Text Classification Based on Machine Learning

Abstract—This project is aimed to achieve text classification based on machine learning method. In this project, sentences from the Rotten Tomatoes dataset were used as corpus. To extract text features, I used bag-of-word model and uni-gram, bi-gram and tri-gram were considered. To reduce feature dimension, I only used TFIDF-IDF of the most common tokens. Also, to achieve higher accuracy, appearance of high frequency words, sentiment words, positive and negative judgment words, Not words and degree words were taken into account. In terms of classifier, softmax regression model was constructed for this multi-class classification task. Here, cross entropy loss function was implemented and batch gradient descent, stochastic gradient descent and mini-batch gradient descent were compared. When it comes to experiments, I used Dev-test set. Informative features and parameters were also discussed. Finally, the best accuracy was achieved at 53.83% for our softmax regression model and 58.40% for sklearn Logistic Regression model.

I. Introduction

Text classification, as the assignment of text files to one or more categories based on information contained from text files, is an important component of information management tasks. Text classification can be achieved by machine learning methods, in which, we aim to find a mapping or function to get category y of a piece of text based on its features x.

In this report, we focus on text classification of movie review of the Rotten Tomatoes, to label phrases on a scale of five values: negative, somewhat negative, neutral, somewhat positive and positive. Indeed it is a multi-class classification task. First, we try to extract features of the text. Then we discuss the softmax regression classifier by focusing on the cross entropy loss function and different ways to minimize it. Finally, we analyze the testing results and discuss the future work.

II. Text Feature Extraction

A. Bag-of-word and N-gram Features

In bag-of-word models, we assume a piece of text is a combination of numerous words, ignoring the order of the words. However, it is too simple and ideal to only consider the single words, as sometimes the order of words can totally change the meaning of the whole sentence. In that case, we often need to introduce N-gram features to include more context information. Having these features, we can use bag-of-word model to represent text in vectors. Here, we tried the CountVectorizer and TFIDFVectorizer of *sklearn* package to extract text features.

1. CountVectorizer

The CountVectorizer can transform words into word frequency matrix. So that we can represent a sentence by a vector in which each dimension shows the count a certain word appears in this sentence. For example, for corpus composed of sentences ‘I have a dog’ and ‘I like running’, these two sentences can be represented respectively as follows:

I	have	a	dog	like	running
1	1	1	1	0	0

Fig 1. Vector of ‘I have a dog’

I	have	a	dog	like	running
1	0	0	0	1	1

Fig 2. Vector of ‘I like running’

Obviously, with the increase of corpus, the dimension of word vector can be explosive; also it can be sparse as many words or tokens don’t appear in a short sentence. As a result, we often pay attention to those high frequency words.

2. TF-IDF

As the CountVectorizer only pay attention to word frequency, some problems may appear. For example, many common but meaningless words like ‘a’, ‘the’ ‘do’ will be emphasized. However, they do not help us

distinguish sentences. It this case, we need to introduce TF-IDF. TF-IDF refers to term frequency–inverse document frequency. This value assesses a word's contribution or importance to a sentence or a file. It can be calculated as follows:

$$tfidf_{i,j} = tf_{i,j} * idf_i$$

Where the term frequency $tf_{i,j} = \frac{n_{i,j}}{\sum_1^k n_{k,j}}$ and inverse document frequency $idf_i = lg\frac{|D|}{|\{j:t_i \in d_j\}|}$. D means the total number of files in corpus and $|\{j: t_i \in d_j\}$ means the number of files containing word t_i.

Here, we use the TFIDFVectorizer of sklearn package to transform words into TFIDF matrix. It's noticeable that we set the *ngram_range*=(1,3), so that we can include uni-gram, bi-gram and tri-gram features. Also, in order to reduce vector dimension and avoid sparsity, we set *max_features*=500, so that we only focus on the 500 most common tokens.

