Project 2

ESE 545, Data Mining: Learning from Massive Datasets

Youjia LI, 21360270; Xiaonan YANG, 54589089

Introduction

The project is aimed to process a massive dataset with programming techniques and then design and implement PEGASOS and AdaGrad algorithms to develop classifiers based on supervised learning. The project consists of 6 parts and are solved by Python. All programs responding each question has been optimized in terms of running time and storage efficiency.

Q1.1 &Q1.2 Dataset Reading and Processing Data

The original file contains 1.6 million entries of tweets and corresponding sentiments with numerical values of either 4 or 0. As required by the question 1.1 and 1.2, the tweet contents and sentiments need to be read into the program. Then the sentiment values need to be transferred to 1 and -1 and tweet content be cleaned up instructed by specific rules. The combination of Q1.1 and Q1.2 is divided into reading-in datasets and processing data and is achieved via following steps.

i. Define a read-in function to read sentiment and tweet contents from training data csv file and collections of stop words from stopwords text file (for cleaning questions). The program returns three separate lists storing the required information.
ii. Define a data-process function which transform the data obtained above. For sentiment list values, simply compute (sentiment-2)/2 will do.
iii. For the clean-up procedure of tweet contents, firstly iterate through every string in the whole tweet list split it into a nested list of words. By doing this, additional white spaces are automatically removed. Next detect words containing features of a web address (www., http// etc) and replace such words with 'URL' and similarly replace words starting with '@' by 'AT-USER'. Then iterate through every nested list and delete adjacent duplicates such as 'very very' as well as words that also exist in stop words collection. Note that by deleting only adjacent duplicates (rather than deleting all duplicates) will keep the occurrences of words and hence ease the process of creating the feature list in later question, which will be explained later. Then convert the nested list back to strings by connecting words by a single space. Iterate through every character and delete punctuations (if any)
iv. Return the whole tweet list and processed sentiment list

Q1.3 Unigram Feature Extraction

Given the cleaned tweet list and corresponding sentiment, the question requires us to firstly establish a bag of words consisting of all words from cleaned tweets without repetition and then create a feature vector for every tweet recording the number of occurrences of each words in bag of words appearing in this particular tweet.

i. Iterate through all strings of tweets and converted words separated by space into lists as separated by space. Store the converted tweet data into another list since we still the non-converted tweets to count occurrences. Then obtain bag of word as the union of sets of all tweet lists. Note that during this process all duplicates in every tweet are removed by sets. This way we have a list of feature words from all tweets without repetition.
ii. Establish a CountVectorizer imported from sklearn module and set the previous bag of words list as vocabulary. Apply the configured CountVectorizer to the cleaned tweet content before conversion (in the form of strings), which gives you a huge sparse csr array of feature vectors, whose rows represents different tweets and columns representing a occurrences of words in bag of words as shown below.
iii. Return a tuple of the features vector array obtained above and the sentiment list. Note that both collections have the same indexes and therefore the sentiment and feature list for each tweet can be matched by indexing.

Q1.4 PEGASOS Trained SVM

Question1.4 requires us to train a SVM for feature vector list using PEGASOS algorithm and evaluate the classifier by training error rate with different numbers of iterations. PEGASOS updates the hyperplane by gradients of objective function based on stochastic gradient descent.

The detailed process are as follows:

i. Define a function to realize the PEGASOS algorithm. The function takes in training data containing feature vector list and sentiment list from Q1.3 and the desired maximum iteration number. Initialize the hyperplane (classifier) w to be an array of zeros in the same size of each feature vector as default value and initialize the regularization coefficient λ to be 0.001.
ii. Updating : For a given iteration number *t*, firstly randomly select B (default value 100) number of samples from the whole feature vector list as mini batch. Initialize a variable *products* to zero to prepare for recording the sum of $y * \boldsymbol{x}$. For each sample, assign its feature vector to x and its sentiment value to y, then compute the following expression as a condition:

