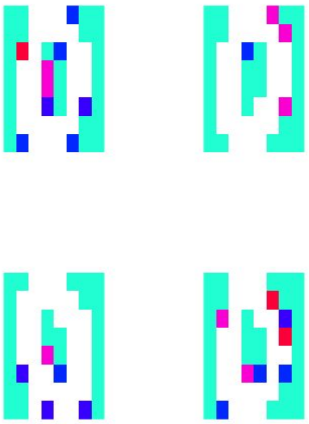
# CS 7641 Assignment 1: Supervised Learning

## Optical Recognition of Handwritten Digits Data Set



Handwriting recognition is a classic classification problem and was an area that saw early use of Artificial Neural Networks (ANN). I retrieved my handwriting dataset from here. In this particular dataset, the classification space was limited to digits 0-9 and the features were already preprocessed. A visualization of the input for fours 0s is to the left. There are 64 features which correspond to the 8x8 pixels of an image of a handwritten digit. Each pixel has a feature value of the intensity of the coloring in the range [0, 16].

I picked this dataset in particular because I believed it would be complex enough to give me a chance to compare algorithm performance and I could easily visualize and understand the feature inputs. Also, computer vision is an entire field of study with tons of real-world applications.

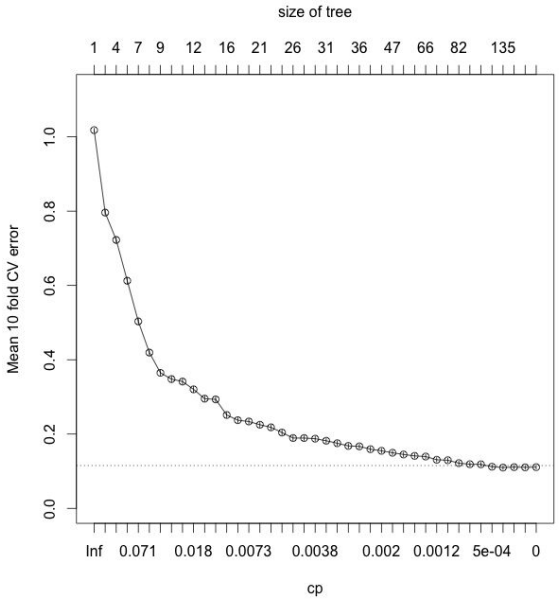
Of particular interest was the test set which, according to the source, contained 13 individual's handwriting who were not in the training set. As opposed to some other datasets where I would have had to arbitrarily split the dataset into training and testing sets, this dataset came with a testing set that is distinctly different from the training set. This gives me an opportunity to see if I can build an algorithm that generalizes to everyone's handwriting, or in this case a new set of 13 people for testing. The only preprocessing that was applied to the data was standardization of each feature column. In theory each feature was supposed to have the same range [0, 16], but in practice this was not the case. Additionally, my neural network back-propagation techniques worked best with standardized features.

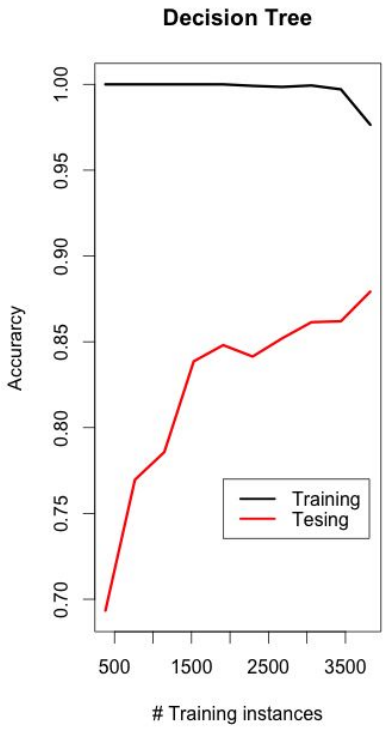
**Analysis Notes:** I sometimes used my own cross validation routines and other times used those provided with my R packages. In all cases, I used 10 folds for cross validation.

### Decision Trees

I used the rpart package that implements Classification and Regression Trees (cart). The algorithm is extremely similar to ID3 and uses information gain to pick the best split attribute. The tree building function in the package takes a complexity parameter which is a threshold for which a split must improve relative error. This complexity parameter is similar to what is discussed in our textbook by Mitchell (page 69): "Quinlan (1986) uses a chi-square test to estimate whether the further expanding of a node is likely to improve performance over the entire instance distribution, or only on the current sample of training data". The rpart package came with its own cross validation routine for picking the best complexity parameter. The graph on the nexts page shows the complexity parameter vs. the size of the tree built using that

complexity parameter vs. the cross validation estimate of error for trees built using that complexity parameter. The graph shows that this particular classification problem requires deep trees in order achieve its best cross validation results. The reason this dataset requires such deep trees is because this problem inherently requires interactions between the different features. For example, in a simple logistic regression or neural network perceptron there are not feature interactions. Each feature contributes some weight to the output of the model, but that contribution is not modified by the value of other features. In this classification problem, however, feature interactions are guaranteed to be important. For example, a tree may be able to separate out the zero class by looking at the intensity value at the center of the image (which is likely to be 0), however, fives, sevens, or eights, for example, all likely have an intensity value greater than 0 there. Therefore, to distinguish a five from a seven or eight, we need to look at other adjacent pixels beyond the center pixel. The model can then tell if the observation is a five by checking if the values at a number of pixel exceed some thresholds and then ANDing together all those boolean statements. The processing of ANDing together the conditions at multiple pixels is what is called multi-feature interaction and since we have 10 classes which each require a number of different conditions that need to be ANDed together, we need deep trees.

