Supervised Learning

Zhijian Li

1 Datasets

Diabetic Retinopathy Data Set (DR dataset): This dataset contains data was taken from the Messidor image set, a collection of DR examinations. The goal is to predict whether the image contains signs of diabetic retinopathy. The features are extracted from the images and is either a detected lesion, a descriptive feature of an anatomical part or an image-level descriptor. The dataset contains 1151 instances and 20 attributes.

Mammographic Mass Data Set (MM dataset): This dataset contains data from mammograms. The goal is to predict whether the patient has malignant breast cancer. The features contain BI-RADS assessments along with related measurements of the patient. The dataset contains 961 instances and 6 attributes.

2 Importance

MM Dataset: This dataset is important because it describes one of the more challenging problems in healthcare, and that is cancer detection. There are many statistics on breast cancer and it is a well-known leading cause of death in women. To be able to apply machine learning techniques to either assist or diagnose patients is becoming more helpful as algorithms become more precise, consistent and reliable. One major problem of Mammograms is the false positive prediction rate. Because of this rate, 70% of biopsies from these predictions are unnecessary because the patient's cancer was wrongly classified to be malignant when it was benign.

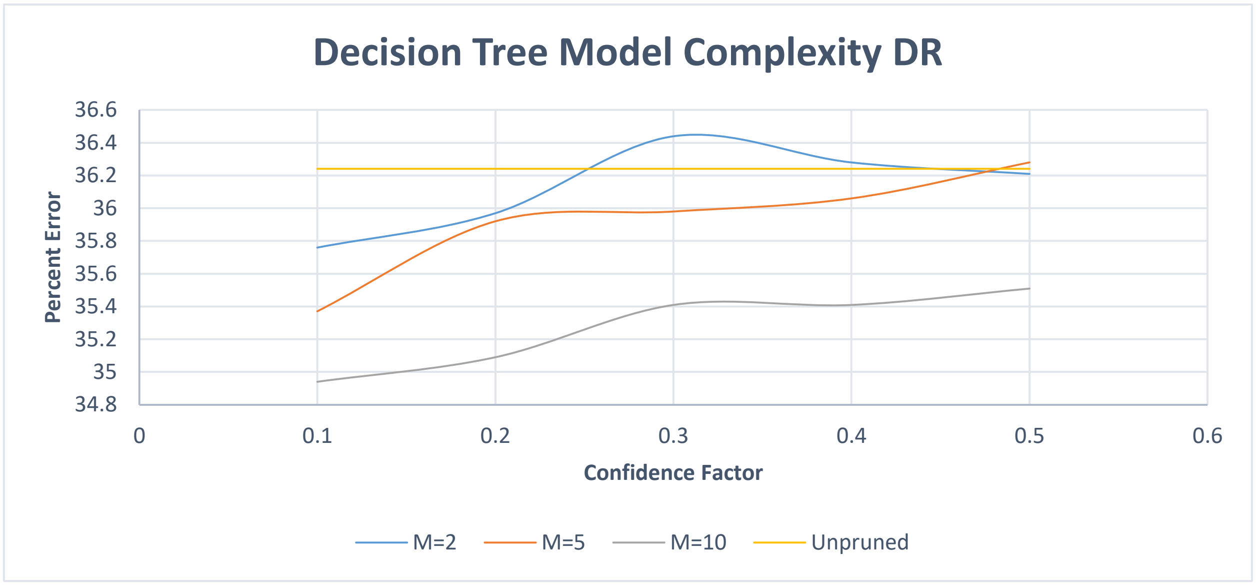
DR Dataset: Diabetic Retinopathy is a diabetic eye disease. Blood vessels at the retina become damaged and ultimately lead to blindness. This dataset is important because it takes images and using feature extraction, predicts whether there are signs of DR. From the data, it is clear that this disease is hard to detect, because pre-screening data of the patient usually does not lead to a correct diagnosis.

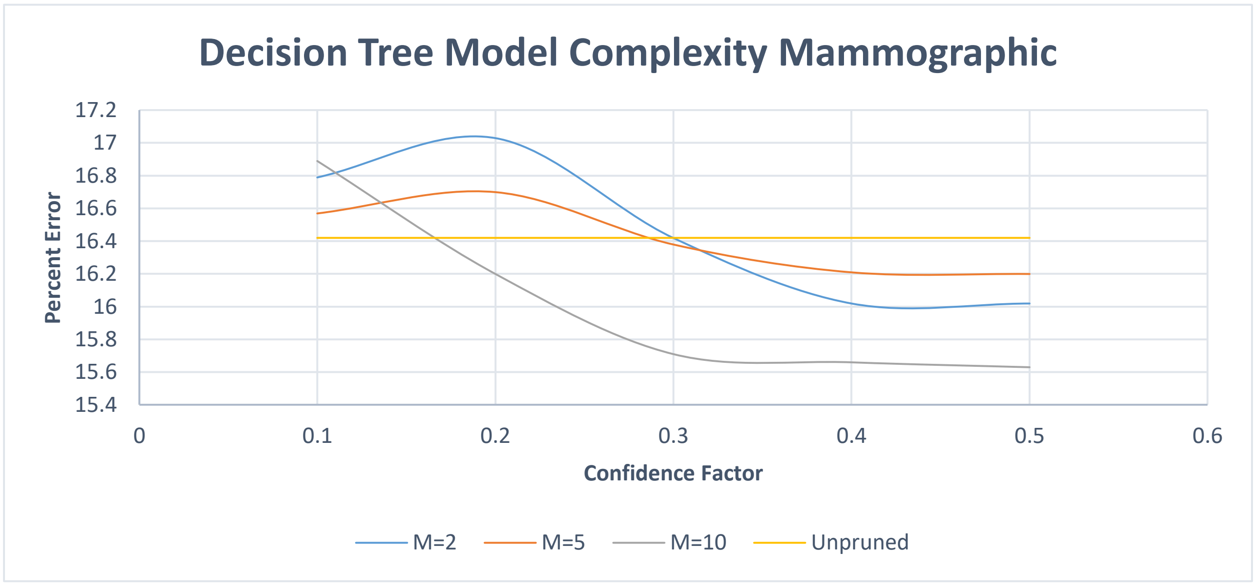
Both datasets are interesting because the algorithms performed differently from MM as opposed to DR.