$$y * \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x} < 1$$

which denotes that a violation to the current classier has occurred or a correct classified data point is too close to the hyperplane. If the condition is satisfied, compute $y * \boldsymbol{x}$ and add up the result to variable *products*. Iterate through every sample in the mini batch and update the classifier with following expressions:

$$\nabla_t = \lambda w_t - \frac{1/t\lambda}{B} \sum y * \boldsymbol{x}$$

$$\boldsymbol{w'_t} = \boldsymbol{w_t} - (\frac{1}{t\lambda}) \nabla_t$$

$$\boldsymbol{w_{t+1}} = min\left\{1, \frac{1/\sqrt{\lambda}}{\|\boldsymbol{w'_t}\|}\right\} \boldsymbol{w'_t}$$

The resultant updating of $\boldsymbol{w}$ is basically ‘moving’ $\boldsymbol{w}$ towards the data point of violation but restricted by the projection $\boldsymbol{w} \leq 1/\sqrt{\lambda}$

iii. Calculating training error: Still within the loop of a given iteration number, randomly select 1000 samples out of original feature vector list. Apply the current classifier to the selected 1000 samples by compute $\boldsymbol{w^T x}$. Call another error-calculating function which takes in current $\boldsymbol{w}$ and selected samples. Check if $y * \boldsymbol{w^T x} < 0$ is satisfied (which means a violation to the classifier occurs). If yes, increase error counter by 1. Iterate through all samples, and then compute the current error rate by dividing error numbers by 1000. The function will then return the error value. Record the error rate corresponding to the current iteration number into a list.

iv. Repeat step ii and iii for next iteration number until it reaches the max iteration number. Plot the training error rate versus the number of iterations.

By setting the maximum iteration number to be 1000, the resulting plot is displayed below:

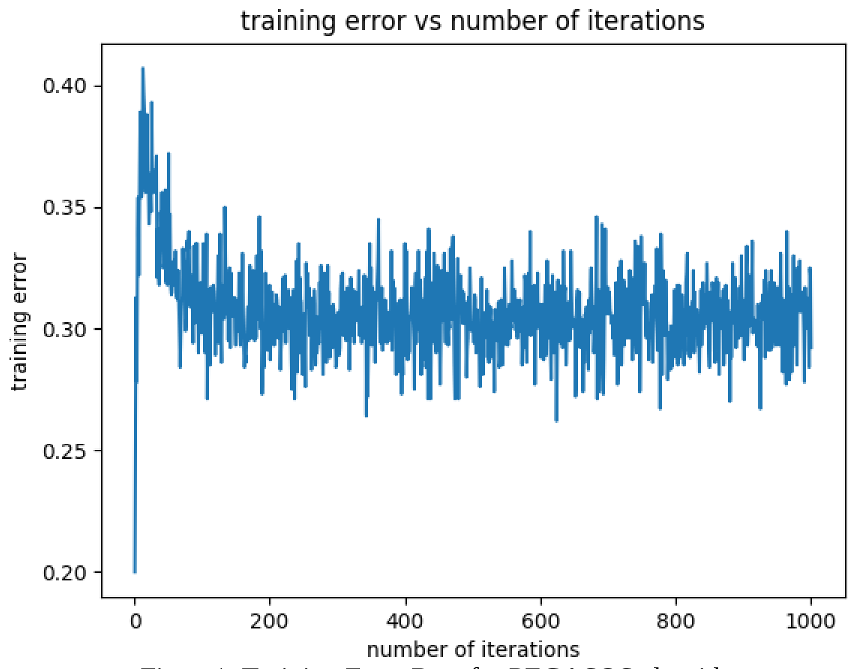

Figure1. Training Error Rate for PEGASOS algorithm

From the plot above, it could be noticed that the error rate is swiftly converged within the first 100 iterations and then stays stabilized at around 0.3.It seems that the error rate cannot be further converged in 1000 iterations, which may be caused by insufficient iteration number or size of mini batch.

