Homework 1: Perceptron Algorithm

Student Name: `Kuan-Lin Liu`
NetID: `kl1482`

Packages

In [1]:

```
#!pip install pandas
#!pip install numpy
```

In [2]:

```
import pandas as pd, numpy as np
import time
import matplotlib.pyplot as plt
```

Question 1

Tasks:

1. Read in spam_train.txt
2. Split into training set (first 4000) and validation set (last 1000)
3. Explanation

In [3]:

```
data_list = []
my_path = "/home/jovyan/shared/spam_train.txt"
with open(my_path, "r") as f:
    for line in f:
        data_list.append([line[0], line[2:].strip("\n").split()])
data_list = np.array(data_list)
```

In [4]:

```
train = data_list[:4000, :]
validation = data_list[4000:, :]
```

> The validation set which is sampled from the original training data help us tune the hyperparameters to avoid building a extremely complex model. In other words, if we only train the model on the training set, we may tend to get the best model so far. However, the model could perform really bad on the test set. In summary, without the validation set, we would be more likely to cause overfitting on the model.

Question 2

Tasks:

1. Transform the words into feature vectors
2. Keep the word appearing in more than or equal to 30 emails.

In [5]:

```
def getWordList(data_list, threshold):
    word_hash = {}
    for target, words in data_list:
        # how many email does a word exist?
        unique_words = np.unique(words)
        for i in unique_words:
            if i not in word_hash.keys():
                word_hash[i] = 1
            else:
                word_hash[i] += 1

    # obtain the word list (>=30)
    return np.array([k for k, v in word_hash.items() if v >= threshold])

# create word list based on the training set
train_word_list = getWordList(train, 30)
```

In [6]:

```
def transformToFeatures(word_list, data_list):
    # build a dict of words for search: O(1)
    word_dict = {value: index for index, value in enumerate(word_list)}

    # initializing a n*m array (n=# of emails; m=# of words)
    array_of_features = np.zeros((len(data_list), len(word_list)))

    # For every words in each email, see if it is in the word_dict by
    index_email = 0
    for target, words in data_list:
        for w in np.unique(words):
            if word_dict.get(w) is not None:
                array_of_features[index_email, word_dict.get(w)] = 1
        index_email += 1
    return array_of_features

train_features_vector = transformToFeatures(train_word_list, train)
validation_features_vector = transformToFeatures(train_word_list, validation)
```

In [7]:

```
print(train_features_vector.shape)
print(validation_features_vector.shape)
```

```
(4000, 2376)
(1000, 2376)
```

Question 3

Tasks:

1. Create a perceptron algorithm, `perceptron_train` , and a testing function, `perceptron_test`

In [8]:

```
def signFunc(x):
    if x >= 0:
        return 1
    else:
        return -1

def perceptron_train(data, max_iter=None, avg_func=False):
    '''
    1. data is a two D array with the target variable in the first column and fe
ature set in the rest
    2. The target variable should be -1 or 1
    2. The target variable should be -1 or 1
    '''

    # initials
    w = np.zeros(data.shape[1]-1)
    error = 1 # for while loop
    batch = 0 # number of iteration
    k = 0 # number of updating
    iteration = 0
    sum_w = np.zeros(data.shape[1]-1) # for counting the final average of weights

    while error != 0:
        error = 0
        batch += 1
        for i in range(data.shape[0]):
            x = data[i, 1:]
            y = data[i, 0] # a number; not an array
            if signFunc(np.dot(w, x)) != y:
                error += 1
                w += y*x
            iteration += 1
            sum_w += w
        k += error

        if batch == max_iter:
            break

    # Return
    if avg_func:
        return sum_w/iteration, k, batch
    else:
        return w, k, batch

def perceptron_test(w, data):
    # initializing
    x = data[:, 1:]
    y = np.array(data[:, 0])

    pred_value = np.dot(x, w)
    sign_vectorizer = np.vectorize(signFunc)
    sign_pred_value = sign_vectorizer(pred_value)

    return 1-(sum(sign_pred_value==y)/len(y)) # 1-number of right prediction
```

