Xinmeng Li xl1575 HW1

In [1]:
```
import numpy as np
from collections import defaultdict
```

In [2]:
```
f = open("/home/jovyan/shared/spam_train.txt",'r')
y_ori = []
x_ori = []
for line in f:
    y = int(line[0])
    if y==0:
        y=-1
    y_ori.append(y)
    x_ori.append(line[2:])
```

1.

In [3]:
```
x_train = x_ori[:len(x_ori)-1000]
y_train = y_ori[:len(y_ori)-1000]
x_val = x_ori[len(x_ori)-1000:]
y_val = y_ori[len(y_ori)-1000:]
print(len(x_train),len(y_train),len(x_val),len(y_val))
```

```
4000 4000 1000 1000
```

If we do not use validation set, the final model is likely to be overfitting and cannot generalizes on the test dataset, which causes a high training accuracy but low testing accuracy.

2.

In [4]:
```
def vocab(x_train):
    words = defaultdict()
    for l in x_train:
        s = l.split()
        for i in range(0,len(s)):
            w = s[i]
            if w not in s[:i]:
                if w in words:
                    words[w] = words[w]+1
                else:
                    words[w] = 1
    return words
```

In [5]:
```
words = vocab(x_train)
words_30 = [w for w in words if words[w]>=30]
```

In [6]:
```
def getFeature(x_train, words_30):
    feature = []
    for i in range(len(x_train)):
        feature.append([1 if w in x_train[i].split() else 0 for w in words_30])
    feature = np.array(feature)
    return feature
```

In [7]:
```
feature =  getFeature(x_train, words_30)
feature.shape
```

Out[7]:
```
(4000, 2376)
```

3.

In [8]:
```
def perceptron_train(data,max_iter,lr):
    w = [0]*len(data[0][0])
    loops = 0
    nummis = 0
    zero = w
    for i in range(0,max_iter):
        loops = loops+1
        mistake = 0
        for d in data:
            yhat = np.dot(w,d[0])
            # If there is no vocabulary in an email, then it lays on the decision boundary.
            if list(d[0]) == zero:
                continue
            elif d[1]*yhat<=0:
                w = w+d[1]*d[0]
                mistake = mistake+1
        if mistake==0:
            loops = loops-1
            break
        nummis = nummis+mistake
    return w, nummis, loops
```

In [9]:
```
def perceptron_test(w, data):
    errs = 0
    zero = [0]*len(data[0][0])
    n = 0
    for d in data:
        yhat = np.dot(w,d[0])
        if list(d[0]) == zero:
            if d[1] == -1:
                errs=errs+1
            continue
        if d[1]*yhat<=0:
            errs = errs +1
    return float(errs)/float(len(data))
```

4.

In [10]:
```
def getData(y_train,feature):
    data = []
    for i in range(0,len(y_train)):
        data.append((feature[i],y_train[i]))
    return data
```

In [11]:
```
data = getData(y_train,feature)
```

In [12]:
```
w, mistakes, iters = perceptron_train(data,10000,1)
print("There are",mistakes,"mistakes and",iters,"iterations.")
```

```
There are 437 mistakes and 10 iterations.
```

In [13]:
```
trainerr = perceptron_test(w, data)
print("The training error is",trainerr)
```

```
The training error is 0.0
```

In [14]:
```
def getVal(x_val,y_val,words_30):
    fval = []
    for i in range(0,len(x_val)):
        e = x_val[i].split()
        fval.append(([1 if w in e else 0 for w in words_30],y_val[i]))
    return fval
fval = getVal(x_val,y_val,words_30)
```

In [15]:
```
valerr = perceptron_test(w, fval)
print("The validation error is",valerr)
```

```
The validation error is 0.013
```

5.