Q1.5 AdaGrad trained Classifier

Question1.4 requires us to train a classifier for feature vector list using AdaGrad algorithm and evaluate the classifier by training error rate with different numbers of iterations. AdaGrad algorithm functions by modifying the values of gradient obtained in stochastic gradient decent.

The detailed process are as follows:

i. Define a function to realize the AdaGrad algorithm. Again, the function takes in training data containing feature vector list and sentiment list from Q1.3 and the desired maximum iteration number. The initialization process is basically the same as in Q1.4 i. Additionally, a fudge factor is initialized to 10^{-6} for numerical stability and the historical gradient variable is initialized to zero. Another step size variable η is assigned to 0.001, which decides the magnitude of updating in $\boldsymbol{w}$ in every iteration.

ii. Updating $\boldsymbol{w}$: For a given iteration number t, repeat step ii in Q1.4 to obtain the gradient with current classifier $\boldsymbol{w}$ and sample feature vectors. Then adding the square of gradient vector to historical gradient and adjust the current gradient value by following expressions.

$$historical\_grad_t = \sum_{i=1}^{t} \nabla_i$$

$$\nabla_t{}' = \nabla_t / (fudge\_factor + \sqrt{historical\_grad_t})$$

Similarly, update $\boldsymbol{w}$ with adjusted gradient with following expression.

$$\boldsymbol{w'_t} = \boldsymbol{w_t} - \eta \nabla_t{}'$$

iii. Calculating training error: The definition and computation of training error rate remains the same as in Q1.4. However, instead of calculating the error rate for each iteration, the error rate is measured and stored every 1000 iterations.

iv. Repeat step ii and iii for next iteration number until it reaches the max iteration number. Write the result to csv file.

v. Re-rum the program for PEGASOS algorithm with same maximum iteration number and measure the training error for every 1000 iterations. Write the result to csv file. Plot the training error rate versus iteration number for two different algorithms and merge these two plots.

By setting the maximum number of iterations to 40,000, the training error rates for these two classifier are plotted below:

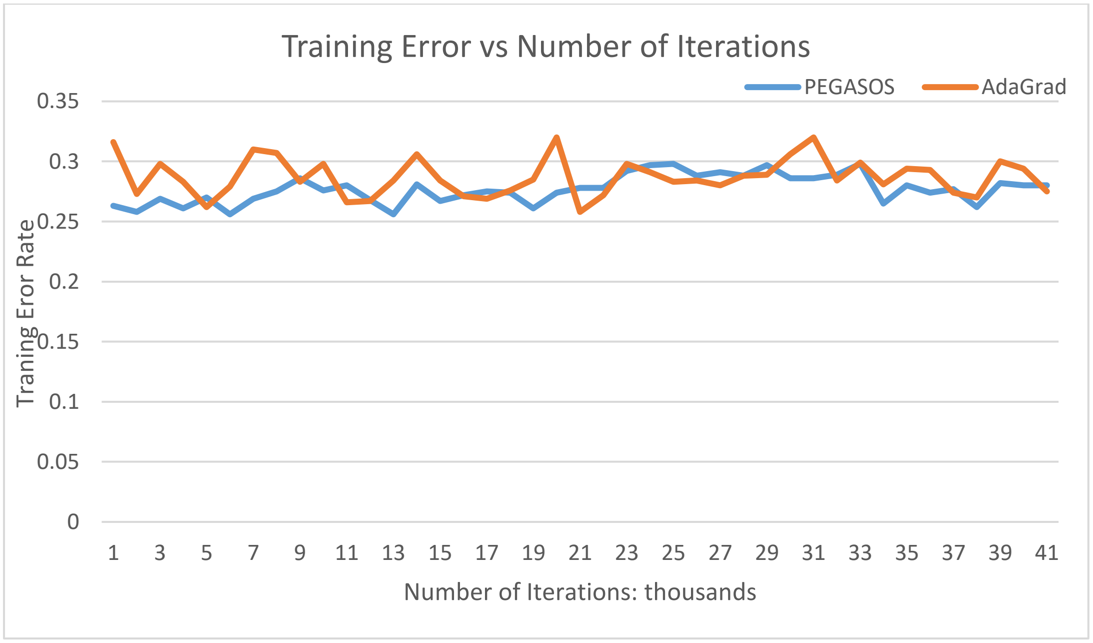

Figure2. Training Error Rate for PEGASOS and AdaGrad

It seems that the performances of these two classifiers are similar in terms of average error rates and convergence rate.

