Loglinear report

程奔 1700012813

March 22, 2020

This is the report for assignment 1 of 2020 EMNLP course. The task is implemented with pure python.

Pre-processing Data

The dataset for this assignment is *20-Newsgroups*. It is a popular dataset organized into 20 different newsgroups, each corresponding to a different topic. To create feature template, it is necessary to pre-process the raw text. Data processing consists of mainly two parts: normalization and extracting features.

Data Normalization

Normalization requires *re* package. I just simply delete space at both side of the text, convert all the characters to lower-case and replace punctuations with spaces.

Extracting Features

As is mentioned in paper *Log-Linear Models* by Michael Collins, the model utilizes uni-gram feature template to create feature vectors. One feature vector for a (text, target) pair is thus of size: label counts multiplies uni-gram

counts.
First, build a vocabulary of the train dataset mapping each uni-gram to a unique integer by one scan. Since there must be words that appear only in the test set, I add a special token [UNK] in the vocabulary to serve as all unknown words.
Second, create a method *encode* to convert one text to a vector of length **vocab_size**. Because for one text, only one dimension (corresponding to its label) is valid and all other dimensions will be zeros, knowing this vector and the label is enough to re-build the whole feature vector. The method will also return another vector containing the valid indexes of this vector. This is useful for dealing with sparsity of the feature vector. Later I will explain how I make use of it.

These two parts are defined in dataset.py and feature.py. Relevant parameters are defined in parameters.py

Log-linear model

Train the model

The trainable parameters are defined as a matrix **v** in the model. As is mentioned above, it is a matrix of size (**n_label**, **vocab_size**). Each vector in the first dimension corresponds to a feature space of its gold label.
I use SGD as optimizing method. Calculate the gradient of each text target pair and update parameter **v**. **Train** method acts like this each epoch:
1.Get all instances from the train dataset and shuffle.
2.Generate gradient δ for each instance through method forward.
3.Update parameters **v** according to learning rate, which is a hyper-parameter:

$$v = v + lr * \delta$$

I choose the unnormalized form of expression of the gradient, because the normalization involves a expensive matrix calculation.

Evaluation

After one epoch training, there will be an evaluation **eval**. This method predicts each class for each text in the train and test dataset. Model accuracy will be simply calculated like this:

$$accuracy = \frac{correct\_num}{data\_num}$$

Forward

The key method in the model is **forward**. **Forward** takes two parameters text and target. The function do as followings:

1. Encode the text to get a feature vector **feature_vec** and valid index **valid_id** of the vector.
2. For each label, calculate corresponding value $exp(v[label] \cdot f)$, then save them.
3. If **target** is not **None**, this is training part. Through a softmax method, we can get the distribution of post probability of y as **dist**. Then we can get the gradient. Here I apply a faster way to free matrix calculation. Initialize a δ matrix, and for each label, calculate $\delta[label][id] = -dist[label]$. Thus the second term of gradient is calculated. Add $f(x^(i), y^(i))$, and we get the final gradient.
4. If **target** is **None**, this is evaluating part. There is no need to calculate probability distribution. The index of the biggest value $exp(v[label] \cdot f)$ is what model predicts.

Extension

Simple save and load methods are implemented for convenience. After each epoch training, model parameter **v** will be automatically saved to directory *save_model*.

The feature vector size of loglinear model is usually too large, and not all words in the text are important. I Chose to filter the words that appear less than 3, 5, 8 times in the train set.

Experiment

By default, the model will be trained 15 times with learning rate 0.001 and filter_times 8.

filter times	train set(%)	test set(%)
3	94.66	77.48
5	94.36	77.59
8	94.05	77.06