B. Other Features

Considering we are doing a text classification task related to sentiment analysis, I have also picked some other features to improve the classification accuracy.

- The frequency of most common words
- The appearance of positive and negative judgment words
- The appearance of positive and negative sentiment words
- The existence of Not words and degree words

For the first one, I calculated the most common words from the already processed sentences using FreqDist provided by nltk. For the later three, I downloaded positive and negative judgment and sentiment dictionaries as well as Not word and degree word lists via the HowNet knowledge Database. As the sentences in our corpus are not long, so we focus on whether these words appear instead of their frequency.

III. Classifier Construction

A. Softmax Regression

Softmax Regression is a kind of Logistic Regression for Multi-Class classification. Given class set $\{1,2,\dots C\}$ and a sample $\boldsymbol{x}$, the conditional probability that x belongs to class c is:

$$p(\mathrm{y} = \mathrm{c}|\boldsymbol{x}) = \mathrm{softmax}({w_c}^T\boldsymbol{x})$$

$$= \frac{\exp({w_c}^T\boldsymbol{x})}{\sum_{c'=1}^{C}\exp({w_{c'}}^T\boldsymbol{x})}$$

Where w_c represents the weight of class c.

In that case, the decision function is:

$$\hat{y} = argmax_{c=1}^{C} p(\mathrm{y} = \mathrm{c}|\boldsymbol{x})$$
$$= argmax_{c=1}^{C} {w_c}^T\boldsymbol{x}$$

In vector representation the conditional probability is:

$$\hat{\boldsymbol{y}} = \frac{\exp(W^T\boldsymbol{x})}{1^T\exp(W^T\boldsymbol{x})}$$

Where the weight matrix *W* is composed of weights of C classes, and this is the parameter that we need to train.

B. Loss Function

We use the cross entropy loss function to learn the best weight matrix W. Here, we use one-hot vector y to indicate the true label of sample x. Using cross entropy loss function, the risk of softmax regression model is:

$$R(W) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C}{\boldsymbol{y}_c}^{(n)} log{\hat{\boldsymbol{y}}_c}^{(n)}$$

$$= -\frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{y}^{(n)})^T\, log\hat{\boldsymbol{y}}^{(n)}$$

However, in order to avoid overflow when computing the softmax function, we need to add a regularization term to restrict the weight matrix. So that our risk turns into:

$$R(W) = -\frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{y}^{(n)})^T\, log\hat{\boldsymbol{y}}^{(n)} + \frac{\lambda}{2}\sum_{i=1}^{C}\sum_{j=1}^{m}{W_{i,j}}^2$$

Therefore, we can get the gradient as:

$$\frac{\partial R(W)}{\partial W} = -\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}^{(n)}(\boldsymbol{y}^{(n)} - \hat{\boldsymbol{y}}^{(n)})^T + \lambda W$$

As a result, our training process is as follows:

- Randomly initiate the weight matrix W_0
- While Iter<maxIter or $||W_{t+1} - W_t|| > \delta$

  Update $W_{t+1} \leftarrow W_t - \alpha(-\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}^{(n)}(\boldsymbol{y}^{(n)} - \hat{\boldsymbol{y}}^{(n)})^T + \lambda W_t)$

C. Gradient Descent

It is noticeable that when doing the gradient descent, we have several ways with different advantages and disadvantages. They are discussed and compared here.

1. Batch gradient descent

In batch gradient descent method, we use the whole train set to update the weight matrix. What we have discussed in last section is this kind of gradient descent. However, it is time consuming as it uses all the samples in every single iteration.

