SUPERVISED LEARNERS

Wan, Yuet Wing Frank  
ywan33@gatech.edu

***Abstract—***To understand the differences between various supervised learning algorithms, algorithms like pruned decision trees, gradient boosted trees, neural networks, support vector classification, and k-Nearest Neighbor algorithms were tested, refined, and compared on a publicly available dataset on breast cancer. The data set was first explored before tested on with the learning algorithms, before the results were finally compared against.

## Detail of Target Data

The load\_breast\_cancer dataset[[1]](#footnote-1) provided by scikit learn dataset was chosen as the dataset as the feature data was cleaned and did not have many missing values. Other datasets from Kaggle machine learning projects were explored but were discarded due to many factors such as bad quality of feature data, small datasets, and lack of features. For example, while looking for datasets regarding COVID infection and mortality among patients, meaningful data like population age and other health indicators would be needed to understand which health factor is most highly correlated with COVID mortality.

The load\_breast\_cancer dataset also contained many distinct features and were fully populated, thus not requiring the need for imputation. As the approach towards data imputation depends on the type of data and requires subject matter experts to weight in, having a fully populated dataset is much more convenient to perform analysis on. The target value being classes of malignant or benign also allows classification methods like SVC and KNN classifiers to be deployed.

One caveat is that the dataset only contains 569 unique rows, which may impact the accuracy of the model. However, different methods like bootstrapping allows for the sample data to predict the wider population representation, and trees can sample with replacement to remediate the small sample size. In the future, a richer dataset of at least 5000 rows would be ideal.

However most available datasets are merely from results of COVID tests rather than hospital data of patient mortality, providing datasets with only countries and their daily COVID infection rates. With a lack of meaningful features[[2]](#footnote-2), those datasets would not be able to meaningfully leverage the most machine learning that are not time-based like Decision Trees. Other potential datasets were restricted[[3]](#footnote-3) from public access as well, potentially due to patient confidentiality concerns.

## Feature Engineering

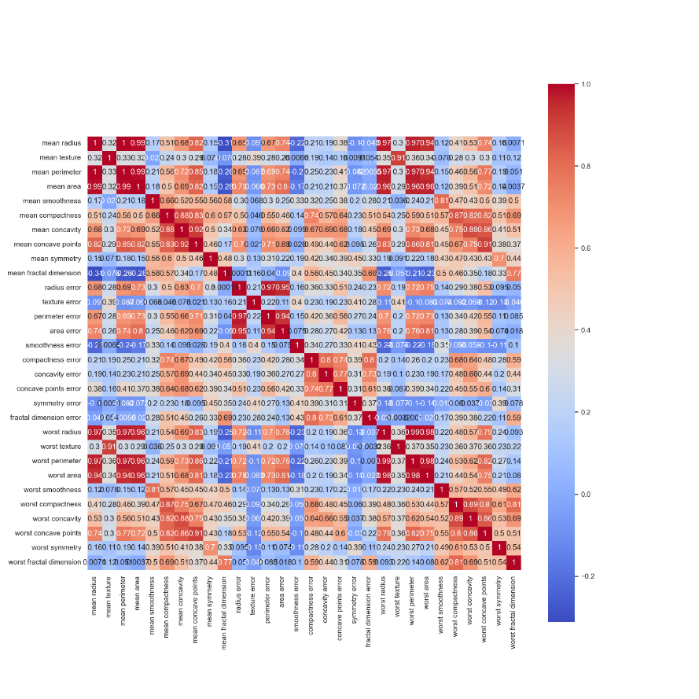
First the heat map was plotted out to find features of high correlation. Feature Engineering must first be performed to ensure meaningful and accurate data is presented to train the data

Figure 1 – Validation curves for Pruned Decision Tree

The heat map of the different features in the cancer database was calculated and plotted out in the heat map above, showing the correlation of 2 features above. High correlation implies that the 2 features are likely stemming from a single factor, and one should be excluded from the final datasets to provide equal weights when estimating. For examples in the figure above, the radius, perimeter, and area all highly correlated, as they are all inter-related to the sizes of the breast mass. Combining those features to one would allow the machine learning algorithms to consider all unique features more evenly.

While there are multiple parameters that can be reduced, this was not done due to technical limitations of deploying the lasso or PCA techniques. The PCA was needed to be combined into a sklearn pipeline, which did not produce a dataset ready for other algorithms to read from. In the future, more optimizations would be ideal in engineering the data for model use.

Package StandardScaler() from sklearn.preprocessing was applied to the dataset within the pipeline initialization to standardize the values before training and testing for better ML training, by centering and scaling the individual features to a Guassian curve with 0 mean and unit variance.