Question 4

Tasks:

1. Combine the target variable and the feature sets into a new array for both training and validation set.
2. Do Training and then testing the model on the validation set
3. Print out the training error and the validation error

In [9]:

```
# Merge the target variable and the feature vectors into a DataFrame
y_train = np.array([[-1 if x==0 else 1 for x in train[:, 0].astype(int)]]) # 2D
train_array = np.concatenate((y_train.T, train_features_vector), axis=1)

y_val = np.array([[-1 if x==0 else 1 for x in validation[:, 0].astype(int)]]) # 2D
val_array = np.concatenate((y_val.T, validation_features_vector), axis=1)
```

In [10]:

```
train_result = perceptron_train(train_array)
w, k, iteration = train_result
print(k, iteration)
```

```
447 11
```

In [11]:

```
print("The training error is:", perceptron_test(train_result[0], train_array))
print("-" * 20)
print("The validation error is:", perceptron_test(train_result[0], val_array))
```

```
The training error is: 0.0
--------------------
The validation error is: 0.020000000000000018
```

Question 5

Tasks:

1. Find the most positive and negative words by the size of weights

In [12]:

```
weight = train_result[0]
desc_weight = sorted(enumerate(weight), key=lambda k: k[1], reverse=True)[:30] # (index, value)
print("The most positive 15 words are:", train_word_list[[index for index, value in desc_weight]])

print("-" * 50)

asc_weight = sorted(enumerate(weight), key=lambda k: k[1], reverse=False)[:15]
print("The most negative 15 words are:", train_word_list[[index for index, value in asc_weight]])
```

```
The most positive 15 words are: ['sight' 'our' 'remov' 'yourself' 'c
lick' 'these' 'nbsp' 'pleas' 'market'
 'guarante' 'your' 'present' 'ever' 'deathtospamdeathtospamdeathtosp
am'
 'am' 'major' 'below' 'brand' 'further' 'internet' 'email' 'sincer'
'will'
 'bodi' 'hour' 'simpl' 'basenumb' 'dollarnumb' 'you' 'contact']
--------------------------------------------------
The most negative 15 words are: ['but' 'wrote' 'prefer' 'and' 'i' 'r
eserv' 'on' 'still' 'technolog' 'sinc'
 'copyright' 'url' 'instead' 'upgrad' 'recipi']
```

Question 6

Tasks:

1. Create the `Averaged Perceptron Algorithm` from perceptron_train

In [13]:

```
def averaged_perceptron_train(data, max_iter=None):
    return perceptron_train(data, max_iter=max_iter, avg_func=True)
```

