Rahul Ramesh - CSE 595 HW1

Kaggle username: rahulrameshaz

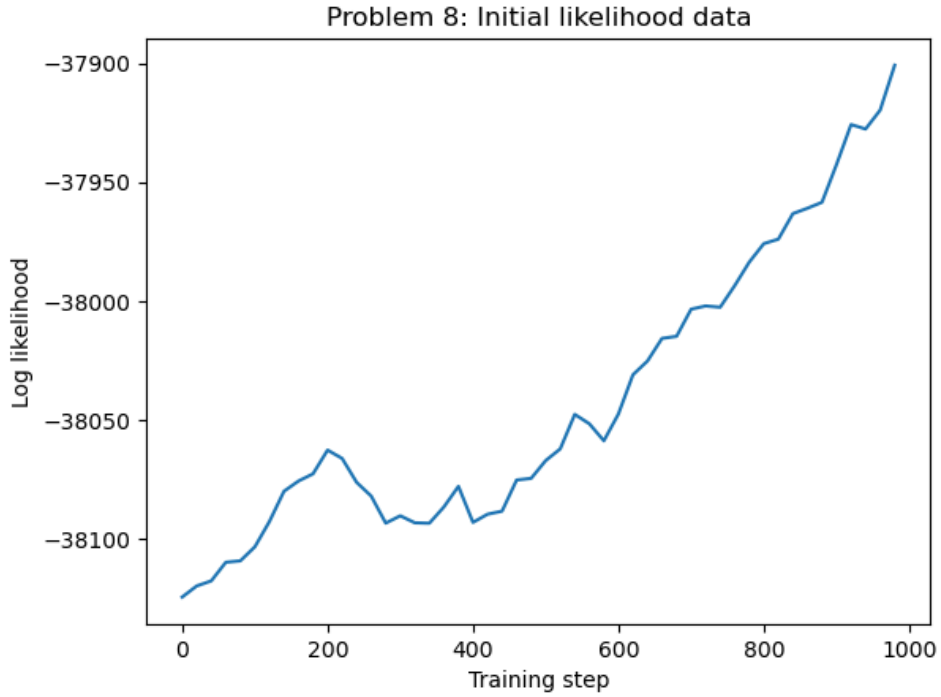
Note: Some of the charts in this writeup look very slightly different from my final submission notebook. If there are any issues, I have an older copy of the notebook I can show you.

Problem 2:

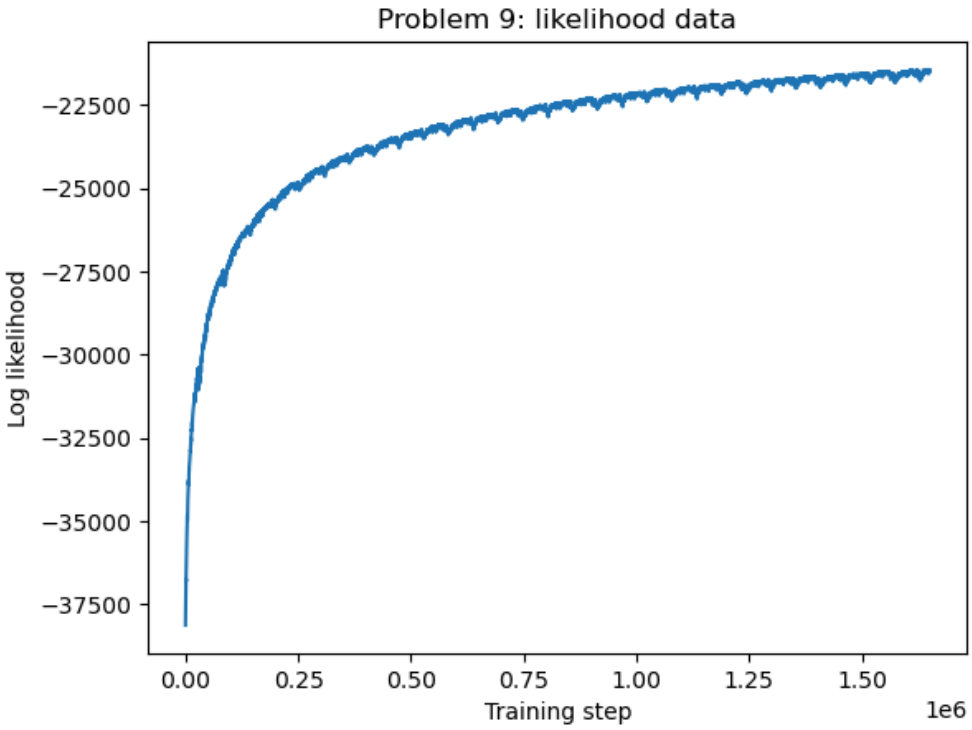
Firstly, I converted everything into lower case to standardize “Good” vs “good”. I also removed all punctuation, numbers, etc. replacing them with spaces. This had the effect of getting rid of any contractions and squishing them into one word. Finally, I removed all words with less than 4 characters, to get rid of short irrelevant words like “the” or “can”. The result of this is that the tokens remaining have more meaning in theory than the tokens with the worse tokenize function.