3 Hyperparameter Tuning

Decision Trees – J48

The specific decision tree algorithm I used was the J48 algorithm from Weka. The parameters I modified were whether the tree is pruned or not, the confidence factor, which is used for pruning, and the minimum objects per leaf (M). All tests used 10 fold cross validation.



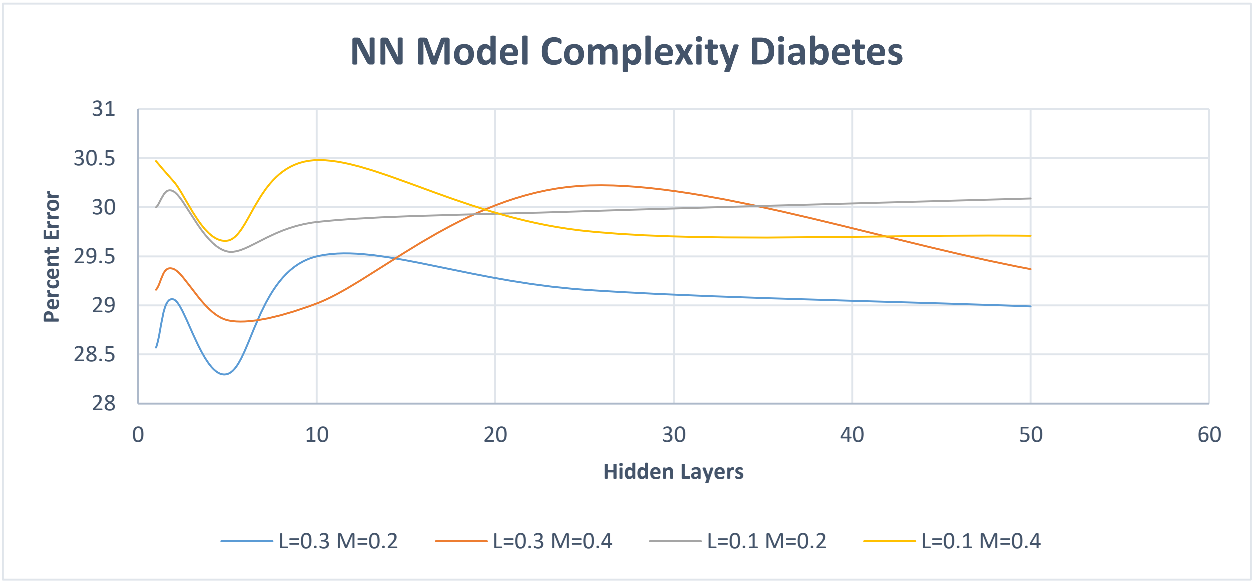


DR Dataset: As seen in the graph, the best parameters for J48 for this dataset is a confidence factor of 0.1 and an M of 10. This is interesting because with lower confidence factor, thee more pruned the tree. It's interesting to see that with the lowest confidence factor, the tree performed the best, which is usually not the case. This is the same for all parameters of M. One reason behind this is the fact that with a higher M, the amount of pruning is reduced, as opposed to a lower M, as it sets a lower bound for how much pruning can be done to the tree. This also reveals something about the data. This also shows that confidence factors > 0.1 lead to overfitting. A possible reason for this result is the fact that there are a lot of features in the DR dataset, and many of those features might not be a good predictor thus by having less pruning, we believe the data more and thus perform worse on the testing set.

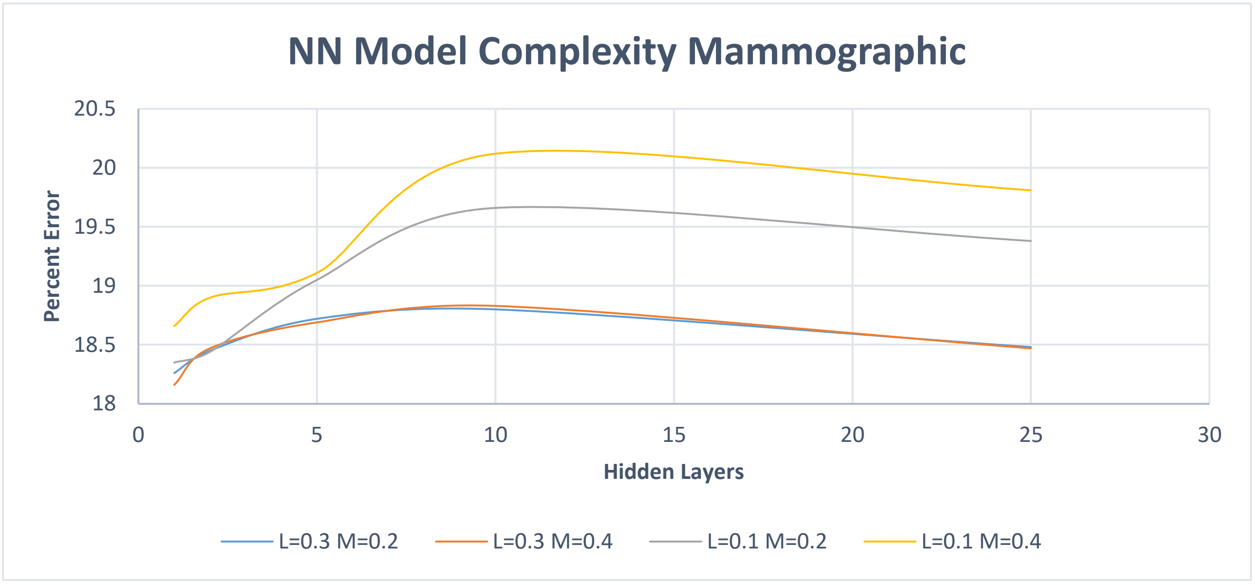
MM Dataset: As opposed to the DR dataset, the best parameters for J48 are a confidence factor of 0.5 and a M of 10. Because of the confidence factor of 0.5, this suggests that a nearly unpruned tree predicts the data the best, although confidence factors of 0.3 to 0.5 have very similar error percentages. Another very interesting thing is that at confidence factors < 0.3, with a M of 2 and 5, the unpruned tree predicted the data much better than a pruned tree. This means that, without a M of 10, there was a much lower limit to how much the tree was pruned and because of this, the tree was pruned so much that its accuracy became worse than if it was unpruned. J48 also did not show signs of overfitting for any parameters, as accuracy continuously decreased. What these things tell us about the data is that, with much fewer attributes, less pruning is advised because there is a higher chance that each attribute contributes more to the prediction.

