xl2590 Introduction to Machine Learning

Perceptron algorithm

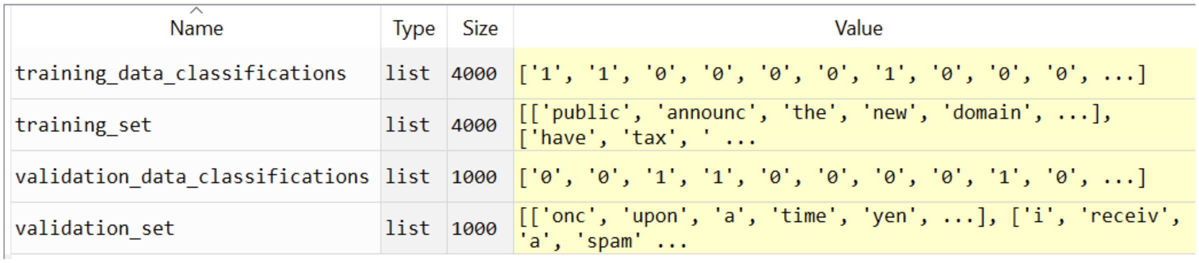
September 18, 2019

question one

This problem set will involve your implementing several variants of the Perceptron algorithm. Before you can build these models and measure their performance, split your training data into a training and validate set, putting the last 1000 emails into the validation set. Explain why measuring the performance of your final classifier would be problematic had you not created this validation set.

```python
import numpy as np
import matplotlib.pyplot as plt
import math
"""
question one
split the training data (i.e. spam train.txt) into a training and validate set,
putting the first 4000 emails into the training set
putting the last 1000 emails into the validation set.
when putting each email into the training set and validation set, split each letter
then seprate the first letter which is 0 or 1, the classification of them
"""
def get_ads():
    training_set = []
    validation_set = []
    training_data_classifications = []
    validation_data_classifications = []
    with open("spam_train.txt") as training_data_file:
        for i,line in enumerate(training_data_file):
            if i < 4000:
                training_set.append(line.split())
                training_data_classifications.append(training_set[i].pop(0))
            if i >= 4000:
                validation_set.append(line.split())
                validation_data_classifications.append(validation_set[i-4000].pop(0))
    return training_set,training_data_classifications, validation_set, validation_data_classifications
training_set,training_data_classifications, validation_set, validation_data_classifications = get_ads()
```

Function 1: question one



Name	Type	Size	Value
training_data_classifications	list	4000	['1', '1', '0', '0', '0', '0', '1', '0', '0', '0', ...]
training_set	list	4000	[['public', 'announc', 'the', 'new', 'domain', ...], ['have', 'tax', ' ...
validation_data_classifications	list	1000	['0', '0', '1', '1', '0', '0', '0', '0', '1', '0', ...]
validation_set	list	1000	[['onc', 'upon', 'a', 'time', 'yen', ...], ['i', 'receiv', 'a', 'spam' ...

Figure 1: question one data frame

I add some explanations in the coding part, and the idea of this question is: first, open the *spam_train.txt* file and read that each line. Second, separate the line from 1-4000 and 4000-5000, according to the question, the first 4000 line is for *training*, and 1000 is for *validation*. Because I will use this function in the further question, so I wrote that one into a function, which return $training\_set$, $training\_data\_classifications$, $validation\_set$, $validation\_data\_classifications$. Which you can see in fig:question one data frame.

To example why we use the validation set. In this homework, we have 5000 data which is email, if we train all of the 5000 feature vectors, and after several iterations, we will have a weight for each item. However, if we get the training error equal to 0.0, that may cause over-fitting in the testing set. So the validation set is using for test whether the weight we get is good enough for the testing set, which is not under-fitting or over-fitting. After we get a good error result with the validation set, we can use the weight to train the testing set.

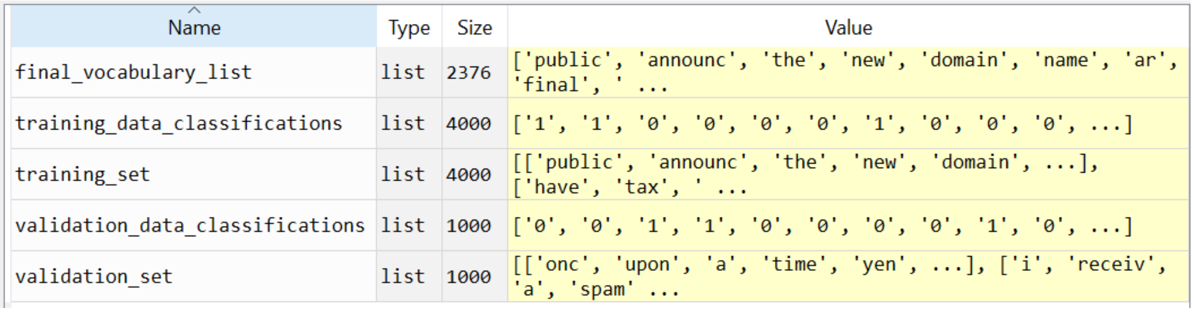
question two

Transform all of the data into feature vectors. Build a vocabulary list using only the 4000 e-mail training set by finding all words that occur across the training set. Ignore all words that appear in fewer than X = 30 e-mails of the 4000 e-mail training set – this is both a means of preventing overfitting and of improving scalability. For each email, transform it into a feature vector .

```python
"""
question two
Transform all of the data into feature vectors.
Build a vocabulary list using only the 4000 e-mail training set by finding all words that occur across the training set.
Ignore all words that appear in fewer than X = 30 emails, so we need to use dict in python, to trans all lines in training_set in to dict, to know the number of the words appear e-mails of the 4000 e-mail training set  this is both a means of preventing overfitting and of improving scalability.
"""
def get_vocabulary_list(X):
    vocabulary_list = []
    """
    using dict.fromkeys() to remove the words appear many times
    example:
        seq = ('Google', 'Runoob', 'Taobao','Google', 'Runoob', 'Taobao')
        >>> dict = dict.fromkeys(seq)
        >>> dict
        {'Google': None, 'Runoob': None, 'Taobao': None}
    """
    for line in training_set:
        vocabulary_list += (list(dict.fromkeys(line)))
    # using dict to compute the number that word appear in different emails
    # if numbers bigger than 30 then store in final_vocabulary_list
    counts = {}
    for word in vocabulary_list:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1
    final_vocabulary_list = []
    for word in counts:
        if counts[word] >= X:
            final_vocabulary_list.append(word)
    return final_vocabulary_list
final_vocabulary_list = get_vocabulary_list(30)
```

Function 2: question two function

Firstly, to building a vocabulary list. According to the instructor, we need to select the words which appear more than 30 times(contain 30) in different emails. So I firstly adding the different words from each email into *vocabulary_list*, and initialing a *dict*(named counts) for counting the appear times for each words in *vocabulary_list*. Then selecting the words which *vocabulary_list*[*word*] is no fewer than 30.