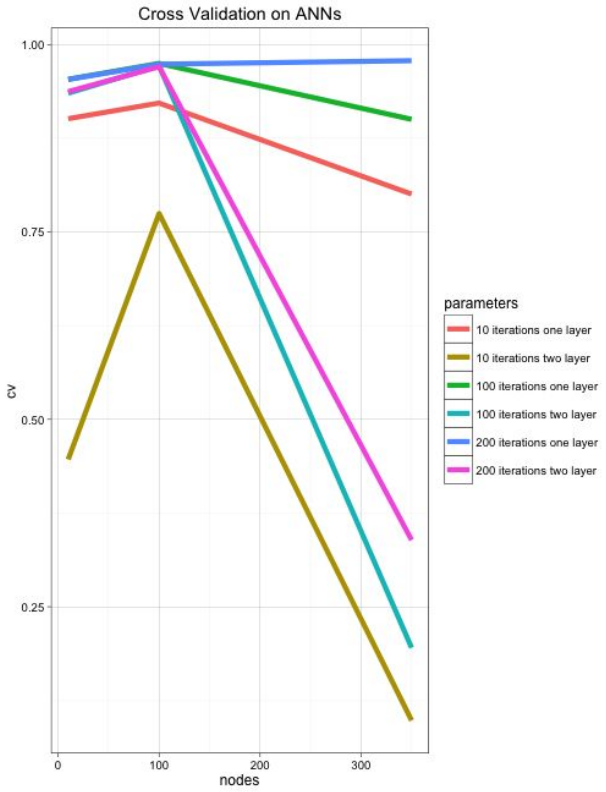


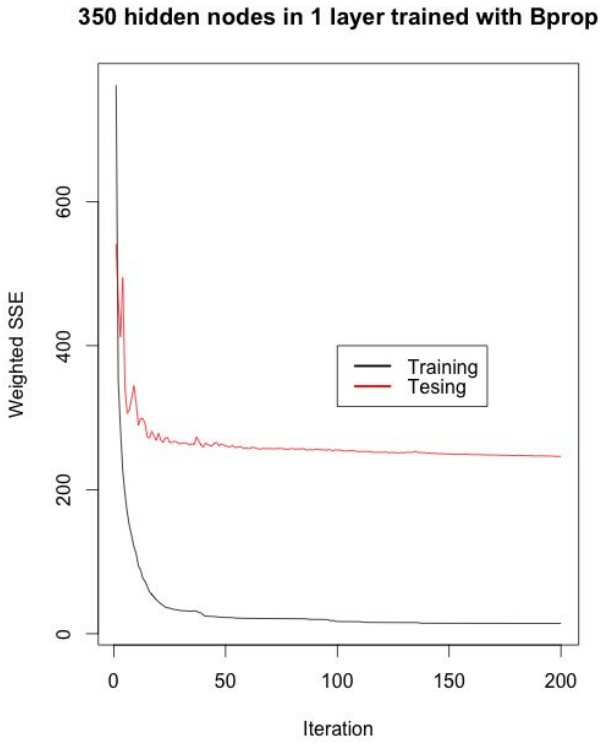

To finalize my analysis, I trained a decision tree on the entire training data with the optimal complexity parameter chosen by cross validation. The accuracy of this model on the testing set was 87.92%. Although tree models are quite simple, they performed quite well on this dataset. Just randomly guessing that each digit was a 9 in this dataset would give an accuracy of approximately 10% and we were able to increase the accuracy by close to 80%. Furthermore, the graph to the right shows that as we increased the amount of training data for the tree, the better we did on predicting the test set. We see degraded performance on the training set as the number of training instances increases because the complexity parameter prevents the tree from overfitting and perfectly predicting the training set.

## Artificial Neural Networks (ANN)

For my ANN analysis I used the Multi-Layer Perceptron function from the library RSNNS which is an R interface to the Stuttgart Neural Network Simulator. There were many parameters to choose from including the number of nodes, the number of layers, and the number of iterations for backpropagation, so I wrote my own cross validation routine to test various combinations of the parameters. My results are displayed graphically to the left (# of nodes corresponds to the number of nodes in **each** layer). 1 layer with 350 hidden nodes for 200 iterations performed the best. One potential reason models with the additional layer had degraded performance was because they were overfit to the training data and did not generalize to the testing data (folds not the actual test set). Another possibility is that they were not run for enough iterations. Either way, our optimal parameters have a great cross validation estimate of accuracy.



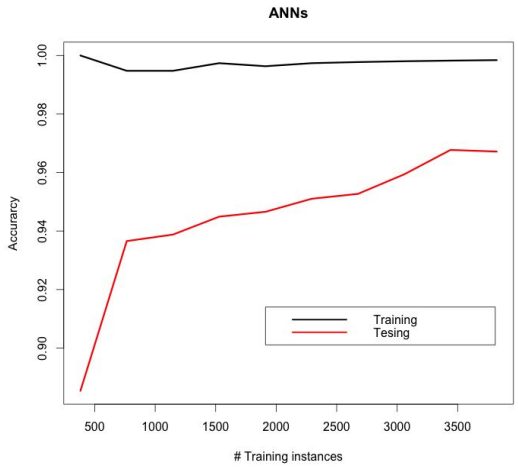

Using a single hidden layer with 350 nodes, I trained a neural network for 200 iterations on the entire training data. Below is a graph of the Weighted SSE for the training and testing set vs. the number of iterations. We can see that 200 iterations in the end was a good choice and that we do not see overfitting on the test set. The accuracy associated with this best net model is 95.15% which is a 5% improvement over our decision tree model.