Neural Nets – Multilayer Perceptron

The specific Neural Net algorithm I used was the Multilayer Perceptron from Weka. The parameters I changed were the amount of hidden layers, the learning rate (L) and the momentum (M).



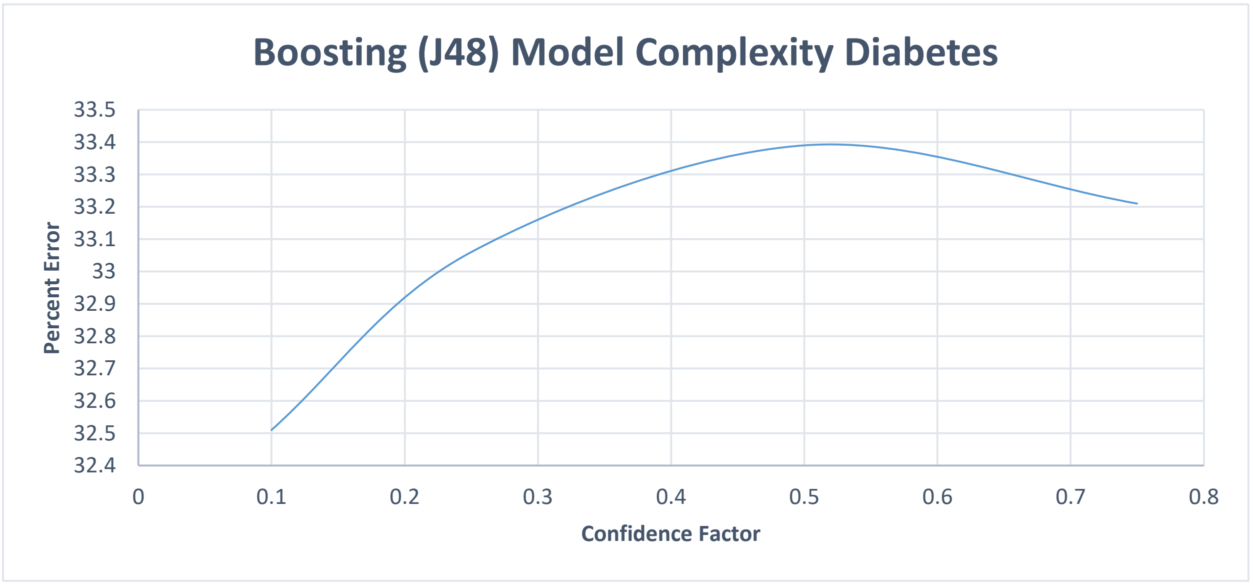
DR Dataset: As seen in the graph, the best parameters for NN is a Learning Rate (L) of 0.3 and Momentum (M) of 0.2 with 5 hidden layers. From the graph, it is clear that NN started to overfit for all parameters of L and M after 5 hidden layers. What is also interesting is that it seems as if the data overfit after 1 hidden layer, but with more complexity, 5 hidden layers, the model actually fit the data better. After 20 hidden layers, the percent error leveled off, with no signs of improving.

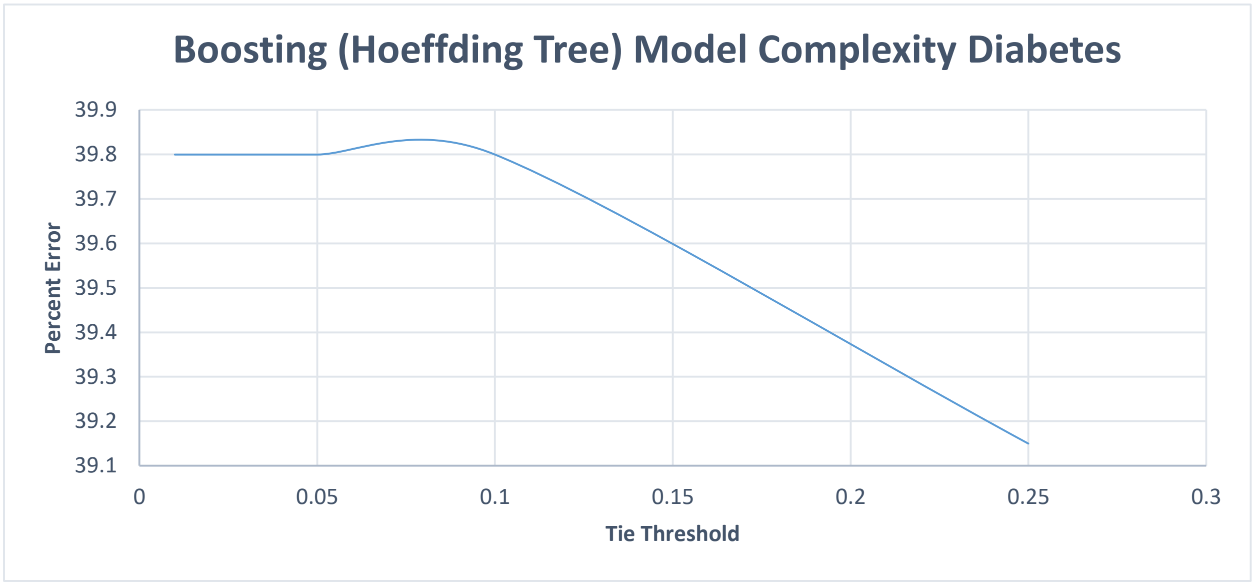


MM Dataset: Similar to the DR dataset, the best parameters for NN for this dataset was close, with a L of 0.3 and an M between 0.2 and 0.4. This is because with an M of 0.2 and 0.4, the model performed equally well. The difference between these two datasets is that for the MM dataset, overfitting occurs very early, at only one hidden layer. This means that with fewer attributes, the MM dataset coincidentally needed less complexity to model the data. While it also seemed like NN overfit after 1 hidden layer for the DR dataset, it actually had the lowest error at 5 hidden layers as opposed to 1 for the MM dataset.