In [16]:
```
weighted = sorted([(v,i) for i,v in enumerate(w)],reverse = True)
weighted_n = sorted([(v,i) for i,v in enumerate(w)])
weighted_words = []
weighted_words_n = []
for v,i in weighted:
    if len(weighted_words) == 15:
        break
    weighted_words.append((words_30[i],v))
for v,i in weighted_n:
    if len(weighted_words_n) == 15:
        break
    weighted_words_n.append((words_30[i],v))
print("15 words with the most positive weights are",weighted_words)
print("15 words with the most negative weights are",weighted_words_n)
```

```
15 words with the most positive weights are [('sight', 22), ('click', 18), ('market', 16), ('remov', 16), ('t
hese', 16), ('our', 15), ('deathtospamdeathtospamdeathtospam', 14), ('most', 13), ('yourself', 12), ('parti',
12), ('ever', 12), ('present', 12), ('basenumb', 11), ('guarante', 11), ('bodi', 11)]
15 words with the most negative weights are [('wrote', -16), ('reserv', -15), ('prefer', -14), ('copyright',
-13), ('i', -12), ('still', -12), ('technolog', -12), ('but', -11), ('comput', -11), ('which', -11), ('somet
h', -11), ('recipi', -11), ('url', -10), ('date', -10), ('coupl', -10)]
```

6.

In [17]:
```
def avg(w_list):
    w_list = np.array(w_list)
    average = [0]*len(w_list[0])
    for w in w_list:
        average = average+w
    average = average/len(w_list)
    return average
```

In [18]:
```
def average_perceptron_train(data,max_iter,lr):
    w = [0]*len(data[0][0])
    w_list = []
    loops = 0
    nummis = 0
    zero = w
    for i in range(0,max_iter):
        loops = loops+1
        mistake = 0
        for d in data:
            w_list.append(w)
            yhat = np.dot(w,d[0])
            # If there is no vocabulary in an email, then it lays on the decision boundary.
            if list(d[0]) == zero:
                continue
            elif d[1]*yhat<=0:
                w = w+d[1]*d[0]
                mistake = mistake+1
        if mistake==0:
            loops=loops-1
            break
        nummis = nummis+mistake
    return avg(w_list), nummis, loops
```

In [19]:
```
w, mistakes, iters = average_perceptron_train(data,10000,1)
print("There are",mistakes,"mistakes and",iters,"iterations.")
```

```
There are 437 mistakes and 10 iterations.
```

In [20]:
```
valerr = perceptron_test(w, fval)
print("The validation error is",valerr)
```

```
The validation error is 0.016
```

In [21]:
```
trainerr = perceptron_test(w, data)
print("The training error is",trainerr)
```

```
The training error is 0.00075
```

7.

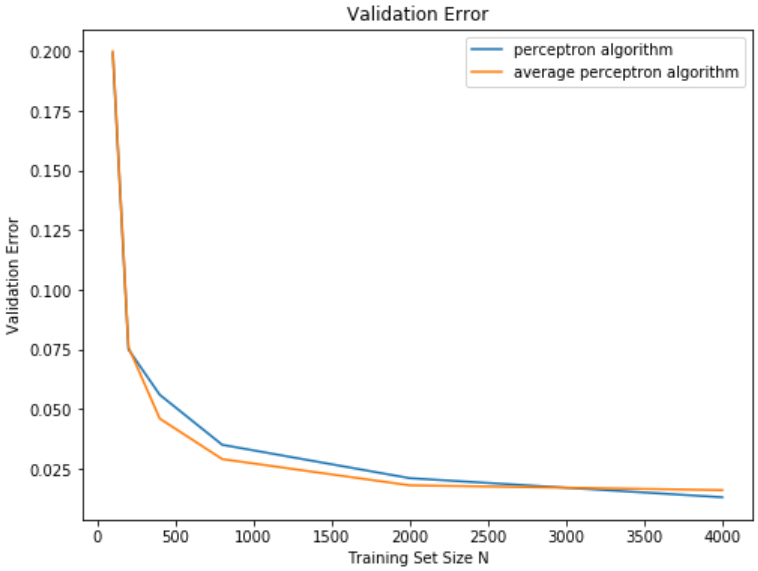
In [22]:
```
def evaluate(train,validation,max_iter,lr):
    w1, mistakes1, iters1 = perceptron_train(train,max_iter,lr)
    valerr1 = perceptron_test(w1, validation)
    w2, mistakes2, iters2 = average_perceptron_train(train,max_iter,lr)
    valerr2 = perceptron_test(w2, validation)
    return [valerr1,valerr2, mistakes1,mistakes2,iters1,iters2]
```