Why was 1 layer the most performant and why did increasing the number of hidden units increase the cross validation estimate of accuracy? Single layer ANNs can represent boolean and continuous functions which I'll argue that in this dataset is enough to be performant. What we need to learn is the combinations of pixels (boolean ANDs) that make up each digit and whether each pixel has a high enough value (continuous function) to contribute to this ANDing process. The reason more hidden nodes produced better cross validation results was because each digit

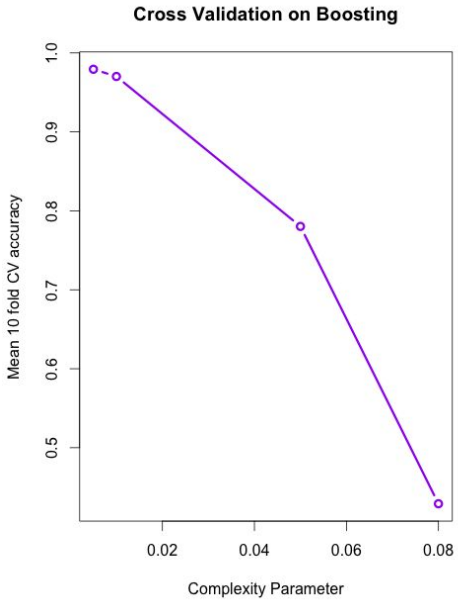
requires its own combination of ANDs and continuous functions.

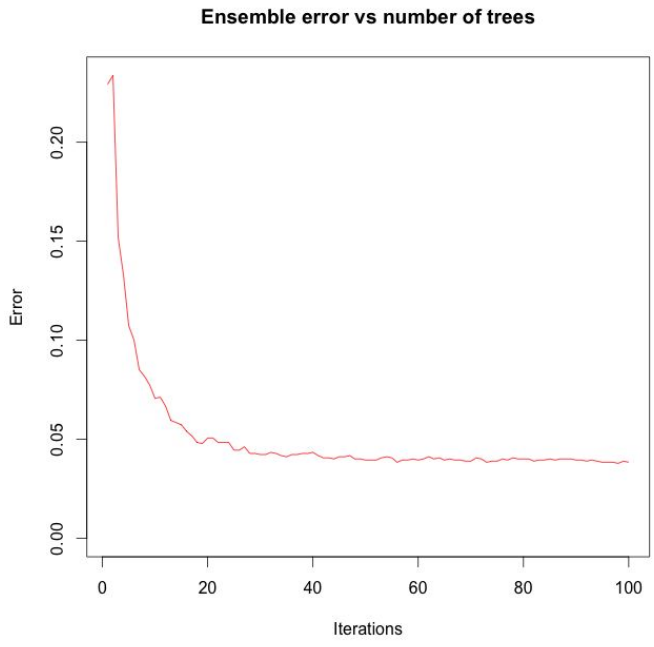
The reason we saw extra hidden layers not improving performance was because we did not need to search the "arbitrary" function hypothesis space provided by deep multi-layer networks, and by restraining ourselves to simpler networks we were able to learn the simpler concepts better. I'd like to point out that this handwriting dataset has been heavily preprocessed and that another similar dataset might require a larger number of layers. For example, all the images were in the same orientation (i.e. no upsidedown sevens). In noisier data, a deeper neural network might have fared better since it could learn rotation and shift functions before booleans and continuous functions. The graph to the left shows a similar trend to what we saw with decision trees: as the number of training instances increase the better we do on the test set.




## Boosting

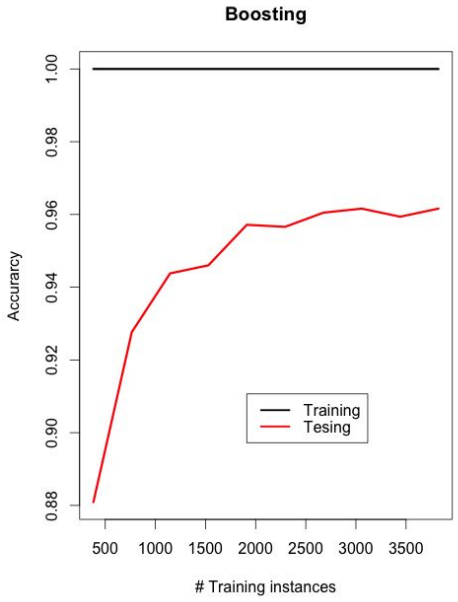
I used the adabag package in R which performs boosting on classification and regression trees (CART) from the rpart package previously discussed. For boosting I varied the complexity parameter previously discussed in the decision trees section and looked at the cross validation results displayed in the graph to the right. The choice of the complexity parameter helps to limit the first decision tree from being overfit, but allows future iterations to still learn. If we perfectly fit the data on the first iteration of boosting, the algorithm will not reweight any of the observations since the classifier perfectly predicted the training set. The results show the best complexity parameter is 0.005. Further interesting analysis would be to vary the number of iterations for which boosting was run. In that analysis, it might be the case that ensembles with higher complexity parameters eventually catch up or do better than the performance of complexity parameter of .005, however, that experiment would take quite some time to run and using .005 as a complexity parameter gives great cross validation results, so I went ahead and used it.



I boosted the entire training set for 100 iterations using a complexity parameter of .005. The graph on the previous page shows the performance on the test set as the number of iterations increase. Of interest is that we do not overfit as the number of iterations increase. This concept was discussed in lecture and as we see in the graph, error on the testing set seems to level off and stay consistent past around 30 iterations. Boosting's accuracy was 96.16% on the test set. This is a 1% improvement over the ANN's performance.

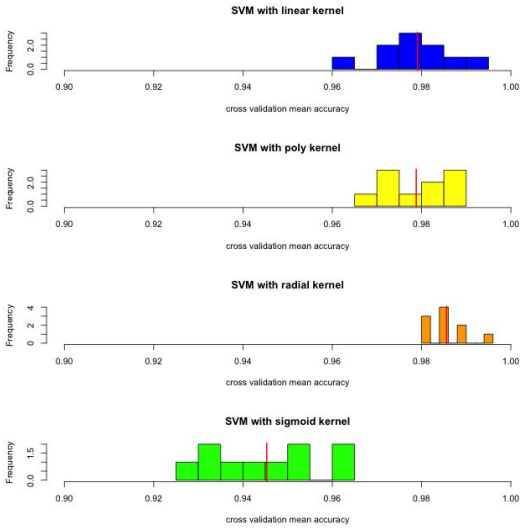
Why did boosting perform better than ANNs and what was it doing? I conjecture that each class of observations are clustered together in feature space (which I'll discuss in more depth in SVMs and KNN) and that boosting was able to separate the data with larger margins between the different classes than ANNs. It's also important to note that we only saw a 1% increase in accuracy so this improvement might be coincidental.