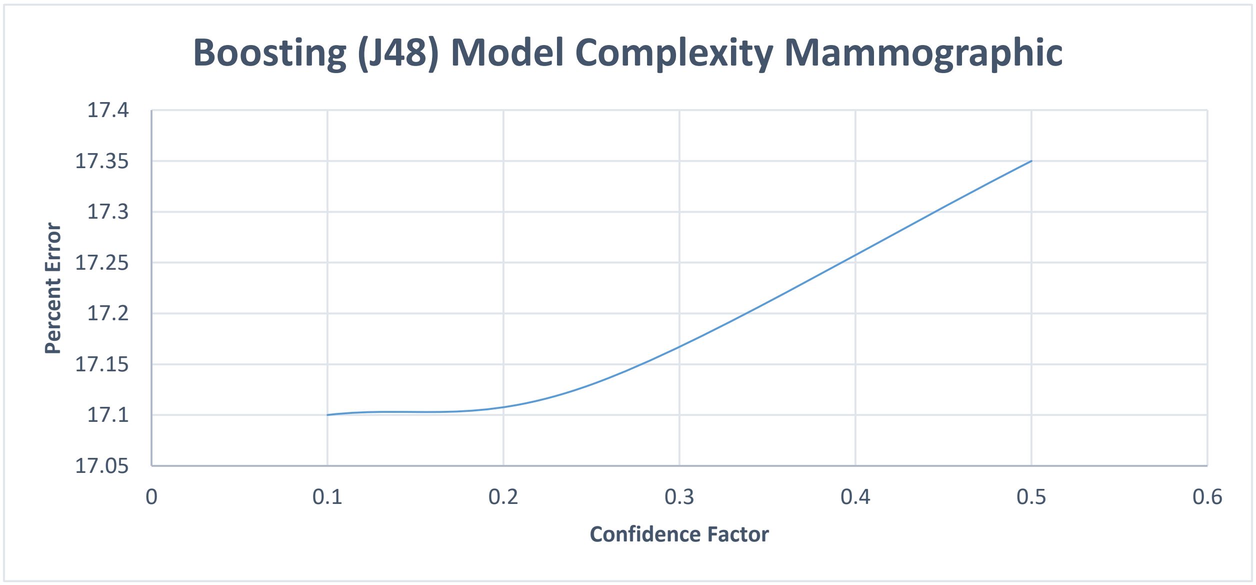
Boosting – ADABoostM1

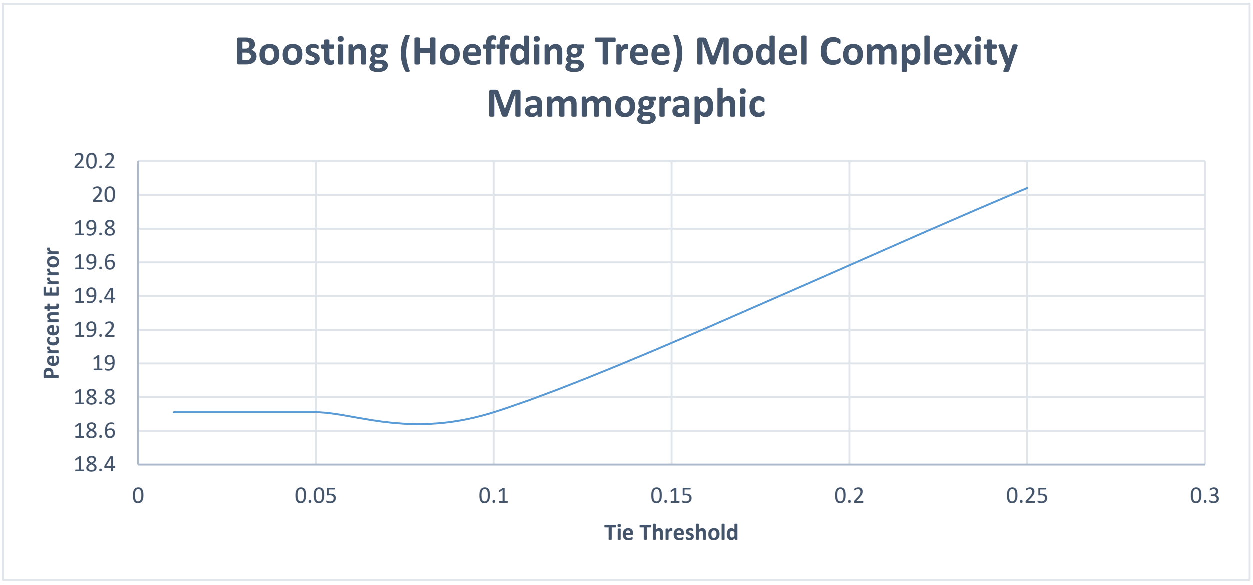
For boosting, I used the ADABoostM1 implementation in Weka. I used two different trees for boosting, J48 and Hoeffding. The parameters I modified for J48 was the confidence factor only. For Hoeffding, I modified the tree height.





DR Dataset: Looking at the graph, J48 and Hoeffding tree had very different curves. J48 immediately started to overfit after a confidence factor of 0.1. This suggests that with less pruning, the algorithm did much worse. Because of boosting, the algorithm overfit the data. With the Hoeffding tree, there was no signs of overfitting at all. But despite that, the Hoeffding tree had much worse error overall. This either suggests that I did not test enough of the range of parameters for the Hoeffding tree to show any signs of overfitting, or that the Hoeffding tree was a bad classifier to use for boosting. The best parameters for the DR dataset was using J48 with a confidence factor at 0.1 with anything greater overfitting the data.

