姓名：王想
学号：2100013146

Homework 2

课程名称：自然语言处理基础
2023 年 4 月 1 日

1 Data Preprocess & Corpus & Feature

First, read .csv file with pandas and drop lines with null value. Then, for each line in file, replace column named 'data' with new_text. The transform from raw_text to new_text includes following steps.

1. Remove all punctuations and line breaks.
2. Remove extra spaces.
3. Turn all letters to lowercase.

After cleaning all text, build the corpus with all texts from training set. Finally, we can extract features with this corpus. In function $extract\_feature$, use parameter use_tfidf to control whether to extract TF-IDF or just 0-1 feature. The implementation of these parts are in $DataPreprocess.py$, $corpus.py$, $feature.py$.

2 Log-linear Model

The parameters of the Loglinear Class are given as follows:

- θ: parameters of log-linear model, it's a $class\_num \times length$ dimensional matrix.
- $class\_num$: Number of classes.
- $length$: Length of a feature.

The functions of the Loglinear Class are given as follows:

- *predict*: given a text, predict classification for each text.
- cal_grad: given a single sample x_k, calculate gradient for parameters θ.
- *update*: given an epoch of training set, update parameters θ.

Here come the details of these functions:

predict For each text-class y of instance x, compute a score: $score(x, y) = \sum_i \theta_{yi} f_i(x, y)$ then return the index of text-class corresponding to the highest score.

cal_grad According to the course slide, we need to maximize:

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

Considering the function of **cal_grad** is just calculate gradient for parameters θ with a single sample x_k, we set $k = 1$ and the formula above will be

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

So the gradients for each θ_j can be written as:

$$\frac{\partial LL(\theta_{\mathbf{j}})}{\partial \theta_{ji}} = f_i(x_k, y_k) - \sum_{y'} f_i(x_k, y') p(y'|x_k; \theta_{\mathbf{j}})$$

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

$$\frac{\partial LL(\theta_{\mathbf{j}})}{\partial \theta_{ji}} = f_i(x_k, y_k) - \sum_{y'} f_i(x_k, y') p(y'|x_k; \theta_{\mathbf{j}}) - \alpha \theta_j$$

The details of function *update* will be introduced in the next section. The Loglinear Class is implemented in *model.py*. In this file, there is another function named *my_dot*, it will be called to calculate dot product between two sparse vectors with high efficiency.

3 Update Algorithm

The Update algorithm is mini-batch gradient descent. At each step, we randomly sample a mini-batch(epoch) from training data. And the number of samples is the same for each type of text-class. Then we feed it to the *update* function. The *update* function will call *cal_grad* function to calculate gradients Δ_k for every single text x_k from the epoch. For each gradient Δ_k, update parameters θ to $\theta + learning\_rate \cdot \Delta_k$. After that, we need to add relular term, and update parameters θ to $\theta - learning\_rate \cdot \alpha \cdot \theta$. The operation above is equal to the update formula on course slide.

4 Evaluation & Results

Function *evaluation* is implemented in *eval.py*: given a list of text, evaluate log-linear model with accuracy and macro-f1. For binary classification tasks, F1 score is defined as following formula:

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$
$$F1 = \frac{2 \times Pre \times Rec}{Pre + Rec}$$

For multi-classification tasks, Macro-F1 score is given by:

$$MacroF1 = \frac{1}{n} \times \sum_{i=1}^{n} F_i 1$$

After trying tons of groups of Hyperparameters, it turns out that $batch\_size$(100,200,300,500 are tested) have little effect on scores, while lr and α have significant effect. The effect of these Hyperparameters are showed below, here I just post out five groups, one of which is the best Hyperparameters I found. The best Hyperparameters: $lr = 0.01,\ \alpha = 0.001$.

Hyperparameters	Train epoch		Test set	
$batch\_size = 200$	Accuracy	Macro-F1	Accuracy	Macro-F1
$lr = 10^{-2}, \alpha = 10^{-2}$	0.995	0.99443	0.76823	0.76531
$lr = 10^{-2}, \alpha = 10^{-3}$	1.0	1.0	0.80098	0.79620
$lr = 10^{-2}, \alpha = 10^{-4}$	0.995	0.99443	0.78620	0.77946
$lr = 10^{-1}, \alpha = 10^{-3}$	1.0	1.0	0.74562	0.74225
$lr = 10^{-3}, \alpha = 10^{-3}$	0.905	0.89695	0.70911	0.69411

Tab 1: **Effect of Hyperparameters on model** lr: learning rate, α: relular parameter
Note: The scores on training epoch are the heighest scores among all training epoches when training. It is not the scores on the whole training set. Keep 5 decimal places.

If you run *run.sh* directly, the default parameters will be $batch\_size = 200,\ lr = 0.01,\ \alpha = 0.001$, and scores on test set will be about $Accuracy = 0.80,\ MacroF1 = 0.79$. There may be small data fluctuations caused by random sampling.