ESE 545 Project 2

By Yiding Zhang & Xiangyang Cui

We finished Project 2 using Python2.7.
The modules we used and their functions were:

csv: Open csv files of the Sentiment140 dataset.
re: Clean up the data.
time: Record the time consumed each part.
math: Mathematical calculation.
pickle: Store and get variables. So we don't have to run the previous parts again.
numpy: For matrix calculation.
sklearn.feature_extraction.text.CountVectorizer: For Question 1.3. Extract unigram features from the bag of words
matplotlib.**pyplot**: Plot error vs number of iteration.

We have a main function to control the whole process and use pickle module to separate each part.

```
import csv
import re
import pickle
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
import time
import math
import matplotlib.pyplot as plt

def main():
    '''
    This is where the program starts
    '''
    print("Reading training data...")
#    (features, sentiments) =
readCleanData('./trainingandtestdata/training.csv')# FOR TESTING!!!! 20000
entities
    (features, sentiments) =
readCleanData('./trainingandtestdata/training.1600000.processed.noemoticon.cs
v')
    print("Reading complete \n")

    print("Reading testing data...")
    (tst_feat, tst_sent) =
readCleanData('./trainingandtestdata/testdata.manual.2009.06.14.csv')
    print("Reading complete \n")

    print("Creating feature matrix...")
    (feature_matrix, word_bag) = geneFeatureMatrix(features)
    print("Feature matrix created \n")

    print("Creating testing matrix...")
    tst_matrix = geneTestingMatrix(tst_feat, word_bag)
    print("Testing matrix created \n")

    print("Training...")
    train(feature_matrix, sentiments, tst_matrix, tst_sent)


if __name__ == '__main__':
    main()
```

Question 1.1 & Question 1.2:

We processed Question 1.1 and Question 1.2 together since they all need to read every entry in the csv file. We have two functions for this part:

```
def readStopWords():
    with open('stopwords.txt', 'r') as text:
        stopwords = text.read().split()
    return stopwords


def readCleanData(file_name):
    start = time.time()
    stopwords = readStopWords()
    sentiments = []
    features = []
    with open(file_name, 'r') as f:
        reader = csv.reader(f)

        for line in reader:
            feature = []
            sentiments.append(-1 if line[0] == '0' else 1);

            tweet = line[5].lower()
            tweet = re.sub(r'((https?:\/\/)|(www\.))\w+(\.\w+)+(\/\w+)*', '
', tweet)
            tweet = re.sub(r'@\S+', ' ', tweet)
            tweet = re.sub(r'[^a-zA-Z]', ' ', tweet)

            last_word = ''

            for word in tweet.split():
                if word != last_word:
                    if word not in stopwords:
                        feature.append(word)
                        last_word = word
            features.append(feature)

    print "Time = " + str(time.time()-start)
    return (features, np.array(sentiments))
```

The readStopWords() is a function to get all stopwords from stopwords.txt, which we have made some changes from the one provided by professor.

The readCleanData() function is the main method for this part, which read entry one by one from csv file, and for each entry, record its sentiment value in sentiments list and get its feature vector as a list and record it in features list.

For each line in the csv file, we have the following processes:

1. Sentiment in the data file is classified as either 0 (negative emotion) or 4 (positive emotion). With the command:

```
sentiments.append(1 if line[0] == '4' else -1);
```

We can convert these to -1 and 1 respectively.

2. Convert all letters into lowercase.

```
tweet = line[5].lower()
```

3. Replace all occurences of 'www.website' or 'https://website//' with a whitespace.

```
tweet = re.sub(r'((https?:\/\/)|(www\.))\w+(\.\w+)+(\/\w+)*', ' ', tweet)
```

4. Replace all @username with a whitespace.

```
tweet = re.sub(r'@\S+', ' ', tweet)
```

5. Replace all non-ASCII characters with whitespace.

```
tweet = re.sub(r'[^a-zA-Z]', ' ', tweet)
```

6. Split tweet into word list. Remove duplicate words (next to each other). And remove all stopwords read from stopwords.txt.

```
last_word = ''
for word in tweet.split():
    if word != last_word:
        if word not in stopwords:
            feature.append(word)
            last_word = word
features.append(feature)
```

After done all 6 steps, sentiments and feature vectors are saved in sentiments list and features list. And we saved it locally by pickle.dump().

We have made some modifications when we clean the tweet. **And all modifications have been approved by instructors on Piazza**.