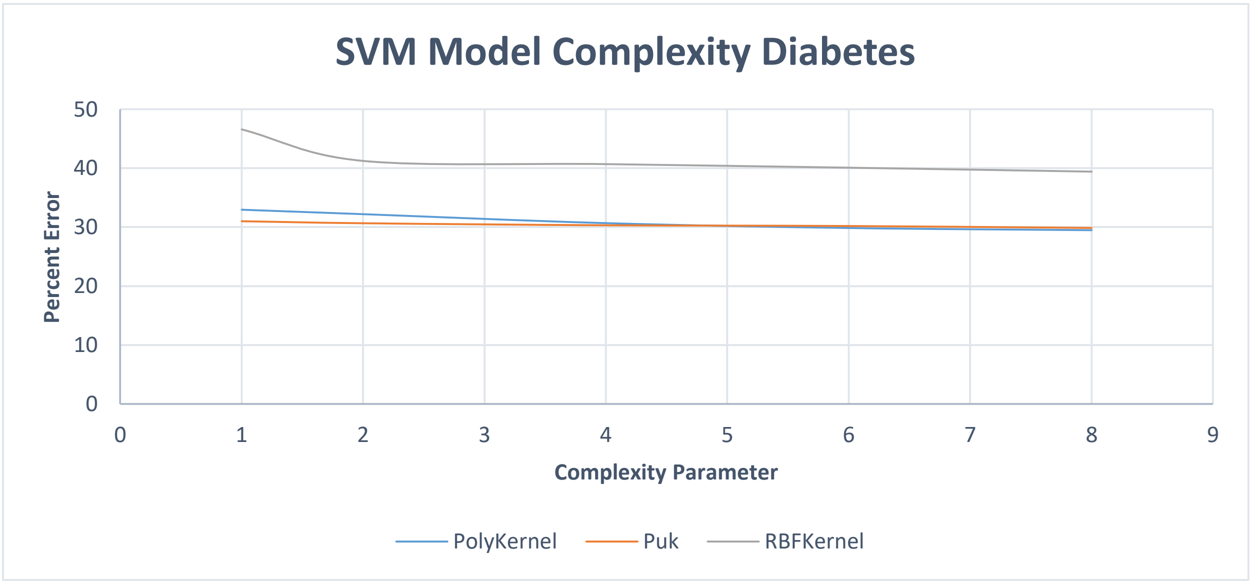




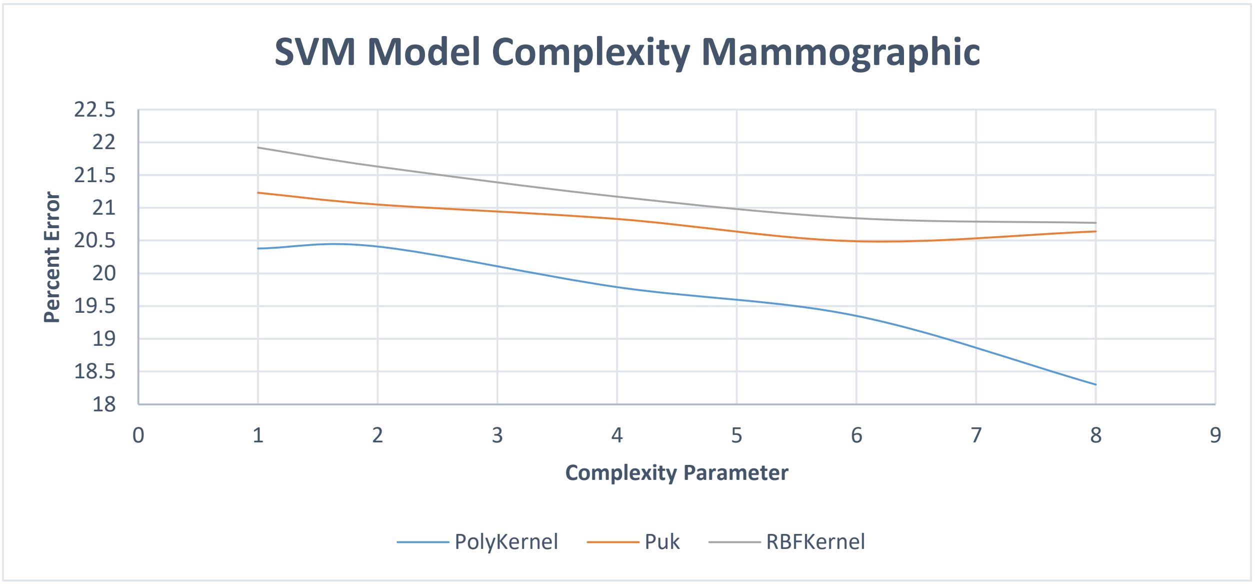
MM Dataset: Comparing that graphs between datasets, the MM dataset graphs for J48 and Hoeffding looked almost identical. Overfitting occurred for both. For J48, overfitting occurs at a confidence factor of 0.2, whereas for Hoeffding, overfitting occurred at a threshold of 0.1. J48 still outperformed Hoeffding, by about 1% error.

Support Vector Machines (SVM) - SMO

The algorithm I used was the SMO algorithm from Weka. The parameters I changed were the different kernels, PolyKernel, Puk and RBFKernel, and the complexity parameter. The complexity parameters I used ranged from 1 to 8.



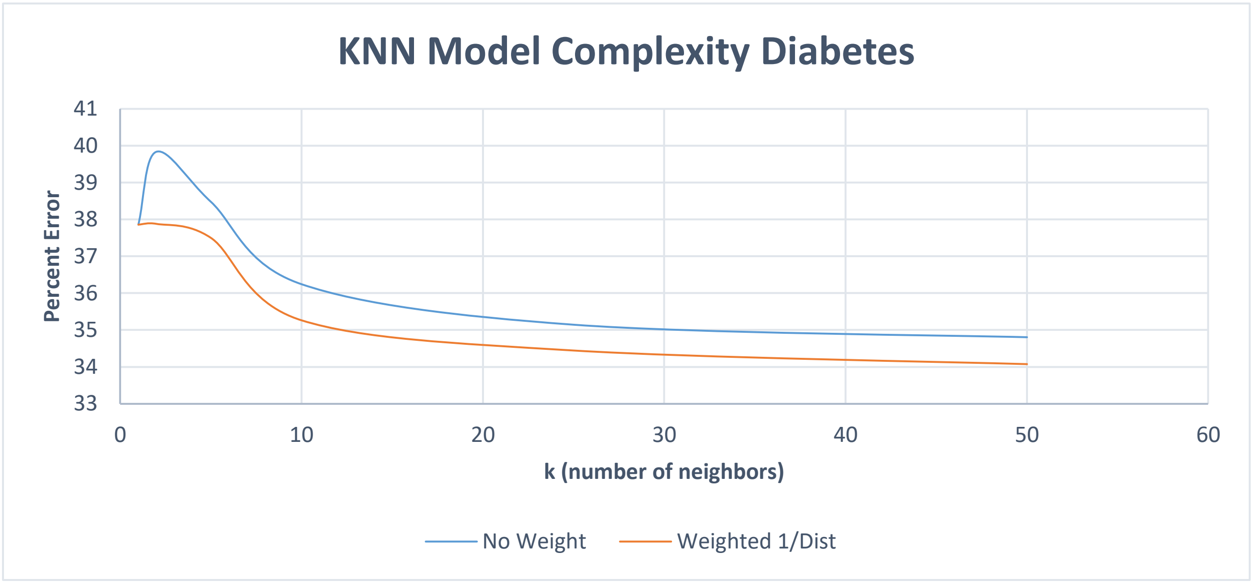
DR Dataset: For this dataset, all three kernel had very similar performance. The best Kernel would be both the PolyKernel and the Puk as they both have almost identical performance. There was no overfitting as all kernels got better with higher complexity. One reason could be that all I was trying to fit these models to the data, but they were all not very good models and could not improve any further. Another reason is that I did not test enough parameters to get any signs of overfitting.