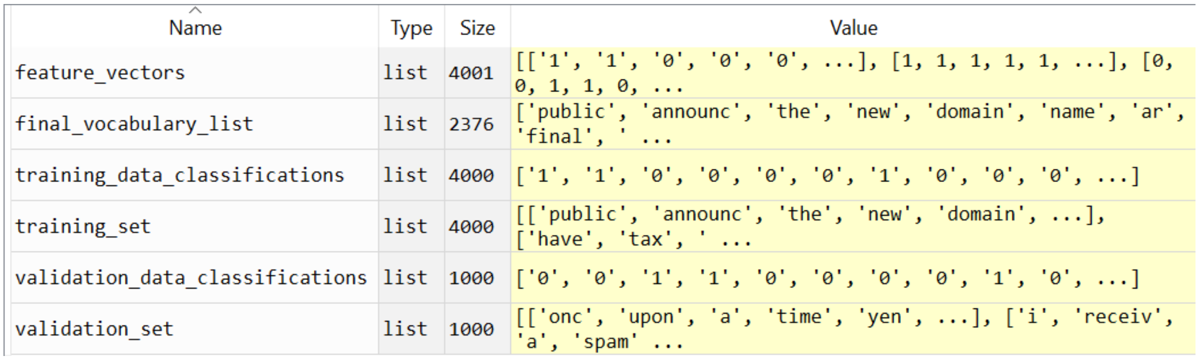


Name	Type	Size	Value
final_vocabulary_list	list	2376	['public', 'announc', 'the', 'new', 'domain', 'name', 'ar', 'final', ' ...
training_data_classifications	list	4000	['1', '1', '0', '0', '0', '0', '1', '0', '0', '0', ...]
training_set	list	4000	[['public', 'announc', 'the', 'new', 'domain', ...], ['have', 'tax', ' ...
validation_data_classifications	list	1000	['0', '0', '1', '1', '0', '0', '0', '0', '1', '0', ...]
validation_set	list	1000	[['onc', 'upon', 'a', 'time', 'yen', ...], ['i', 'receiv', 'a', 'spam' ...

Figure 2: question two data frame

```
"""
For each email, transform it into a feature vector
x where the ith entry, xi, is 1 if the ith word in the vocabulary occurs in the email, and 0 otherwise.
"""
def get_feature_vectors(training_set):
    feature_vectors = []
    feature_vectors = [[1 if word in vector else 0 for word in final_vocabulary_list] for vector in training_set]
    # adding bach the classification in the first space
    feature_vectors.insert(0,training_data_classifications)
    return feature_vectors

feature_vectors = get_feature_vectors(training_set)
```

Function 3: trans the data set to feature vectors



Name	Type	Size	Value
feature_vectors	list	4001	[['1', '1', '0', '0', '0', ...], [1, 1, 1, 1, 1, ...], [0, 0, 1, 1, 0, ...
final_vocabulary_list	list	2376	['public', 'announc', 'the', 'new', 'domain', 'name', 'ar', 'final', ' ...
training_data_classifications	list	4000	['1', '1', '0', '0', '0', '0', '1', '0', '0', '0', ...]
training_set	list	4000	[['public', 'announc', 'the', 'new', 'domain', ...], ['have', 'tax', ' ...
validation_data_classifications	list	1000	['0', '0', '1', '1', '0', '0', '0', '0', '1', '0', ...]
validation_set	list	1000	[['onc', 'upon', 'a', 'time', 'yen', ...], ['i', 'receiv', 'a', 'spam' ...

Figure 3: question two with feature vectors

Secondly, after we have the *final_vocabulary_list*, we can trans the *training_set* into the *feature_vectors* which we use further. For each words in *training_set*, if the words in *final_vocabulary_list*, then the same position in *feature_vectors* will equal to 1, else equal to 0. And the function will return the *feature_vectors*.

question three

Implement the functions perceptron train(data) and perceptron test(w, data). The function perceptron train(data) trains a perceptron classifier using the examples provided to the function, and should return w , k, and iter, the final classification vector, the number of updates (mistakes) performed, and the number of passes through the data, For the corner case of w ·x = 0, predict the +1 (spam) class. For this exercise, you do not need to add a bias feature to the feature vector (it turns out not to improve classification accuracy, possibly because a frequently occurring word already serves this purpose).

```
"""
question three
Implement the functions perceptron train(data) and perceptron test(w, data).
The function perceptron train(data) trains a perceptron classifier using the examples provided to the
    ↪ function,
For the corner case of wx = 0, predict the +1 (spam) class.
The function perceptron test(w, data) should take as input the weight vector w
 (the classification vector to be used) and a set of examples.
return :
    w: the final classification vector, theta
    k: the number of updates (mistakes) performed
    iter: the number of passes through the data, respectively
"""
def perceptron_train(data,data_classification):
    # seprate the classification from each data for further use
    # and the vector is already delete the first space which is label
    classifications = data_classification
    # change the label from 0 to -1, according to the instructor
    classifications = ['-1' if x=='0' else x for x in classifications]
    #print(classifications)
    # return items
    w = [0]*len(data[0]) #weight
    k = 0 #number of update
    iter = 0 # mistakes
    finish = False # need a flag for the algorithm to stop
    while finish is False:
        finish = True
        # data = [[],[],...,[],[]]
        for t,vector in enumerate(data):
            activation = 0
            activation = np.dot(w,vector)
            if activation * int(classifications[t]) <= 0 and np.sum(vector) > 0 or (activation == 0 and
                ↪ classifications[t] == '-1'):
                for i in range(0,len(vector)):
                    # update the weight
                    w[i] = w[i] + (vector[i]*int(classifications[t]))
                k = k + 1 # mistake count +1
                finish = False # till done equal to true, stop
        iter = iter + 1
    print(iter)
    return w,k,iter

def perceptron_test(w, data,data_classification):
    classifications = data_classification
    prediction_label = []
    count = 1
    for vector in data:
        activation = 0
        for i in range(0,len(vector)):
            activation += w[i]*vector[i]
        if activation >= 0:
            prediction_label.append('1')
        else:
            prediction_label.append('0')
    num = len(classifications)

    combine_label_classifications = zip(prediction_label,classifications)
    for i,j in combine_label_classifications:
        if i == j:
            count += 1
    # count is the number which is classified right
    return (num - count)/num
```

Function 4: question three functions

The goal of percetron algorithm is to minimize the number of classification mistakes. The perceptron algorithm starts with an initial guess $w_1 = 0$, and does the following on receiving example x_i:

1. Predict $sign(w_i \cdot x)$ as the label for example x_i.
2. If incorrect, update $w_{i+1} = w_i + l(x_i)x_i$ else $w_{i+1} = w_i$. $l(x_i)$ is the label of x_i.