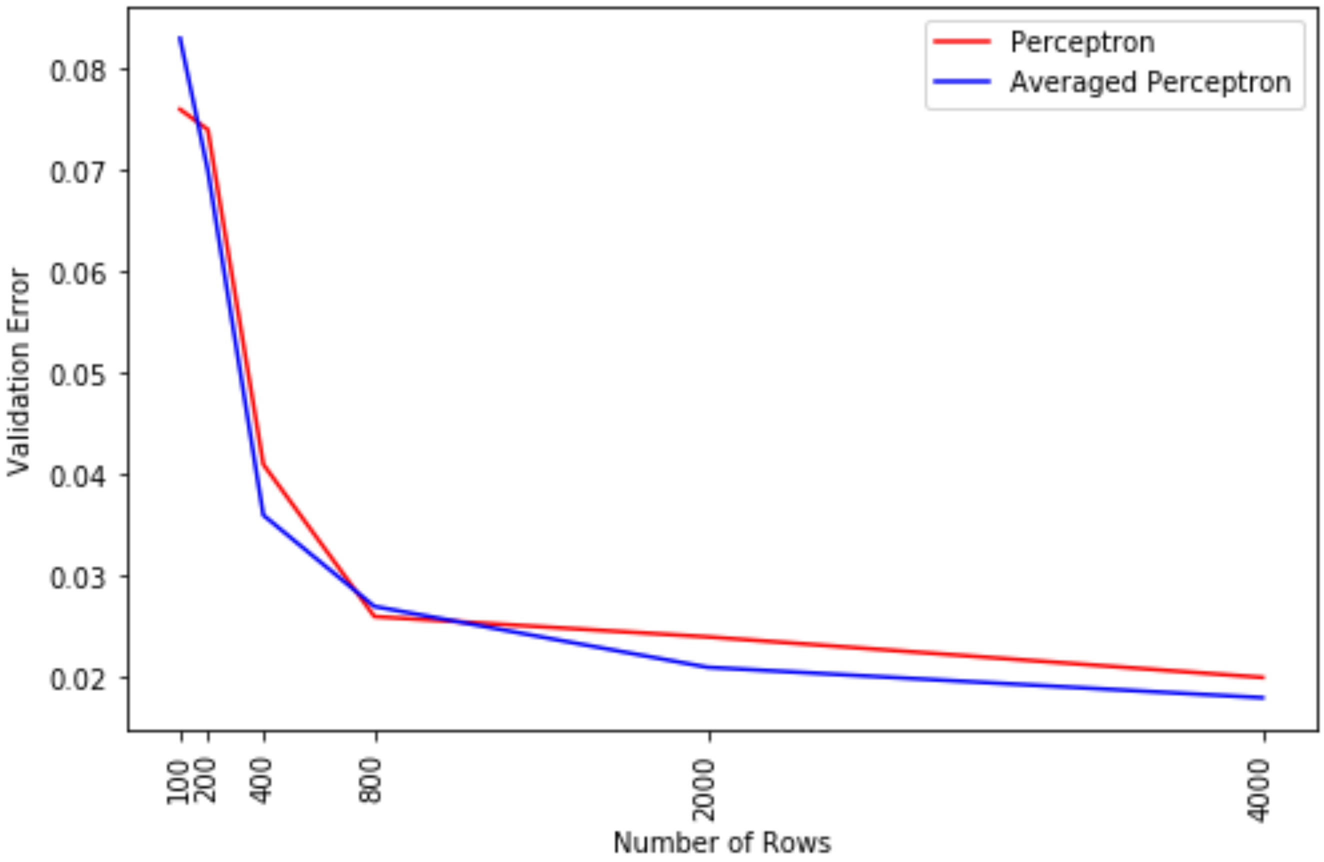
Question 7

Tasks:

1. Create a learning curve for different number of iteration

In [14]:

```
n_sample = [100, 200, 400, 800, 2000, 4000]
val_error = []
avg_val_error = []
for n in n_sample:
    # perceptron
    perceptron_result = perceptron_train(train_array[:n])
    val_error.append(perceptron_test(perceptron_result[0], val_array))
    # averaged perceptron
    avg_perceptron_result = averaged_perceptron_train(train_array[:n])
    avg_val_error.append(perceptron_test(avg_perceptron_result[0], val_array))

plt.figure(figsize=(8,5))
plt.plot(n_sample, val_error, c="red")
plt.plot(n_sample, avg_val_error, c = "blue")
plt.xlabel("Number of Rows")
plt.ylabel("Validation Error")
plt.xticks(ticks=n_sample,rotation=90)
plt.legend(labels=["Perceptron", "Averaged Perceptron"])
plt.show()
```



