Course Project Report (Part I)

ERG3020 Web Analytics and Intelligence Project

Author: Piao Chuxin (115010058), Ye Xiaoxing (115010270)

In this part, we implemented the Naive Bayes Classifier using tweets data, using the code from sailor2017 (https://github.com/abisee/sailors2017). Copies of exported PDFs are attached to the report, and you can also view the files online. Due to some technique problems, the PDF is not wrapping lines and some characters will be missing.

What is this project?

The original project is a workshop in sailors2017, focus on the introduction to Natural Language Processing. The task was to automatically classify real tweets from Sandy using a Naive Bayes model.

The project is structured in six parts, and each part deals with a small task. In the first part, the project introduces some of the basic usage of how to define simple functions. Through the process of dealing with functions that can successfully classify the grade levels according to score, the project then develops to classify tweets with specific words in sentences at the end of part one.

In the second part, the project introduces `list` which is one of the useful data structure in this project, because 'list' is an excellent way to tied item with labels. Besides the basic usage of list, some new concepts of model evaluation are also listed in this part, such as precision, recall and F1 score.

In the third part, the detailed process of naive bayes algorithm is developed, but in this part, the program using the classifier only in identifying which box different balls come from. Begin with the function of 'Counter', the naive bayes algorithm can correctly count the number of a specific item in the whole dataset, which will provide an approach to calculate the conditional probability directly in the next step. Using for loop, the naive bayes program can calculate all the conditional probability of different item by dividing the number of whole dataset to the result of 'counter' function for each item. After getting the conditional probability, the program then calculates the most probable class of the new items. Setting the equal prior probability, the program multiplies the conditional probability to get the final probability of the item in one class. Finally, find out the class that has the largest posterior probability and that class is the result of the naive bayes prediction.

In the fourth part, the program first constructs the 'counter' unigram_probs, which maps each word to its probability. The program then developed into bigram_probs and trigram_probs, which maps pairs and triples of words to their probability. This part of the program is a foundation of developing classifier with different structure of sentences.

In the fifth part, the program uses the same train of thought in the third part to the tweeter dataset. Similar as before, we first calculate the prior probability of tweeters in separate class, such as food, medical and so on. After calculating the prior probability, the program then focusses on the posterior probability which equals to the prior probability multiply the conditional probability. After getting the posterior probability, the program finds the max posterior probability of one class and use this class as the result of bayes classifier. Finally, to evaluate the model, the program use methods, such as precision and recall in the second part. The overall precision and recall, are all over 90% which indicate the naive bayes classifier have a good prediction.

In the sixth part, originated from the fourth and fifth part, the program incorporating bigrams with the naive bayes classifier, which may improve the average F1 score of the test set. The results show that two fifth of the F1 score have been significantly improved. To visualize the specific way the prediction improves, confusion matrix is a useful tool. A confusion matrix shows you for each true category c how many of the tweets in c were classified into the five distinct categories. To go further, visualizing individual tweets can help the programmer understand why a classifier made a correct or wrong prediction.

For the error analysis, we want to figure out why the classifier made mistakes. For one possible reason, the way of recognition of the specific words may sometimes make mistakes. For example, we define the word 'milk' as the sign of food category, but actually, some sentence contains 'milk' may not indicate the category of food, such as 'milky way' have nothing to do with food, and this may be one of the possible reason for the classifier making mistakes.

What we have also done?

Besides implementing the codes, we also adopted the original codes to Python 3. The original one is in Python 2 and it is going to be deprecated.

For the bonus part, we have finished a tweet sentiment analysis project, which is attached as well. The project collects 4695447 tweets on the Election Day from 2435717 users. A sentiment analysis is conducted on all tweets, and their political leanings are also analyzed. After that, we will figure out whether the social network can predict the election results.

Jupyter Notebook Viewers

- lesson1_rulebased.ipynb: https://github.com/Yexiaoxing/sailors2017/blob/Miley/lesson1_rulebased.ipynb
- lesson2_evaluation.ipynb: https://github.com/Yexiaoxing/sailors2017/blob/Miley/lesson2_evaluation.ipynb
- lesson3_naivebayes_exercises.ipynb: https://github.com/Yexiaoxing/sailors2017/blob/Miley/lesson3_naivebayes_exercises.ipynb
- lesson4_languagemodel.ipynb: https://github.com/Yexiaoxing/sailors2017/blob/Miley/lesson4_languagemodel.ipynb
- lesson5_naivebayes.ipynb: https://github.com/Yexiaoxing/sailors2017/blob/Miley/lesson5_naivebayes.ipynb

- lesson6_visualization.ipynb: https://github.com/Yexiaoxing/sailors2017/blob/Miley/lesson6_visualization.ipynb