By varying the number of training instances for the boosting algorithm we see the test set accuracy increase as shown in the graph to the left. The interesting result of the graph, however, is that boosting always perfectly predicts the training set. Boosting avoids overfitting but still correctly classifies the entire training set because the algorithm performed enough iterations (100 in this case) to find a shape to fit the training data with large margins.

## Support Vector Machines

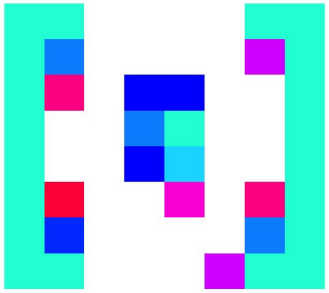
For support vector machines I tested 4 different kernels: linear, polynomial, radial basis, and sigmoid and chose one for my final model by performing cross validation.

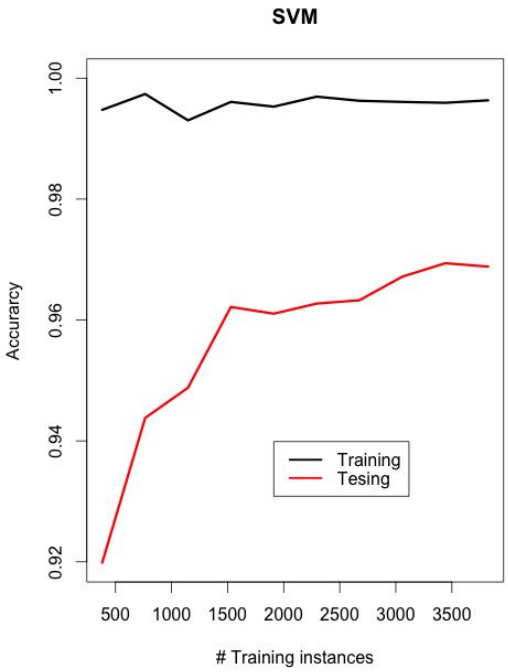



The radial basis function kernel had the highest cross validation accuracy (as shown to the left), so I used it to create a model trained on all the data. This model's accuracy on the test set was 96.88%. The radial basis function kernel looks at the similarity of two observations in part by looking at the euclidean distance from each other. I conjecture that radial basis function was the best kernel function because the digits likely form 10 clusters that are quite separate in distance from one another, so measuring similarity using distance to other observations works quite well. We have not yet covered unsupervised learning, but I decided to test this hypothesis by running k-means with 10 clusters on the raw data and looking at the centers (or characteristic) vectors of each cluster. On the next page is an image of the center of a cluster that conveniently looks like a 0. I hypothesize that

k-nearest-neighbors will perform extremely well on this dataset because the radial basis kernel worked so well, and we have these convenient centers from the k-means algorithm.

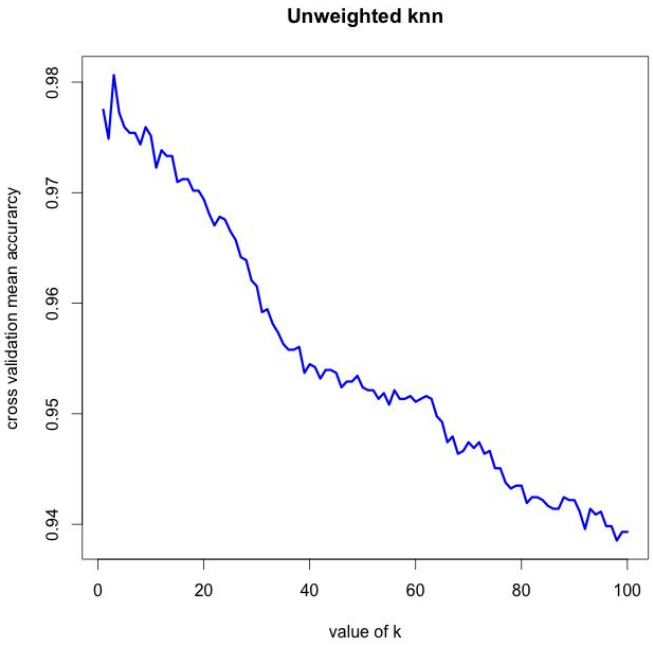
By varying the number of training instances for the svm process (shown on the next page) we see test set accuracy increase in a similar fashion to other algorithms. What's interesting to note, however, is that unlike boosting where the boundaries between classes created by the algorithm are completely flexible in shape and can fit all the training data, the radial basis function kernel confines the shape of the class boundaries created by the algorithm. This confinement of the shape of the boundaries causes the algorithm to not perfectly fit the training data. This confinement may have help the algorithm avoid overfitting on training set and help contribute to SVM's performance on the test set.






## K-nearest neighbors (knn)

For knn, I varied the k used for classification and performed cross validation. There is a defined peak when **k** = 3 (shown to the right), so I built a knn model on all the data using a k of 3. This model achieved an accuracy of 96.66% on the test set. Of note is that even with extremely high values of k, we still see what would be considered great cross validation performance. This is likely because there is such a large number of observations in the dataset, so the 100 closest neighbors are likely the same class. If the dataset was say 150 observations instead of 3823 observations, we would likely not see this trend.




As discussed previously in the SVM section, knn works so well here because the there are distinct clusters of the same digit with separation between them. Additionally, we do not have any features that are irrelevant to the class outcome. Said another

way, each pixel matters when predicting the class. Likely if we had a number of noise variables we would not have seen knn and rbf SVMs being on par with Boosting and ANNs.