## Classification Models and Hyper-Parameter Optimization Framework

To build a consistent approach in analyzing different learning models, each supervised ML model was created under a super class of BaseClassifier. Different parameters depending on the ML pipeline being tested were then optimized to prepare the classification model to have the highest accuracy.

The main method to fine-tune the model is to perform hyper-parameterization on the different parameters available when initializing the pipeline. For Decision Trees, the max depth, minimum samples, and ccp\_alpha were optimized. For Gradient Boosted Tree, the maximum depth of the tree, number of estimators, learning rate was optimized.

When the experiments method is called, the different model parameters are calculated by function validation\_curve() from sklearn.model\_selection, while function plot\_model\_complexity() then is called to evaluate the accuracy of the train and test set with both default and optimal parameter values. The test scores across the different folds were grouped to find the mean and standard deviation before plotted out in a graph of accuracy against increasing depth of tree.

Both validation\_curve() and plot\_model\_complexity() were calculated with 10-Fold stratified cross validation without replacement, so the test and train split ratio is 9:1. This causes the bags to have the similar mean values for each learning algorithm to iterate over for similar outcomes.[[4]](#footnote-4)

## Decision Trees with Pruning

After setting up the pipeline with Decision Trees, grid search was performed on both the maximum tree depth, minimum samples per leaf, and pruning degree for the decision tree. The classic decision trees using ID3 to determine splits were selected as this brings more consistency with the gradient boosted trees.

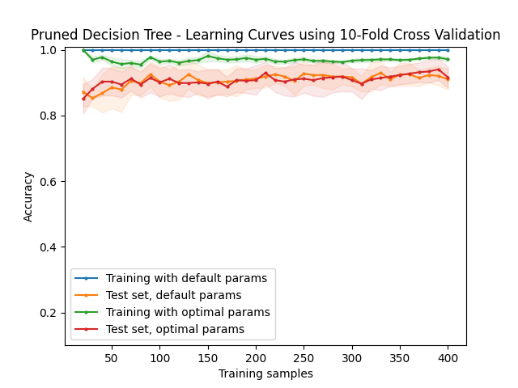
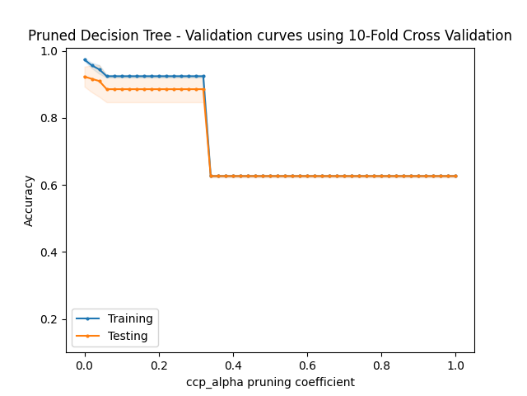
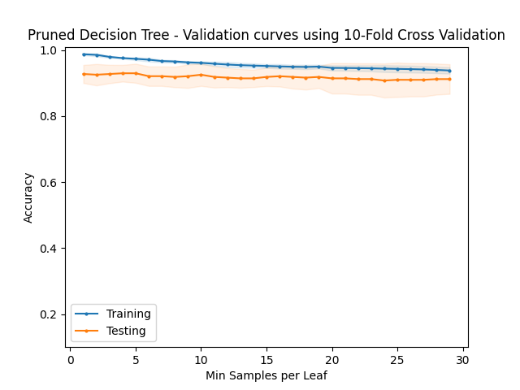
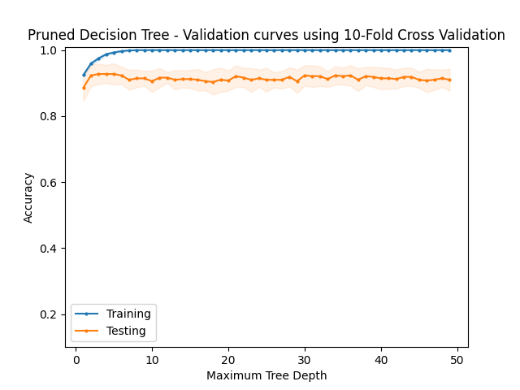


Figure 2 – Validation curves for Pruned Decision Tree

The max depth of the tree represents how many splits and nodes will be made, where too small a number would lead to incomplete splits, but too large would lead to nodes overfitting to random noises in the data. From the top left of figure 2, we can see the validation scores reach a maximum at tree depth of 3 before decreasing and plateauing after 10 maximum tree depths. This observation is expected, where the accuracy of the model decreases due to overfitting and the plateauing effect is caused by the trees no longer being able to sort any better should all the leaf contents already be the same.