Problem 8:

As seen by the following chart, the log likelihood does not stabilize when running on only 1000 messages. This implies that several more steps could be helpful in order to achieve a stable result.

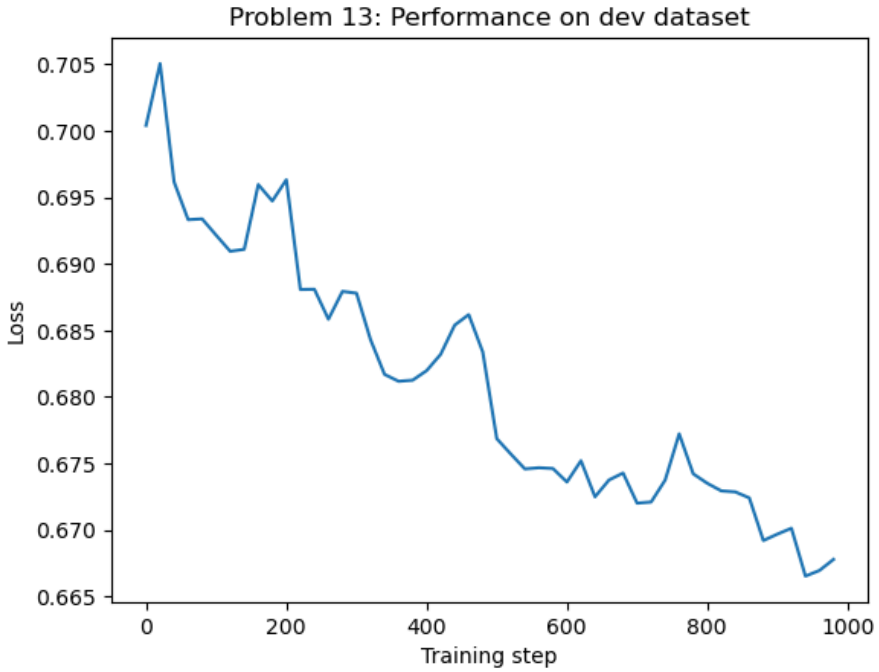


Problem 9:

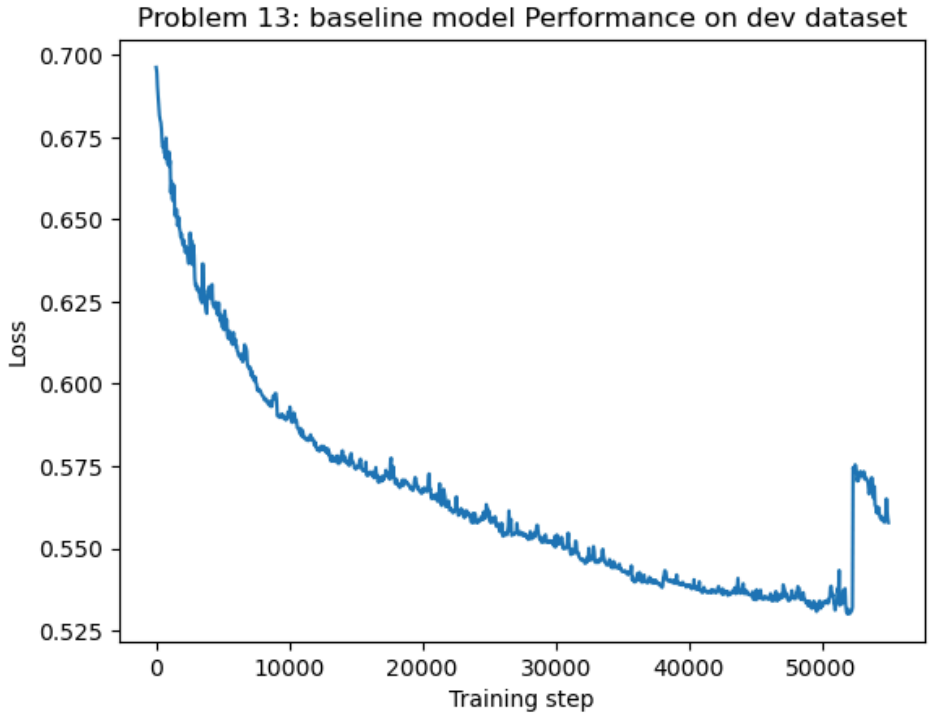


F1 Score on validation dataset: 0.7739

Problem 13:

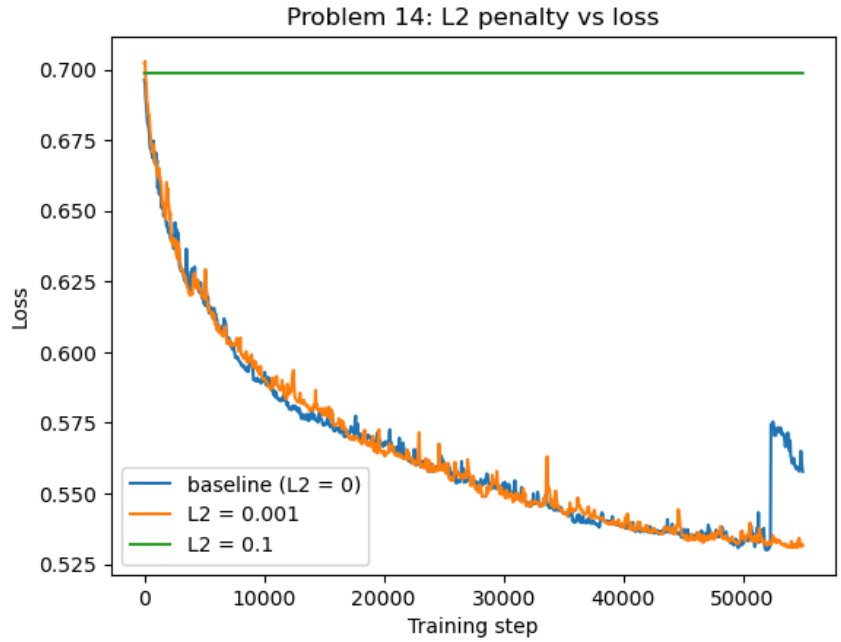


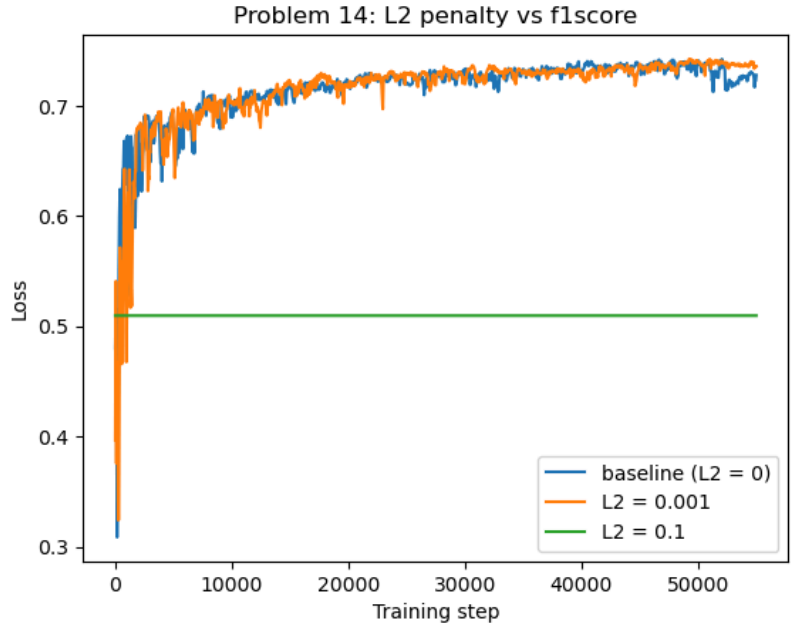
This plot confirms that our loss is decreasing, which confirms our training procedure is moving in a promising direction. Here's what happens when running for one epoch:



Problem 14:

It appears that setting a smaller L2 value (0.001) appears to prevent the model from overfitting after many iterations (~50000), as the loss function on the dev set is more stable. However, a large L2 value (0.1) prevent the model from learning any information about the chosen features, leading to a flat loss function and f1score.

