CS 498-AML HW2

Paul Novacovici - novacov2
Atul Nambudiri - nambudi2
Raghav Narain - rmnarai2

A. SVM Implementation:
We read in all of the data from the data file. We removed all of the rows with missing elements, and converted everything to numeric.
We then split up this data into three different data sets:

68% Training
16% Validation
16% Testing

We tested out three different values for lambda: .005, .01, and .5. We trained our SVM with each of these lambdas on the training set, and then tested its accuracy on the validation set. We chose the lambda with the highest accuracy and used the a and b we calculated from that trial to test on our testing data.

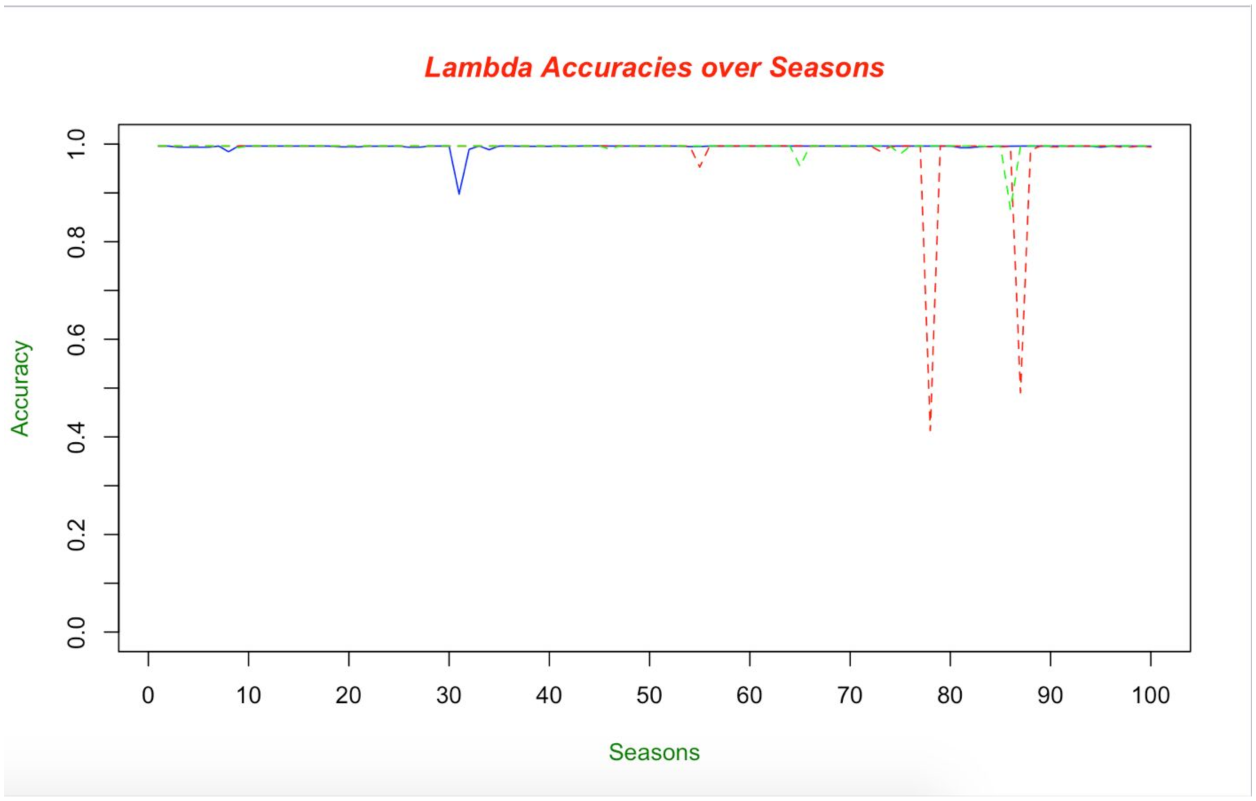
We trained our SVM using Stochastic Gradient Descent. We started out with random values for a and b. We split up our training into 100 seasons. Every season we had 200 steps.

We used a steplength in our implementation, which we defined as m/(season +n). M and n are constants, which we took from the book. M was 1 and n was 50.

For every step, we chose a random row in the training data. We calculated the gradient for that row, and then updated the a and b using this gradient * steplength. This was, as the seasons increase, changes in the gradient affect a and b less.

We touch 200 * 100 * 1 = 20000 data points during training, which is around 62.5% of the data. We used slightly more than 50%, as we used sampling with replacement.

Results:
We got the best results by using a lambda of .5.
We calculated the accuracy after every season for every lambda, by testing our implementation on the validation set. This is the resulting plot of the accuracy:



Blue is a lambda of .005. Red is .01 and Green is .5

We got an overall accuracy of 0.9939819.

B. Naive Bayes Implementation:

We partitioned the data into training and testing data. Training had 84% of the data, whereas, testing had the remaining 16%. We then trained the testing data using the library function “train” with 10 folds. We then tested the model on the testing set using the model the train function created to classify the feature vector.

The prediction accuracy was 57%. This was the confusion matrix:

	Reference	
Prediction	active	inactive
active	59	6618
inactive	16	8885

C. Comparison:

Based on both accuracies we can tell that the SVM implementation is a lot more accurate than the Naive Bayes implementation. Also, running our SVM implementation was significantly quicker than running the Naive Bayes implementation. The reason we believe this happened is due to the fact that the data is lopsided. There are many more inactive data points than active points. This causes the Naive Bayes implementation to choose Inactive too many times which means it misses out on a lot of the Active labels.