anyway", "anywhere", "are", "around", "as", "at",
"back","be","became", "because","become","becomes", "becoming", "been", "before", "beforehand",
"behind", "being", "below", "beside", "besides", "between", "beyond", "bill", "both", "bottom","but",
"by", "call", "can", "cannot", "cant", "co", "con", "could", "couldnt", "cry", "de", "describe", "detail",
"do", "done", "down", "due", "during", "each", "eg", "eight", "either", "eleven","else", "elsewhere",
"empty", "enough", "etc", "even", "ever", "every", "everyone", "everything", "everywhere", "except",
"few", "fifteen", "fify", "fill", "find", "fire", "first", "five", "for", "former", "formerly", "forty", "found",
"four", "from", "front", "full", "further", "get", "give", "go", "had", "has", "hasnt", "have", "he", "hence",
"her", "here", "hereafter", "hereby", "herein", "hereupon", "hers", "herself", "him", "himself", "his",
"how", "however", "hundred","i" "ie", "if", "in", "inc", "indeed", "interest", "into", "is", "it", "its", "itself",
"keep", "last", "latter", "latterly", "least", "less", "ltd", "made", "many", "may", "me", "meanwhile",
"might", "mill", "mine", "more", "moreover", "most", "mostly", "move", "much", "must", "my", "myself",
"name", "namely", "neither", "never", "nevertheless", "next", "nine", "no", "nobody", "none", "noone",
"nor", "not", "nothing", "now", "nowhere", "of", "off", "often", "on", "once", "one", "only", "onto", "or",
"other", "others", "otherwise", "our", "ours", "ourselves", "out", "over", "own","part", "per", "perhaps",
"please", "put", "rather", "re", "same", "see", "seem", "seemed", "seeming", "seems", "several", "she",
"should", "show", "since", "six", "sixty", "so", "some", "somehow", "someone", "something",
"sometime", "sometimes", "somewhere", "still", "such", "system", "take", "ten", "than", "that","thats",
"the", "their", "them", "themselves", "then", "thence", "there", "thereafter", "thereby", "therefore",
"therein", "thereupon", "these", "they", "thickv", "thin", "third", "this", "those", "though", "three",
"through", "throughout", "thru", "thus", "to", "too", "top", "toward", "towards", "twelve", "twenty",
"two", "un", "under", "until", "up", "upon", "us", "very", "via", "was", "we", "well", "were", "what",
"whatever", "when", "whence", "whenever", "where", "whereafter", "whereas", "whereby", "wherein",
"whereupon", "wherever", "whether", "which", "while", "whither", "who", "whoever", "whole",
"whom", "whose", "why", "will", "with", "within","without", "would", "yet", "you", "your", "yours",
"yourself", "yourselves", "the"]


        for j in range(0,len(temp)):
            for i in range(0,len(words_to_be_removed)):
                if(temp[j]==words_to_be_removed[i]):
                    temp[j]=''



        temp=[x for x in temp if x]
        return temp


def laplace_smoothing(counter_name, alpha=0):
    word_to_count = counter_name.copy() ## Make sure you use a "copy()" not to affect your original
dictionary
    V = len(word_to_count) ## V means the number of unique words (== a size of vocab)
    #print(V)
    for w, c in word_to_count.items():
        word_to_count[w] += alpha  ## == (n_ij + alpha) == numerator
    word_to_prob = {}  ## build a new dictionary for new values after smoothing
    total_counts = float(sum(word_to_count.values()))
    ##Since we've added alpha to all of the frequencies, total_counts implicitly is sum_freq + alpha*V
    for w, c in word_to_count.items():
        word_to_prob[w] = c / total_counts
    return word_to_prob

def train_classify(smoothing_alpha=0):
    with open('train.csv', mode='r') as infile:
        reader = list(csv.reader(infile))
        header = reader[0]
        troll=0
        normal=0
        total_count=0
        troll_list=[]
        normal_list=[]
        for row in reader[1:]:
            if(row[0]=='1'):
                troll = troll +1
                total_count=total_count+1
                tweet= row[1]
                troll_list=troll_list+better_tokenize(tweet)
                tweet


            else:
                normal =normal +1
                total_count= total_count+1
                tweet=row[1]
                normal_list=normal_list+better_tokenize(tweet)
        troll_dictionary= Counter(troll_list)
        normal_dictionary=Counter(normal_list)
        #print(normal_dictionary)