The algorithm can be described as following codes:

Input: Train set D$\{(x_i, y_i)\}$, maxIter, learning rate **α,** penalty coefficient **λ**
Randomly initiate W_0
repeat

gradient=$-\frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}^{(n)}(\boldsymbol{y}^{(n)} - \hat{\boldsymbol{y}}^{(n)})^T + \lambda W_t$

$W_{t+1} \leftarrow W_t - \alpha \cdot gradient$

Iter←Iter+1
While Iter<maxIter
Output: weight matrix W

Algorithm 1. Batch gradient descent

2. Stochastic gradient descent

In order to reduce training time, stochastic gradient descent is proposed. While the batch gradient descent uses all samples when iterate, this method only use a sample each time for updating. In this way, we use this formula to update the weight matrix:

$$W_{t+1} \leftarrow W_t - \alpha(-x^{(i)}(y^{(i)} - \hat{y}^{(i)})^T + \lambda W_t)$$

As a result, the training speed is improved. But it also has some problems. More noise makes it not always move towards the direction of overall optimization. It seems blinder.

The algorithm can be described as following codes:

Input: Train set D$\{(x_i, y_i)\}$, maxIter, learning rate **α,** penalty coefficient **λ**
Randomly shuffle dataset, Randomly initiate W_0, i← 0
repeat

gradient=$-\boldsymbol{x}^{(i)}(\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)})^T + \lambda W_t$

$W_{t+1} \leftarrow W_t - \alpha \cdot gradient$

i←i+1
While i<maxIter
Output: weight matrix W

Algorithm 2. Stochastic gradient descent

3. Mini-batch gradient descent

Mini-batch gradient descent is a compromise between batch gradient descent and stochastic gradient descent. Taking both training time and accuracy into consideration, Mini-batch gradient descent may be a proper way to find the best parameter. Here, at each iteration, we use a subset of train set to update the weight matrix. For example, we use 10 sample each time, and the update rule changes into:

$$W_{t+1} \leftarrow W_t - \alpha(-\frac{1}{10}\sum_{n=1}^{10} \boldsymbol{x}^{(n)}(\boldsymbol{y}^{(n)} - \hat{\boldsymbol{y}}^{(n)})^T + \lambda W_t)$$

The algorithm can be described as following codes:

Input: Train set D$\{(x_i, y_i)\}$, maxIter, batch size,learning rate **α,** penalty coefficient **λ**
Randomly shuffle dataset, Randomly initiate W_0, i← 0
Repeat
m=batch size

gradient=$-\frac{1}{m}\sum_{j=i*m}^{(i+1)*m-1} \boldsymbol{x}^{(j)}(\boldsymbol{y}^{j} - \hat{\boldsymbol{y}}^{(j)})^T + \lambda W_t$

$W_{t+1} \leftarrow W_t - \alpha \cdot gradient$

i←i+1
While i<maxIter
Output: weight matrix W

Algorithm 3. Mini-batch gradient descent

IV. Experiment

A. Use Dev-Test set

The corpus is divided into Development set and Test set, go a step further, we divided the Development set into Training set and Dev-Test set.

Hence, we do training on training set, adjusting parameters and choosing the best classifier base on the results from Dev-Test set. At last, we use our trained best classifier to do test on testing set.

B. Find Informative Features and Parameters

Here, I tried many possible combinations of features, and the result I got on Dev-Test set through different models is as the following table.

In the table, TFIDF_500 means we only use 500 words with the highest TFIDF value as features, and Freq_500 means we use word frequency of the 500 most common words as features. Similarly, Freq_1000 means we use the 1000 most common words. *Particular* means we only consider the appearance of those particular words, including sentiment words, not words and degree words. *All* means we use the whole features set as mentioned before.

Tab 1. Different combinations of features of Softmax Regression

Features	TFIDF_500	TFIDF_1000	Freq_500	Freq_1000	Particular	All
Accuracy	51.31%	51.35%	51.33%	51.38%	52.24%	53.83%

Also, I have tried to use Logistic Regression classifier of *sklearn* package, and the results are below:

Tab 2. Different combinations of features of Logistic Regression

Features	TFIDF_500	TFIDF_1000	Freq_500	Freq_1000	Particular	All
Accuracy	56.03%	58.08%	56.24%	57.72%	52.30%	58.40%

Besides, as learning rate may influence convergence of the model, I have tried to use different learning rate to train our models. Following picture shows accuracy of different models under different learning rates when we use all features we extracted.