As for pruning, the ccp\_alpha hyper-parameter was optimized in a similar fashion. The alpha value stems from ­a minimal cost-complexity pruning algorithm that prevents overfitting.[[5]](#footnote-5) The pruning involves finding the node with the smallest alpha, decreasing the complexity of the model while minimizing model impurity increase. As seen in the figure above, as the alpha increases, so does the tolerance for impurities increase in pruning a branch node as the alpha value increases from 0 to 1 in 0.02 increments. While the initial attempt was to increase accuracy by reducing overfitting, the figure above shows that the accuracy only decreases once alpha increases from 0. While disappointing, this may be due to the low amount of training data, meaning each branch node is too important to prune out.

## Gradient Boosted Trees

The Gradient Boost Decision Tree was tested for the boosted decision tree learner. This specific boosting method was selected as it is like the relatively popular XGBoost, which is renown for its modelling performance in industry.[[6]](#footnote-6) Gradient Boost is a simpler model that does not have the more granular fine-tuning methods that XGBoost has, including the use of sparse matrices with sparsity aware algorithms that bypass the need for differentiability of cost functions[[7]](#footnote-7) and better processor cache utilization which makes it faster.[[8]](#footnote-8) However, XGBoost is not available in the sklearn package, and the simpler tuning parameters allows for easier analysis of its performance, so Gradient Boost was ultimately chosen and its metrics are plotted as of below.

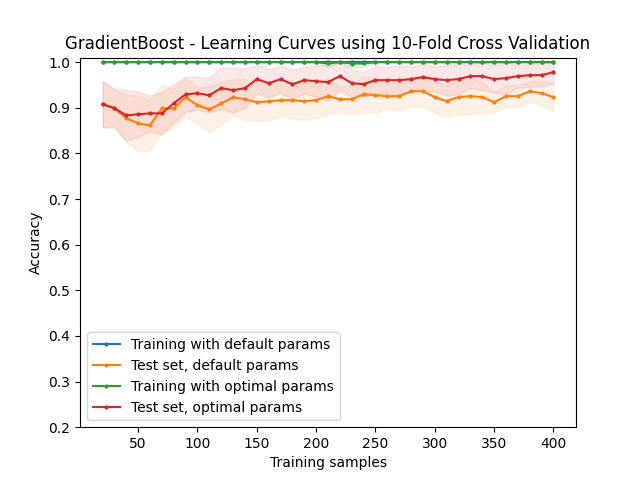
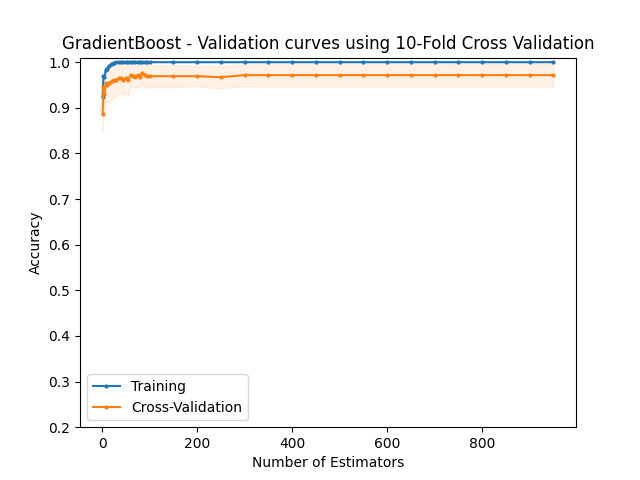
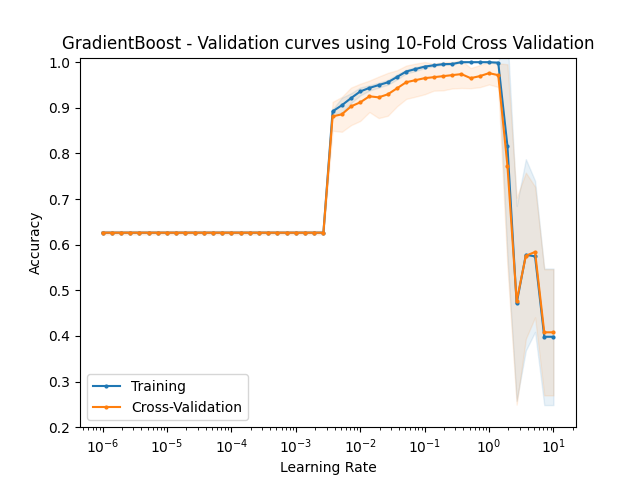
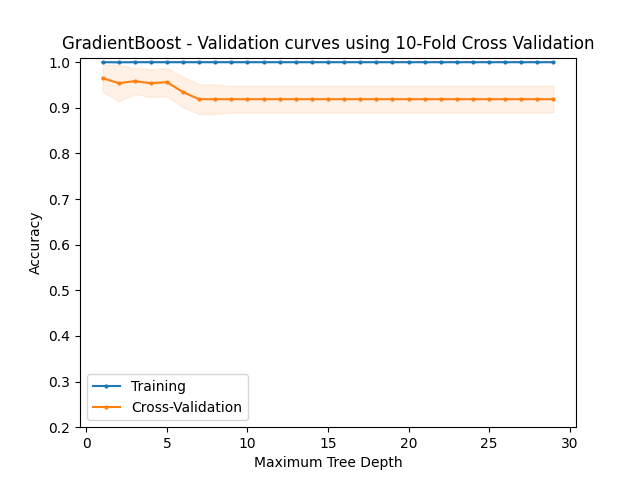


