Discrete Mathematics PA3 (Github : https://github.com/LeeLoHoon/PA3)

[Group 35]
Jun Hyung Kim | Seo Ye Ji | Seo Jun Pyo | Shin Hee Ju | Lee Ji Hoon

1. Naive Bayes Text Classifier

Introduction

- Problem Analysis

Problem

Build a model that classifies whether the message is negative or non-negative based on the Twitter US Airline Sentiment Dataset.

Input Output format

INPUT : sentence entered in standard input format.
OUTPUT: determine whether negative or non-negative in output format

Analysis

Process special characters, stopwords, numbers, and uppercase letters in given sentences and express then as word prototypes, not sentences. Calculate and insert the negative, non-negative probability of the processed word.

Approach

- Solution design

Trainer

The trainer implementation was divided into these 3 sections.

1) Tokenization

(1) Convert a line of text to a set of words

The given count.c code is code for inserting tokenized words into the hash table, counting advent of each word. For a better model, we conducted the process of recognizing words as one, which is recognized differently because of various input from various users. This whole process was done before putting words into the hash table.
The first thing we did is checking the value of ASCII code, and deleting special characters and numbers. Next, we used the isupper() function to recognize capital letters and change them to lowercase letters.

(2) Reduce dictionary

Made a new hashtable to save stopword. Comparing the word in the hashtable with the tokenized words, and if the words are in the hashtable, not added. The word with 'http' was meaningless because there was no overlap rate, so not added.

2) Normalization

Stemmer is a process to extract the origin form of words in which the variation of word occurs, such as as plural noun, addition of tense, article. It is a process to unify words used as same meaning. Using a snowball stemmer, which is mostly used check the tokenized words one by one and insert the modified word into hash table.

3) Probability estimation

Through the above process, words entered the hash table through 4 steps. (1-special characters, number, uppercase recognition, 2-delete duplicate words in ont message, 3-reduce dictionary, 4-stemmer)
We make the count of words' appearing divided by the number of messages and multiply 100 to calculate probability. The reason why number of messages was used as divisor is because we want to obtain probability of appearance of message containing that word.
The model structure desired in the PA3 was a set of word, negative probability, non-negative probability is written in one line. To implement this, we have made two separate hash tables, one for negative words, other for non-negative words. First, get the probability of a negative word through process above and check whether this word is in a non-negative hash table. If not existed, write the word and negative probability calculated and '-100' as non-negative probability. If existed, we get the number of advent from non-negative hash table, process it using by log and write it.

In addition, we deleted this word from a non-negative hash table. If this process is done for all negative hash tables, only words that are not included in negative hash table left. Therefore, these words were written as '0' in negative probability.

Predictor

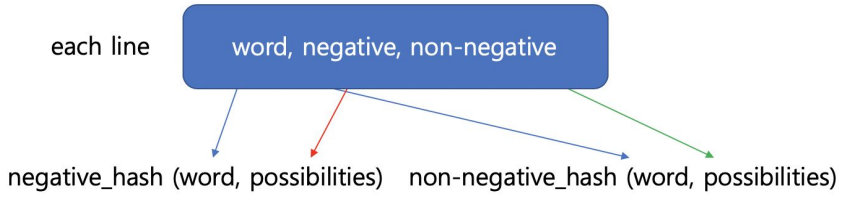
1) Load trained model

(1) Approach

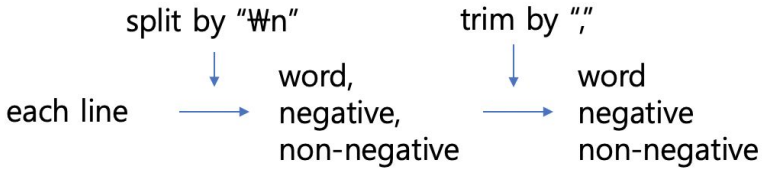
Unlike when we make a trained model data, we need to read data from one csv file which is "model.csv" and load data into the hash table structure for predict possibilities of words.
The ultimate goal for this step is to successfully extract data from "model.csv" and store it to hash table using minimal memory and time complexity as possible.
The obvious difference from trainers is that there is no need to separate code about hash_table structure and its variances between the non-negative and negative words. By taking this advantages, we can avoid unnecessary loop.