In [23]:
```
import matplotlib.pyplot as plt
N = [100, 200, 400, 800, 2000, 4000]
result = []
for n in N:
    words = vocab(x_train[:n])
    words_30 = [w for w in words if words[w]>=30]
    feature =  getFeature(x_train[:n], words_30)
    print(feature.shape)
    d = getData(y_train[:n],feature)
    result.append(evaluate(d,getVal(x_val,y_val,words_30),10000,1))
```

```
(100, 50)
(200, 129)
(400, 297)
(800, 658)
(2000, 1409)
(4000, 2376)
```

Obviously, as the size of the training set increases, the size of the vocabulary list increases.

In [24]:
```
plt.figure(figsize = (8,6))
plt.plot(N,[r[0] for r in result],label = 'perceptron algorithm')
plt.plot(N,[r[1] for r in result],label = 'average perceptron algorithm')
plt.xlabel("Training Set Size N")
plt.ylabel("Validation Error")
plt.title('Validation Error')
plt.legend()
plt.show()
```

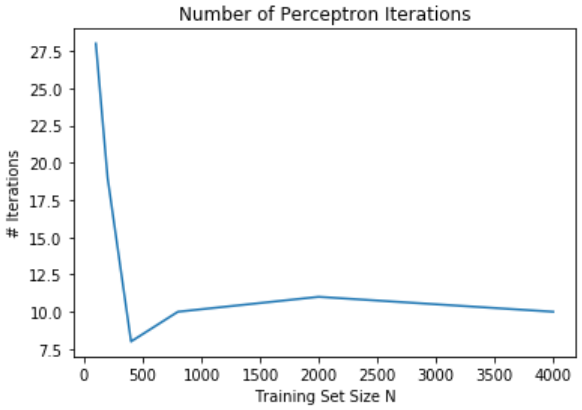


In [25]:
```
print("About Perceptron Algorithm:")
print("validation error",[r[0] for r in result],", mistakes",[r[2] for r in result],", # iterations",[r[4] for r in result])
print("About Average Perceptron Algorithm:")
print("validation error",[r[1] for r in result],", mistakes",[r[3] for r in result],", # iterations",[r[5] for r in result])
```

```
About Perceptron Algorithm:
validation error [0.199, 0.075, 0.056, 0.035, 0.021, 0.013] , mistakes [174, 133, 136, 150, 288, 437] , # iterations [28, 19, 8, 10, 11, 10]
About Average Perceptron Algorithm:
validation error [0.2, 0.076, 0.046, 0.029, 0.018, 0.016] , mistakes [174, 133, 136, 150, 288, 437] , # iterations [28, 19, 8, 10, 11, 10]
```

8.

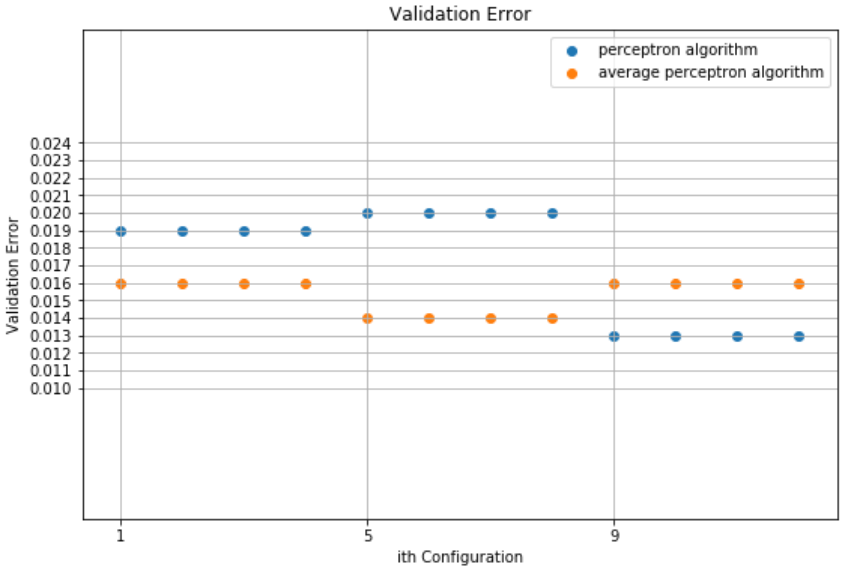
In [26]:
```
plt.plot(N,[r[4] for r in result])
plt.xlabel("Training Set Size N")
plt.ylabel("# Iterations")
plt.title("Number of Perceptron Iterations")
plt.show()
```



9.

