Terrence Klock
CS 6375.004

Project 1: Naive Bayes and Logistic Regression for Text Classification

The series of programs covered by this report are MultinomialNaiveBayesHamSpam.py, DiscreteNaiveBayesHamSpam.py, and SGDClassifierHamSpam.py. Each runs the corresponding algorithms and returns the confusion matrix information.

I was not able to get Bernoulli working for the SGDClassifier due to issues with the arrays. Rather, getting the data into a format where the SGDClassifier would accept it was difficult.

The results are listed below:

	Accuracy	Precision	Recall	F1 Score
Multinomial Naive Bayes	0.6908	0.9029	0.6059	0.7251
	0.7218	0.9216	0.6753	0.7795
	0.7219	0.5039	0.4210	0.4587
Discrete Naive Bayes	0.8004	0.7903	0.9580	0.8661
	0.8180	0.8042	0.9914	0.8880
	0.7348	0.6818	0.0987	0.1724
Log Regression (Bag of Words)	0.7522	0.7321	0.9967	0.8441
	0.5648	1.0	0.4022	0.5737
	0.5948	0.4081	0.9934	0.5785
Log Regression (Bernoulli)	0.9452	0.9578	0.9609	0.9593
	0.9393	0.9369	0.9827	0.9593
	0.9705	0.9594	0.9342	0.9466
SGDClassifier	0.9118	0.9240	0.9576	0.9404
	0.9788	0.9883	0.9826	0.9854
	0.9852	1.0	0.8384	0.9683

When implementing Log Regression, I split the training and testing sets into 70% and 30% as suggested, using the 30% to validate lambda. I tested the data with lambdas of 0, 5, and 10 and received these results: {0: 0.7037037037037037, 5: 0.7037037037037037, 10: 0.7185185185185186}

I performed this test on the other datasets as well as with other lambda values, but 10 remained the overall best for accuracy. I considered implementing in the code a ‘true lambda optimizer’ that would use multiple logistic regressions to narrow down the best possible lambda for any dataset, but the runtime was already so unnecessarily long. This idea also impacted the maximum iterations I put on the regression, which ended up being 1000 as I didn’t notice much accuracy loss compared to even 10000, and it helped my sanity greatly.

For the SGDClassifier, I ran several option settings on each of the datasets to try and optimize the results. Hinge loss and L2 regularization seemed to perform best (and were notably the default). An alpha of 0.1 worked best, with alpha =1 and alpha = 0.01 performing notably worse. Increasing the maximum number of iterations did increase accuracy and precision, but only marginally. Increasing to values over 100 also sometimes lowered the results. As such, I settled on 15 iterations since it gave good results while not taking very long.

Bernoulli tended to perform better than Bag of Words, likely due to the nature of an email tending to repeat words that throw off the data considerably. The presence or non-presence of a term often matters more. For instance, a spam email may say ‘free money’ once for 1 word each - but those words are much more important for determination than that metric would suggest. For Bernoulli, Log-Regression seemed to perform best, but it didn’t work well on Bag of Words. Rather, SGDClassifier performed best for it.

Bag of Words for Logistic Regression was so bad, in fact, that for most of the datasets Multinomial Naive Bayes performed better than it with accuracies ~14% higher. Multinomial Naive Bayes was also much more precise reaching ~90% precision on two of the datasets. The low precision on the Logistic Regression model does suggest however that changing the maximum number of iterations or the learning rate may have produced better results.

Logistic Regression did outperform Discrete Naive Bayes when both were using Bernoulli. Boasting much accuracy, precision, recall, and F1 scores all above 90%, its safe to say that Logistic Regression works extremely well for Bernoulli, especially compared to the accuracies ~80% that Discrete Naive Bayes gave. However, Discrete Naive Bayes took fractions of the time compared to Logistic Regression while still maintaining usable performance.

The libraries I used for the project were Numpy, Pandas, and Scikit-Learn