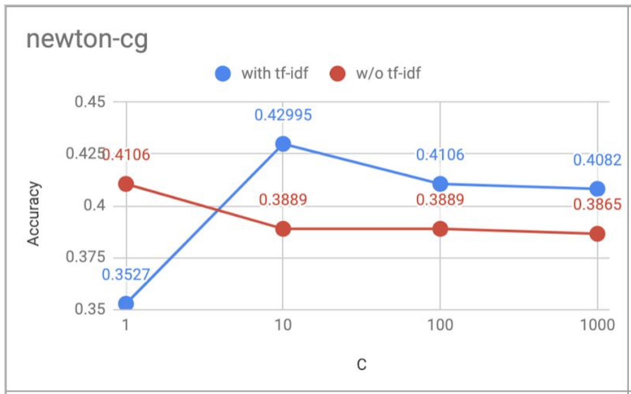
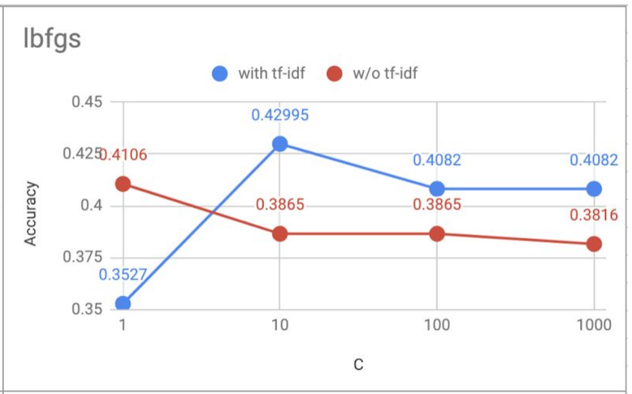
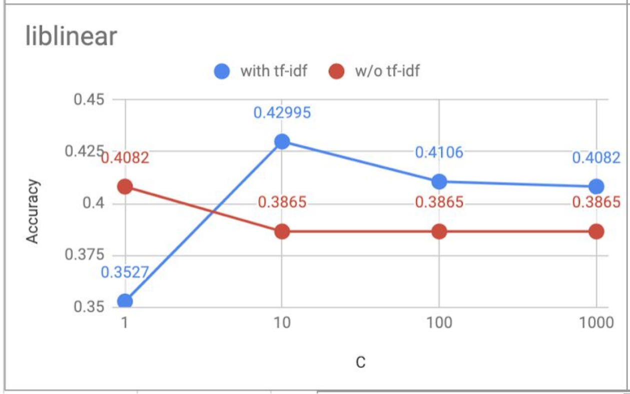
Caleb Nelson

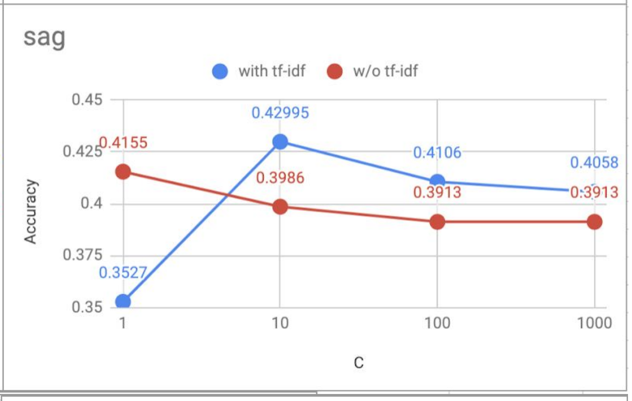
CS 272 HW 1

For the first part, I improved the basic logistic regression model in three different ways - modifying the value of the regularization parameter C, trying different solvers for the optimization algorithm, and using TF-IDF weighting. The different values for C I experimented with were 1, 10, 100, 1000. I tried values of C lower than 1, but the model would not converge and the accuracy on the training set would barely get past 50%, so I didn't take those results into account. Experimenting with these values of C, the 5 different solvers that the sklearn implementation of logistic regression implements, and whether or not to use TF-IDF weight gave me 40 different models in total to evaluate. The 5 graphs on the following page show the accuracy on the dev set for all 40 of these models, with each graph referring to a different solver. The default model uses the liblinear solver without TF-IDF and C=1, which you can see on the graph corresponds to an accuracy on the dev set of .4082. Most models did about as well or worse than this number, but notably, using the saga solver with TF-IDF and a C value of 10 led to an accuracy of .4324, which is higher than both the default and the rest of the models. In general, increasing the value of C led to worse results and adding TF-IDF led to better results. However, when C=1 the model will not converge with TF-IDF, which causes this model to achieve poor performance on both the training set and the dev set. This leads to a combination of TF-IDF and a C equal to 10 to perform the best for all solvers. While choice of solver did not affect results as much as changing the C value or adding TF-IDF, the saga solver still had a noticeably higher peak than the other solvers.



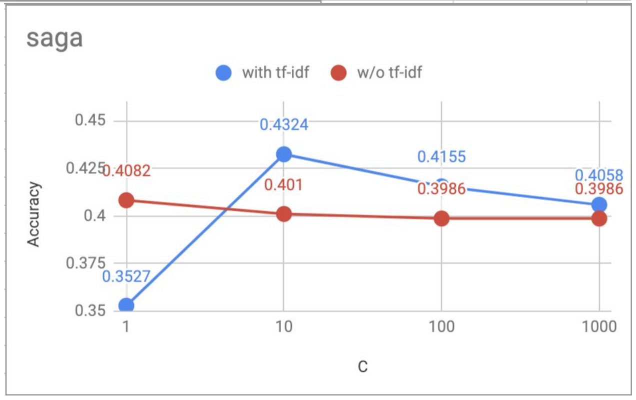






All accuracies listed are on the dev set.

Graphs are separated by solver, which is listed in the top left corner for each graph.



For the second part, I tried a couple of things. I first tried expanding the labelled data using the repeated classification as described in 2.2.1 in the homework, but I quickly discovered that it wasn't going to work. After talking about it with some classmates, we figured that since the classifier only achieved a 43% accuracy at its peak, more than half of the classifications would be wrong. This would cause a negative feedback loop, as garbage into the classifier would result in garbage out, making it worse with repeated iterations. Next, I tried using label propagation using sklearn's label propagation package. However, when I tried to pass in all of the data at once, its runtime became unacceptably long. This is because for one step of label propagation, it needs an NxN matrix where N is the total number of samples. With 47712 total samples, this matrix would have over 2 billion elements, making it impossible to store in memory. In order to get around this, I took the 4370 labeled samples and combined them with all but 2 of the unlabeled samples to get 47710 samples, and then split them into 10 batches of 437 labelled samples and 4334 unlabeled samples each. While suboptimal, this led to a much more manageable runtime and the label propagation was able to be run in around 40 minutes time. Unfortunately, the model is still training so I do not have its results yet, but I will upload the model to kaggle as soon as it finishes training and am willing to send you the results if needed.