Figure 3 – Validation curves Gradient Boosted Trees

From the top left of figure 3, we can see the validation scores spike at maximum tree depth of 3, before plateauing after the maximum tree depth of 7. The plateauing effect is expected because at one point the depth of trees will no longer be able to sort results any better should all the values of the nodes be correct. Additionally, more tree depth would lead to overfitting of tree nodes around random effects of data, decreasing accuracy as seen as the accuracy dips from after a tree depth of 3.

The number of estimators and learning\_rate of the model is also tested through hyper-parameterization search, with the above steps repeated for both those parameters for the model. Estimators for boosted ensemble tree learns from the misclassifications of the previous split before repeating until either a perfect split is achieved, or the max number of estimators is reached.[[9]](#footnote-9) While accuracy generally increases as the estimators increases, performance generally decreases as each node would be recalculated multiple times.

As seen in the figure above, the number estimators increase the accuracy of the model from 0 to 100 before plateauing afterwards. Meanwhile, the learning rate is the coefficient in which the final iteration of the model learns from the previous one.[[10]](#footnote-10) It is suggested to keep the learning rate low so that more previous model iterations can be considered before finding the result.[[11]](#footnote-11) The accuracy of the final model over increasing learning rate is shown below.

The sharp decrease in accuracy as the learning rate nears 10 is expected, as trees from previous training iterations would have a lesser impact in the final calculation output, decreasing the boosting effect of the adaptive boosting model, making the forest a weaker learner. However, it is interesting how the accuracy peaks at 1 even as the learning rate continues to decrease, allowing more previous training tree iterations to factor into the final prediction. This is likely due to newer iterations learning from the previous node splits, splitting the nodes better. As the learning rate decreases, older tree iterations are weighted more heavily into the final prediction, potentially causing the accuracy to dip as seen.

# Neural Networks

Sklearn’s neural network setup contained a lot more numeric parameters such as the size of the hidden layers, alpha, and learning rate. Following Occam’s razor principle, the parameters were kept as simple as possible before tuning for better performances. Due to the small amount of dataset that contained 30 features, an initial amount of 15 nodes were chosen in a single layer of hidden node as suggested from online resources.[[12]](#footnote-12) Linearity was assumed in the dataset for convenience, solver ‘lbfgs’ was used due to faster convergence and higher performance for smaller datasets as mentioned in scikit documents.[[13]](#footnote-13) Relu activation method was chosen out due to its reported higher reliability and prevention of the vanishing gradient problem.[[14]](#footnote-14)

The number of nodes in the first hidden layer, second hidden layer, and learning rate was adjusted to examine their effect on accuracy. These parameters were prioritized due to their mention in the course lectures and personal curiosity. The non-linear property of the L-BFGS may have contributed to the rapid learning high accuracy of the neural network, but further investigation and research is needed before a more accurate conclusion can be drawn.