### The Best Algorithm

| Algorithm | Performance on Test Set | Average Train [10 trials run] (on entire training set) runtime in seconds | Average Prediction [10 trials run] (on test set) runtime in seconds |
|---|---|---|---|
| Decision Tree | 87.92% | 0.03868 | 0.00332 |
| ANN | 95.15% | 160.16 | 34.6754 |
| Boosting | 96.16% | 12.28518 | 0.50654 |
| SVM | 96.88% | 0.71714 | 0.10884 |
| Knn | 96.66% | n/a | 0.7454 |

The best algorithm in my analysis of handwritten digits is a SVM with a radial basis function kernel. A SVM model had the highest accuracy on the test set, second fastest training time, and the second fastest prediction time. It's likely that any application of computer vision is going to be concerned with speed as well as accuracy.

The algorithm is simple in terms of structure and uses a variant on the idea that observations are similar if they are similar in proximity to one another. This embodies feature interactions without the explicit interaction hypothesis structure of decision trees or ANNs.

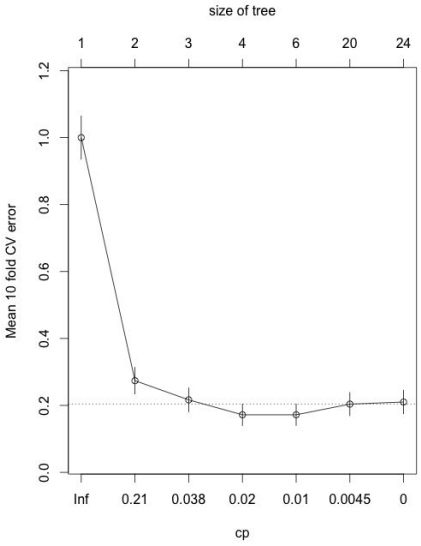
## Breast Cancer Wisconsin Data Set

Medical diagnosis is another common classification problem. I decided to analyze a classic cancer diagnosis dataset that uses a number of tumor attributes to predict whether a tumor is benign or malignant. I retrieved the dataset here. I picked this dataset in particular because it was a binary classification problem as opposed to the multi-class classification problem of handwritten digit recognition. Also, practically, medical diagnosis is still a field with thriving research.

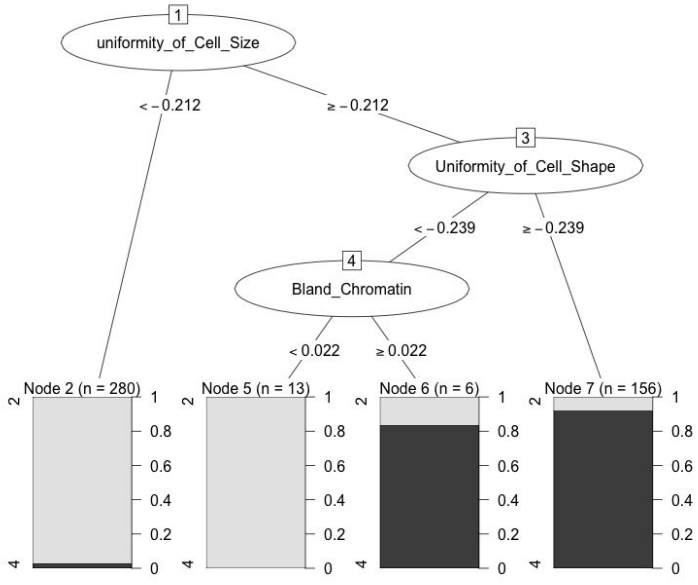
I performed 3 pre-processing tasks on this dataset. There were originally 699 observations, but 16 had missing attributes. I removed these 16 observations to create a total of 683 observations. As in the digits dataset, I also standardized each feature. Finally, this dataset did not come with a pre-split test set as my previous dataset did, so I withheld a random ⅓ for a test set and used the remaining ⅔ for a training set.

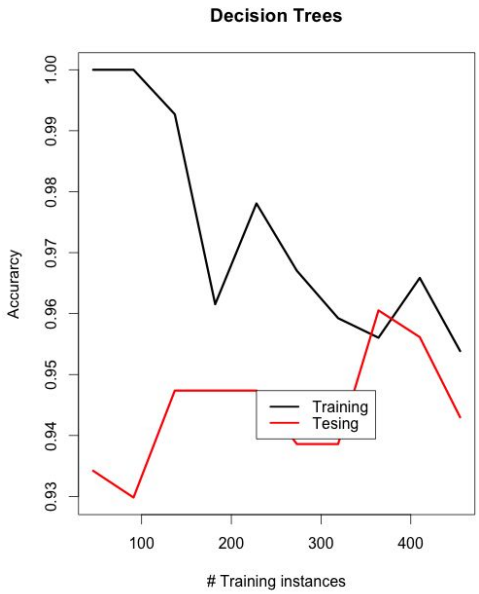
## Decision Trees

I again used rpart's cross validation routine to pick my optimal complexity parameter. The graph below shows that in contrast to handwritten digits dataset, relatively small trees actually outperform their deeper counterparts, likely, because they avoid overfitting the data. To finalize my analysis, I trained a decision tree on the entire training data with the optimal complexity parameter chosen by cross validation. The accuracy of this model on the testing set was 94.30%. The prior probability of a tumor being be benign in the whole dataset was 65.00%, so with a simple tree algorithm we see an increase in performance of approximately 30%. Below is a visualization of the actual decision tree generated (2 stands for benign and 4 for malignant). As shown, using only 3 features the algorithm provides a great model.