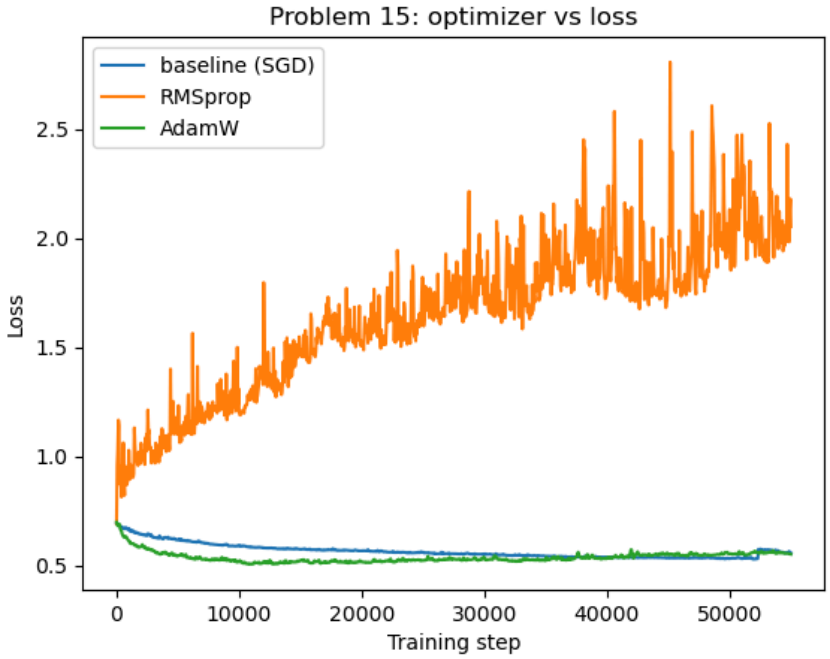


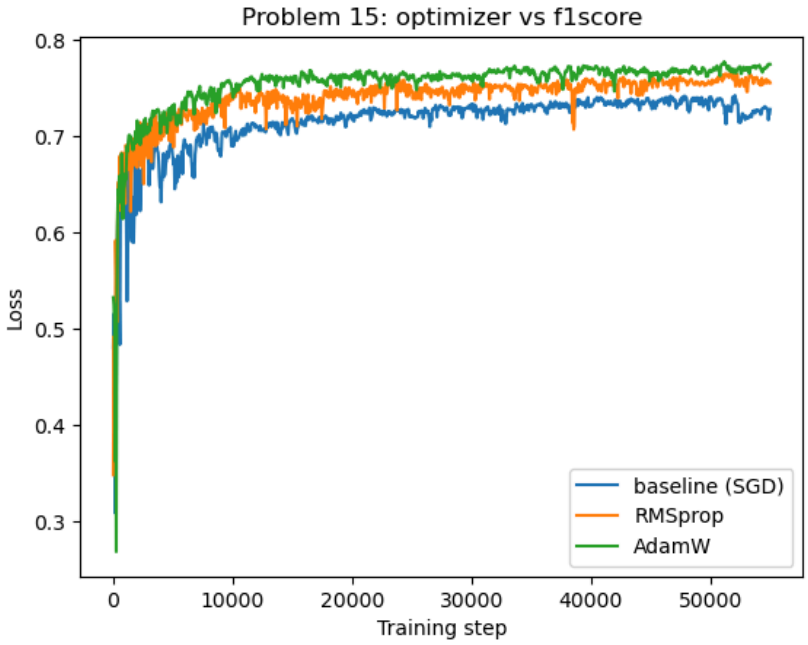


Problem 15:

AdamW clearly converges faster than the baseline model. Meanwhile, RMSprop does not appear to converge, and in fact increases in loss significantly, which indicates its weakness as a model for the text classification problem.

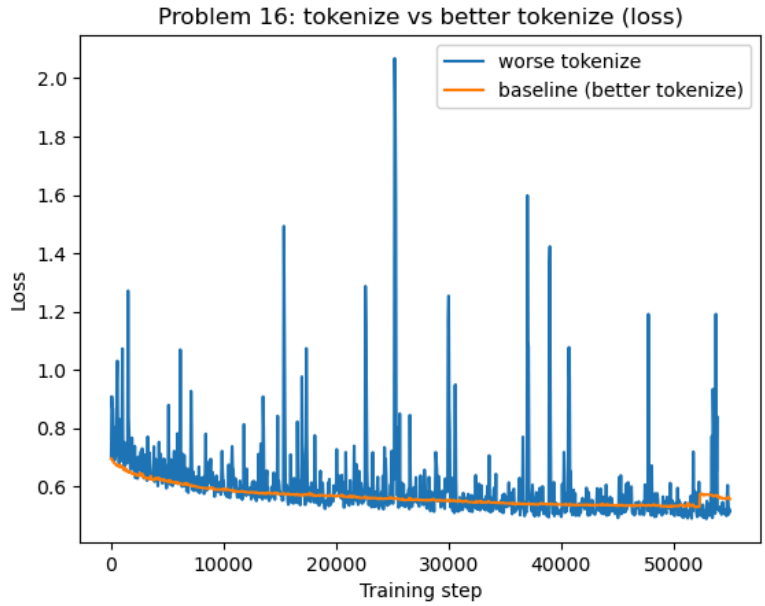
In terms of overall model performance, AdamW appears to perform the best on the dev dataset. RMSprop does actually predict the dev dataset better than SGD as well.

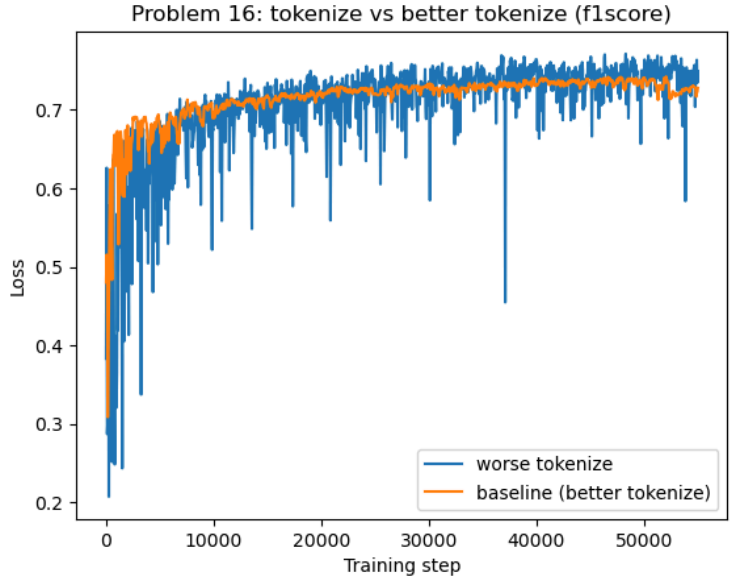




Problem 16:

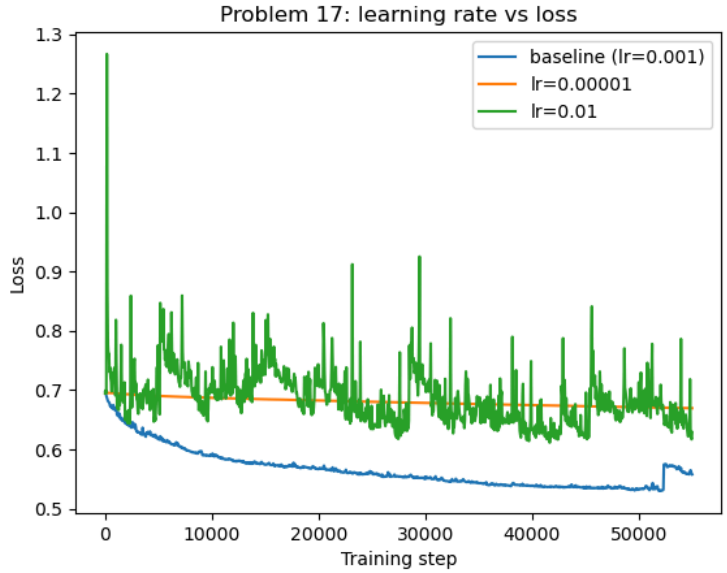
Using a more strict (i.e. better) tokenize function appears to keep the data very stable. Loss in general monotonically decreases, and f1score increases, over the number of iterations trained. However, the basic tokenize function (i.e. worse) has massive spikes in both loss and f1 score while following the same overall trend as the better function. This data would indicate that the worse tokenize function has a tendency to overfit the model to the training set and perform much more inconsistently with the validation dataset.





Problem 17:

Clearly, the quickest convergence is seen with the baseline learning rate. With a high learning rate, the loss is erratic and appears to slowly decrease. With a low learning rate, the loss is predictably stable and does not even converge, even after an epoch. Interestingly enough, the f1 score of the low learning rate model appears to lag behind and only start to increase after many iterations. However, both the baseline and high learning rate models perform well with f1 score, though the high learning rate model has higher variation.



Problem 18:

Submitted on kaggle (~0.77 best attempt)

Problem 19:

L2=0.001 ['ingredients', 'instructions', 'including', 'recipe', 'sure', 'overall', 'enjoy', 'novel', 'significant', 'else']
L2=0.1 ['again', 'generate', 'cool', 'draw', 'strongly', 'lack', 'lots', 'argue', 'healthy', 'achieve']
No L2 penalty ['ingredients', 'instructions', 'including', 'sure', 'recipe', 'overall', 'enjoy', 'novel', 'significant', 'been']

Clearly, no L2 penalty and L2=0.001 appear to have a lot of similarities when looking at the words with the highest beta values. In fact, they are nearly the same, with some slight changes in ordering. L2=0.1 appears to have a completely different set of words. It appears that the former's words relate to recipes, instructions, etc. when analyzing the training data.
I don't think the words particularly work well as features. To be honest, I don't use LLMs too much so I may not know the patterns as well as others. I do notice a couple words like "sure" that could be evidence of LLM usage; an LLM might say "sure, I can do that for you" in response to a query. With the large L2 penalty data, I don't see any words that are particularly compelling features.

In my opinion, the L2=0.001 model would generalize the best. This is because, when looking at the charts for problem 14, the model's loss on the dev dataset tends to decrease the most consistently, and the f1 score tends to not vary as much. I do admit that this conclusion isn't really supported by the provided words above, which implies that the model without L2 penalty still performs adequately on a feature by feature basis.