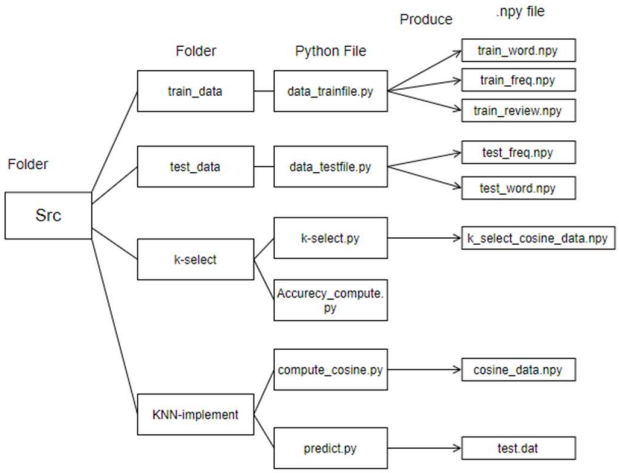
Miner2 username : Bear
Mason user id : chsiung2
Gnumber : G01272835
Best public score : 0.74

1. Introduction

- Similarity Function : I choose cosine similarity function as my approach to compute the similarity between two sentiments.
- Model Selection : I split 20% of “train_file.dat” as test data and 80% as training data. Then, compute the accuracy from k=1 to 40.
- Use majority vote in Model Selection and K-NN implement.
- The following picture is the structure of my src folder :
  train_data、test_data folder : handle text data.
  k-select folder : model selection(choose k).
  KNN-implement folder : implement KNN algorithm.
  Readme in src folder contains more details about program.



2. Approaches

- Handling “test_file.dat” and “train_file.dat”: steps i~v
  i. Convert sentiments into lowercase and use function”re.sub()” to remove punctuations and numbers. Then, use “list.split()” to get each word.
  ii. Use “nltk.corpus.stopwords.words()” to get rid of useless words. Ex : I’m, she’s, am, a, the…
  iii. Convert words to their original type by “lemmatize.lemmatize()”. Ex : went → go
  iv. Use list to store each word and its frequency. Each sentiment has its own lists of words and frequency. For example : “I am studying in George Mason University.” will be :
  [‘study’, ’George’, ’Mason’, ’University’] and [‘1’, ’1’, ’1’, ’1’]
  v. Use “np.array()” to save these data as “train_word.npy”, “train_freq.npy”, ”test_word.npy”, “test_freq.npy”. There is one more file named “train_review.npy” storing the rating of each sentiment in “train_file.dat”

- Compute cosine value (Cosine similarity function) :
  i. Compute cosine value of each sentiment in “test_file.dat” to each one in “train_file.dat” : Combine two lists of words and create two lists of frequency. For example :
     → combine [‘go’,‘school’] [‘nice’,’book’] to [‘go’,‘school’,‘nice’,’book’] and create two lists of frequency : train - [‘1’,’1’,’0’,’0’] & test - [‘0’,’0’,’1’,’1’]
  ii. Only save the sentiments in “test_file.dat” with higher cosine values.
- Predict rating.
  i. Summing the rating of k nearest neighbors of each sentiment in test data to predict the rating. If sum is larger than 0, then predict the rating to be +1. otherwise, -1.
  ii. Model selection : In “k-select.py”, use 20% of “train_file.dat” as test data and 80% as train data. Save the top 40 cosine values as “k_select_cosine_data.npy”. In “Accuracy_compute.py”, predict the rating of test data and then compare them to their original rating. Get the accuracy of k from 1 to 40. Results show in part 3.
  iii. KNN algorithm : In “compute_cosine.py”, save the top 30 cosine value as “cosine_data.npy”. In “predict.py”, predict the rating of each sentiment in “test_file.dat” and save them as “test.dat”.

3. Experimental Results

- The following picture is the accuracy of model selection from k = 1 to 40.

```
 1 : 0.6275.   2 : 0.4876.   3 : 0.6537.   4 : 0.5562.   5 : 0.6675.
 6 : 0.5956.   7 : 0.6737.   8 : 0.6167.   9 : 0.6769.  10 : 0.6329.
11 : 0.6910.  12 : 0.6502.  13 : 0.6939.  14 : 0.6640.  15 : 0.6994.
16 : 0.6672.  17 : 0.7021.  18 : 0.6702.  19 : 0.7039.  20 : 0.6796.
21 : 0.7102.  22 : 0.6829.  23 : 0.7093.  24 : 0.6858.  25 : 0.7110.
26 : 0.6896.  27 : 0.7145.  28 : 0.6867.  29 : 0.7107.  30 : 0.6904.
31 : 0.7099.  32 : 0.6883.  33 : 0.7104.  34 : 0.6918.  35 : 0.7118.
36 : 0.6904.  37 : 0.7134.  38 : 0.6942.  39 : 0.7142.  40 : 0.6956.
```

4. Conclusion

- According to experimental results, the highest accuracy occurs when k = 27. Also, most of them are in the range from 0.65 to 0.72. Because I only use 20% of “train_file.dat” as test data. The test set is smaller so that the difference is a bit higher. But when it comes to “test.dat” generating by KNN algorithm, the results on the miner2 are close due to the larger test set. All my submissions are from 0.71 to 0.74 and many k values have the same accuracy : 0.74. The highest accuracy k = 27 in model selection got the same 0.74 accuracy on the miner2.
- Therefore, using other similarity functions could probably be the way to improve the accuracy. But I think handling the sentiments more precisely would be better. Although some words like “go”, “buy”, “book” are not meaningless. But they can not give decisive clues. Maybe I should give more weights to some adjectives, like “good”, “excellent”, “awful”, “worst”. These words are more critical for me to decide whether it’s a positive sentiment or a negative one.