Also, the first time I implemented the function follow the step:

1. initial the w with all 0
2. if w >= 0 then the predict_label is 1, else the predict_label = -1
3. compare the predict_label with the data set classification, if they are same, then do not change w, else $w+ = data\_set\_classification[x] \cdot [x]$ for the update

Two ways all work for me, and the results are same. And for the perceptron_test function, I give the input with w, $data$, $data\_classification$. And do the following:

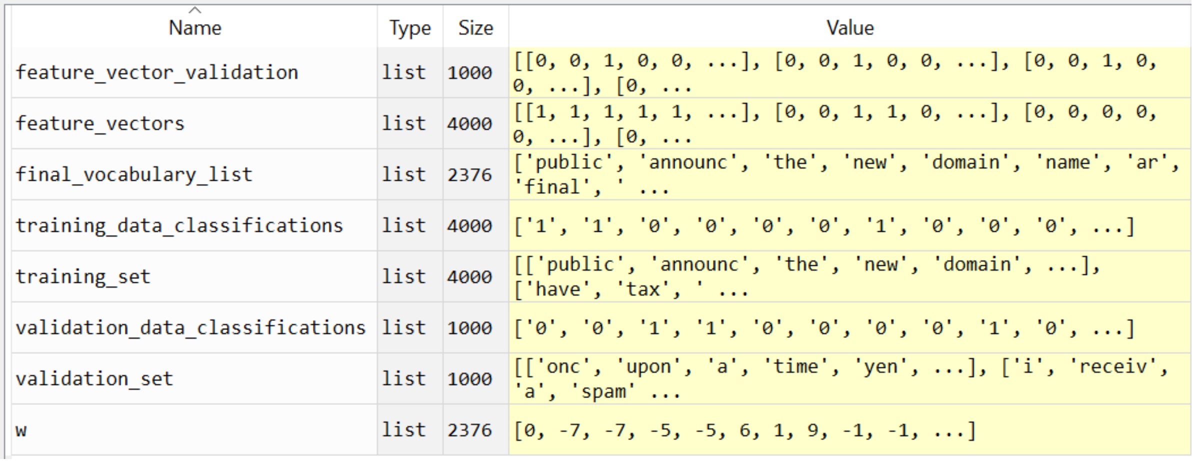
1. initial *prediction_label*
2. for each feature_vector in data
    (a) compute the activation af each, same with the perceptron algorithm, $activation = w[i] \cdot vector[i]$
    (b) is the activation bigger than 0, which add "1" in *prediction_label*, else add "0".

question four

Train the linear classifier using your training set. How many mistakes are made before the algorithm terminates? Test your implementation of perceptron test by running it with the learned parameters and the training data, making sure that the training error is zero. Next, classify the emails in your validation set. What is the validation error?

```

"""
question four
Train the linear classifier using your training set.
Test your implementation of perceptron test by running it with the learned parameters and the training data,
making sure that the training error is zero.
Next, classify the emails in your validation set.
"""

# adding bach the classification in the first space
def train_perceptron(feature_vectors):
    w,k,iter = perceptron_train(feature_vectors,training_data_classifications)
    error = perceptron_test(w,feature_vectors,training_data_classifications)
    print("Mistakes made while training the training data: ",k)
    print("Training error when testing the w and training data: ",error)
    return w

def validation_percetron(w, feature_vectors):
    # manage validation data same with question two
    # using the same w for the validation data
    error = perceptron_test(w,feature_vector_validation,validation_data_classifications)
    print("Validation error with the former w and validation_data_classification: ",error)

feature_vectors.pop(0)
w = train_perceptron(feature_vectors)
feature_vector_validation = []
feature_vector_validation = [[1 if word in vector else 0 for word in final_vocabulary_list] for vector in validation_set]
validation_percetron(w, feature_vector_validation)
```

Function 5: question four



Name	Type	Size	Value
feature_vector_validation	list	1000	[[0, 0, 1, 0, 0, ...], [0, 0, 1, 0, 0, ...], [0, 0, 1, 0, 0, ...], [0, ...
feature_vectors	list	4000	[[1, 1, 1, 1, 1, ...], [0, 0, 1, 1, 0, ...], [0, 0, 0, 0, 0, ...], [0, ...
final_vocabulary_list	list	2376	['public', 'announc', 'the', 'new', 'domain', 'name', 'ar', 'final', ' ...
training_data_classifications	list	4000	['1', '1', '0', '0', '0', '0', '1', '0', '0', '0', ...]
training_set	list	4000	[['public', 'announc', 'the', 'new', 'domain', ...], ['have', 'tax', ' ...
validation_data_classifications	list	1000	['0', '0', '1', '1', '0', '0', '0', '0', '1', '0', ...]
validation_set	list	1000	[['onc', 'upon', 'a', 'time', 'yen', ...], ['i', 'receiv', 'a', 'spam' ...
w	list	2376	[0, -7, -7, -5, -5, 6, 1, 9, -1, -1, ...]

Figure 4: percetron train with training set and validation set

After running the functions, it goes 11 iterations for getting a final result. Which means it runs 44000 feature vectors.
$Mistakes$ made while training the training data: 437.
$Training\ error$ when testing the w and training data: 0.0.
$Validationerror$ with the former w and validation_data_classification: 0.013.

question five

To better understand how the spam classifier works, we can inspect the parameters to see which words the classifier thinks are the most predictive of spam. Using the vocabulary list together with the parameters learned in the previous question, output the 15 words with the most positive weights. What are they? Which 15 words have the most negative weights?

```
"""
question five:
output the 15 words with the most positive weights.
each time get the maximum weights and pop that out the w list
"""
def get_highest_15(w,final_vocabulary_list):
    positive_weights = []
    w_array = np.array(w)
    argsort_w =np.argsort(w_array)
    index= argsort_w[::-1]
    for item in index[:15]:
        positive_weights.append(final_vocabulary_list[item])
    print(positive_weights)

get_highest_15(w,final_vocabulary_list)
```

Function 6: question five function

After training the training_set, we will have the weight. And getting the index of the 15th highest number in the $weight$, backing the search the index in $final\_vocabulary\_list$. The 15 words with the most positive weights are: $['sight','click','market','these','remov','our','deathtospamdeathtospamdeathtospam','most','present','yourself','ever','parti','basenumb','guarante','bodi']$.

