ITP 20002-02 Discrete Mathematics, Fall 2020

Homework 4

21900429 Yang si wan, 21900429@handong.edu

1. Introduction

In this homework, we use the given data to learn the machine and check whether the string input is negative or not using the machine-learning data. The given data is a preprocessed US airline sentiment corpus, and we can let the program match whether people have negative comments on this airline or non-negative comments.

For this program, (1) using the given dataset to create a trainer to make prediction model that is a large table format. (2) using prediction model to evaluate whether a given string is negative or non-negative.

2. Approach

1. implement trainer module(trainer.c)

1) tokenization

function : void tokenization(FILE * src,FILE * dst)

Read train_negative and train_nonnegative file using fopen() and tokenization all sentences using strtok(str," "). This separated words by whitespaces. Also, I use **void remove_p_make_lwcase(char*p)** function to remove punctuations and remove char that is not alphabet(use isalpha()) and convert each word to lower case while implementing tokenization.

remove_p_make_lwcase(char*p): If there is puntuation in the input string, the function removes it from the string and if there is an uppercase letter, changes it to lowercase and finally and it restores the result string in the original string. Also, remove char that is not ascii.(use isascii() function)

2) Normalization

I use system() function in c standard library to execute "stemwords" executable file. **[$ stemword -i input file -o output file]** I execute this shell command. I used this command to save normalizazed word to a file. (norm : negative normalized word file, n_norm : non-neg normalized word file)

3) Stopword removal

function : void remove_stopword(FILE * src, FILE * dst)

After reading the file, read the normalized string one by one, and if the string is a stopword, removed the word in a way that does not write it in the stopword dst file. I use **int isStopword(char * ch)** function to distinguish stopword. This function uses a 'strcmp()' function in the c library to compare all stopwords with input words.

4) Vocabulary reduction

function : int word_distint(FILE * src, FILE *dst, int num[],char ch[][50]);

(1)Open a file that collects words that have been removed the stopword and check how many times each word comes out of the file. For check the frequency of each word, I use function **word_distint(),** so only words that have appeared more than five times are stored in a string array ch[][50]. (2)Store frequency of the word in int array num[]. (3) Store number of all word in file's first line.(not more than 5 times word, just all word = neg_len, nonneg_len). (4)Then return the number of all words that appear five or more in file. (= neg_count, nonneg_count)

5) Construct prediction model

Use double for statements to find the probability that each word is negative and positive. I use struct [word], and struct **word** have voca[50], n_pro, p_pro.

p(word | C_{neg}) : (float)(neg_num[i] / neg_len)

p(word | C_{nonneg}) : (float)(nonneg_num[i] / nonneg_len)

<find neg word and non-neg word same as each neg word>

(1) Laplace smoothing (if word is in non-neg file)

$$P(w_i|Pos) = \frac{Count(w_i,Pos) + \alpha}{\sum_{w\in V}(Count(w,Pos) + \alpha)}$$

$$= \frac{Count(w_i,Pos)+1}{\sum_{w\in v}(Count(w,Pos)+1)} \quad = \frac{Count(w_i,Pos)+1}{\sum_{w\in v}(Count(w,Pos))+|V|}$$

- I choose $\alpha = 1$, and according to the smoothing formula, |V| is the only number of words.

- Because there is no duplicate word in this file, double the number of neg words and number of non-neg words to be used as denominator.

(2) Save the words and probabilities determined as neg to a file. At this time, the probability is obtained by adding 1 to the number of negs for smoothing.

(3) If there is a non-neg word that matches the word that is determined to be neg, obtain the non-neg probability and enter it into the file.

(4) write word, n_probablity, p_probablity in file (predict.txt)

<find nonneg word that is not same with neg word>

- Steps to enter words that have not been entered so far

(1) Repeat by the number of non-neg words, and make sure that each word matches the neg-determined word.

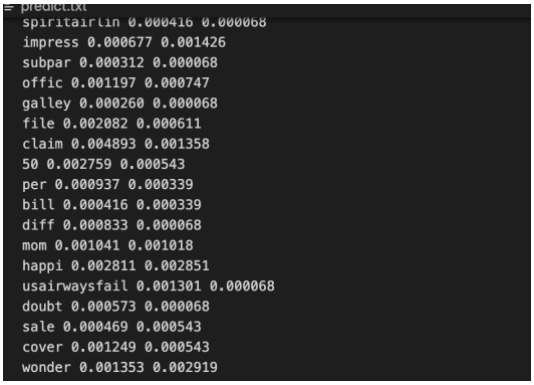
(2) If all the neg words are checked and there is no neg word that matches the non-neg word, the probability of this non-neg word is found. ((float)[nonneg_num]+1) / (float) nonneg_len)

(3) At this time, the probability that this word is neg is 0, so it is calculated as 1/(float)neg_len by smoothing.

(4) write word, n_probablity, p_probablity in file (**predict.txt**)

- structure of prediction model (distint by whitespace)

word neg_probablity pos_probablity



- interpretation of prediction model

In predictor module, Every string in the file was checked line by line. After receiving a string, splitting the string into words, filtering the words through several steps. Then, the result of filtering the words was saved to a file, and the file was read and stored all words in an array.

While iterating over the array, in the case of the same word in the predict model, the probability written next to the word was entered into the variable.(use find_neg, find_pos function())

2. implement predictor module(predictor.c)

1)Enter the file name

2) Text is entered line by line and stored in a "string" file.