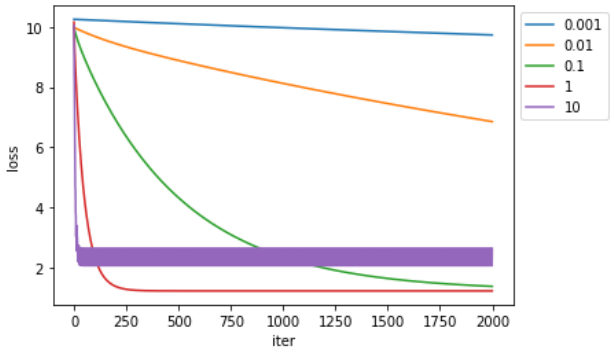

Fig 3. Loss of different learning rates

As we can see from this figure, when we set the learning rate too little, for example 0.001, the model converges really slowly. On the contrary, if we set it too large, for example 10, the he value of loss function will fluctuate continuously and cannot reach the optimal point, just as the purple line shows. From the figure, learning rate of 0.1 or 1 may be relatively proper for our task.

Next we focus on the penalty coefficient of weight—**λ,** which can work on overfitting problems of models. If **λ** is too small, the model is easy to overfit; on the contrary, if it is relatively large, we may not achieve our goal of training as it focus too much on the limitation of weight. So, I tried to use different **λ** under the learning rate of 1, and results are below:

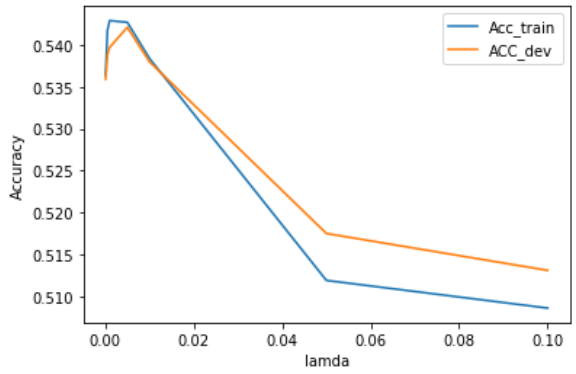

Fig 4.Accuracy when using different penalty coefficient

We can see that large values of **λ** indeed limit the weight too much and reduce accuracy. The figure shows that 0.001 or 0.01 may be a proper value for **λ**.

C. Different Gradient Descent Methods

Next, I pay attention to three gradient descent methods discussed before. From results shown in table 3, we validate that batch gradient descent indeed consume more time, but it brings more information and reach a higher accuracy. On the contrary, the other two methods train much faster, but sacrifice some accuracy.

Tab 3. Results of different gradient descent methods

Gradient descent	Batch gradient descent	Stochastic gradient descent	Mini-batch gradient descent when batch size=10
Accuracy	53.83%	51.43%	52.15%
Time to convergence	144.09s	71.63s	71.23s

V. Conclusion

The main objective of this project is to apply machine learning method on text classification. Firstly, we extracted text features like word frequency, tfidf value of common words, and appearance of sentiment words and so on. And then we explored softmax regression algorithm for text classification. Here, we mainly use Numpy package to do the matrix computation. Cross entropy function was chosen as our loss function and we tried three different gradient descent methods. When it comes to other parameters, learning rate and penalty coefficient of l2 regularization were discussed.

However, there does remain scope for improving the performance of this text classifier. On the one hand, we need to consider our text features more seriously, for example, including sentiment value or topic features. On the other hand, as I have tried other classifiers provided by *sklearn* on our features and the accuracy can even exceed 60%, so there remains some work on our softmax classifier. Maybe we need to pay attention to class weight or sample weight as our training corpus is imbalanced.