1. We didn't ignore text after the first comma. Since We use csv module, we can get tweet as a whole. So we just take the first comma as a common punctuation.
2. We removed urls and usernames directly rather than replace them with 'URL' and 'AT-USER'. Because urls and usernames will be removed eventually.
3. We didn't remove whitespaces explicitly, since we split the tweet by whitespaces.
4. We didn't remove duplicate words which are not next to each other. Since we want to use the features vectors in Question 1.3. This won't affect the bag of words we generated in Question 1.3.
5. In stopwords.txt, we find it doesn't include words like "couldn't", "wasn't", etc. But obviously these words appear a lot. So we added these in our stopwords.txt.

am	hadn	shan	wouldn
aren	hasn	shouldn	ve
couldn	haven	wasn	re

doesn	isn	weren	ll
don	mustn	won	

Take “couldn’t” for example, the punctuation “’” will be replaced by whitespace so it became “couldn” and “t” after split, “t” is a stopword and “couldn” is added as a stopword, so “couldn’t” will be removed eventually.

6. We removed numbers and non-ASCII characters. Since numbers are random and don’t have as much meaning as words. For non-ASCII characters, other students set `decode_error = 'ignore'`, and I just removed them directly.

Question 1.3

```
def geneFeatureMatrix(features):
    start = time.time()

    cv = CountVectorizer()
    features_join = [' '.join(row) for row in features]
    feature_matrix = cv.fit_transform(features_join)
    word_bag = cv.get_feature_names()
    print "Feature matrix shape =", feature_matrix.shape
    print "Time = ", time.time()-start
    return (feature_matrix, word_bag)
```

In this question, we used a module, sklearn.feature_extraction.text.CountVectorizer:

```
from sklearn.feature_extraction.text import CountVectorizer
```

Since CountVector() only read a list of strings as corpus, so we have to modify the features list we got from Question 1.2:

```
features_join = [' '.join(row) for row in features]
```

First we need a CountVectorizer() Object:

```
cv = CountVectorizer()
```

Then we need use this Object to read our corpus:

```
feature_fit = cv.fit_transform(features_join)
```

After this we can get the unigram features and feature vectors:

```
word_bag = cv.get_feature_names()
matrix = feature_fit.toarray()
```

The sentiment attached to these features are the same as Question 1.2.

Question 1.4

We separate this question into two parts: First, Use PEGASOS to train an SVM on the features extracted above. Second, Make a plot of training error vs number of iterations. (We put the plot part in Question 1.5, since we coded most part of Questions 1.4 and 1.5 together.)

1. Use PEGASOS or AdaGrad to train an SVM

As shown in class, The PEGASOS Algorithm is:

INPUT: training set $S = \{(\boldsymbol{x_1}, y_1), \cdots (\boldsymbol{x_n}, y_n), \}$,
Regularization parameter λ,
Number of iteration T
Size of subset B
INITIALIZE: Choose $w_1\ st. \|w_1\| \leq 1/\sqrt{\lambda}$
FOR $t = 1,2, \cdots, T$
Choose $A_t \subseteq S$
$A_t^+ = \{(\boldsymbol{x}, y) \in A_t : y\langle w_t, x\rangle < 1\}$
$\nabla_t = \lambda w_t - \frac{\eta_t}{|A_t|}\sum_{(x,y)\in A_t^+} yx$
$\eta_t = \frac{1}{t\lambda}$
$w_t' = w_t - \eta_t \nabla_t$
$w_{t+1} = \min\{1, \frac{1/\sqrt{\lambda}}{\|w_t'\|}\} w_t'$
OUTPUT: w_{T+1}

The input B is what we added, since as we can see, each iteration, we need Choose $A_t \subseteq S$, the B represents how many data points we choose.

So first, we need define λ (here, we denote it as L), B and T.

```
L = 1E-4
B = 100
T = 1000
```

Then, we need initialize the w_1:

```
(m, n) = feature_matrix.shape
w = np.ones((1, n)) / math.sqrt(L * n)
```

In each iteration, we have:

```
        idx = np.random.randint(n, size=B)
        X = feature_matrix[idx,:].todense()
        y = sentiments[idx]

        Aplus = np.where(y * np.squeeze(np.asarray(X.dot(w.T))) < 1)
        g = y[Aplus] * X[Aplus]
        _w = (1 - Eta * L) * w + (Eta / B) * g
        norm = math.sqrt(L * _w.dot(_w.T))
        w = _w if 1 < norm else _w * norm
        if t % 10 == 0:
            training_error.append(predict(w, X, y))
            testing_error.append(predict(w, tst_matrix, tst_sent))
```

In which, we use *idx = np.random.randint(m, size=B)*, to randomly choose B indices. And use these indices to choose A_t (which is X in code) and find out A_t^+ (which is Aplus in code). η_t is Eta in code.