3) Tokenization by using “string” file (s_token.txt)

4) Normalization by using “s_token.txt” file (s_norm.txt)

- At this time, use system() function to use libstemmer **[$ stemword -i input file -o output file]**

5) Remove stop word by using “s_norm.txt”(s_stop.txt)

- Use **void remove_stopword(FILE * src, FILE * dst)**

6) In “s_stop.txt” file, bring all the words into the array.

7) Use the functions below to find negative, non-negative probabilities by word in the prdiction model.

float find_neg(char * ch) : return negative conditional probability of the parameter “ch”

1) open the prediction model (predict.txt)

2) Use feof() to read the file until it reaches the end of the file, and check if there is a string equal to the input string.

3) When receiving input, all of the string, n_pro, and p_pro are input.

4) If a string, such as a string that was entered as a parameter of the function, exists in the prediction model, then return the n_pro.

float find_pos(char * ch) : return non-negative conditional probability of the parameter “ch”

- simillar with find_neg function.

8) Log scaling

$$p(c_x|t_i) \propto \sum_{w\ i,j\ \in t} \log P(w_{i,j} \mid c_x)$$

- By this formula, The neg, non-neg probability of text can be added to each word by taking log. (log(ab) = log(a) + log(b))

- So, with the find function above, get the neg, non-neg probability for each word, take the log, and then add it to each.

- If the log values are added to total_neg and total_pos, then two are compared to determine if this text is neg, non-neg. (neg = 0, non-neg = 1, store in judge[] array)

9) Store result (result.txt)

- Write down whether each sentence is neg or non-neg, and save it to a file with good readability.

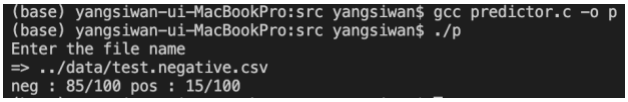
- print the results how many text are determined as neg and non-neg in this file.

3. Evaluation(detailed result is in result.txt)

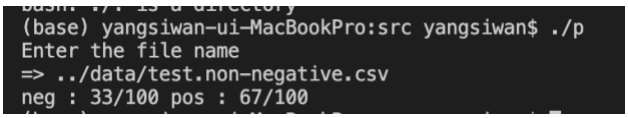
make => ./trainer => gcc predictor.c -o p => ./p

1. Finding the probability of a word when it appears 5 or more times

1) test.negative.csv



2) test.non-negative.csv



	True	False
True	True Positive 67/100	False Positive 15/100
False	True Negative 33/100	False Negative 85/100

Precision = $\frac{\frac{67}{100}}{\frac{67+15}{100}}$ = 0.817073170 = 81.7 %

Recall = $\frac{\frac{67}{100}}{\frac{67+85}{100}}$ = 0.4407894736 = 44.1 %

2. Finding the probability of a word when it appears 10 or more times

1) test.negative.csv

```
(base) yangsiwan-ui-MacBookPro:src yangsiwan$ ./p
Enter the file name
=> ../data/test.negative.csv
neg : 76/100 pos : 24/100
```

2) test.non-negative.csv

```
(base) yangsiwan-ui-MacBookPro:src
Enter the file name
=> ../data/test.non-negative.csv
neg : 27/100 pos : 73/100
```

	True	False
True	True Positive 73/100	False Positive 24/100
False	True Negative 27/100	False Negative 76/100

Precision = $\frac{\frac{73}{100}}{\frac{73+24}{100}}$ =0.75257= 75.3 %

Recall = $\frac{\frac{73}{100}}{\frac{73+76}{100}}$ = 0.489= 49 %

3. Finding the probability of a word when it appears 2 or more times

1) test.negative.csv

```
(base) yangsiwan-ui-MacBookPro:src yangsiwan$ ./p
Enter the file name
=> ../data/test.negative.csv
neg : 92/100 pos : 8/100
```

2) test.non-negative.csv

```
(base) yangsiwan-ui-MacBookPro:src yangsiwan$ ./p
Enter the file name
=> ../data/test.non-negative.csv
neg : 41/100 pos : 59/100
```

	True	False
True	True Positive 59/100	False Positive 8/100
False	True Negative 41/100	False Negative 92/100

Precision = $\frac{\frac{59}{100}}{\frac{59+8}{100}}$ = 0.88059= 88.1%

Recall = $\frac{\frac{59}{100}}{\frac{59+92}{100}}$ = 0.3907284= 39 %

- Through these results, it can be confirmed that the precision and recall change according to the thresh value of the trainer's vocabulary reduction.

- When the classification threshold is decreased, the precision value increases and the recall value decreases.

- When the classification threshold is increased, the precision value decreases and the recall value increases.

- It can be seen that the precision and recall values conflict with each other.

- Therefore, it can be seen that the performance varies depending on how the threshold of the classifier is determined.

4. Discussion

Navies Bayes' limitation

1) If the value of the newly given feature type does not exist in the previously learned feature, the probability is 0, and multiplying it will result in a final probability of 0. So, I use Laplace smoothing for all probability.

2) Since the probability derived using the classifier is less than 1, if there is a lot of probability to multiply, the value continues down, and the value comes out so small that it is difficult to distinguish. So, by taking logs on all the probabilities, I prevented underflow.

5. Conclusion

Using the Naive Bayes Classifier, we can distinguish spam messages. In addition, it can be seen that the precision and recall are conflicting and different according to the classification threshold. If we learn with more data, it will be a more accurate spam classifier.