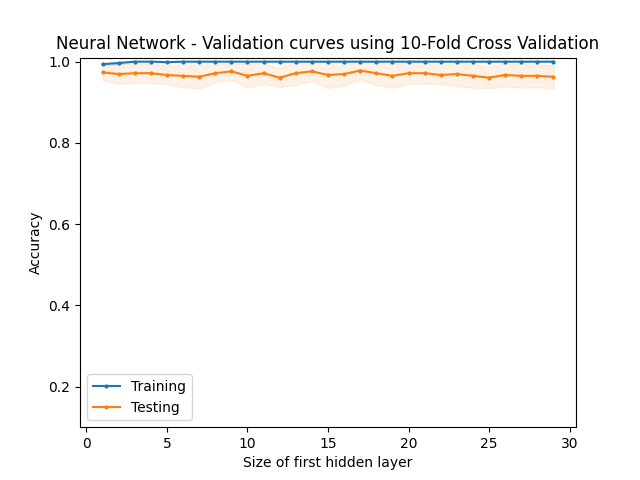


Figure 4 – Validation curves for Pruned Decision Tree

The rule of thumb of the size of the first hidden layer is to be half of a data set[[15]](#footnote-15), so the range was selected to run from 1 to 30, the total number of features in the dataset. As seen in figure 4, the testing accuracy remains relatively constant, with peak performance at 17 nodes in the first hidden layer. The relatively constant accuracy is expected for higher number of learning nodes as the weights of each node are adjusted to fit the feature inputs thought back-propagation of the errors to adjust the weights of each node to reduce error in the next iteration out of 100 as was set.[[16]](#footnote-16) However what was not expected was the consistent performance even for a small amount of hidden layers, where hidden layers from 0 to 10 also maintain an accuracy of over 90% in the test dataset.

The number of nodes in the second layer was varied from 0 to 30 in a similar fashion, but the same results of the consistent accuracy of the model above 90% was again observed. Due to the minimal impact of having the second layer over the first and to keep the model simple, the second layer was removed entirely from the later optimizations.

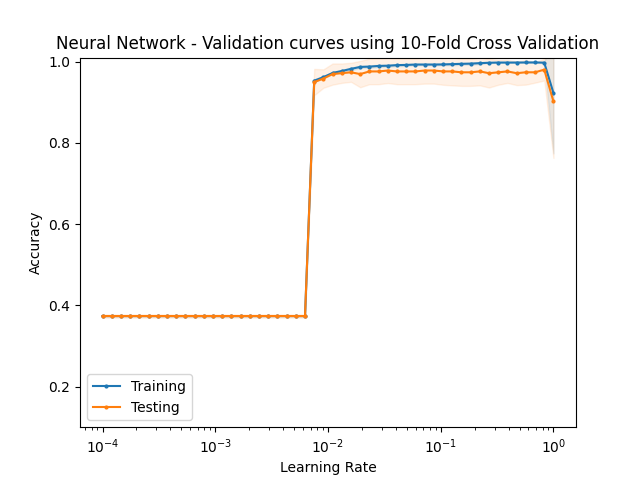
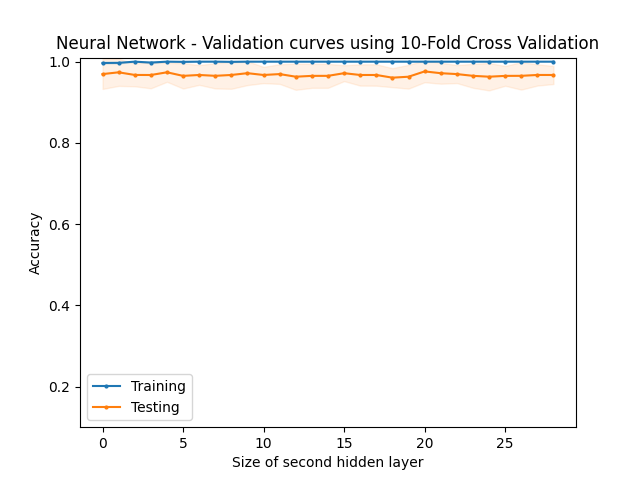
Afterwards the learning rate was tested with the validation curve, as this concept was taught explained more thoroughly in the course lectures. The value of learning rate was varied from 0 to 1 and its effect on performance was plotted out below. To use the learning rate, the stochastic gradient descent solver with a learning rate of ‘constant’ was selected over lbfgs, where the previous optimal hidden layer size was used.

Figure 5 – Validation curves for Neural Networks

As seen in figure 6, the accuracy of the model accuracy jumps to around 95% as learning rate reaches around 0.01, with the optimal learning rate equal to 0.829. The large increase in accuracy was expected, as 0.01 is the usual value in which the model strikes a good balance between changing fast enough from previous errors using backpropagation and settling too quickly potentially in a local minimum. The high value of the optimal learning rate of 0.8 was relatively unexpected, as the optimal learning rate was expected to be around 0.01, but this may suggest that there may not be a lot of local minima within the dataset. The accuracy does dip when the learning rate nears 1 as expected, as it may have fallen into a local minimum and assumed it as a global minimum when failing to check outside that range.

