SI 630: Homework 1 – Classification Report

Name: Chenyun Tao Kaggle Username: chenyuntao Uniqname: cyuntao

Task 1: Naive Bayes Classifier

Part 1

The performance of my classifier on the development data with no smoothing in terms of F1 is about 0.07, which is quite low.

Figure 1 is a plot of my classifier's performance on the development data where (i) my model's performance is on the y-axis and (ii) the choice in `smoothing_alpha` is on the x-axis. As I change the value of `smoothing_alpha` from 10^{-4} to 100, the performance first increases, but when `smoothing_alpha` reaches about 1, the performance begins to decrease.

Most practitioners use `smoothing_alpha` = 1 in practice, and this value also gives good performance to me. In fact, the `smoothing_alpha` I finally choose for part 1 is quite close to 1, which is $10^{-0.1} \approx 0.794$. The predictions on the test data using my model with this `smoothing_alpha` receive a score of 0.67937 on Kaggle.

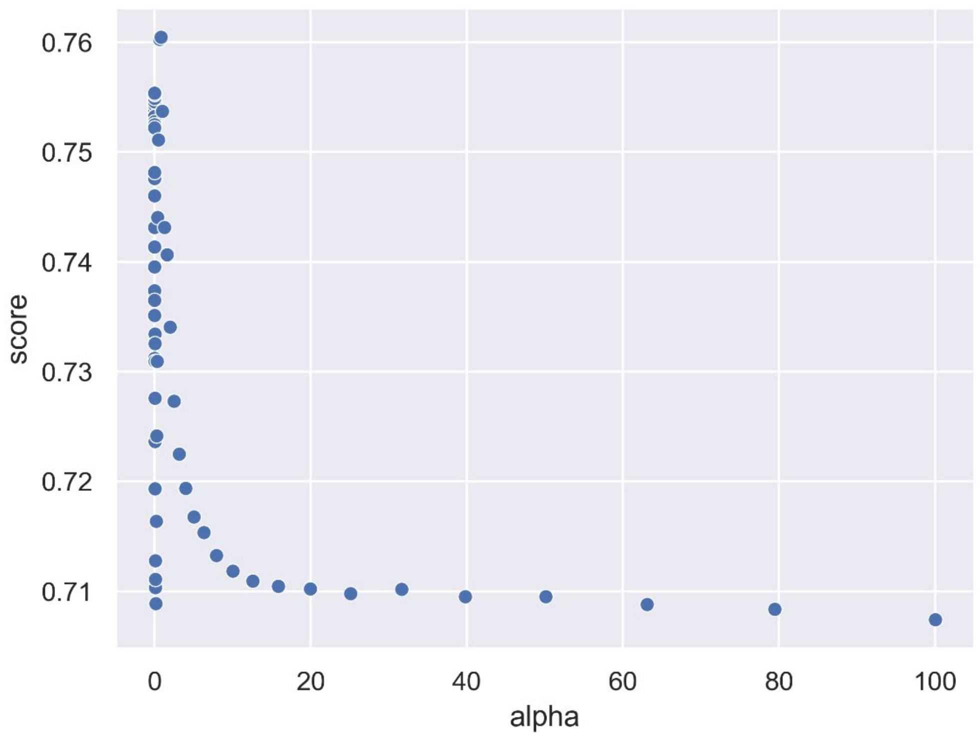

Figure 1: Classifier's Performance on the Development Data (Task 1-1)

Part 2

In the `better_tokenize` function, I fix the following issues, and add some filtering rules:

- The words like “Good”, “good” and “good,” should not be considered as different words. Hence, I transform the words into lowercase, and filter out the punctuations surrounding the words.
- The comparison and superlative forms of adjectives should not be considered as different words from the original adjectives. Hence, I apply the simplest filtering rules to treat words like “lower” and “lowest” to be the same as “low”.
- The web addresses should not be added into vocabulary.

Inspired by SI630 Homework 1 description, NLTK documentation (“nltk.tokenize package”), and some tested ideas by intuition (*i.e.* these ideas raise the score on the development data), I decide to make the above choices.

Figure 2 shows the performance on the development data using the `better_tokenize` method. Compared to Figure 1, we can see the performance increased a bit for the Naïve Bayes classifier. In fact, the performance of the logistic regression classifier also increased a bit with the `better_tokenize` method. This time the prediction on the test data using the best `smoothing_alpha` receives a score of 0.70062 (It was actually not my best Kaggle score on the test data, which was 0.70187, but I decide to adopt the tokenization method receiving this score. This is because in the previous version, I did not consider punctuation in front of the word, and I think this feature should be included).

