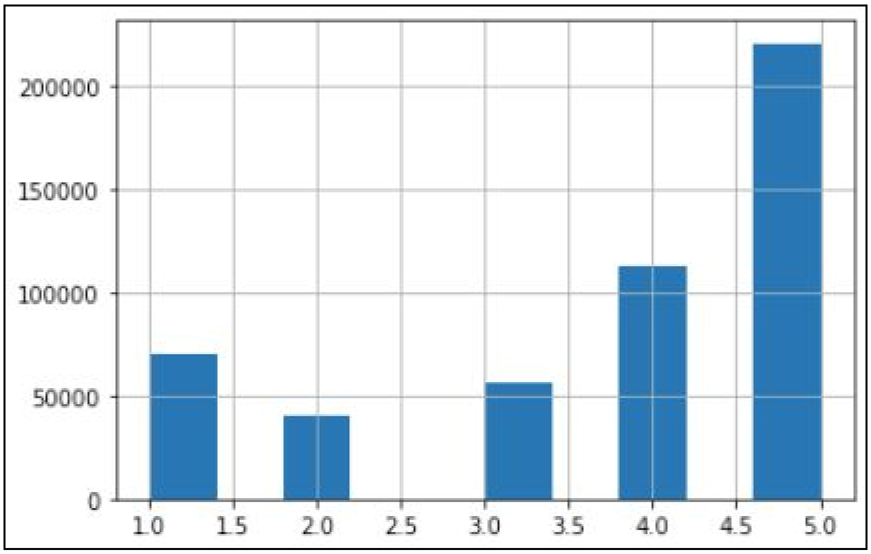
Homework 3 - Text Analytics - GitHub Link
Kristiyan Dimitrov - ktd5131

For this homework I am working with the Yelp review dataset, which contains just over 8 million reviews with their corresponding text and 1-5 star ratings. The json review dataset is ~6.33 GB and contains. Due to computational resource constraints I will be working with just the first 500,000 reviews. The distribution of the ratings is show below:



1: 14% | 2: 8% | 3: 11% | 4: 23% | 5: 44%

Within these first 500k reviews there are 26.6 million words with 150 million characters for an average word length of 5.63 characters. I preprocessed these reviews using gensim.utils.simple_preprocess, which basically tokenizes and converts to lowercase letters. Then, I removed all stopwords. This took ~25 minutes using the multiprocessing module with a 12 core CPU. I saved the preprocessed reviews to a pickle file which is ~422 MB.

You can find code for the above in dataset_stats.py and preprocessing.py

I prepared a few different corpuses for my model training: unigram only & unigram + bigram; Bag of Words & TF-IDF. In the end I have a total of 4 corpuses to train on. It's important to note that I removed any words from the dictionary that occur in more than 50% of documents and fewer than 100 times across all documents. This dramatically reduced the size of the vocabularies and leaves only relevant words. My unigram dictionary has 11,147 words and my unigram+bigram dictionary has 34,936 words. Code in generate_corpuses.py.

I tried using Cross-Validation, but it took too long, so I resorted to a standard train-test split of 3:1. I trained a Logistic Regression & Linear SVM with a few different values for the regularization strength parameters. For the Logistic Regression I also tried L1 & L2 regularization as well as different number of iterations for the solver. The results are summarized in the following tables. The overall conclusion, however, is that the tuning parameters don't really influence performance in a significant way and the defaults would probably be ok. Furthermore, the TF-IDF unigram+bigram corpus consistently did slightly better than the rest. The code for generating the final SVM model on the TF-IDF unigram+bigram corpus is in train_model.py. You can use predict_svm.py to make rating predictions for new reviews. I recommend checking the README file in the repo. Code for all the below iterations, model exploration, and results can be found in Homework3.ipynb.