The optimal learning rate was attempted to be lowered by using different learning rate methods like invscaling or adaptive, but these methods only decrease the learning rate over training iterations.

# Support Vector Classification

Support Vector Classification can be deployed for this dataset as it can classify the tumor as malignant or benign. Different SVC kernels with varying penalty C parameters were compared with each other with default values otherwise, as each of the parameters are specific for different kernels.

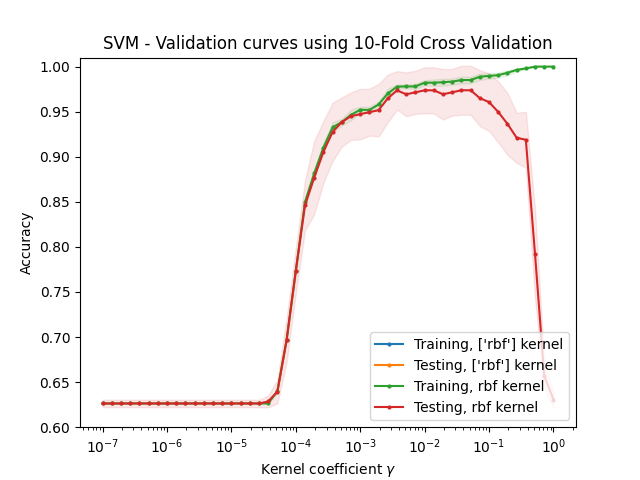
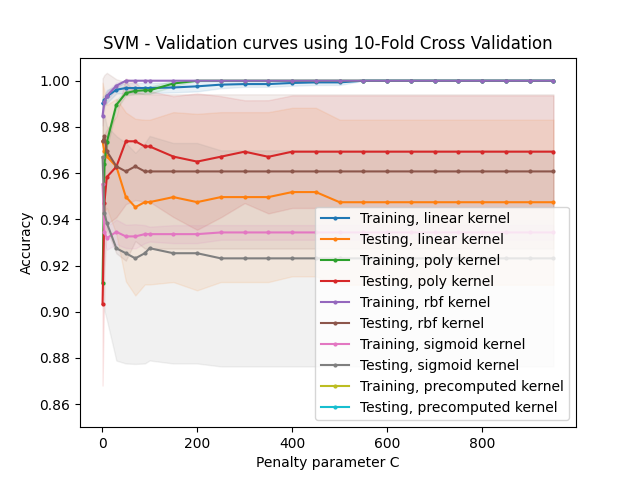


Figure 6 – Validation curves for Support Vector Machines

The highest performing SVM kernel was the RBF which is the Radial Basis Function, which uses a radial formula to calculate a data point is malignant or benign. The penalty value C was found to be optimal at 5 for the testing datasets, which controls the margin in which an outcome is determined to prevent misclassification.[[17]](#footnote-17) This moderate value of C is appropriate as a low C makes the decision surface smooth while a high C would lead to overfitting of the model as observed in the left portion of figure 6 where the training datasets are much more inaccurate than the testing datasets.

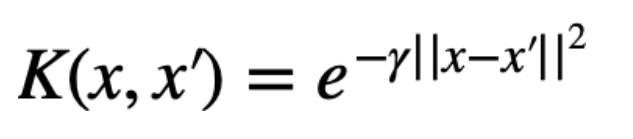


Figure 7 – Radial Basis Function kernel

The radial basis function can be described with the function, which informs how the model should evaluate nearby data points to determine the outcome.[[18]](#footnote-18)

Gamma controls the how similar two data points must be to be classified together, where a smaller gamma is more tolerant of data points differences and group them as benign.[[19]](#footnote-19) The optimal gamma was then optimized for an SVM with an RBF kernel as shown below, as it was the only parameter designed for RBF usage.

As shown in the figure above, the accuracy increases as the gamma coefficient increases for both testing and training datasets, before dipped for the testing dataset after gamma increases after 0.0372. The divergence of accuracy between the testing and training dataset is indicative of overfitting of the model, which is expected as a large gamma causes the model to be more tolerant of differences between the datapoints, causing more misclassifications and overfitting in the testing dataset.

An interesting observation is how the accuracy of the model does not go under 63% no matter the gamma value. This is likely due to the model classifying all the data points as either malignant or benign when the tolerance for the other class is too low. If the test and train dataset had 63% of benign cases, this would explain why the accuracy never goes under 63% as the model would guess all the outcomes to be benign.

## K Nearest Neighbor

The KNN model was optimized for parameters n\_neighbor and p power parameter with the Minkowski metric, with the kNN weight\_functions tested with both distance and uniform. The other parameters were kept to their default values for simplicities sake.