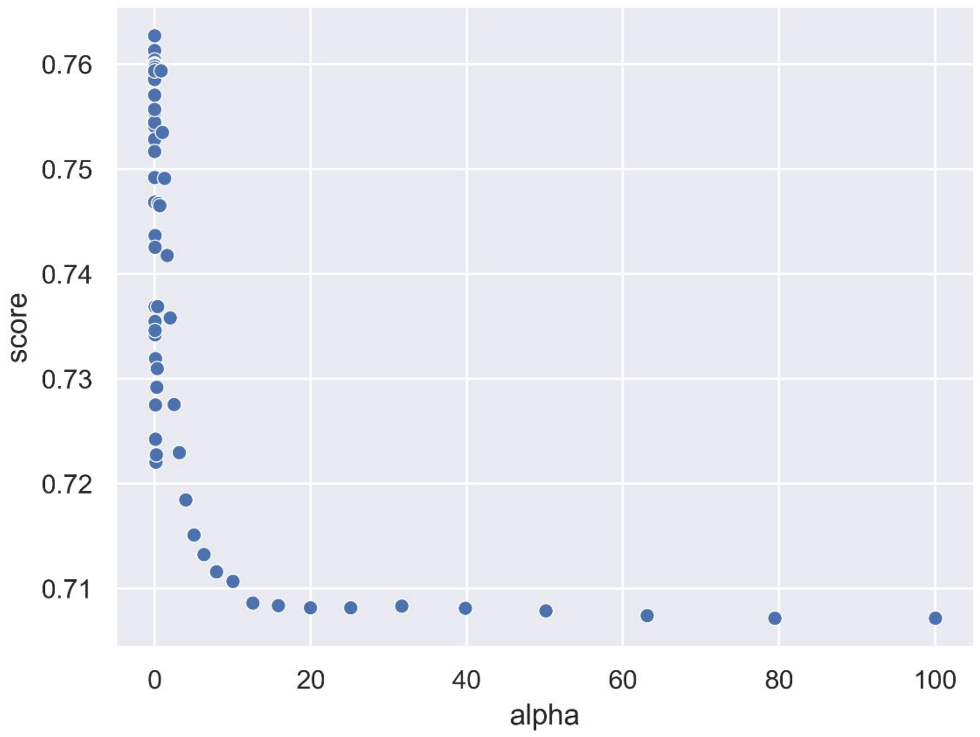

Figure 2: Classifier’s Performance on the Development Data (Task 1-2)

Task 2: Logistic Regression

The leftmost subplot in Figure 3 is plot of the log likelihood for the full data every 100 steps when training my model on the training data with `learning_rate`=5e-5 and `num_steps` = 1000. The model did not converge at any point in this plot.

Then I change the `learning_rate` to a much larger number 5e-3 (corresponding to the subplot in the middle of Figure 3), and a much smaller value 5e-7 (corresponding to the rightmost subplot of Figure 3) and repeat the training procedure. From the 3 curves, we can see that the larger the learning rate, the faster the speed of converging.

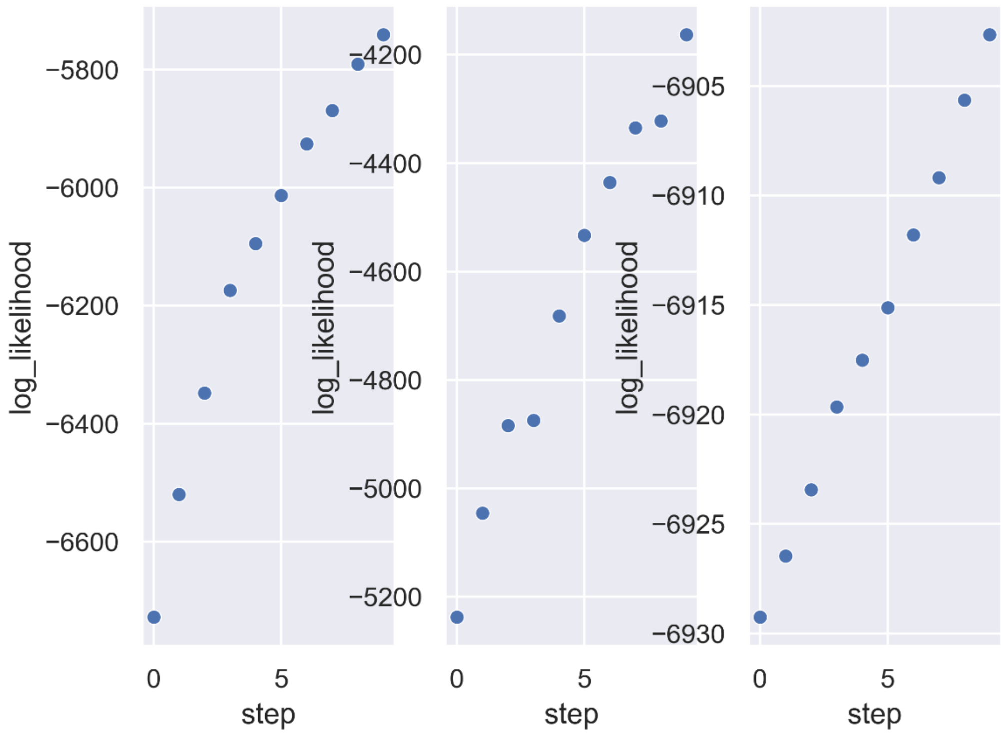

Figure 3: Plot of Log Likelihood with Different Learning Rates

Based on these observations, I train several models until them converge. As my model runs slowly, I just upload 4 various predictions to Kaggle to get the scores there, with `learning_rate` of 5e-4, 1e-3, 5e-3, and 1e-2. Among them, 5e-3 is the best, and the predictions on the test data receive a score of 0.90562 on Kaggle. We can see this performance is much better than that of the Naïve Bayes classifier.

References

“nltk.tokenize package.” *NLTK 3.5 Documentation*, www.nltk.org/api/nltk.tokenize.html#nltk.tokenize.api.StringTokenizer.tokenize.