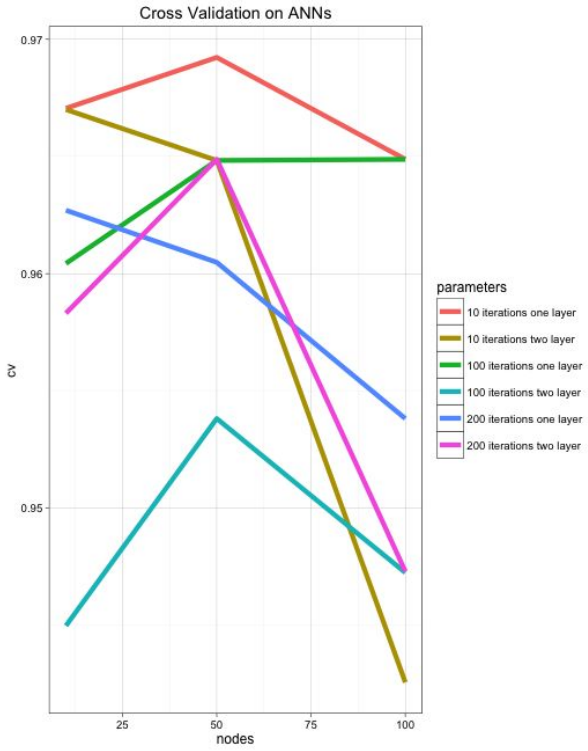

By varying the number of training instances provided to the algorithm we see training set accuracy degrade while testing set accuracy seems to increase slightly (shown below). This is likely because the complexity parameter prevents the tree from overfitting the training set and as we get more training example we have a better idea of the boundaries on the features values (and are able to generalize on the test set). The testing set accuracy increase is likely not as smooth as we saw in the handwritten digit recognition dataset (for all algorithms) because the increment on this dataset was 45 more observations versus 380 in the previous dataset. Regardless, we see a slight increase in performance by increasing the number of training observations.

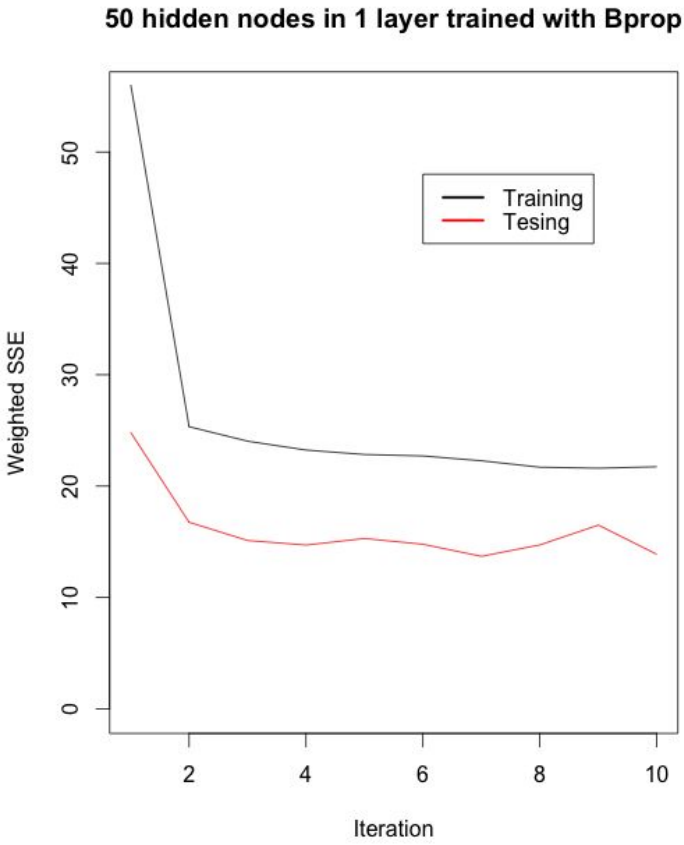


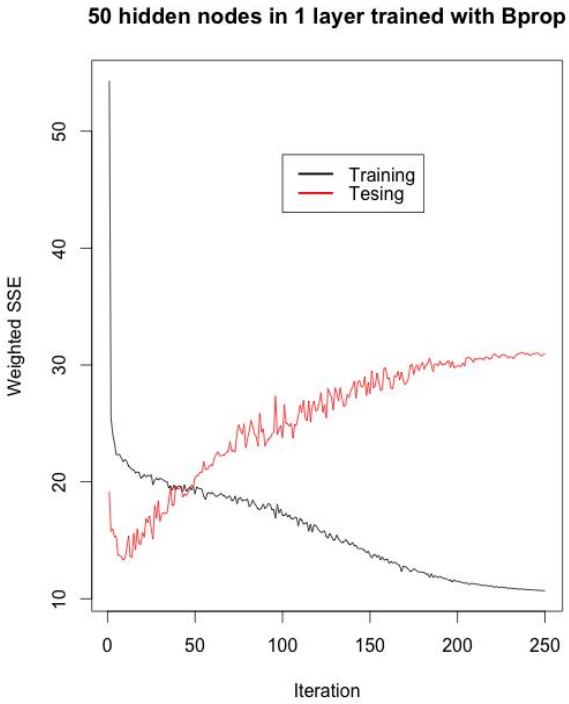



## Artificial Neural Networks (ANN)

I used cross validation to choose the number of hidden nodes, number of layers, and the number of iterations. The clear winner was 10 iterations with 50 hidden nodes in 1 layer (as shown in the graph to the left, note: # of nodes corresponds to the number of nodes in **each** layer). I trained a neural network using these optimal parameters and achieved an accuracy of 98.24% on the test set. Below is a plot of the performance vs. the number of iterations for both the training and testing set. As an experiment, I also run another neural network with the optimal parameters except I ran it for 250 iterations and the results are displayed to the left. Clearly the model has overfit the test data in this case and using cross validation to pick the number of iterations was essential to help tease out the correct number of iterations for backpropagation.