        troll_word_to_prob = laplace_smoothing(troll_dictionary,smoothing_alpha)
        #print(troll_word_to_prob)
        normal_word_to_prob = laplace_smoothing(normal_dictionary,smoothing_alpha)
        troll_prob= troll/total_count
        normal_prob= normal/total_count
        print(len(troll_dictionary))
        print("----------------")
        print(len(normal_dictionary))
        print("----------------")
        print(troll_prob)
        print("----------------")
        print(normal_prob)
        print("----------------")
        print(len(troll_word_to_prob))
        print("----------------")
        print(len(normal_word_to_prob))
        print("----------------")
    infile.close()
    with open('classify-output1.csv', 'w', newline='') as file2:
        fieldnames=['Troll','Tweet']
        writer=csv.DictWriter(file2, fieldnames=fieldnames)
        writer.writeheader()
        with open('testnb.csv') as infile:
            reader = list(csv.reader(infile))
            header = reader[0]
            accuracy=0
            for row in reader[1:]:
                prob1=1
                prob2=1
                tokens=better_tokenize(row[1])
                #print(tokens)
                for tok in troll_dictionary:
                    if tok in tokens:
                        prob1=prob1*troll_word_to_prob[tok]*tokens.count(tok)
                    else:
                        prob1=prob1*(1-troll_word_to_prob[tok])
                for tok in normal_dictionary:
                    if tok in tokens:
                        prob2=prob2*normal_word_to_prob[tok]*tokens.count(tok)
                    else:
                        prob2=prob2*(1-normal_word_to_prob[tok])

                troll_out= prob1*troll_prob/(prob1*troll_prob+prob2*normal_prob+0.0000000000001)
                normal_out= prob2*normal_prob/(prob1*troll_prob+prob2*normal_prob+0.0000000000001)
                if(troll_out>0.5):
                    writer.writerow({'Troll':'1','Tweet':row[1]})
                else:
                    writer.writerow({'Troll':'0','Tweet':row[1]})
def train_classify_dev(smoothing_alpha=0):
    with open('train.csv', mode='r') as infile:
        reader = list(csv.reader(infile))
        header = reader[0]
        troll=0
        normal=0
        total_count=0
        troll_list=[]
        normal_list=[]
        for row in reader[1:]:
            if(row[0]=='1'):
                troll = troll +1
                total_count=total_count+1
                tweet= row[1]
                troll_list=troll_list+better_tokenize(tweet)
                #print(type(troll_list))


            else:
                normal =normal +1
                total_count= total_count+1
                tweet=row[1]
                normal_list=normal_list+better_tokenize(tweet)
        troll_dictionary= Counter(troll_list)
        normal_dictionary=Counter(normal_list)
        print(normal_dictionary)

        troll_word_to_prob = laplace_smoothing(troll_dictionary,smoothing_alpha)
        #print(troll_word_to_prob)
        normal_word_to_prob = laplace_smoothing(normal_dictionary,smoothing_alpha)
        troll_prob= troll/total_count
        normal_prob= normal/total_count
        print(len(troll_dictionary))
        print("----------------")
        print(len(normal_dictionary))
        print("----------------")
        print(troll_prob)
        print("----------------")
        print(normal_prob)
        print("----------------")
        print(len(troll_word_to_prob))
        print("----------------")
        print(len(normal_word_to_prob))
        print("----------------")
    infile.close()
    with open('classify-output2.csv', 'w', newline='') as file2:
        fieldnames=['troll','Tweet']
        writer=csv.DictWriter(file2, fieldnames=fieldnames)
        writer.writeheader()
        with open('devnb.csv') as infile:
            reader = list(csv.reader(infile))
            header = reader[0]
            accuracy=0
            for row in reader[1:]:
                prob1=1
                prob2=1
                tokens=better_tokenize(row[1])
                #print(tokens)
                for tok in troll_dictionary:
                    if tok in tokens:
                        prob1=prob1*troll_word_to_prob[tok]*tokens.count(tok)
                    else:
                        prob1=prob1*(1-troll_word_to_prob[tok])
                for tok in normal_dictionary:
                    if tok in tokens:
                        prob2=prob2*normal_word_to_prob[tok]*tokens.count(tok)
                    else:
                        prob2=prob2*(1-normal_word_to_prob[tok])

                troll_out= prob1*troll_prob/(prob1*troll_prob+prob2*normal_prob+0.0000000000001)
                normal_out= prob2*normal_prob/(prob1*troll_prob+prob2*normal_prob+0.0000000000001)
                #print(troll_out,normal_out)
                if(normal_out<troll_out):
                    if(row[0]=='1'):
                        accuracy+=1
                else:
                    if(row[0]=='0'):
                        accuracy+=1

        print(accuracy)
```

TASK 2:

1. Make a plot of the log-likelihood every step. Did the model converge at some point (i.e., does the log likelihood remain stable)?

Ans: Plot of Learning rate = 5e-5



Code for Logistic Regression:

```
import numpy as np
import matplotlib.pyplot as plt
import re
def tokenize(text):
   temp = text.split(" ")
   return temp
def better_tokenize(text):
   temp = text.split(" ")
   temp = [(word.lower()) for word in temp]
   temp = [(re.sub('[^A-Za-z]+', '', word)) for word in temp]

   words_to_be_removed=["having",'didnt','doesnt','n','u','ur','i','I',"youre","im","a","an", "about",
"above", "across", "after", "afterwards", "again", "against", "all", "almost", "alone", "along", "already",
"also","although","always","am","among", "amongst", "amoungst", "amount", "an", "and", "another",
"any","anyhow","anyone","anything","anyway", "anywhere", "are", "around", "as", "at",
"back","be","became", "because","become","becomes", "becoming", "been", "before", "beforehand",
"behind", "being", "below", "beside", "besides", "between", "beyond", "bill", "both", "bottom","but",
"by", "call", "can", "cannot", "cant", "co", "con", "could", "couldnt", "cry", "de", "describe", "detail",
"do", "done", "down", "due", "during", "each", "eg", "eight", "either", "eleven","else", "elsewhere",
"empty", "enough", "etc", "even", "ever", "every", "everyone", "everything", "everywhere", "except",
"few", "fifteen", "fify", "fill", "find", "fire", "first", "five", "for", "former", "formerly", "forty", "found",
"four", "from", "front", "full", "further", "get", "give", "go", "had", "has", "hasnt", "have", "he", "hence",
"her", "here", "hereafter", "hereby", "herein", "hereupon", "hers", "herself", "him", "himself", "his",
"how", "however", "hundred","i" "ie", "if", "in", "inc", "indeed", "interest", "into", "is", "it", "its", "itself",
"keep", "last", "latter", "latterly", "least", "less", "ltd", "made", "many", "may", "me", "meanwhile",
"might", "mill", "mine", "more", "moreover", "most", "mostly", "move", "much", "must", "my", "myself",
"name", "namely", "neither", "never", "nevertheless", "next", "nine", "no", "nobody", "none", "noone",
"nor", "not", "nothing", "now", "nowhere", "of", "off", "often", "on", "once", "one", "only", "onto", "or",
"other", "others", "otherwise", "our", "ours", "ourselves", "out", "over", "own","part", "per", "perhaps",
"please", "put", "rather", "re", "same", "see", "seem", "seemed", "seeming", "seems", "several", "she",
"should", "show", "since", "six", "sixty", "so", "some", "somehow", "someone", "something",
"sometime", "sometimes", "somewhere", "still", "such", "system", "take", "ten", "than", "that","thats",
"the", "their", "them", "themselves", "then", "thence", "there", "thereafter", "thereby", "therefore",
"therein", "thereupon", "these", "they", "thickv", "thin", "third", "this", "those", "though", "three",
"through", "throughout", "thru", "thus", "to", "too", "top", "toward", "towards", "twelve", "twenty",
"two", "un", "under", "until", "up", "upon", "us", "very", "via", "was", "we", "well", "were", "what",
"whatever", "when", "whence", "whenever", "where", "whereafter", "whereas", "whereby", "wherein",
"whereupon", "wherever", "whether", "which", "while", "whither", "who", "whoever", "whole",
"whom", "whose", "why", "will", "with", "within","without", "would", "yet", "you", "your", "yours",
"yourself", "yourselves", "the"]

    for j in range(0,len(temp)):
        for i in range(0,len(words_to_be_removed)):
            if(temp[j]==words_to_be_removed[i]):
                temp[j]=''



    temp=[x for x in temp if x]
    return temp
#Define sigmoid
def sigmoidFunct(t):
    return 1./(1+np.exp(-t))
# negative log-likelihood
def log_likelihood(X_bias, y, Beta):
    temp = np.dot(X_bias, Beta)
    ll = np.sum( y*temp - np.log(1 + np.exp(temp)) )
    return ll
def train_classify(alpha):
    vocab_list=[]
    log_likelihood_value=[]
    with open('train.csv', mode='r') as infile:
        reader = list(csv.reader(infile))
        header = reader[0]
        troll=0
        normal=0
        total_count=0
```