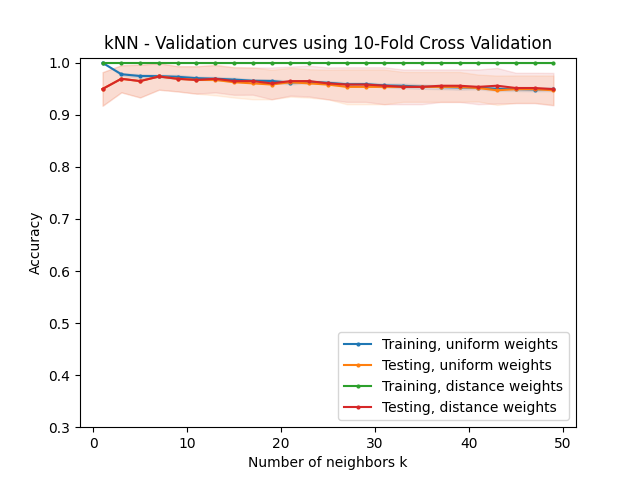


Figure 8 – Validation curves for k Nearest Neighbor

From the figure above, the accuracy of the validation peaks at 97.4% with a k value of 7. Surprisingly, the uniform weights performed similarly to the distance weight functions. It was expected that distance weights would perform better as it would account for similarity between points before making a prediction, but it appears that the accuracy is about the same regardless. This is likely due to the lack of training data.

What is expected however is the appearance of overfitting as k decreases from 3 to 1 nearest neighbor, where the accuracy of the test set increase to 100% due to no averaging of the data points, while the test set accuracy decreases as the results are more influenced by a lesser amount of data points instead of an average of nearby data points. The accuracy also continues to dip as k increase from 7 onwards, due to influence of too many data points. After k nearest neighbors goes above 40, we can also observe a slight dip when the uniform weights under performs the distance weights. This is due to distance weights factoring in the similarity between the data point and query point when calculating their influence on the result.

Confusion Matrix is also very accurate with all values being predicted correctly aside from 3 values.

## Comparisons and Observations

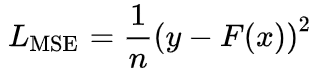
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Training Time (s) | | Fitting Time (s) | Optimal Accuracy |
| Pruned Decision Tree | | 0.0083 | 0.0001 | 92.1% |
| Gradient Boosted Tree | | 0.1976 | 0.0007 | 95.6% |
| Neural Networks | | 0.2447 | 0.0011 | 95.6% |
| Support Vector Classifications | | 0.0064 | 0.012 | 98.2% |
| K Nearest Neighbors | | 0.008 | 0.1726 | 97.3% |

Figure 9 – Table of Algorithms and their performance metrics

One observation from figure 8 is how pruned decision trees have the lowest accuracy, likely due to the small dataset causing less redundant branch nodes. Any pruning performed on those nodes would lower the accuracy. An interesting detail is how fast the training and fitting time for that is, especially since without pruning, the decision tree is simple a weak learner that can be trained and fitted quickly, with only slight delay through the stratified cross-validation bagging method employed.

This also explains why the gradient boosted tree take over 20 times longer to train than the pruned decision tree, as the gradient boosted tree attempts to retrain on the poorest performing branch node splits by creating new split conditions. This process is also repeated in the 10-fold cross validation, which results in a much longer training time than doing a simple decision tree training in the pruned decision tree. However, the accuracy is much improved in the gradient boosted tree compared to the pruned method.

The loss function of the gradient boosted tree is:



The gradient boosting tree uses this function to decrease the LMSE in the next iteration of the branch. This is achieved by the derivative of the LMSE ­value, where a value of 0 would signify a local minimal has been reached, while a negative value would indicate the function can still further be optimized.

Another observation is how k Nearest Neighbor trains very quickly yet takes the longest time to predict. This may be caused by its lazy learner properties where it computes the model only when queried with a datapoint, whereas the other algorithms are eager learners that compute a decision tree or deterministic function during the training procedure. While this method does not take the longest to run, it may not be ideal for scenarios where queries are constantly required from a constant model.

The Neural Network has the highest training time of the five, and this is due to the repeated optimization of the weights through back-propagation of the results, leading to extended training times but allows for fine-tuning of the weights in the neural nodes based on the loss obtained in the previous epoch. With the multiple layer nodes, there is also the forward-propagation where the weighted sum of the first layer nodes activation functions into the second layer of nodes.[[20]](#footnote-20) The final error rate is effective lowered by optimized weights, making the model reliable by increasing its generalization but taking a significant time to train.