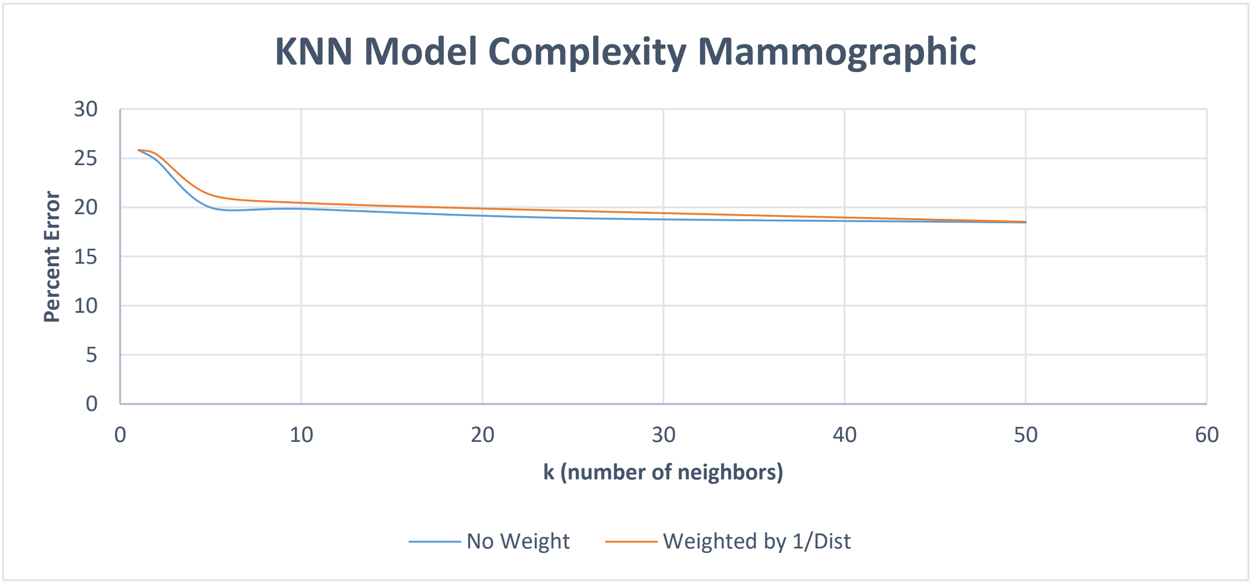
MM Dataset: For this dataset, the different kernels diverge in performance. The best kernel was the PolyKernel with a complexity of 8. All three kernels got better with higher complexity. Even with cross-validation, these kernels showed no sign of overfitting. While Puk and RBFKernel started to level off in performance, the PolyKernel kept becoming a better and better model. I can only extrapolate that it will eventually level off or hit a point of overfitting with higher complexity.

KNN – IBk

The KNN algorithm I used was IBk from Weka. The parameters I changed were k, the number of neighbors to look at, and the weighting of each value. Both curves turned out to be very similar.



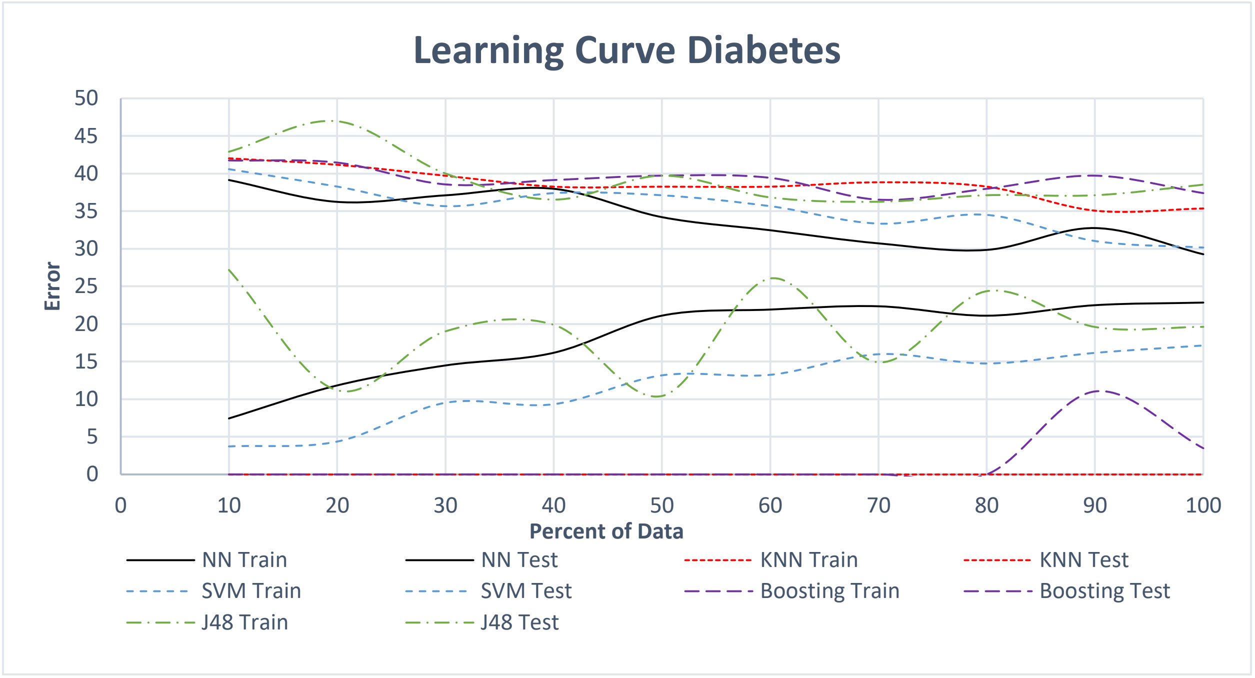
DR Dataset: The best parameters for KNN for this dataset was a k of 50 and having a weight of 1 / distance from that point. There weren't really any signs of overfitting, as accuracy continued to increase as k increased. The only point that is overfitting would when k is small, which makes sense because you are basing your classification on a small amount of data. What's interesting is that the error differences between these two curves remained the same for k > 10. This also means that, for the DR Dataset, it made sense that data points farther from the original has less of an impact on the classification.