question six

Implement the averaged perceptron algorithm, which is the same as your current implementation but which, rather than returning the final weight vector, returns the average of all weight vectors considered during the algorithm (including examples where no mistake was made). Averaging reduces the variance between the different vectors.

```
def average_perceptron_train(data,data_calssification):
    classifications = data_calssification
    classifications = ['-1' if x=='0' else x for x in classifications]

    w = [0]*len(data[0])
    #average_w = []
    k = 0
    iter = 0
    done = False
    cache_w = [0]*len(data[0])
    count = 1
    while not done:
        done = True
        for t,vector in enumerate(data):
            activation = 0
            activation = np.dot(w,vector)

            if activation * int(classifications[t]) <= 0 and np.sum(vector) > 0 or (activation == 0 and classifications[t] == '-1'):
                for i in range(0,len(vector)):
                    # update the weight
                    w[i] = w[i] + (vector[i]*int(classifications[t]))
                    cache_w[i] = cache_w[i] + count*(vector[i]*int(classifications[t]))
                k = k + 1
                done = False
            count += 1
        #average_w.append(w)
        iter = iter + 1
        cache_w = np.array(cache_w)
        average_change = np.array(w) - (1/count)*cache_w
    return list(average_change),k,iter

#another way to implement the average perceptron
def average_perceptron_train_try(data,data_calssification):
    classifications = data_calssification
    classifications = ['-1' if x=='0' else x for x in classifications]
    w = [0]*len(data[0])
    k = 0
    iter = 0
    done = False
    cache_w = [0]*len(data[0])
    count = 1
    while not done:
        done = True
        for t,vector in enumerate(data):
            activation = 0
            activation = np.dot(w,vector)
            if activation * int(classifications[t]) <= 0 and np.sum(vector) > 0 or (activation == 0 and
                ↪ classifications[t] == '−1'):
                for i in range(0,len(vector)):
                    w[i] = w[i] + (vector[i]*int(classifications[t]))
                k = k + 1
                done = False
            count += 1
            cache_w = np.array(cache_w)
        iter = iter + 1
        cache_w += np.array(w)
        average_change = cache_w/(iter*len(data))
    return list(average_change),k,iter

def train_average_perceptron(feature_vectors,training_data_classifications,feature_vector_validation,
    ↪ validation_data_classifications):
    w,k,iter = average_perceptron_train(feature_vectors,training_data_classifications)
    error_average_train = perceptron_test(w,feature_vectors,training_data_classifications)
    print("Mistakes made while training the training data with the average perceptron algoright:", k)
    print("Training error when teating the w and training data:", error_average_train)
    print("the number passes throught:", iter)
    error_average_validate = perceptron_test(w,feature_vector_validation,
        ↪ validation_data_classifications)
    print("Validation error with the former w and validation_data_classification ( average percetron): ",
        ↪ error_average_validate)

train_average_perceptron(feature_vectors,training_data_classifications,feature_vector_validation,
    ↪ validation_data_classifications)
```

Function 7: question six function

To average the weights, we need to store the *weights* we get from each iteration, and then compute the average of them. I do the following:

1. initial *cache_w* with all 0
2. at each time when encounter with error, add the *weight* in the *cache_w*

Then divide them with the error numbers. Also, I use another way to implement the average perceptron, due to difference reference, there are some different to implement. You can see in the third function, I just add the weight after each iteration, and get the average of the $cache_{w}$. But the results are the same.
$Mistakes$ made while training the training data with the average perceptron algoright: 437.
$Trainingerror$ when testing the w and training data: 0.0 the number passes throught: 11.
$Validationerror$ with the former w and validation_data_classification (average percetron): 0.013

question seven

Add an argument to both the perceptron and the averaged perceptron that controls the maximum number of passes over the data. This is an important hyperparameter because for large training sets, the perceptron algorithm can take many iterations just changing a small subset of the point – leading to overfitting.

```
"""
question severn
Add an argument to both the perceptron and the averaged perceptron
that controls the maximum number of passes over the data.
This is an important hyperparameter because for large training sets,
the perceptron algorithm can take many iterations just changing a small subset of the point --
leading to overfitting.
"""
def perceptron_train_with_argument(data,data_classification,max_iterations):
    # seprate the classification from each data for further use
    # and the vector is already delete the first space which is label
    classifications = data_classification
    # change the label from 0 to -1, according to the instructor
    classifications = ['-1' if x=='0' else x for x in classifications]
    #print(classifications)
    # return items
    w = [0]*len(data[0]) #weight
    k = 0 #number of mistakes
    iter = 0 #update
    # run 10 rounds and whole 40000 passes
    while iter < max_iterations:
    # data = [[],[],...,[],[]]
        for t,vector in enumerate(data):
            activation = 0
            activation = np.dot(w,vector)
            if activation * int(classifications[t]) <= 0 and np.sum(vector) > 0 or (activation == 0 and classifications[t] == '-1'):
                for i in range(0,len(vector)):
                    # update the weight
                    w[i] = w[i] + (vector[i]*int(classifications[t]))
                k = k + 1 # mistake count +1
        iter = iter + 1
    return w,k,iter

def perceptron_train_averaged_with_argument(data,data_classification,max_iterations):
    classifications = data_classification
    classifications = ['-1' if x=='0' else x for x in classifications]

    w = [0]*len(data[0])
    k = 0
    iter = 0
    cache_w = [0]*len(data[0])
    count = 1
    while iter < max_iterations:
        for t,vector in enumerate(data):
            activation = 0
            activation = np.dot(w,vector)
            if activation * int(classifications[t]) <= 0 and np.sum(vector) > 0 or (activation == 0 and
                ↪ classifications[t] == '-1'):
                for i in range(0,len(vector)):
                    # update the weight
                    w[i] = w[i] + (vector[i]*int(classifications[t]))
                    cache_w[i] = cache_w[i] + count*(vector[i]*int(classifications[t]))
                k = k + 1
            count += 1
        #average_w.append(w)
        iter = iter + 1
        cache_w = np.array(cache_w)
        average_change = np.array(w) - (1/count)*cache_w
    return list(average_change),k,iter

# another way to implement the average perceptron with the argument
def perceptron_train_averaged_with_argument_try(data,data_classification,max_iterations):
    classifications = data_classification
    classifications = ['-1' if x=='0' else x for x in classifications]
    w = [0]*len(data[0])
    k = 0
    iter = 0
    cache_w = [0]*len(data[0])
    count = 1
    while iter < max_iterations:
        for t,vector in enumerate(data):
            activation = 0
            activation = np.dot(w,vector)
            if activation * int(classifications[t]) <= 0 and np.sum(vector) > 0 or (activation == 0 and
                ↪ classifications[t] == '-1'):
                for i in range(0,len(vector)):
                    w[i] = w[i] + (vector[i]*int(classifications[t]))
                k = k + 1
            count += 1
            cache_w = np.array(cache_w)
            cache_w += np.array(w)
        #average_w.append(w)
        iter = iter + 1
    average_change = cache_w/(iter*len(data))
    return list(average_change),k,iter
```

Function 8: question seven function

I added one parameter *max_iteration*in the function which is used before, and change the loop with *while iter* ≪ *max_iteration*.

question eight

Experiment with various maximum iterations on the two algorithms checking performance on the validation set.

```
"""
question eight:
Experiment with various maximum iterations on the two algorithms checking performance on the validation set.
Optionally you can try to change X from question 2. Report the best validation error for the two algorithms
"""
def train_with_argument(feature_vectors,training_data_classifications,feature_vector_validation,validation_data_classifications):
    for i in range(1,12):
        # adding bach the classification in the first space
        w,k,iter = perceptron_train_with_argument(feature_vectors,training_data_classifications,i)
        error_train = perceptron_test(w,feature_vectors,training_data_classifications)
        print("number passes iteration",iter)
        print("error from training set:", error_train)
        # using the same w for the validation data
        error_validation = perceptron_test(w,feature_vector_validation,validation_data_classifications)
        print("Mistakes made while training the training data with the perceptron algorighm:", k)
        print("Validation error with the former w and validation_data_classification: ",error_validation)
        print("---")
        w,k,iter = perceptron_train_averaged_with_argument(feature_vectors,training_data_classifications,i)
        error_train_average = perceptron_test(w,feature_vectors, training_data_classifications)
        print("Validation error with the former w and validation_data_classification: ",error_train_average)
        # using the same w for the validation data
        error_validation_average = perceptron_test(w,feature_vector_validation,validation_data_classifications)
        print("Mistakes made while training the training data with the perceptron algorighm:", k)
        print("Validation error with the former w and validation_data_classification: ",error_validation_average)
        print("------")
        print([error_train,error_validation,error_train_average,error_validation_average])

print("------")
print(math.ceil(len(feature_vectors)/500) - 1)
train_with_argument(feature_vectors,training_data_classifications,feature_vector_validation,validation_data_classifications)
```