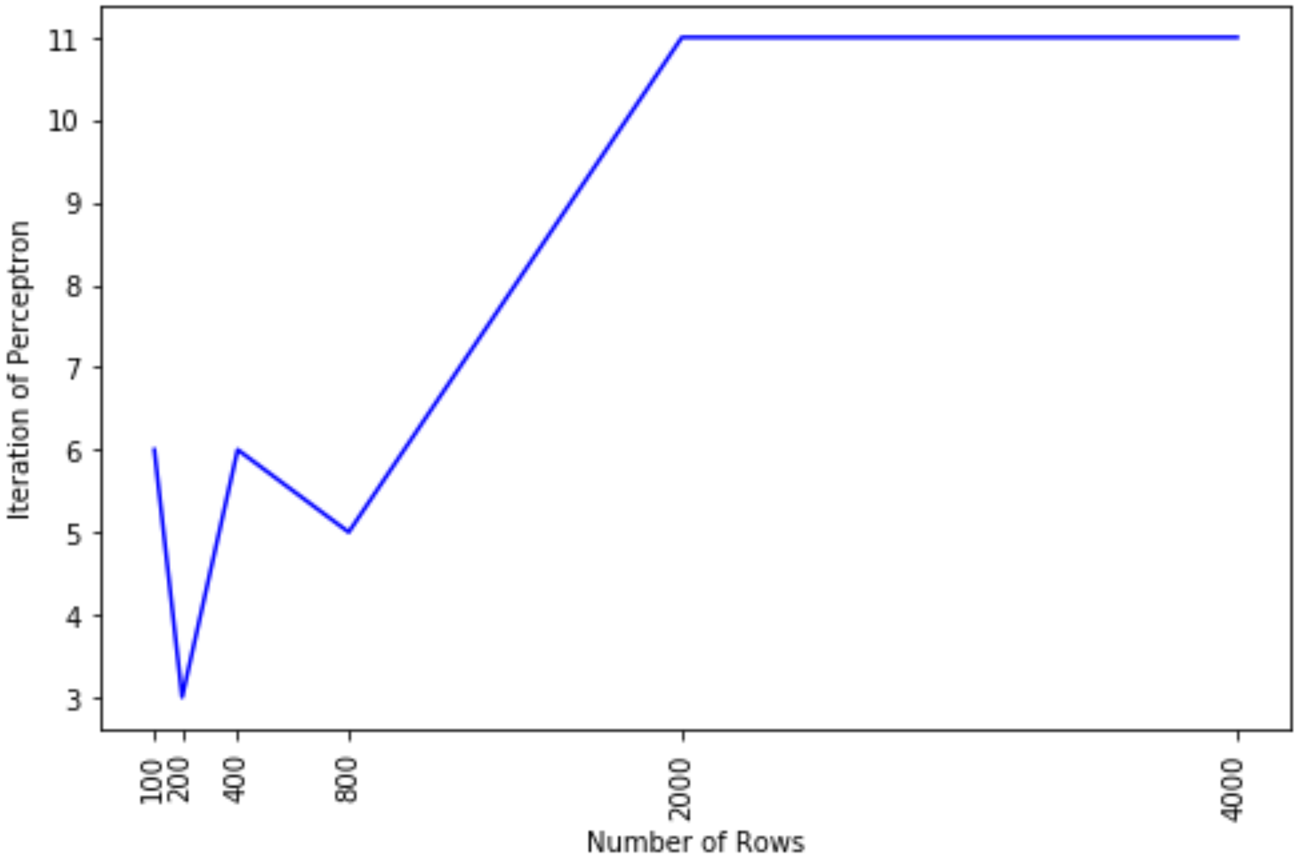
Question 8

Tasks:

1. Get the number of iteration with different number of rows

In [15]:

```
n_sample = [100, 200, 400, 800, 2000, 4000]
iter_freq = []
for n in n_sample:
    iter_freq.append(perceptron_train(train_array[:n])[2])

plt.figure(figsize=(8,5))
plt.plot(n_sample, iter_freq, c="blue")
plt.xlabel("Number of Rows")
plt.ylabel("Iteration of Perceptron")
plt.xticks(ticks=n_sample,rotation=90)
plt.show()
```



