王昱琛
2100013153

Homework 1

自然语言处理基础
2023 年 4 月 7 日

第二次作业

2023 年 4 月 7 日

Introduction

This task is to implement a log-linear model for the 20 newsgroups dataset for text classification. I use TF-IDF as features for training, after several tries for parameters tuning, the model reaches 83.02% for my own evaluation function, while the F1 score is 82.34% and the sklearn accuracy score is 83.02%.To run this code, you need to pip install nltk and sklearn first.

Data preprocessing

The data preprocessing is done in utils.py's *preprocessing* function I use nltk to tokenize the data and remove some stopwords which is normal for every catagory and will affect the result. After that, I stem and lemmatize the words to normalize the word.Stemming is the process of reducing a word to its root or base form, regardless of its grammatical role or meaning. Lemmatization is the process of transforming a word to its dictionary or canonical form, based on its grammatical role and meaning.

With the preprocessed data, I build the sorted vocabulary based on the train.csv, then in the $compute\_tfidf$ function, the first loop through the corpu s is to calculate the df value for each word. For each word, compute the TF-IDF value by multiplying its term frequency (TF) in the document by its IDF value. Add the resulting TF-IDF value to a dictionary called tfidf, with the word as the key and the TF-IDF value as the value. Add this dictionary

to a list called $tfidf\_corpus$. We can get features from the $tfide\_corpus$ then.Addtionly, I use the top frequency 20000 words in the vocabulary.

The implementation of log-linear model

For each possible y of instance x, we can compute a score:

$$score(x, y) = \sum_i w_{ij} f_i(x, y)$$

in the *predict* function to calculate the possiblility of this text sorted to any of the 20 catagories.

The probalistic model is:

$$p(y|x) = \frac{exp\ score(x, y)}{\sum_{y'} exp\ score(x, y')}$$

And we need to maximize:

$$LL(\hat{w}) = \sum_k \hat{w} \cdot f(x_k, y_k) - \sum_k log \sum_{y'} exp(\hat{w} \cdot f(x_k, y'))$$

So the gradients for each w can be written as:

$$\frac{\partial LL(\hat{w})}{\partial w_i(x, y)} = \sum_k f_i(x_k, y_k) - \sum_k \sum_{y'} f_i(x_k, y') p(y'|x_k; \hat{w})$$

After adding relular term to avoid overfitting,the gradient can be written as:

$$\frac{\partial LL(\hat{w})}{\partial w_i(x, y)} = \sum_k f_i(x_k, y_k) - \sum_k \sum_{y'} f_i(x_k, y') p(y'|x_k; \hat{w}) - \alpha w_{f_i(x,y)}$$

a is the regularization term coefficient.

When we train, we first predict and then ungrade the gradients, and then we use learning rate and lemma to ungrade the w parameters.

The update algorithm

The function first calls the predict method, which predicts the scores of each class for each sample in the input batch. The function then initializes a gradient matrix with dimensions

n_classes by n_features, where n_classes is the number of classes in the classifier, and n_features is the number of input features. This gradient matrix is used to compute the gradient of the loss function with respect to the model's coefficients. We can do that by iterating over each sample in the input batch and each feature in the sample, and updating the corresponding gradient value according to the derivative of the loss function with respect to the model's coefficients.

Then, we can update the model's coefficients using the computed gradient and the learning rate and regularization parameter specified in the model.

Evaluation

What I use is an easy Evaluation function, which uses the trained parameters to predict the labels for each test sample and compute the possiblility of making the correct decision.

The results on both train and test set

	Eva	F1	Eva(sklearn)
train	0.9871840197984798	0.986957903118298	0.9871840197984798
test	0.8301911842804036	0.8233857085232472	0.8301911842804036