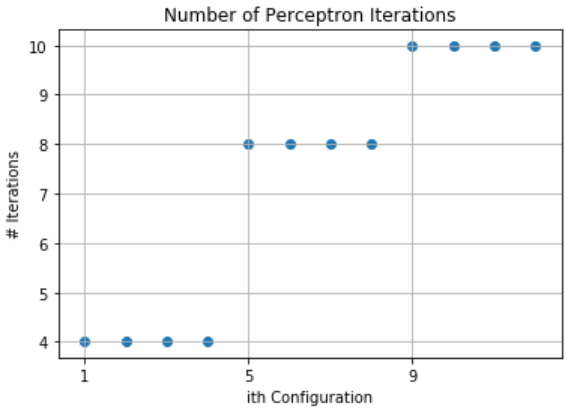
In [27]:
```
lr = [1e-2,1e-1,1e0,1e1]
compare = []
conf = []
for m in range(2,5):
    mi = 2**m
    for l in lr:
        conf.append((mi,l))
        compare.append(evaluate(data,fval,mi,l))
```

In [47]:
```
fig = plt.figure(figsize = (9,6))
plt.scatter(np.arange(1,len(conf)+1),[r[0] for r in compare],label = 'perceptron algorithm')
plt.scatter(np.arange(1,len(conf)+1),[r[1] for r in compare],label = 'average perceptron algorithm')
ax = fig.gca()
ax.set_xticks(np.arange(1,len(conf)+1,len(lr)))
ax.set_yticks(np.arange(0.01,0.025,0.001))
plt.xlabel("ith Configuration")
plt.ylabel("Validation Error")
plt.title('Validation Error')
plt.grid()
plt.legend()
plt.show()
```



We observe that the average perceptron algorithm is more stable than the perceptron algorithm. The value of learning rate does not have much influence on the validation error.

In [30]:
```
fig = plt.figure()
ax = fig.gca()
ax.set_xticks(np.arange(1,len(conf)+1,len(lr)))
plt.scatter(np.arange(1,len(conf)+1),[r[4] for r in compare])
plt.xlabel("ith Configuration")
plt.ylabel("# Iterations")
plt.title("Number of Perceptron Iterations")
plt.grid()
plt.show()
```

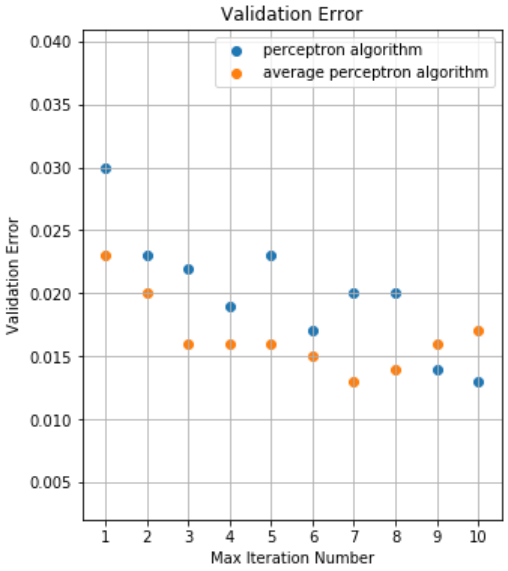


We observe that the iteration numbers stays at 10 when max iteration number is 16, which implies that the model converges after 10 iterations no matter what the learning rate is.

Since the learning rate is not important, we set lr = 1 and further explore the performance with all possible max iteration numbers.