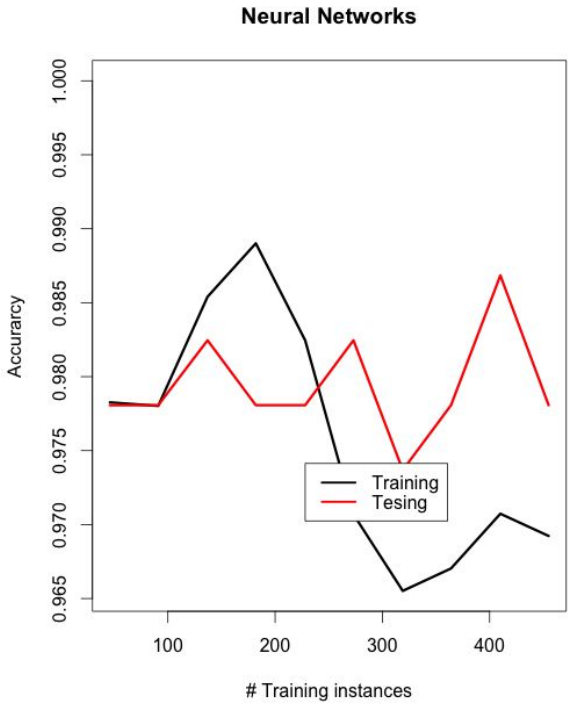



By varying the number of training instances, we see results that contrast everything we have seen in the

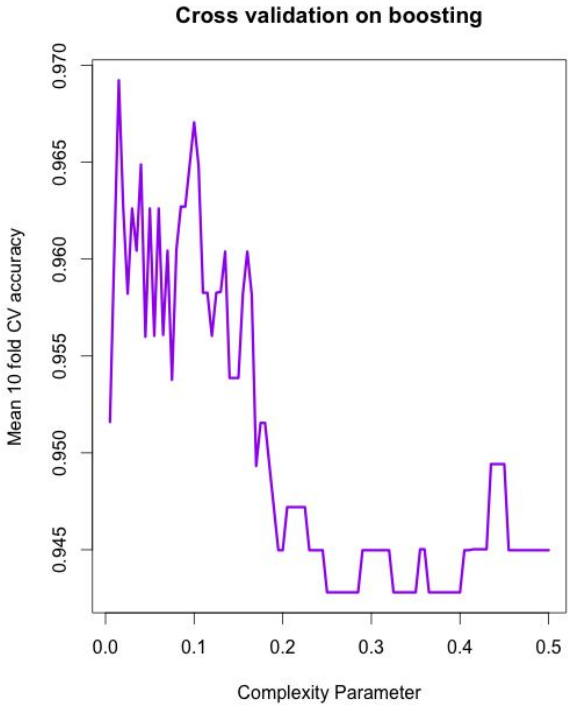
digits dataset or what we expect. Training set and testing set accuracy show no obvious positive or negative relationship with the number of training instances. The only conjecture I can make is that the incremental increase in the number of training observations on this already small dataset was not able to demonstrate the relationship we expected.

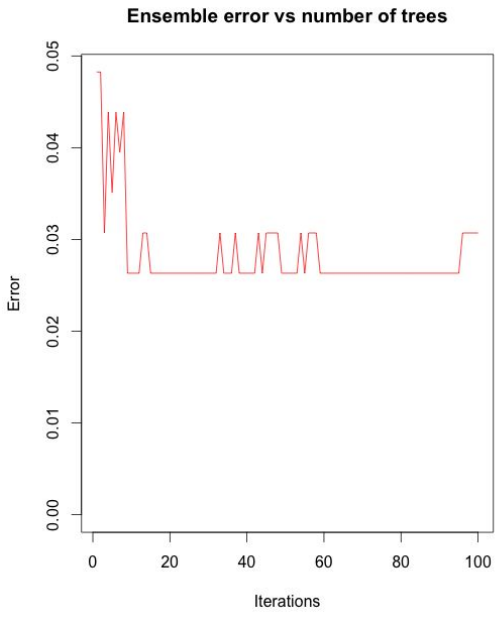



Why was 50 nodes the best, why 1 layer, and why so few iterations? As we saw in the decision tree section, a quite simple model with only three features does quite well at predicting whether a tumor is malignant. I conjecture that the neural network is learning a similar concept to predict most of the tumor cases and potentially learning a few more obscure conditions to predict the few that do not fit that model. So, in essence, we do not have too much more to learn than what we have with a simple tree, so deep networks and many iterations are not needed.

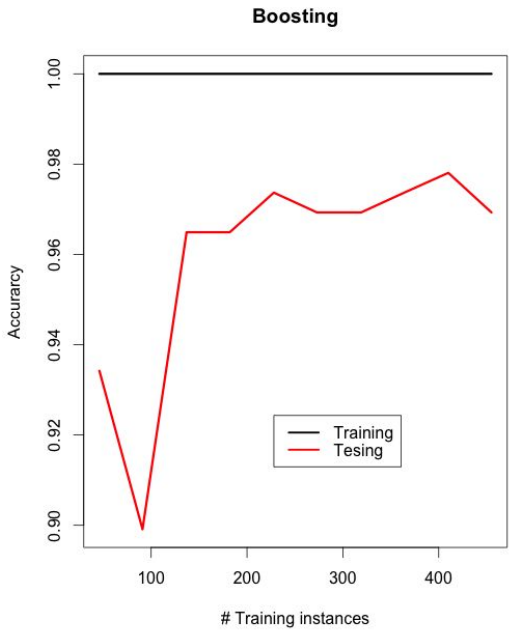
## Boosting

For boosting I picked the complexity parameter through cross validation as shown in the graph to the right. For this dataset, I experimented with a larger number of complexity parameters. The optimal complexity parameter was 0.015. I trained a boosted set of decision trees on the entire training set using this optimal complexity parameter. The model