After several iterations (here we record errors every 10 iterations), we can use the w to predict the feature vectors and see the error between our prediction and the actual sentiments.

```
def predict(w, pred_matrix, pred_sent):
    n = len(pred_sent)
    pred = pred_matrix.dot(w.T)
    diff = (np.sign(pred).T - pred_sent) / 2
    error = diff.dot(diff.T) / n
    return error[0,0]
```

Here, we did part of Question 1.6, since we not only calculated the training errors based on the training dataset, but also calculated the test errors based on the test dataset read from testdata.manual.2009.06.14.csv and cleaned as in Questions 1.2 and 1.3. (In Question 1.6, we explained why we can use the same methods for test dataset directly.)
Here we use two list to record errors:

```
errTrain_P = []
errTst_P = []
```

Question 1.5

AdaGrad is very similar with PEGASOS. The only difference is how they update the w. For PEGASOS, it updates every element of w the same way, which means every dimension has the same learning rate. But AdaGrad update every dimension with different learning rate based on the gradient of each dimension, which can be shown as following:

$$w'_{t+1} = w_t - \eta G^{-1} \nabla f_t(w_t)$$
$$w_{t+1} = arg\min_{w \in S} \|w - w'_{t+1}\|_G = arg\min_{w \in S} \|G(w - w'_{t+1})\|_2^2$$

In which,

$$G = diag(\sum_{s=1}^{t-1} \nabla f_s(w_s)\, \nabla f_s(w_s)^T)^{1/2}$$

Similarly, we can code the update part as following:

```
def AdaGrad(w, X, y, Eta, s):
    '''
    Train a SVM through AdaGrad
    '''
    Aplus = np.where(y * np.squeeze(np.asarray(X.dot(w.T))) < 1)
    grad = -(y[Aplus] * X[Aplus])/B + L*w
    G_inv= 1 / (np.sqrt(s) + epson)
    w -= Eta * np.multiply(G_inv, grad)
    s += np.square(grad)
    return (w, s)
```

The rest part is the same as PEGASOS.
As in PEGASOS, we have two lists to record training errors and test errors:

```
errTrain_A = []
errTst_A = []
```