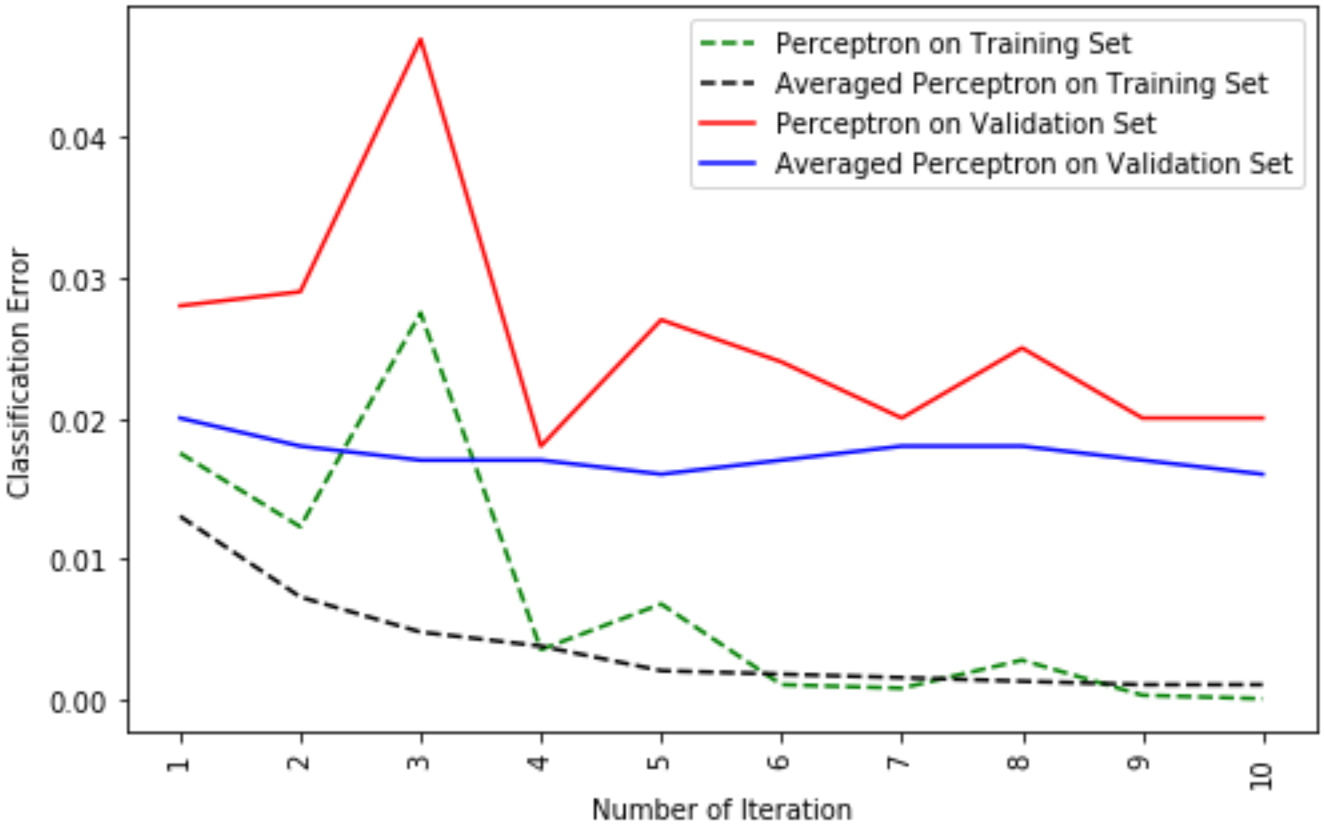
Question 9

Tasks:

1. Compare the two algorithms and find the best configuration

In [16]:

```
batch = range(1, 11)
train_val_error = []
avg_train_val_error = []
for i in batch:
    # perceptron
    perc_train_result = perceptron_train(train_array, max_iter=i)
    train_val_error.append((perceptron_test(perc_train_result[0], train_array),
                            perceptron_test(perc_train_result[0], val_array)))
    # averaged perceptron
    avg_perc_train_result = perceptron_train(train_array, max_iter=i, avg_func=True)
    avg_train_val_error.append((perceptron_test(avg_perc_train_result[0], train_array),
                                perceptron_test(avg_perc_train_result[0], val_array)))

plt.figure(figsize=(8,5))
plt.plot(batch, [train for train, val in train_val_error], c="green", ls="--")
plt.plot(batch, [train for train, val in avg_train_val_error], c="black", ls="--")
plt.plot(batch, [val for train, val in train_val_error], c="red")
plt.plot(batch, [val for train, val in avg_train_val_error], c="blue")
plt.xlabel("Number of Iteration")
plt.ylabel("Classification Error")
plt.xticks(ticks=batch,rotation=90)
plt.legend(labels=["Perceptron on Training Set", "Averaged Perceptron on Training Set",
                   "Perceptron on Validation Set", "Averaged Perceptron on Validation Set"])
plt.show()
```