(2) Implement

The program can be implemented efficiently when data is refined and added to the hash table at the same time while program read one line.
Every line in the model.csv have a same format which contains word and probability of negative and non-negative. The format is always same as "word, negative, non-negative" and this is the key point of reduce time complexity.



To do this, each line must be properly trimmed and splitted. Since each line have a same string format, the data can be refined without exception.



Therefore, word is always added to the [0] index of the temporal array, negative is added to the [1], and non-negative is added to the [2]. At the same time, each string is also added to the hash table.

2) Predict

The predictor reads the text file through command-line argument. Then, each words from the text file will be tokenized and stemmed by program. After that, each words will be the key for the hash table to look up the probability that the word itself had in training. The probability of message is negative is calculated with Bayes formula. But as rescaling has been done by log function, we just have to add up all the probability of "the word is from negative message" and "the word is from non-negative message". Then, the probability is calculated by probability the words are from negative message divided by (probability the words are from negative message + probability the words are from non-negative message).
After the probability is calculated, the program checks with the decision boundary(= 0.47) that we setted. The program predicts the message is negative if the probability is lower than or equal to the decision boundary value. If the probability is larger than the decision boundary value, it predicts the message is non-negative.

Result

- Demonstration

  (1) Model result : result of count.c
  Here are some of the contents of model.csv
  bad 1.773518, 0.557053
  good 1.597268, 2.767196
  love 0.837189, 3.755615
  worst 2.511566, 0
  best 0.68297, 2.228212

  (2) test result : result of calculate.c

  test data

  We created the probability.csv file and wrote the probability of messages in the test.negative.csv file and test.non-negative.csv file. If word in the message is not in the model, we write its probability as '-1'. Follow the higher probability of negative or non-negative. The result was like this: FP = 26, FN = 31 (FP means the number of wrong predictions of negative message predicted as non-negative, FN means the number of wrong predictions of non-negative message predicted as negative.)

  test custom sentence

  The program receives *.txt file with the sentence written for the test in standard input and shows the result. ex)

Sentence	Result
I'm very disappointed with your service. Bad meals!!!! no!!	NEGATIVE 0.454036
I met my love in this plane. beautiful.. good service. I wanna know her phone number.	NON_NEGATIVE 0.518313
Excellent! apple was delicious	NON-NEGATIVE 0.544521
It was like hell	NEGATIVE 0.452918

Discussion

- The problem of Naive Bayes model is that the model doesn't consider the context of the message during classification. But there is a machine learning model named Recurrent Neural Network(RNN) that can handle timestamp data well, which means the model does consider the context of the message by bringing information from the past. So RNN may classified better as it doesn't assume naively.
- The smoothing technique we used is called Laplace smoothing. Laplace came up with this technique when he tried to solve the sunrise problem, the problem to find a probability that the sun will rise tomorrow.
- The first way to remove the stopword was to delete a word with a duplicate rate above a certain range. However, it did not completely filter out the stopwords because the duplicate rate was similar to the general word. So, we made a new hashtable to save the stopword.
- Through this assignment, we could understand how to design a machine learning model and learning process.
- During the data science class of System Programming 1, we have learned the concept of machine learning, and executed some functions. Using C language instead of Python without numpy, I realized that I become familiar with the concept of machine learning.
- We want to implement predictor can updating the trained model in real time. By using the predicted results of input as train data. Like machine learning, we can expand data and make more accurately just by predicting input without adding new data to the train data every time.
  Also, implementing weighting system can be helpful to increase the predictive accuracy. We can get more accurate results by adding weights according to the combination and number of positive and negative words in the sentence.
  In order to do this, we need to expand the data structure of the trained model. But we think that the csv file has limitations when the model needs to implement more complex data structures like vector, which for data of frequency, associated words.