Q1.6 Performance on Test Data

Note in previous questions the performances of classifiers are evaluated by train error using training datasets. Question 1.6 asks us to evaluate the classifier in Q1.4 and Q1.5 with test data by plotting test error versus number of iterations, which is achieved via following steps:

i. Make use of the read-in and data-process functions to read in, convert and clean test data following the same rule.
ii. Re-run two algorithm functions as in Q1.5, but this time use processed test data as the input parameter of error calculation function.
iii. Again, write the results to csv file. Plot the test error rate versus iteration number for two different algorithms and merge these two plots

Again, the maximum iteration number is set to 40,000. And the test error rates from two different algorithms are shown in Figure3

It seems that the performances of these two classifiers are similar in terms of average error rates and convergence rate. Compared with results in Figure 2, the overall error rate with test data seems to higher than that with training data.

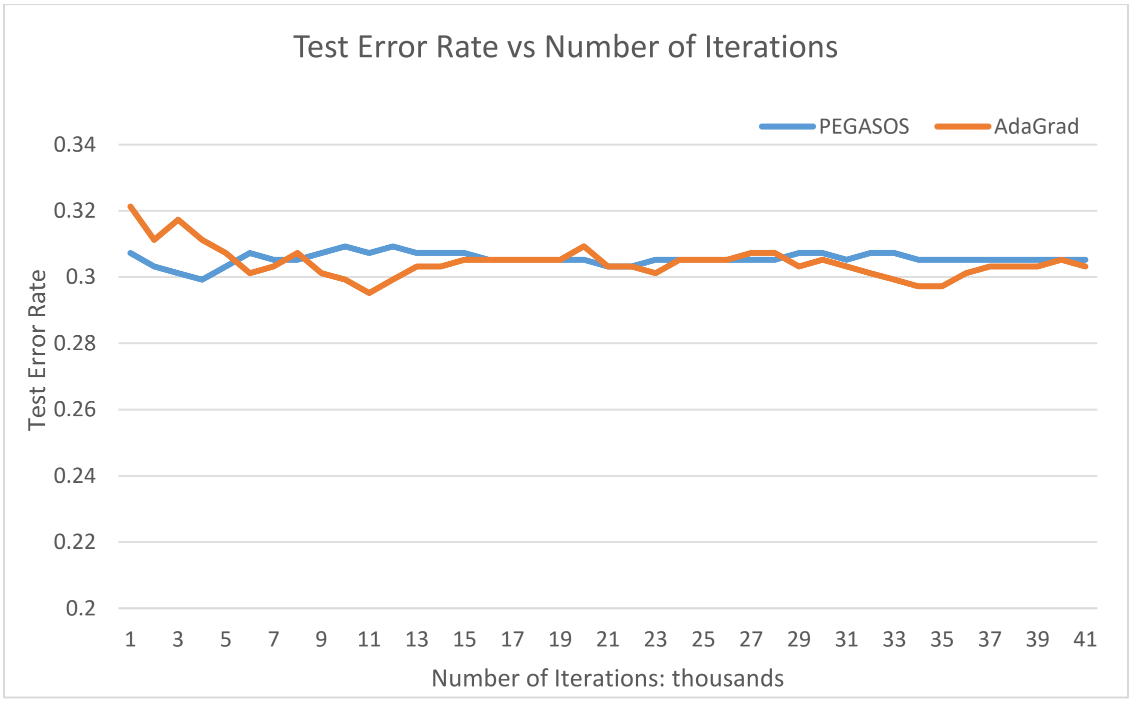

Figure3. Test Error Rate for PEGASOS and AdaGrad algorithms