Josh Veltri (jkv13)
Justin Wang (jsw104)

EECS 440 Programming Assignment 3

(a) *What is the accuracy of naïve Bayes with Laplace smoothing and logistic regression (λ =1) on the different learning problems? For each problem, perform a t-test to determine if either is superior with 95% confidence.*

		Voting	Volcanoes	Spam
Naive Bayes with Laplace Smoothing and 15 bins	Average Accuracy	0.9373	0.6294	0.6822
	Standard Deviation	0.0279	0.0255	0.00687
Logistic Regression with *λ =1*	Average Accuracy	0.9254	0.7357	0.3751
	Standard Deviation	0.0277	0.00895	0.00029

We used 5-fold cross validation, which means our measurements have 4 degrees of freedom. This means that the 95% confidence interval extends for +/- 2.132 standard deviations from the mean (value from a t-test table). Using the data above, this yields the following 95% confidence intervals:

Voting
Naive Bayes: 0.9373 +/- 2.132 * 0.0279 = 0.9373 +/- 0.05948 = [0.8778, 0.9967]
Logistic Regression: 0.9254 +/- 2.132 * 0.0277 = 0.9254 +/- 0.05906 = [0.8661, 0.9845]
These confidence intervals overlap, so the two classifiers are not statistically different at the 95% confidence level

Volcanoes:
Naive Bayes: 0.6264 +/- 2.132 * 0.0255 = 0.6264 +/- 0.054366 = [0.5720, 0.6808]
Logistic Regression: 0.7357 +/- 2.132 * 0.00895 = 0.7357 +/- 0.0191 = [0.7166, 0.7548]
The confidence interval for Logistic Regression lies entirely above that for Naive Bayes. Thus, for the volcanoes data, the Logistic Regression classifier is superior at the 95% confidence level.

Spam:
Naive Bayes: 0.6822 +/- 2.132 * 0.00687 = 0.6822 +/- 0.01374 = [0.6685, 0.6959]
Logistic Regression: 0.3751 +/- 2.132 * 0.00029 = 0.3751 +/- 0.000618 = [0.3745, 0.3757]
The confidence interval for Naive Bayes lies entirely above that for Logistic Regression. Thus, for the volcanoes data, the Naive Bayes classifier is superior at the 95% confidence level.

(b) *Examine the effect of the number of bins when discretizing continuous features in naïve Bayes. Do this by comparing accuracy across several different values of this parameter using volcanoes.*

In the following measurements, Laplace smoothing was used.

Number of Bins	Volcanoes Accuracy	Area Under ROC
2	0.6218	0.6877
6	0.6290	0.6841
10	0.6303	0.6786
17	0.6308	0.6730
25	0.6285	0.6460
35	0.6321	0.5884
50	0.6456	0.4667

We also include Area-Under-ROC measurements in the above table because we found that the accuracy measurement alone did not tell the full story. The overall accuracy measure trends upwards as the number of bins is increased. However, we observed that this rising accuracy achieved, is for larger numbers of bins, achieved by classifying nearly all examples as “negative”. This lack of sophisticated decision logic is reflected by the low Area-Under-ROC measures for the larger numbers of bins. The main takeaway is that that there is a balance to be struck in choosing the number of bins - more bins can increase accuracy, but too many bins will sacrifice desirable ROC characteristics.

(c) *Examine the effect of m in the naïve Bayes m-estimates. Do this by comparing accuracy across m=0, Laplace smoothing, m=10 and 100 on the given problems.*

In the following measurements, 15 bins were used when discretizing continuous features.

m	Voting Accuracy	Volcanoes Accuracy	Spam Accuracy
0	0.9373	0.6632	0.6819
Laplace Smoothing	0.9373	0.6294	0.6822
10	0.9419	0.6303	0.6821
100	0.9256	0.6276	0.6815

For the voting and spam data sets, the highest accuracy was achieved with *m*=10, with both larger and smaller values yielding slightly worse results. The volcanoes data set, on the other hand, had significantly higher accuracy when *m*=0 than for any other value of *m*.

(d) *Examine the effect of λ on logistic regression. Do this by comparing accuracy across λ=0, 1, and 10 for the given problems.*

λ	Voting Accuracy	Volcanoes Accuracy	Spam Accuracy
0	0.9627	0.7731	0.5131
1	0.9254	0.7357	0.3751
10	0.5558	0.6943	0.3750

We find that using a large lambda, such as 1 or 10, results in lower accuracy on all three data sets. This is an intuitively logical result. The confidence output is between zero and one, so the magnitude of the weights can be quite large in comparison. This means the large-magnitude weight penalty was dominating the classification error in the loss function. We suggest that using a lambda smaller than one, perhaps something more like 0.01, would be more effective.

(e) *Write down any further insights or observations you made while implementing and running the algorithms, such as time and memory requirements, the complexity of the code, etc.*

For Bayes, the key idea in terms of minimizing time requirements was to minimize the number of iterations we had to make through the example set in order to generate the frequency table and all of the rest of the probabilities we needed. We did this by iterating through the example set

once per feature and constructing a massive hash of hashes, making sure to organize the hashes so that calculating the probability of a feature given a classification was trivial, and that the overall probability of the classification was trivial as well. This meant that after iterating over all of the features in the schema, that we were ready to evaluate the examples in our test set and each evaluation was essentially a hash table lookup. This however swells the memory complexity, but this was not too much of an issue as we only stored integer counters in the hash tables, which meant that the memory footprint was still fairly minimal overall as these feature hashes were still very small compared to the size of the example set since there are far fewer features than examples.

The logistic regression classifier time complexity is a little more complicated. It basically depends on how long it takes for our weights to converge, meaning that if there is very little change in the weights between successive training iterations, the weights are said to have converged. A lower lambda results in more iterations needing to be completed in order for convergence, whereas a higher lambda results in less iterations needing to be completed. Additionally, we had to tinker a couple of values in order to see the best results; the step size used in our gradient descent, and the max threshold for changing weights. We found the best results for having a step size of 0.0025, and a weightDiff threshold of 0.015, meaning that we only stopped iterating if none of the weights changed by more than 0.015. Additionally, randomizing the weights across a range of -0.1,0.1 reduced the number of iterations needed for convergence. Similar to the Bayes algorithm, the weights we stored essentially add up to the memory footprint of 1 example, so the memory complexity was still dominated by the example set.

In terms of complexity of the code, we have found lots of success with our object oriented approach. We've made our classes as generic as possible which has allowed us to simply copy a few classes over from the previous programming assignments for doing tasks such as input normalization, constructing folds for cross validation, and parsing the command line, letting us focus on the important aspects of the assignment. Additionally we have tried to break up the complex parts of our code as much as possible so it is really easy to pinpoint exactly what lines of code are causing bugs. For this assignment especially, the complexity of our code was not much of a time hindrance, it was mostly just figuring out the derivatives for logistic regression.