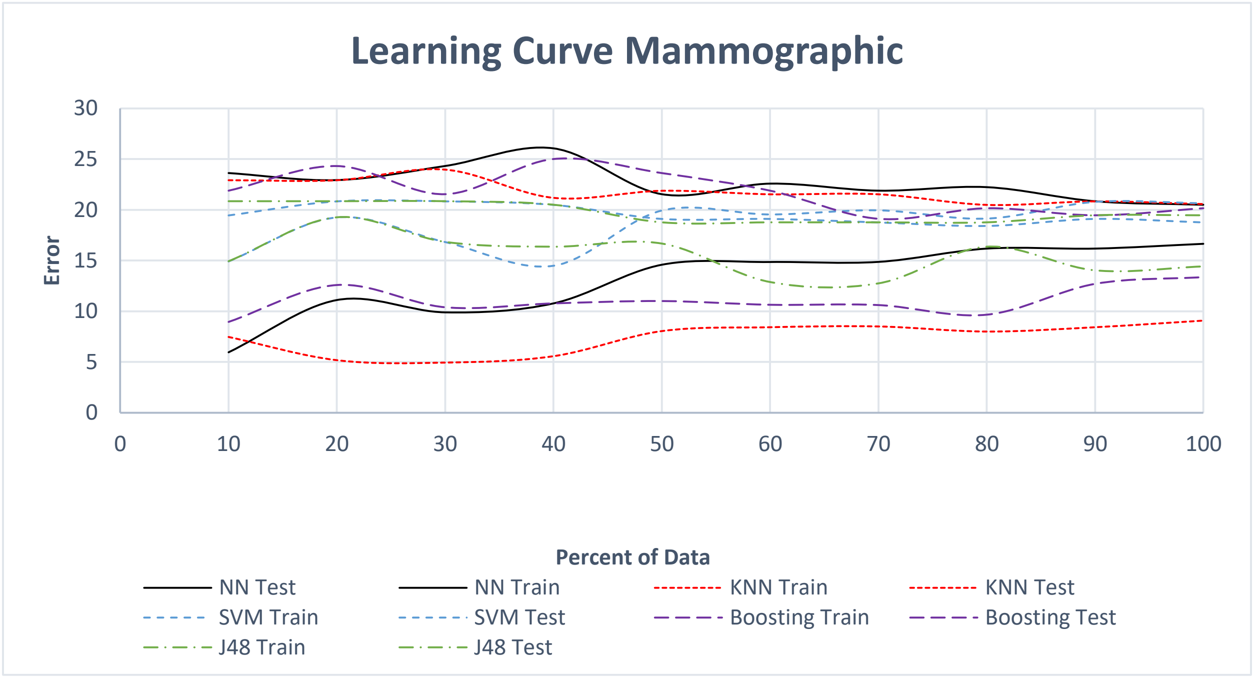


MM Dataset: The graph for KNN is very similar to that for the DR Dataset, except there is barely any gap between the weighted and unweighted KNN. The best parameters might even be not weighting the points. This is interesting because it says that, for any neighbors nearby, the value will be predicted correctly even if it is not weighted. This could also mean that no matter how much you try to improve this algorithm it just doesn't fit the data well enough.

4 Learning Curve

The Learning curves for each algorithm was generating using percentages of the original training data. The data was split into percentages of 10, from 10% to 100% and the training and testing error was plotted.