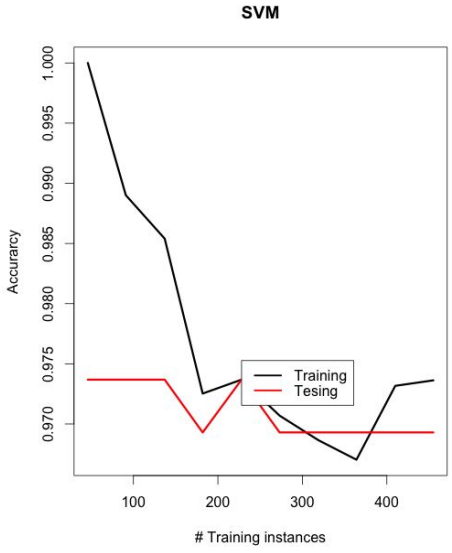
achieved a test set accuracy of 97.37%. The graph on the previous page shows the performance on the test set as the number of boosting iterations increases. Of interest is that we do not overfit as the number of iterations increase, instead, error on the testing set seems to level off and stay consistent past around 10 iterations (with some small variation from iteration 30 to 100). This graph of test set error versus the number of iterations is most likely less elegant and smooth as compared to the same graph I used in the digit dataset, because this dataset has less observations in both the test and training set. Also of interest, is that that after the first iteration we have an error rate of approximately 5%. In this boosting example, it looks like we started by building the same tree we used in the Decision Tree section. Boosting improves accuracy on the test set by either improving the margins between the different classes or learning a few more obscure concepts to predict the hard cases.

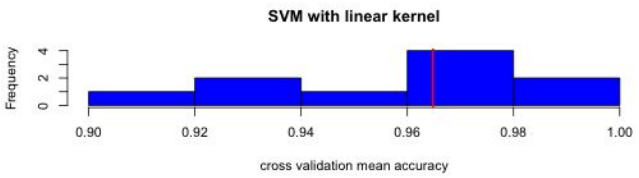


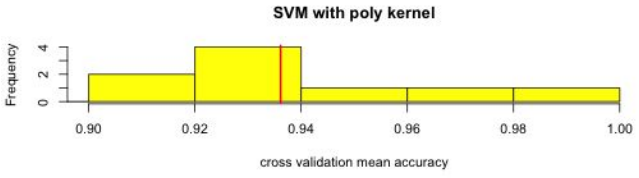

By varying the number of training instances for the boosting process we see the test set accuracy increase (as shown to the left). The interesting result of the graph, however, is that boosting increases performance on the test set more smoothly than the other algorithms.

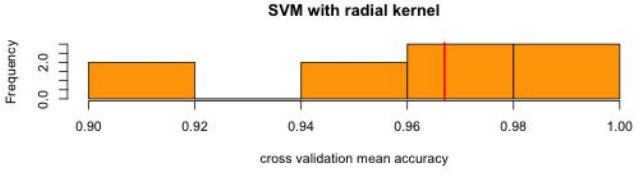
## Support Vector Machines

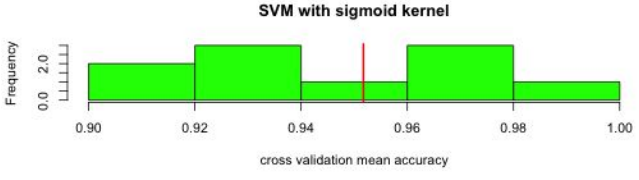
For support vector machines I tested 4 different kernels: linear, polynomial, radial basis, and sigmoid and chose one for my final model by performing cross validation (see the graph below). The best kernel on this dataset was the same as the digits dataset: the radial basis function. I trained a SVM using the radial basis function kernel and it achieved an accuracy of 96.93%. By varying the number of training instances, we see that training accuracy goes down while testing set accuracy seems to stay relatively the same (shown below).



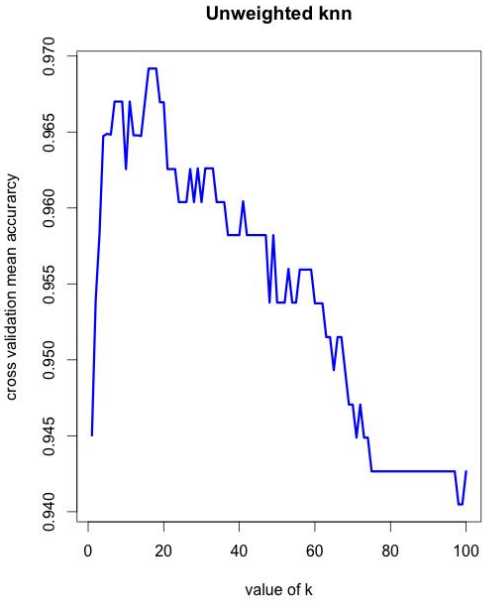



## K-nearest neighbors (knn)




For knn, I varied the k used for classification and performed cross validation. There is a defined peak when **k** = 16 (shown to the left), so I built a knn model on all the data using a k of 16. This model achieved an accuracy of 97.81% on the test set.

Why did knn and SVMs work so well? It's impossible to visualize the observations in 9 dimensional space, however, we can visualize the classes in pairwise plots of the different features. You can see this visualization at the end of my cancer.R code. In this visualization it's clear that the classes are naturally clustered in euclidean space in each pairwise plot, so the distance metric used by both knn and the radial basis function captured similarity very well. To capture the same space, a decision tree would need to be nearly full and certainly complex. However, as I mention in the next section, decision trees give good rules of thumb that could be easily be used by medical practitioners.

## The Best Algorithm

| **Algorithm** | **Performance on Test Set** | **Average Train [10 trials run] (on entire training set)runtime in seconds** | **Average Prediction [10 trials run] (on test set for 10 trials) runtime in seconds** |
|---|---|---|---|
| Decision Tree | 94.30% | 0.00236 | 8e-04 |
| ANN | 98.24% | 2.51216 | 1.06028 |
| Boosting | 97.37% | 9.21894 | 0.07748 |
| SVM | 96.93% | 0.00466 | 0.00106 |
| Knn | 97.81% | n/a | 0.0042 |

In this classification problem, the best algorithm is an Artificial Neural Network **and** a decision tree. The decision tree performs extremely well with only 3 features. The visualization of the tree could easily be shown to doctors or lab technicians to help reinforce what they likely already know and provide them rules of thumb. Based on occam's razor this simple model is the best. However, the neural network could also be used in a lab/clinical setting to help inform doctors or lab technicians of tumors that are potentially malignant, but fall outside of the normal characteristics that define the malignant class.