Function 9: question eight function

```

error_train = [0]*11
error_validation = [0]*11
error_train_average = [0]*11
```

```
error_validation_average = [0]*11
average_error_perceptron = [0] * 11
average_error_averaged_perceptron = [0] * 11

for i,item in enumerate(error_count_from_one):
    error_train[i] = item[0]
    error_validation[i] = item[1]
    error_train_average[i] = item[2]
    error_validation_average[i] = item[3]
    average_error_perceptron[i] = [(item[0]+item[1])/2]
    average_error_averaged_perceptron[i] = (item[2]+item[3])/2
x = np.arange(1,12,1)
y1 = error_train
y2 = error_validation
y3 = error_train_average
y4 = error_validation_average
y5 = average_error_perceptron
y6 = average_error_averaged_perceptron

plt.plot(x, y1, color = "blue",linestyle="-", marker ="^", label = "train error")
plt.plot(x, y2, color = "orange",linestyle="-", marker = "s", label = "validation error")
plt.plot(x, y3, color = "green",linestyle="-", marker ="^",label = "average train error")
plt.plot(x, y4, color = "red",linestyle="-", marker = "s", label = "average validation error")
plt.plot(x, y5, color = "black",linestyle="-", marker = "*")
plt.plot(x, y6, color = "black",linestyle="-", marker = "+")

plt.legend(loc='upper right')
plt.xlabel("x")
plt.ylabel("error")
plt.show()
```

Function 10: question eight for plotting

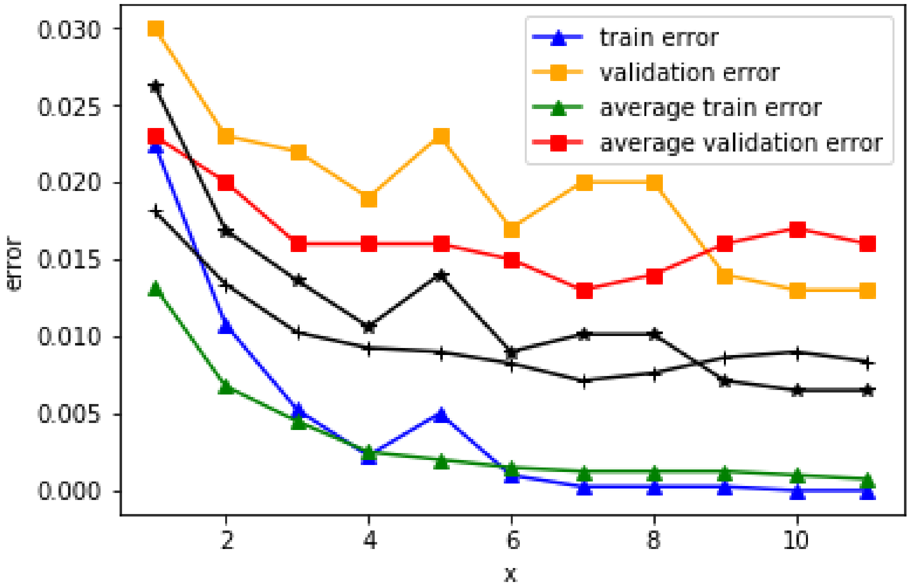


Figure 5: the plot for the error with different iteration

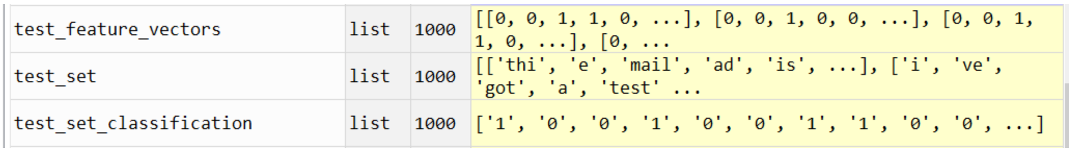
After running the function, we can find the error from ten iteration is the best which is the same with eleven iteration.The red line is average validation error , the orange line is validation error, the green line is average training error, and the blue line is train error.

question nine

Combine the training set and the validation set (i.e. us all of spamtrain.txt) and learn using the best of the configurations previously found. You do not need to rebuild the vocabulary when re-training on the train + validate set. What is the error on the test set.
So we can use 10 iteration for the perceptron algorithm, and 7 iteration for the average perceptron algorithm, which we can easily see in the plot.

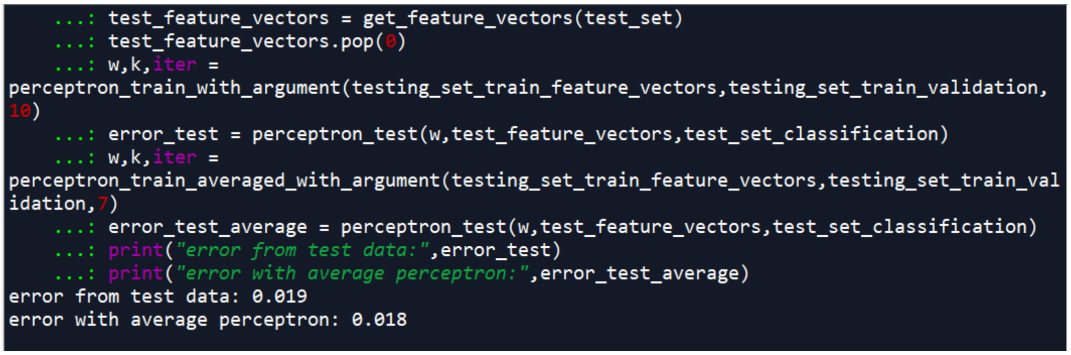
```python
testing_set_train_feature_vectors = feature_vectors + test_feature_vectors
testing_set_train_validation = training_data_classifications + validation_data_classifications

def train_spam_test():
    test_set = []
    test_set_classification = []
    with open("spam_test.txt") as testing_data_file:
        for i,line in enumerate(testing_data_file):
            test_set.append(line.split())
            test_set_classification.append(test_set[i].pop(0))
    return test_set, test_set_classification

test_set, test_set_classification = train_spam_test()
test_feature_vectors = get_feature_vectors(test_set)
test_feature_vectors.pop(0)
w,k,iter = perceptron_train_with_argument(testing_set_train_feature_vectors,
    ↪ testing_set_train_validation,10)
error_test = perceptron_test(w,test_feature_vectors,test_set_classification)
w,k,iter = perceptron_train_averaged_with_argument(testing_set_train_feature_vectors,
    ↪ testing_set_train_validation,7)
error_test_average = perceptron_test(w,test_feature_vectors,test_set_classification)
print("error from test data:",error_test)
print("error with average perceptron:",error_test_average)
```

Function 11: question nine function



test_feature_vectors	list	1000	[[0, 0, 1, 1, 0, ...], [0, 0, 1, 0, 0, ...], [0, 0, 1, 1, 0, ...], [0, ...
test_set	list	1000	[['thi', 'e', 'mail', 'ad', 'is', ...], ['i', 've', 'got', 'a', 'test' ...
test_set_classification	list	1000	['1', '0', '0', '1', '0', '0', '1', '1', '0', '0', ...]