Logistic Regression on 4 corpuses, with different number of solver iterations

corpus	n_iter	Train Acc. %	Test Acc. %	Prec. %	Recall %	F1
Unigram BOW	100	69.66	66	63.79	66.28	.643
Unigram BOW	200	70.64	66	63.6	65.92	.642
Unigram BOW	500	70.94	66	63.5	65.82	.641
Uni + Bi BOW	100	74.85	67	65.13	66.80	.657
Uni + Bi BOW	200	78.13	66	64.21	65.75	.648
Uni + Bi BOW	300	79.22	65	64.01	65.48	.646
Unigram TFIDF	100	68.96	67	64.88	66.94	.655
Unigram TFIDF	200	69.88	67	64.93	66.97	.655
Unigram TFIDF	300	70.20	67	65.95	66.95	.656
Uni + Bi TFIDF	100	71.74	68	66.11	67.89	.666
Uni + Bi TFIDF	200	74.02	68	66.33	68.15	.668
Uni + Bi TFIDF	300	74.67	68	66.34	68.10	.669

Note: It turns out that Accuracy = Precision = Recall = F1 score when using average='micro'. That's why I am reporting with average='weighted' all metrics except accuracy. Note that the weighted recall is also equal to accuracy.

Observations based on above table:

- More solver iterations increases training accuracy, but this is merely a sign of overfitting; the performance on the test data actually worsens
- The metrics appear to be best for the Uni+Bigram TFIDF Corpus

Logistic Regression with Liblinear solver and L1 regularization

Corpus	C	Train Acc. %	Test Acc. %	Prec. %	F1
Unigram BOW	.5	69.22	66	62.8	.631
Unigram BOW	1	69.59	66	62.69	.631
Unigram BOW	2	69.73	65	62.43	.631
Uni + Bi BOW	.5	75.33	67	64.42	.63
Uni + Bi BOW	1	77.06	66	63.71	.644
Uni + Bi BOW	2	78.04	65	63.00	.637
Unigram TFIDF	.5	67.78	66	63.8	.641
Unigram TFIDF	1	68.87	67	64	.644
Unigram TFIDF	2	69.71	67	63.93	.644
Uni + Bi TFIDF	.5	69.12	68	65.26	.654
Uni + Bi TFIDF	1	71.54	68	65.8	.661
Uni + Bi TFIDF	2	74.41	68	65.41	.659

Note: Smaller C ⇔ Stronger Regularization

Observations:

- I repeated the exact same calculations with L2 regularization instead of L1 regularization. The performance was almost identical
- Overall, it does not appear that regularization is improving the model quality.
- It appears that I need more regularization for the uni + bi gram corpuses; This makes sense - I have a lot more features there as opposed to the unigram only. The result it, it appears to be overfitting (training accuracy significantly higher than test accuracy)

LinearSVC with default parameters and L2 regularization

Corpus	C	Train Acc. %	Test Acc. %	Prec. %	F1
Unigram BOW	.5	69.19	65	61.87	.621
Unigram BOW	1	69.18	65	61.85	.621
Unigram BOW	2	69.34	65	62.08	.625
Uni + Bi BOW	.5	78.08	64	62.2	.629
Uni + Bi BOW	1	78.24	64	62.0	.627

Uni + Bi BOW	2	78.21	64	61.8	.626
Unigram TFIDF	.5	69.6	66	63.2	.636
Unigram TFIDF	1	69.78	66	63.1	636
Unigram TFIDF	2	69/.89	66	63.0	.636
Uni + Bi TFIDF	.5	75.85	67	64.5	.650
Uni + Bi TFIDF	1	76.76	66	64.1	.647
Uni + Bi TFIDF	2	77.37	66	63.7	.644

Observations:

- Once again, the addition of regularization doesn't impact results significantly
- The Uni & Bi gram models overfit
- Models on the TFIDF corpus perform slightly better.

Example predictions with the final SVM model based on the entire TFIDF uni_and_bigram_corpus:

- This Italian restaurant really sucks. The salad was terrible. --> Predicted rating is: 1
- I really loved my time at this bakery. The bread was fluffy and tasty. Will definitely come back for more. --> Predicted rating is: 5
- What did I think of this restaurant? I'm not really sure. --> Predicted rating is: 4

These are ratings I just made up. I'm rather happy with the predictions!