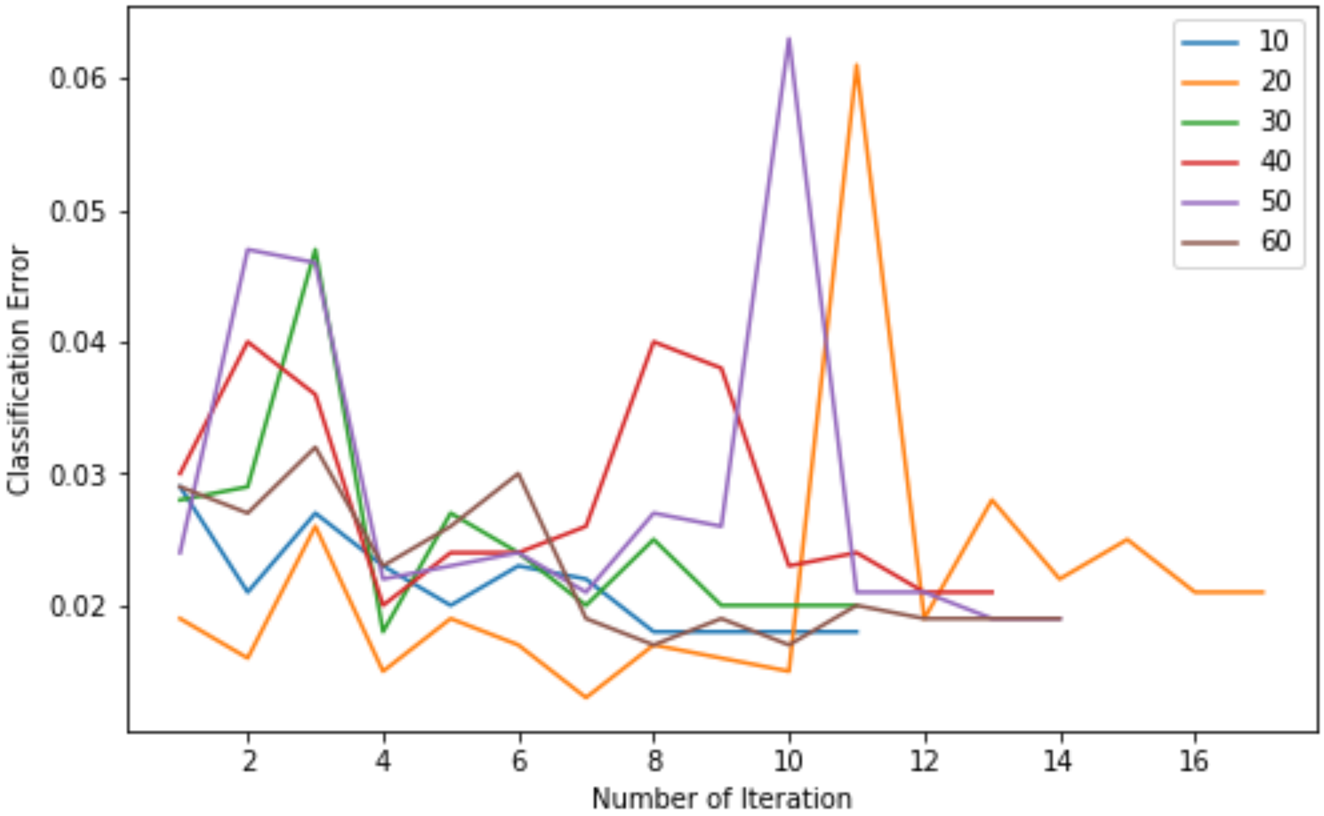


Overall, the averaged perceptron model surpass the standard perceptron with lower validation error on each iteration. Especially, when I take the number of iteration as `5` or `10`, I can obtain the lowest validation error on the averaged perceptron.

Question 10

In [17]:

```
import matplotlib.colors as mcolors
# initializing hyperparameter
max_word = [10, 20, 30, 40, 50, 60]
val_error_list = []

plt.figure(figsize=(8,5))
for m in max_word:
    # transform to feature vectors
    cust_word_list = getWordList(train, m)
    cust_train_features_vector = transformToFeatures(cust_word_list, train)
    cust_validation_features_vector = transformToFeatures(cust_word_list, valida
tion)

    # combine y and x
    cust_train_array = np.concatenate((y_train.T, cust_train_features_vector), a
xis=1)
    cust_val_array = np.concatenate((y_val.T, cust_validation_features_vector),
axis=1)

    cust_train_result = perceptron_train(cust_train_array)
    w, k, iteration = cust_train_result

    # tuning on iteration(copy code from question 9)
    iter_list = []
    for i in range(1, iteration+1):
        result = perceptron_train(cust_train_array, max_iter = i)
        iter_list.append(perceptron_test(result[0], cust_val_array))

    val_error_list.append(round(min(iter_list), 3))

    plt.plot(range(1, iteration+1), iter_list, ls="-")

plt.xlabel("Number of Iteration")
plt.ylabel("Classification Error")
plt.legend(labels=max_word)
plt.show()

print("The lowest validation error for each max word setting:", val_error_list)
```



```
The lowest validation error for each max word setting: [0.018, 0.013
, 0.018, 0.02, 0.019, 0.017]
```

Question 11

Tasks:

1. Train the model on the whole training set
2. Test the model by spam_test.txt

In [18]:

```
# Continuing from question 9
# Reading in the test set
my_path = "/home/jovyan/shared/spam_test.txt"
test_data = []
with open(my_path, "r") as f:
    for line in f:
        test_data.append([line[0], line[2:].strip("\n").split()])

# test data
test_data = np.array(test_data)

# get word list
new_word_list = getWordList(data_list, 30)

# Preprocessing
new_train_features_vector = transformToFeatures(new_word_list, data_list)
new_y_train = np.array([[-1 if x==0 else 1 for x in data_list[:, 0].astype(int)]
]) # 2D
new_train_array = np.concatenate((new_y_train.T, new_train_features_vector), axi
s=1)

test_features_vector = transformToFeatures(new_word_list, test_data)
new_y_test = np.array([[-1 if x==0 else 1 for x in test_data[:, 0].astype(int)]]
) # 2D
test_array = np.concatenate((new_y_test.T, test_features_vector), axis=1)
```

In [19]:

```
# training the model
result_5 = perceptron_train(new_train_array, max_iter=5, avg_func=True)
result_10 = perceptron_train(new_train_array, max_iter=5, avg_func=True)
print("The testing error for 5 iteration is:", round(perceptron_test(result_5[0]
, test_array), 3))
print("The testing error for 10 iteration is:", round(perceptron_test(result_10[
0], test_array), 3))
```

```
The testing error for 5 iteration is: 0.018
The testing error for 10 iteration is: 0.018
```