Figure 6: test set

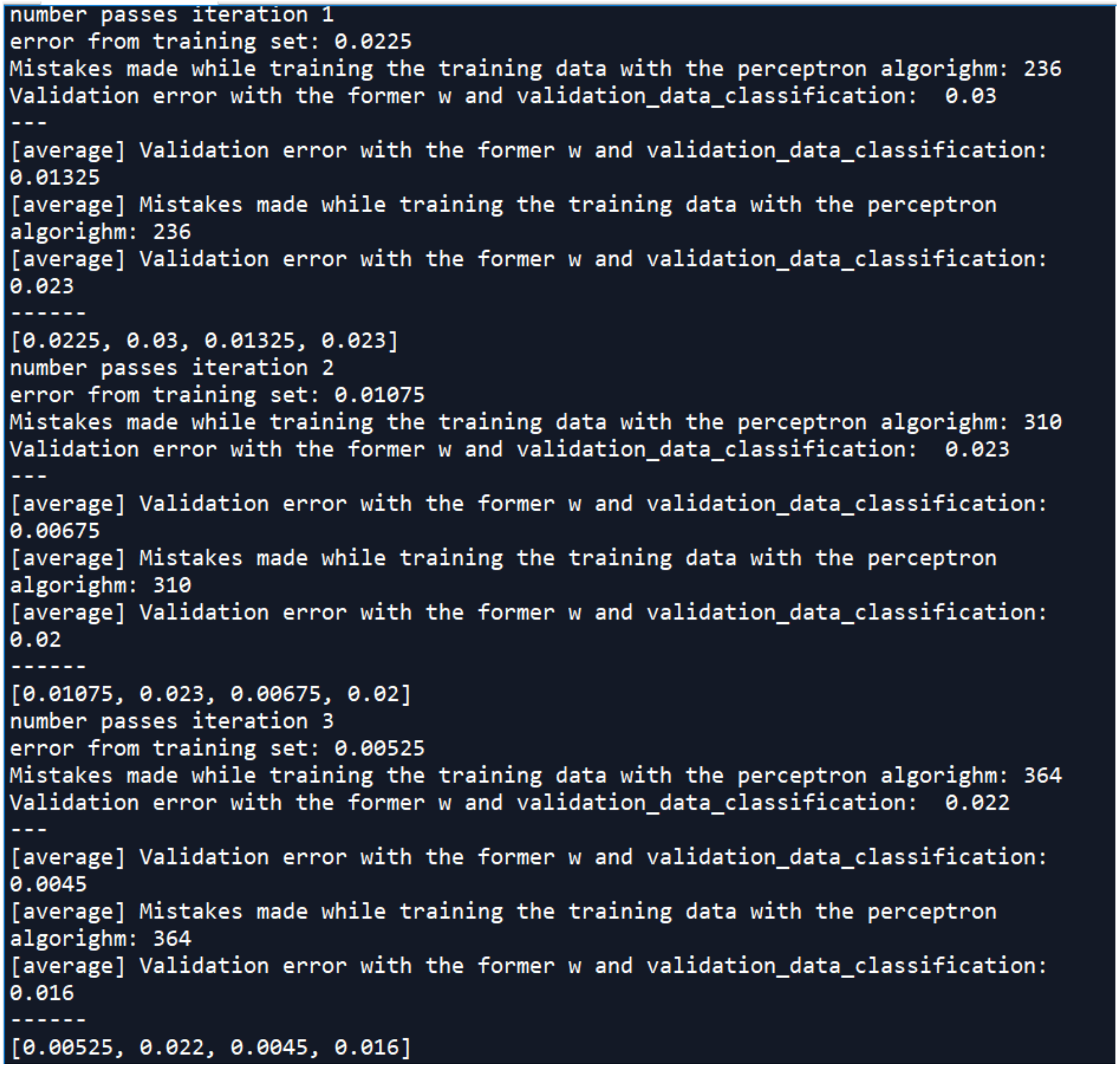


```
    ...: test_feature_vectors = get_feature_vectors(test_set)
    ...: test_feature_vectors.pop(0)
    ...: w,k,iter =
perceptron_train_with_argument(testing_set_train_feature_vectors,testing_set_train_validation,
10)
    ...: error_test = perceptron_test(w,test_feature_vectors,test_set_classification)
    ...: w,k,iter =
perceptron_train_averaged_with_argument(testing_set_train_feature_vectors,testing_set_train_val
idation,7)
    ...: error_test_average = perceptron_test(w,test_feature_vectors,test_set_classification)
    ...: print("error from test data:",error_test)
    ...: print("error with average perceptron:",error_test_average)
error from test data: 0.019
error with average perceptron: 0.018
```