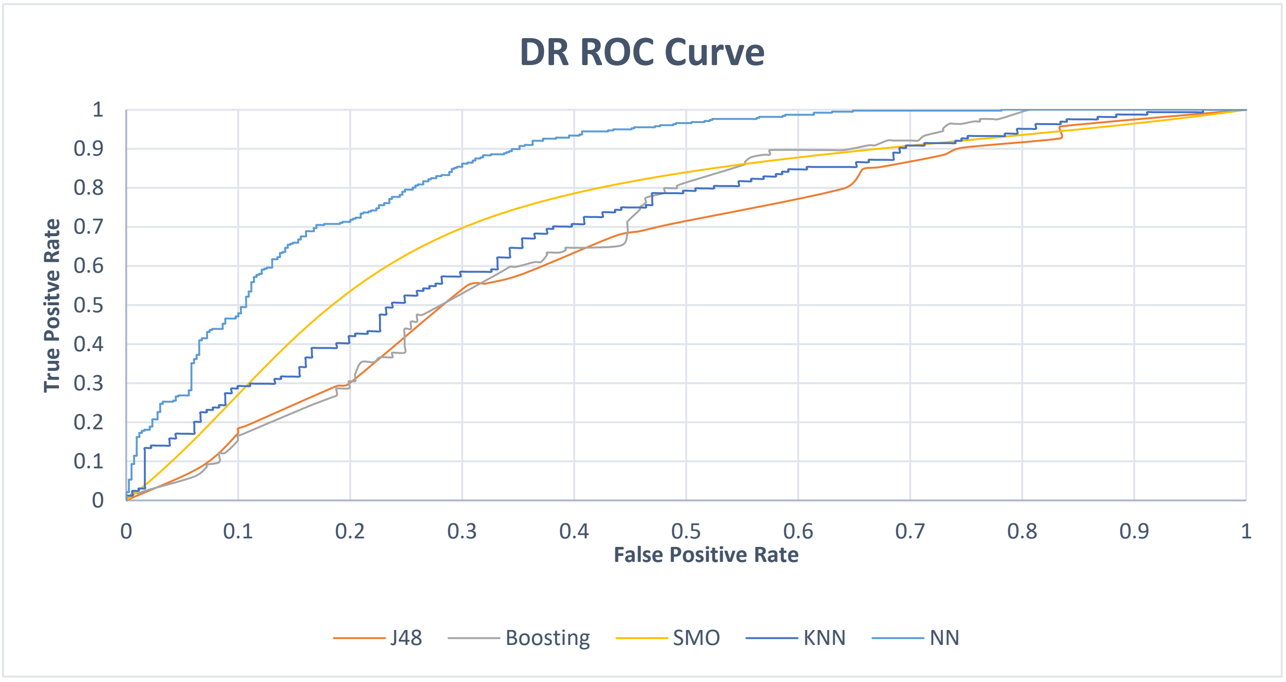


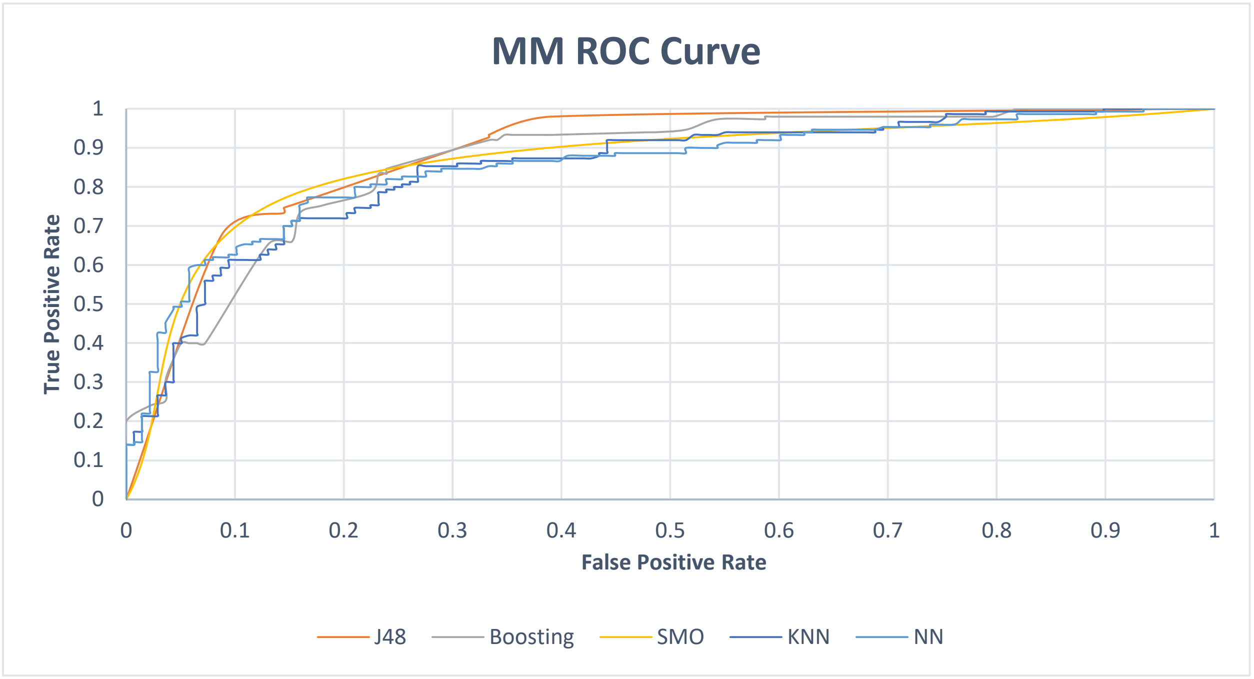
DR Dataset: For this dataset, the algorithms ranked from 1st to last with the best testing accuracy were NN, SVM, KNN, J48 and Boosting. Interesting things to note are the curves for KNN and Boosting. The gap between testing and training was large and did not decrease over time. This suggests that there was high variance in data and if more data was presented, their learning curves would match more closely with NN, SVM and J48. Another interesting thing is that although they had the worst testing error, they have almost no training error. It cannot be the case that KNN and Boosting just couldn't fit the data because if that was the case, the training and testing error curves should eventually meet up with each other, resulting in high bias. With NN, SVM and J48, the testing error slowly decreased, but it seemed like NN eventually fit the model the best. This is interesting because it usually takes Neural Nets a lot of data to be able to fit the model well, but this is almost opposite, with KNN and Boosting not even having enough data to make a difference.

MM Dataset: With this dataset, Neural Nets performed almost the worse out of all the algorithms. It almost looks like NN has a high bias, as training and testing errors were coming closer and closer. Unlike the DR Dataset, both Boosting and KNN showed signs of improvement. The best algorithm was SVM, and its testing error was almost the same and the training error for NN. J48 also showed some signs of high bias, and training and testing error were very close for 50% of the training data and 80% of the training data.

5 ROC Curves

Roc curves were generated for all algorithms for both datasets.





DR Dataset: All algorithms have a curve > 0.5. For this data set, it is clear that NN performed by far the best. This is because from the graph, it has the greatest AUC. The DR dataset suggests that it was a very complex problem because NN could model it much better than most of the other algorithms.

MM Dataset: All algorithms performed very similarly to one another. It isn't clear which algorithm is the best exactly from the graph, but J48 performed the best. Unlike the DR Dataset, NN performed much worse, almost last compared with the other algorithms. This could be due to overfitting through high complexity.

5 Conclusion

Both Datasets gave insight on the algorithms strengths and weaknesses. The DR dataset really showed how NN can model high complexity problems much better than any other algorithms in this scenario. It also showed that the data had high variance, so high that KNN and Boosting could not model it very well, even with the best parameters. In the other dataset, all algorithms have a consistent result, and while all classifiers can model the datasets well, NN did not stand out as much. But despite this, NN still showed consistency through both data sets, in the learning curve and ROC curve.

References:

1 https://archive.ics.uci.edu/ml/datasets/Mammographic+Mass

2 https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set

3 http://machinelearningmastery.com/assessing-comparing-classifier-performance-roc-curves-2/