In [43]:
```
finalre1 = []
finalre2 = []
for m in range(1,11):
    w, mistakes, iters = perceptron_train(data,m,1)
    finalre1.append(perceptron_test(w, fval))
    w, mistakes, iters = average_perceptron_train(data,m,1)
    finalre2.append(perceptron_test(w, fval))
```

In [52]:
```
fig = plt.figure(figsize = (5,6))
plt.scatter(np.arange(1,11),finalre1,label = "perceptron algorithm")
plt.scatter(np.arange(1,11),finalre2,label = "average perceptron algorithm")
plt.xlabel("Max Iteration Number")
plt.ylabel("Validation Error")
plt.title('Validation Error')
ax = fig.gca()
ax.set_xticks(np.arange(1,11,1))
plt.legend()
plt.grid()
plt.show()
```



In [53]:
```
print("With max iteration number 10 on perceptron algorithm, we have the lowest validation error",min(finalre1
))
print("When max iteration number 7, the minimum validation error for average perceptron algorithm is",min(fina
lre2))
```

```
With max iteration number 10 on perceptron algorithm, we have the lowest validation error 0.013
When max iteration number 7, the minimum validation error for average perceptron algorithm is 0.013
```

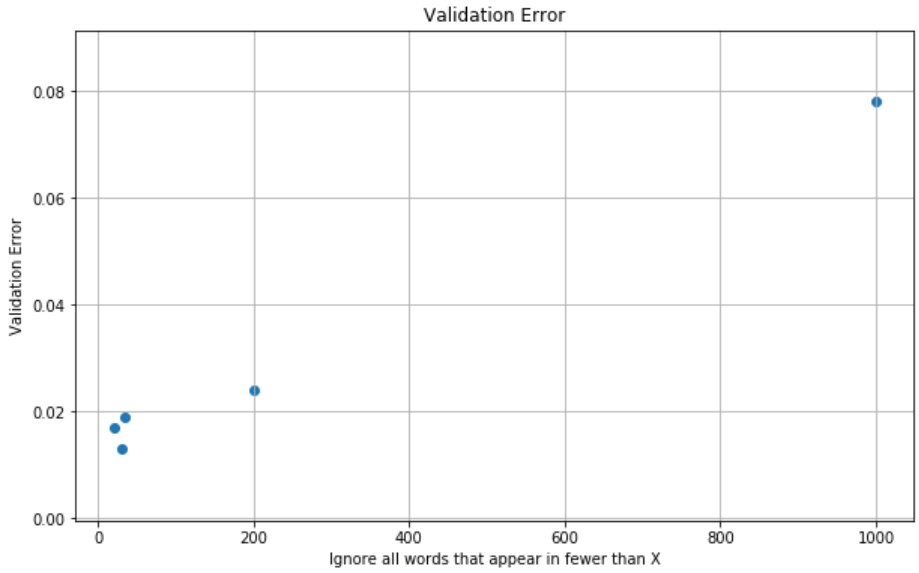
We observe that the average perceptron algorithm performs better than the perceptron algorithm most of the time. Both algorithms have validation error 0.13 as the best result.

10.

In [31]:
```
re = []
X = [20,30,35,200,1000]
words = vocab(x_train)
# Due to the RAM limit, I cannot do more experiments with the values of X. The kernel always dies when I try o
ne more value.
for n in X:
    words_n = [w for w in words if words[w]>=n]
    feature =  getFeature(x_train, words_n)
    print(feature.shape)
    d = getData(y_train,feature)
    w, mistakes, iters = perceptron_train(d,20,1)
    valerr = perceptron_test(w, getVal(x_val,y_val,words_n))
    re.append([valerr,iters])
```

```
(4000, 3115)
(4000, 2376)
(4000, 2148)
(4000, 523)
(4000, 67)
```