Figure 7: question nine: testing set error

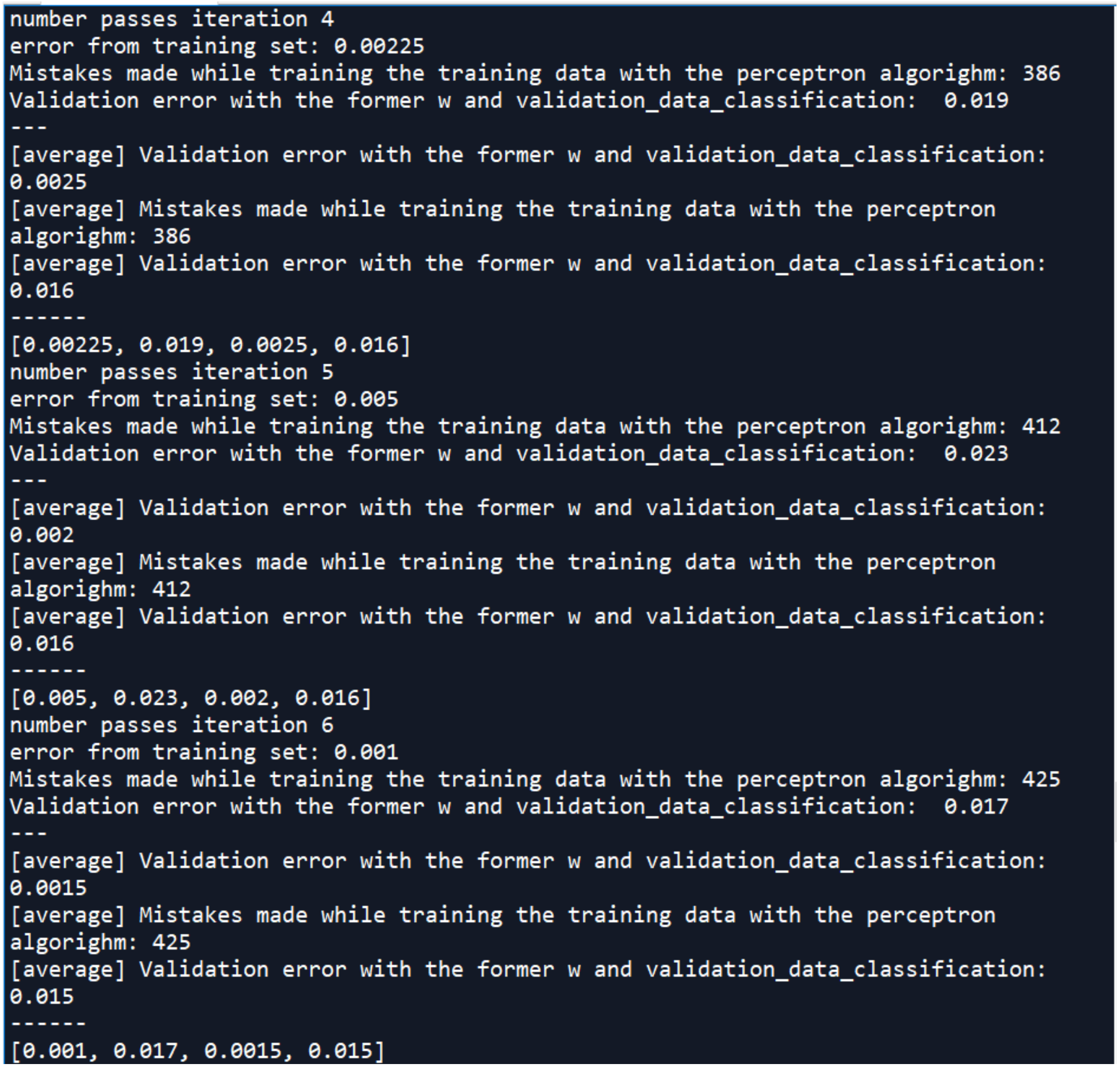
I combine the training set feature vectors and the validation set feature vectors. ALso, combining the classifications of them in order to train the w. The error from test data: 0.019, and 0.018 with the average perceptron algorithm.

extra detail



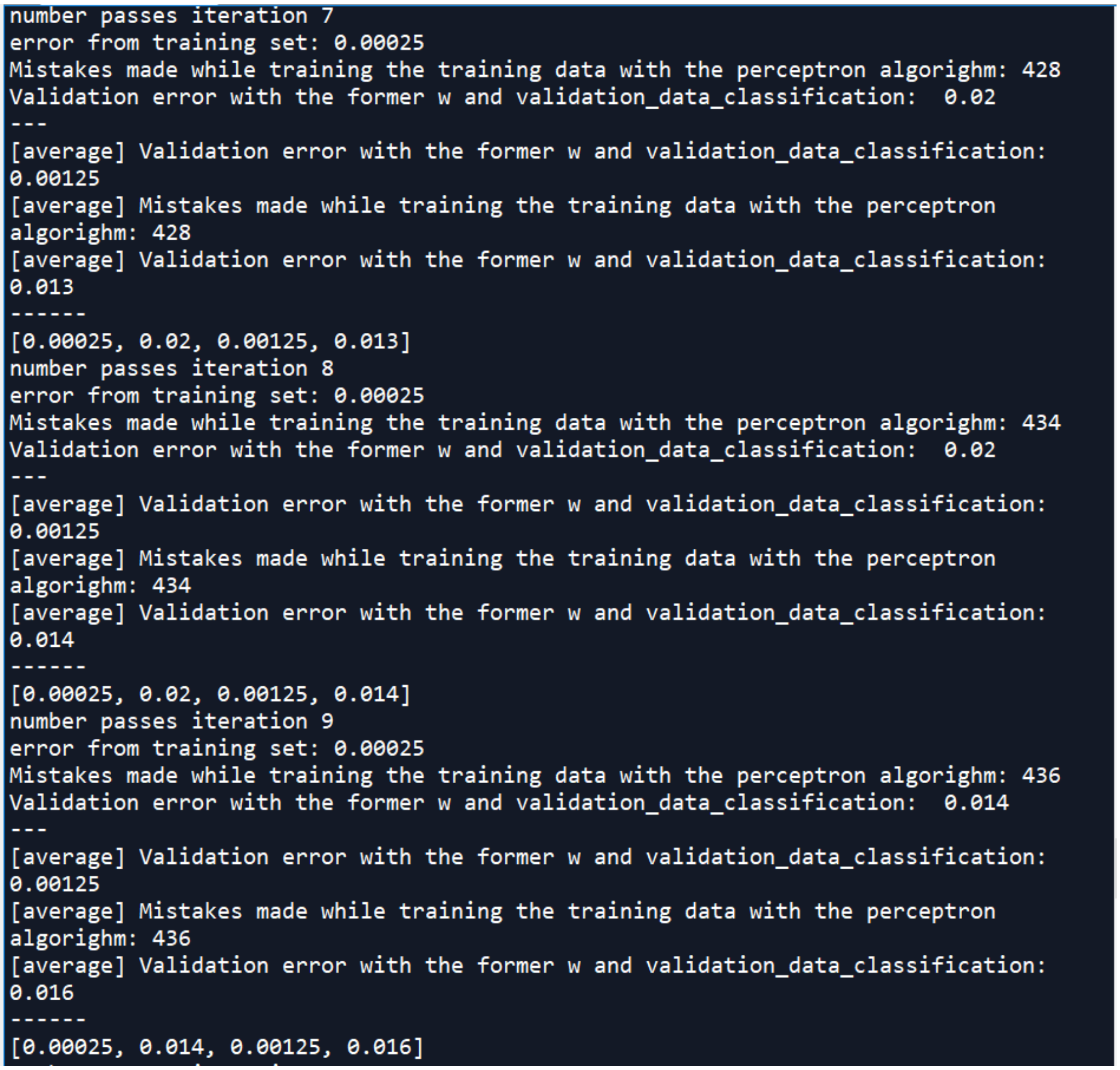
```
number passes iteration 1
error from training set: 0.0225
Mistakes made while training the training data with the perceptron algorighm: 236
Validation error with the former w and validation_data_classification:  0.03
---
[average] Validation error with the former w and validation_data_classification:
0.01325
[average] Mistakes made while training the training data with the perceptron
algorighm: 236
[average] Validation error with the former w and validation_data_classification:
0.023
------
[0.0225, 0.03, 0.01325, 0.023]
number passes iteration 2
error from training set: 0.01075
Mistakes made while training the training data with the perceptron algorighm: 310
Validation error with the former w and validation_data_classification:  0.023
---
[average] Validation error with the former w and validation_data_classification:
0.00675
[average] Mistakes made while training the training data with the perceptron
algorighm: 310
[average] Validation error with the former w and validation_data_classification:
0.02
------
[0.01075, 0.023, 0.00675, 0.02]
number passes iteration 3
error from training set: 0.00525
Mistakes made while training the training data with the perceptron algorighm: 364
Validation error with the former w and validation_data_classification:  0.022
---
[average] Validation error with the former w and validation_data_classification:
0.0045
[average] Mistakes made while training the training data with the perceptron
algorighm: 364
[average] Validation error with the former w and validation_data_classification:
0.016
------
[0.00525, 0.022, 0.0045, 0.016]
```