```
word_list=[]
Beta=np.array(())
Y=[]
for row in reader[1:]:
    Y.append(int(row[0]))
    tweet= row[1]
    word_list=word_list+better_tokenize(tweet)
word_dictionary= Counter(word_list)
print(len(Y))
y=np.asarray(Y)
V=len(word_dictionary)
n=10000
X=np.zeros((n,V))
#print(np.ones((1,10000)).shape)
#bias=np.ones((n))
X_bias=np.concatenate([np.ones((10000,1)),X],axis =1)
#print(X_bias)
vocab_list=list(word_dictionary.keys())
#print(word_dictionary)
Beta= np.zeros((V+1))
count=0
#10001
for row in reader[1:10001]:
    tweet= row[1]
    words=better_tokenize(tweet)
    for word in words:
        j=vocab_list.index(word)
        #print(count)
        X_bias[count][j+1]+=1
    count+=1
    #print(count)
#alpha=5e-5
l=0
while(l<10):

    print(Beta.shape,X_bias.shape)
    #10000
    for k in range(0,10000):
        t= np.matmul(X_bias,np.transpose(Beta))
        #print(y.shape,t.shape)
        temp=np.subtract(y,sigmoidFunct(t))
        Beta_1=Beta+alpha*np.matmul(np.transpose(temp),X_bias)
        error=np.sum(temp**2)
        print(l,k,error)
```

```
            Beta=Beta_1

        l+=1
        log_likelihood_value.append(log_likelihood(X_bias,y,Beta))
    plt.plot(log_likelihood_value)
    plt.ylabel('log-likelihood')
    plt.show()

infile.close()
with open('devnb.csv', mode='r') as infile:
    reader = list(csv.reader(infile))
    header = reader[0]
    Y_dev=[]
    n=2000
    count=0
    X_dev=np.zeros((n,V))
    X_dev_bias=np.concatenate([np.ones((2000,1)),X_dev],axis =1)
    #print(vocab_list.index('fuck'))
    for row in reader[1:2001]:
        tweet=row[1]
        words=better_tokenize(tweet)
        Y_dev.append(int(row[0]))
        for word in words:
            if(word in vocab_list):
                j=vocab_list.index(word)
                #print(count)
                X_dev_bias[count][j+1]+=1
        count+=1
    #word_dictionary= Counter(word_list)
    print(len(Y))

    y_dev=np.asarray(Y_dev)
    #Beta= np.zeros((V+1))
    t= np.matmul(X_dev_bias,np.transpose(Beta))

    y_dev_hat=sigmoidFunct(t)
    error=np.sum((y_dev-y_dev_hat)**2)
    print(error)
infile.close()
with open('classify-output_log2.csv', 'w', newline='') as file2:
    fieldnames=['Troll','Tweet']
    writer=csv.DictWriter(file2, fieldnames=fieldnames)
    writer.writeheader()
    with open('testnb.csv') as infile:
            reader = list(csv.reader(infile))
            header = reader[0]
            #Y_test=[]
            n=8000
            count=0
            X_test=np.zeros((n,V))
            X_test_bias=np.concatenate([np.ones((8000,1)),X_test],axis =1)
            #print(vocab_list.index('fuck'))
            for row in reader[1:8001]:
                tweet=row[1]
                words=better_tokenize(tweet)
                #Y_test.append(int(row[0]))
                for word in words:
                    if(word in vocab_list):
                        j=vocab_list.index(word)
                        #print(count)
                        X_test_bias[count][j+1]+=1
                count+=1
            #word_dictionary= Counter(word_list)
            #print(len(Y))

            #y_test=np.asarray(Y_test)
            #Beta= np.zeros((V+1))
            t= np.matmul(X_test_bias,np.transpose(Beta))

            y_test_hat=sigmoidFunct(t)
            print(y_test_hat)
            print("ther")
            h=0
            for row in reader[1:8001]:
                if(y_test_hat[h]>0.4):
                    writer.writerow({'Troll':'1','Tweet':row[1]})
                else:
                    writer.writerow({'Troll':'0','Tweet':row[1]})
                h+=1



    return(vocab_list,Beta_1,V)

def classify(vocab_list,V,Beta):
    with open('devnb.csv', mode='r') as infile:
        reader = list(csv.reader(infile))
        header = reader[0]
```

```
Y_dev=[]
n=2000
count=0
X_dev=np.zeros((n,V))
X_dev_bias=np.concatenate([np.ones((2000,1)),X_dev],axis =1)
for row in reader[1:2001]:
    tweet=row[1]
    words=better_tokenize(tweet)
    Y_dev.append(int(row[0]))
    for word in words:
        j=vocab_list.index(word)
        #print(count)
        X_dev_bias[count][j+1]+=1
    count+=1
#word_dictionary= Counter(word_list)
print(len(Y))

y_dev=np.asarray(Y_dev)
#Beta= np.zeros((V+1))
t= np.matmul(X_dev_bias,np.transpose(Beta))

y_hat=sigmoidFunct(t)
error=np.sum((y-y_hat)**2)
print(error)
```