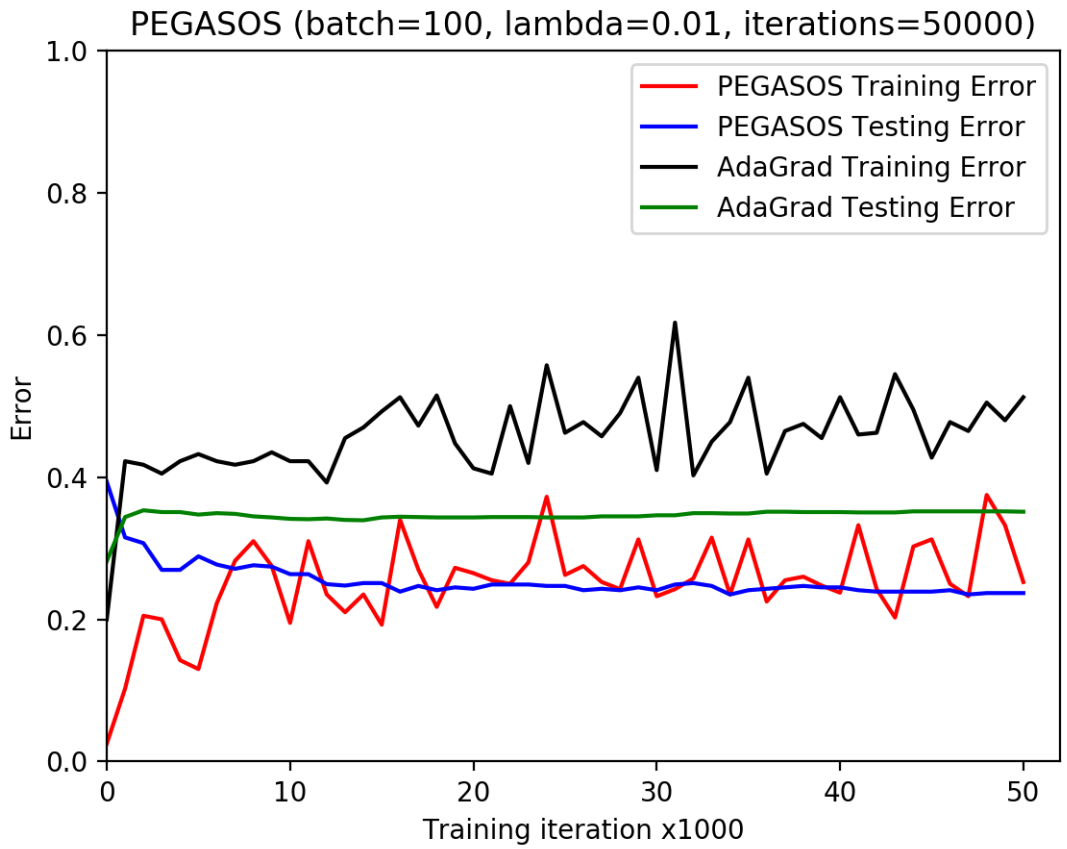
So in total code for Question 1.4 and 1.5 is:

```
def train(feature_matrix, sentiments, tst_matrix, tst_sent):
    '''
    Train SVMs to predict sentiments with given feature words
    '''
    start = time.time()
    (n, d) = feature_matrix.shape
    w_A = np.zeros((1, d))
    w_P = np.ones((1, d)) / math.sqrt(L * d)
    s = np.ones((1,d))
    errTrain_P = []
    errTst_P = []
    errTrain_A = []
    errTst_A = []

    for t in range(T+1):
        idx = np.random.randint(n, size=B)
        X = feature_matrix[idx,:].todense()
        y = sentiments[idx]
        Eta = 1 / (L * (t+1))

        w_P = PEGASOS(w_P, X, y, Eta)
        (w_A, s) = AdaGrad(w_A, X, y, Eta, s)

        if t % 10 == 0:
            errTrain_P.append(predict(w_P, X, y))
            errTst_P.append(predict(w_P, tst_matrix, tst_sent))
            errTrain_A.append(predict(w_A, X, y))
            errTst_A.append(predict(w_A, tst_matrix, tst_sent))
            print t*B

    accuracy_P = 1 - predict(w_P, tst_matrix, tst_sent)
    accuracy_A = 1 - predict(w_A, tst_matrix, tst_sent)
    print "PEGASOS_accuracy = ", accuracy_P
    print "AdaGrad_accuracy = ", accuracy_A
    print "Time = ", time.time()-start
    plotCurves(errTrain_P, errTst_P, errTrain_A, errTst_A)


def PEGASOS(w, X, y, Eta):
    '''
    Train a SVM through PEGASOS
    '''
    Aplus = np.where(y * np.squeeze(np.asarray(X.dot(w.T))) < 1)
    g = y[Aplus] * X[Aplus]
    _w = (1 - Eta * L) * w + (Eta / B) * g
```

2. Make a plot of training error vs number of iterations

When we finish our iteration, we already got four lists of training errors and test error, so we can plot them vs number of iteration:

```
def plotCurves(errTrain_P, errTst_P, errTrain_A, errTst_A):
    maxY = 1
    maxX = len(errTst_P)
    xs = np.arange(maxX)
    plt.plot(xs, errTrain_P, 'r-o')
    plt.hold(True)
    plt.plot(xs, errTst_P, 'b-o')
    plt.hold(True)
    plt.plot(xs, errTrain_A, 'k-x')
    plt.hold(True)
    plt.plot(xs, errTst_A, 'g-x')
    plt.legend(['PEGASOS Training Error', 'PEGASOS Testing Error', 'AdaGrad
Training Error', 'AdaGrad Testing Error'], loc = 'best')
    plt.title('PEGASOS (batch='+str(B)+', lambda='+str(L)+',
iterations='+str(T)+')')
    plt.xlabel('Training iteration x50')
    plt.ylabel('Error')
    plt.ylim((0,maxY))
    plt.xlim((1,maxX+1))
```



Question 1.6

For Question 1.6, actually we did it in Question 1.4 and Question 1.5. It does not need any extra step, since in the readCleanData(), we processed the sentiments like this:

```
sentiments.append(-1 if line[0] == '0' else 1);
```

So, for sentiments of 0's will be recoreded as -1, and for sentiments of 2's and 4's will be recorded as 1. So both training dataset file and test dataset file can use it directly.

Recall the result in Question 1.4 and Question 1.5, we can get the accuracy of PEGASOS was: 0.763.

The accuracy of AdaGrad was: 0.648