Figure 8: question four: perceptron train with training set and validation set



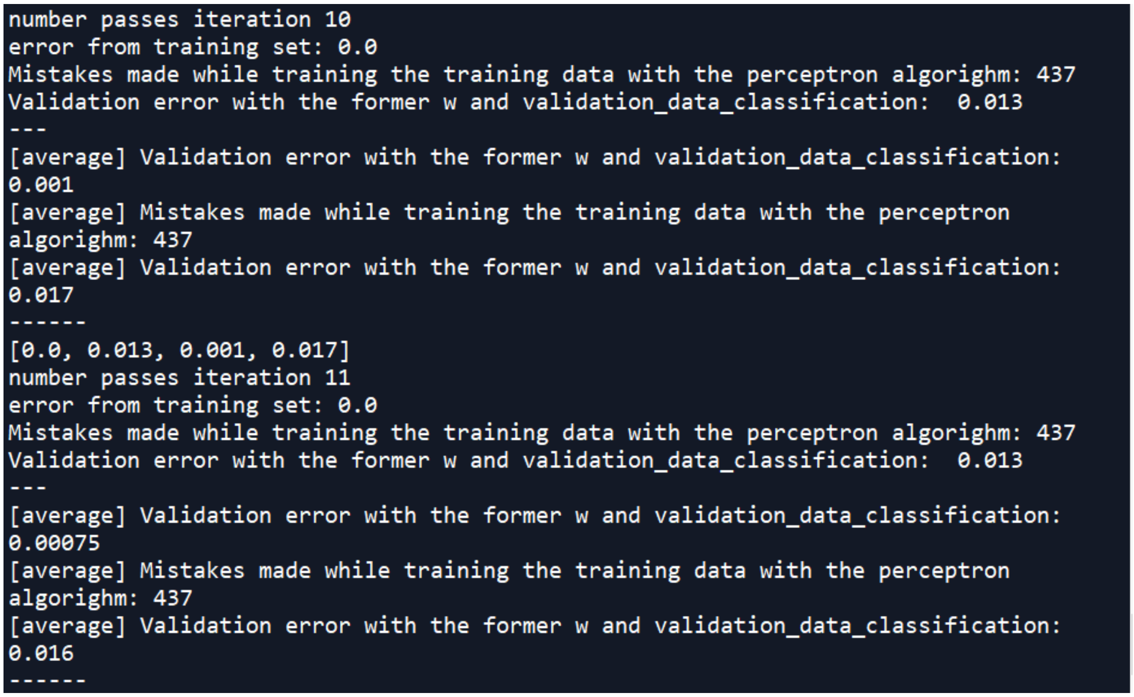
```
number passes iteration 4
error from training set: 0.00225
Mistakes made while training the training data with the perceptron algorighm: 386
Validation error with the former w and validation_data_classification:  0.019
---
[average] Validation error with the former w and validation_data_classification:
0.0025
[average] Mistakes made while training the training data with the perceptron
algorighm: 386
[average] Validation error with the former w and validation_data_classification:
0.016
------
[0.00225, 0.019, 0.0025, 0.016]
number passes iteration 5
error from training set: 0.005
Mistakes made while training the training data with the perceptron algorighm: 412
Validation error with the former w and validation_data_classification:  0.023
---
[average] Validation error with the former w and validation_data_classification:
0.002
[average] Mistakes made while training the training data with the perceptron
algorighm: 412
[average] Validation error with the former w and validation_data_classification:
0.016
------
[0.005, 0.023, 0.002, 0.016]
number passes iteration 6
error from training set: 0.001
Mistakes made while training the training data with the perceptron algorighm: 425
Validation error with the former w and validation_data_classification:  0.017
---
[average] Validation error with the former w and validation_data_classification:
0.0015
[average] Mistakes made while training the training data with the perceptron
algorighm: 425
[average] Validation error with the former w and validation_data_classification:
0.015
------
[0.001, 0.017, 0.0015, 0.015]
```

Figure 9: question four: perceptron train with training set and validation set



```
number passes iteration 7
error from training set: 0.00025
Mistakes made while training the training data with the perceptron algorighm: 428
Validation error with the former w and validation_data_classification:  0.02
---
[average] Validation error with the former w and validation_data_classification:
0.00125
[average] Mistakes made while training the training data with the perceptron
algorighm: 428
[average] Validation error with the former w and validation_data_classification:
0.013
------
[0.00025, 0.02, 0.00125, 0.013]
number passes iteration 8
error from training set: 0.00025
Mistakes made while training the training data with the perceptron algorighm: 434
Validation error with the former w and validation_data_classification:  0.02
---
[average] Validation error with the former w and validation_data_classification:
0.00125
[average] Mistakes made while training the training data with the perceptron
algorighm: 434
[average] Validation error with the former w and validation_data_classification:
0.014
------
[0.00025, 0.02, 0.00125, 0.014]
number passes iteration 9
error from training set: 0.00025
Mistakes made while training the training data with the perceptron algorighm: 436
Validation error with the former w and validation_data_classification:  0.014
---
[average] Validation error with the former w and validation_data_classification:
0.00125
[average] Mistakes made while training the training data with the perceptron
algorighm: 436
[average] Validation error with the former w and validation_data_classification:
0.016
------
[0.00025, 0.014, 0.00125, 0.016]
```

Figure 10: question four: percetron train with training set and validation set



```
number passes iteration 10
error from training set: 0.0
Mistakes made while training the training data with the perceptron algorighm: 437
Validation error with the former w and validation_data_classification:  0.013
---
[average] Validation error with the former w and validation_data_classification:
0.001
[average] Mistakes made while training the training data with the perceptron
algorighm: 437
[average] Validation error with the former w and validation_data_classification:
0.017
------
[0.0, 0.013, 0.001, 0.017]
number passes iteration 11
error from training set: 0.0
Mistakes made while training the training data with the perceptron algorighm: 437
Validation error with the former w and validation_data_classification:  0.013
---
[average] Validation error with the former w and validation_data_classification:
0.00075
[average] Mistakes made while training the training data with the perceptron
algorighm: 437
[average] Validation error with the former w and validation_data_classification:
0.016
------
```

Figure 11: question four: percetron train with training set and validation set