Report

109550005 張可晴

1. **Give an input and output example after applying each preprocessing method in the report.**

   I used 4 methods in total to preprocess the given text to get rid of the relatively unimportant words, which would be beneficial to the following training part.

   (1) Removing stopwords (def remove_stopwords):
   "There is an apple on the table." => "There apple table".

   (2) Removing punctuations and HTML tags (def preprocessing_function):
   "Hi, I am Cassie.
 nice to meet you" => "Hi I am Cassie nice to meet you"

   (3) Stemming (def function_stem): it's a way to remove the suffix of a word
   "transferring" => "transfer", "delightful" => "delight", "definitely" => "definit", "remains" => "remain"

   (4) Removing digits (def function_digits):
   "In 1994, I was born in London." => "In , I was born in London."

   (5) Turn to lowercase(def function_ lowercase):
   "I believed that he is stUpid." => "I believed that he is stupid."

2. The performance of using different numbers of feature_num in sentiment classification.

feature_num	perplexity	F1 score	precision	recall
500 Without preprocess	96.06494923764191	0.7062	0.7092	0.707
500 With preprocess	207.11657385016622	0.6882	0.6977	0.6909
450 Without preprocess	96.06494923764191	0.6984	0.7018	0.6993
450 With preprocess	207.11657385016622	0.6843	0.6964	0.6878
200 Without preprocess	96.06494923764191	0.6713	0.6716	0.6714
200 With preprocess	207.11657385016622	0.6547	0.6756	0.6616

Ngram

feature_num	F1 score	precision	recall	loss
500 Without preprocess	0.9328	0.9328	0.9328	0.2269
500 With preprocess	0.8859	0.889	0.8861	0.3173
450 Without preprocess	0.9325	0.9325	0.9325	0.2291
450 With preprocess	0.889	0.8923	0.8892	0.3186
200 Without preprocess	0.9335	0.934	0.9335	0.2304
200 With preprocess	0.8834	0.8878	0.8837	0.3179

DistilBert

3. Discuss what you observed with perplexity, F1-score, precision, and recall of different methods in the report.

Before the discussion, what is "perplexity"?

Perplexity is a method to evaluate a language model.

First of all, calculate the entropy which can be perceived as the extent of surprise. The higher the entropy is, the more the surprise is. Besides, the perplexity is just the exponential of the entropy.

Below is the formula of perplexity:

$$Perplexity = 2^{l} \quad \text{where} \quad l = \frac{-1}{M}\sum_{i=1}^{m}\log p(w_i)$$

In conclusion, the lower the perplexity is, the lower the entropy is, the more surprise the text is, our prediction is closer to the reality.

Discussion about perplexity

The perplexity of Ngram with preprocess is higher than the one without preprocess, which meets the meaning of perplexity. Because we remove the unemotional words, in this way, it's harder for us to understand the sentence that is more incoherent.

We can also observe that the perplexity won't change no matter how the feature_num is, in my humble opinion, which is because the perplexity has no relations with the sentimental judgement, instead it's about if the sentence is fluent or not.

Discussion about the performance of Ngram and DistilBert

DistilBert will always outperform Ngram no matter the feature_num is.

The higher the feature_num is the better the performance is for Ngram. However, I can't observe the relations between the feature_num and performance for DistilBert.

Both Ngram and DistilBert will perform worse with preprocessing.

The F1-score, precision, recall are almost the same when the texts are without preprocessing by DistilBert

4. Discuss the difference between the bi-gram model and DistilBert (a lightening variant of BERT).

a. What are the reasons you think bi-gram cannot outperform DistilBert?

DistilBert has the self-attention mechanism which will apply attention weights to decide whether this word is vital to understand the whole sentence, and it can also identify the context of a sentence. By contrast, bi-gram can only take the adjacent word into consideration.

b. Can bi-gram consider long-term dependencies? Why or why not?

Bi-gram cannot consider long-term dependencies since we just put adjacent word into dictionary to train the model. For instance, consider “My mom is pretty, but my sister is ugly.” and “My mom is ugly, but my sister is pretty.”. bi-gram will treat the two sentences as the same because the relation of adjacent word is identical. Nonetheless, the two sentences are totally different meaning.

c. Would the preprocessing methods improve the performance of the bi-gram model? Why or why not?

It is surprise that the preprocessing won’t improve the performance of the bi-gram. I guess the reason is that removing the stopwords may prevent from correct judgement due to the incomplete context. As a result, I have a little test that just use removing punctuation, HTML tag, and digits and stemming for preprocessing.

feature_num	F1 score	precision	recall
200 Without preprocess	0.6713	0.6716	0.6714
200 preprocess With stopwords	0.6547	0.6756	0.6616
200 preprocess Without stopwords	0.6792	0.6875	0.6817

From the above data, I believe my hypothesis is proved by F1-score.

d. If you convert all words that appeared less than 10 times as [UNK] (a special symbol for out-of-vocabulary words), would it in general increase or decrease the perplexity on the previously unseen data compared to an approach that converts only a fraction of the words that appeared just once as [UNK]? Why or why not?

Converting all words that appeared less than 10 time as [UNK] will have lower perplexity than merely converting a fraction of the words that appeared just once as [UNK]. The reason is that it can greatly decrease the appearance of unfamiliar words and make them into a bigger group than the latter, which highly increases the probability, reducing the entropy, and thus reduces the perplexity.

5. Describe problems you meet and how you solve them.

(1) In part 0, I was confused and had no idea what I needed to do at the first glance. The most challenging task in this part, in my opinion, is to figure out which type of word can be removed for better training. There’s a trivial problem was when removing the digits. I judged where it’s digits by string.digits, but the digits came as multi-digits (e.g. 1994), which would disable the judgement. To solve the problem, I appended [0] after the string.

(2) In part 1, the hardest thing was to use dictionary correctly for me due to the unfamiliar with it. It took me lots of time to test and apply the 2D-dictionary.

(3) In part 2, to address the problem of encountering zero division error, I use Laplace smoothing. Below is the formula:

$$\hat{P}(\text{mat} \mid \text{cat on a}) = \frac{\delta + N(\text{cat on a mat})}{\delta \cdot |V| + N(\text{cat on a})}$$

Given δ=1, which indicates that we assume we saw all the bigram at least one time preventing from the zero division.

(4) In part 3, at the first time, I didn't get the meaning of the most n pattern, so I just used random.choices to pick up the features. Later on, I realized what it means so that I turned to use Counter, also taking me some time.

(5) As for run the program, first of all, I didn't turn on the window subsystem for Linux so that I run the program in vain. Secondly, I wanted to test the part of data firstly because I wanted to save time so that I directly deleted the data in the same file, leading the error of Nan. To cope with this problem, I turned on another file and stored some data into it. Moreover, I ran for one night and the perplexity didn't come up. As a result, I turned to use colab to run the .sh file by GPU. Next, my train_sentiment part WAS also very slow, I found the typo error "features[:feature_num]", turning into "features = features[:feature_num]"

(6) When I write the report, I found my Ngram performance was much lower than my roommate. The problem was that I use Couter, while it won't store the data by order. Then, I added sort function to solve the problem.

(7) I found in the last moment that TA told us if applying stem() function won't deserve the score, so I firstly added additional parameter to control the typo error about lowercase and uppercase. Nevertheless, it will report error, so I added a function to coped with the typo error to meet the TA's expectation that there are at least three methods to preprocess the text.