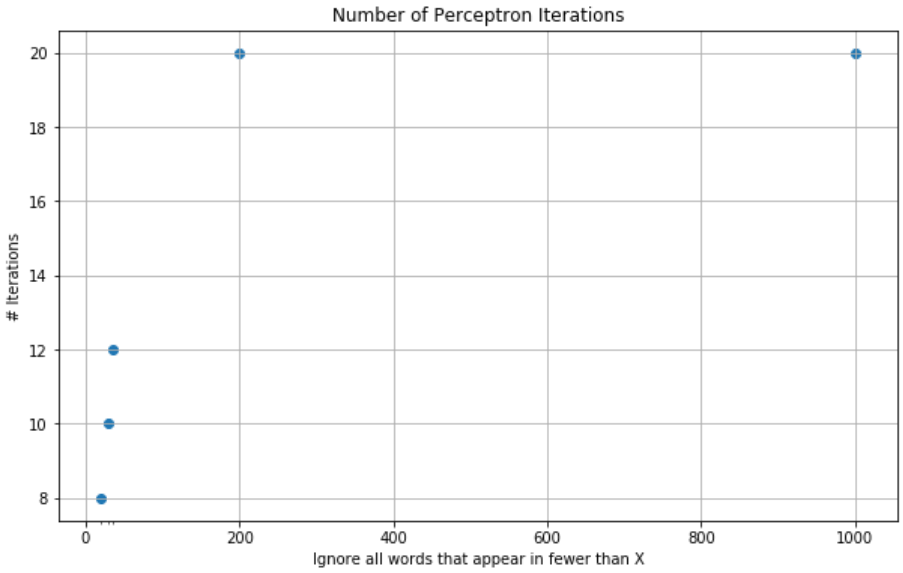
In [32]:
```
fig = plt.figure(figsize = (10,6))
plt.scatter(X,[r[0] for r in re])
plt.xlabel("Ignore all words that appear in fewer than X")
plt.ylabel("Validation Error")
plt.title('Validation Error')
plt.grid()
plt.show()
```



In [33]:
```
print("When X =",30,", validation error",re[1][0],"is the lowest.")
```

```
When X = 30 , validation error 0.013 is the lowest.
```

In [34]:
```
fig = plt.figure(figsize = (10,6))
ax = fig.gca()
ax.set_xticks(X,6)
plt.scatter(X,[r[1] for r in re])
plt.xlabel("Ignore all words that appear in fewer than X")
plt.ylabel("# Iterations")
plt.title("Number of Perceptron Iterations")
plt.grid()
plt.show()
```



In [35]:
```
print("When X =",30,"it takes",re[1][1],"iterations to converge.")
```

```
When X = 30 it takes 10 iterations to converge.
```

We observe that as X increases, the iteration number tends to increase.

11.

In [39]:
```
f = open("/home/jovyan/shared/spam_test.txt",'r')
y_test = []
x_test = []
for line in f:
    y = int(line[0])
    if y==0:
        y=-1
    y_test.append(y)
    x_test.append(line[2:])
```

In [40]:
```
words = vocab(x_ori)
words_n = [w for w in words if words[w]>=30]
feature_30 =  getFeature(x_ori, words_n)
print(feature_30.shape)
d = getData(y_ori,feature_30)
testdata = getVal(x_test,y_test,words_n)
```

```
(5000, 2769)
```

Since both of these two algorithms reach validation error = 0.13, we want to try both on the test set and compare their performances. Let's try the perceptron algorithm optimal configuration (X=30, max_iter = 10) first.

In [54]:
```
w, mistakes, iters = perceptron_train(d,10,1)
testerr = perceptron_test(w, testdata)
print("K:",mistakes,", # Iterations:",iters,", Test Error:",testerr)
```

```
K: 511 , # Iterations: 10 , Test Error: 0.021
```

Since the average perceptron algorithm avoids overfitting and generally performs slightly better than the perceptron algorithm in the previous experiments, we expect the average perceptron to perform better on the test set. Let's take a look at the performance with the average perceptron algorithm under the optimal hyperparameter setting (X=30, max_iter = 7).

In [55]:
```
w, mistakes, iters = average_perceptron_train(d,7,1)
testerr = perceptron_test(w, testdata)
print("K:",mistakes,", # Iterations:",iters,", Test Error:",testerr)
```

```
K: 477 , # Iterations: 7 , Test Error: 0.017
```

It turns out that the average perceptron algorithm performs better than the perceptron algorithm on the test dataset.