The time to query is also average, as the query point must be multiplied and aggregated across the different layer of nodes and the activation function before the result is generated. Ultimately, its performance matches that of Gradient Boosting, but takes a longer time to train and query.

Support Vector Classifications meanwhile has the highest accuracy and lowest total time to train and query of all the other models. Its fast time to train is likely due to its simple algorithm in finding an algorithm to determine the prediction plane that separates the outcomes with the radial basis function without the need for model retuning like neural networks or boosting decision trees. Its moderate query time may be attributed to calculating the distance between the query point and decision plane, although it was a personally a bit surprising to see the neural network query time be nearly ten times faster. This may likely be due to the simple neural network as the number of nodes was minimized and layer of hidden nodes set to 1, requiring less calculations to generate a prediction compared with the SVM.

The high accuracy may be attributed to the many different kernel functions available, with the radial basis function being the most appropriate in classifying this specific dataset.

## Conclusion

After much analysis, the highest performing model was determined to be Support Vector Machines with Radial Basis Function kernel, as it had the highest accuracy and lowest time to train and time to query on this dataset.

To improve the next iteration of this analysis, the dataset should be significantly larger to see how models suitable for large data analysis improve, like the pruned decision trees and neural networks.

1. *Sklearn.datasets.load\_breast\_cancer*. scikit. (n.d.). Retrieved February 13, 2022 [↑](#footnote-ref-1)
2. https://www.kaggle.com/imdevskp/corona-virus-report [↑](#footnote-ref-2)
3. https://www.kaggle.com/c/covid-19-mortality-risk/data?select=test.csv [↑](#footnote-ref-3)
4. Araújo (1961, January 1). Understanding stratified cross-validation. Cross Validated. Retrieved February 13, 2022, [↑](#footnote-ref-4)
5. L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984. [↑](#footnote-ref-5)
6. Dmlc. (n.d.). *Xgboost/demo at master · DMLC/xgboost*. GitHub. Retrieved February 13, 2022 [↑](#footnote-ref-6)
7. S. Theodoridis, Y. Kopsinis, K. Slavakis, S. Chouvardas, Sparsity-Aware Adaptive Learning: A Set Theoretic Estimation Approach., IFAC Proceedings Volumes, Volume 46, Issue 11, 2013 [↑](#footnote-ref-7)
8. agcalaagcala *GBM vs XGBOOST? key differences?* Data Science Stack Exchange. Retrieved February 13, 2022 [↑](#footnote-ref-8)
9. Velidou, D. S. (2018, June 7). *Uncover the structure of tree ensembles in Python*. Medium. Retrieved February 12, 2022 [↑](#footnote-ref-9)
10. Miguel Trejo (1966, May 1). *Learning rate in Adaboost sklearn*. Cross Validated. Retrieved February 13, 2022 [↑](#footnote-ref-10)
11. & TylerrTylerr 1. (1969, May 1). *What's the purpose of learning rate in sklearn adaboost implementation*. Cross Validated. Retrieved February 13, 2022 [↑](#footnote-ref-11)
12. sapysapy 17722 silver badges33 bronze badges. (1958, June 1). *How to choose the number of hidden layers and nodes in a feedforward neural network?* Cross Validated. Retrieved February 13, 2022 [↑](#footnote-ref-12)
13. https://scikit-learn.org/stable/modules/generated/sklearn.neural\_network.MLPClassifier.html [↑](#footnote-ref-13)
14. http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf [↑](#footnote-ref-14)
15. Brownlee, J. (2019, August 6). *How to configure the number of layers and nodes in a neural network*. Machine Learning Mastery. Retrieved February 12, 2022, from https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/ [↑](#footnote-ref-15)
16. *Comparison searching process of linear ... - iopscience*. (n.d.). Retrieved February 12, 2022 [↑](#footnote-ref-16)
17. ASXASX 43122 gold badges44 silver badges77 bronze badges, & JavierfdrJavierfdr 1. (1963, January 1). *Intuition for the regularization parameter in SVM*. Data Science Stack Exchange. Retrieved February 12, 2022 [↑](#footnote-ref-17)
18. Dobilas, S. (2022, February 5). *SVM classifier and RBF kernel - how to make better models in Python*. Medium. Retrieved February 12, 2022 [↑](#footnote-ref-18)
19. Sreenivasa, S. (2020, October 12). *Radial basis function (RBF) kernel: The go-to kernel*. Medium. Retrieved February 12, 2022 [↑](#footnote-ref-19)
20. Al-Masri, A. (2019, January 29). *How does back-propagation in Artificial Neural Networks Work?* Medium. Retrieved February 12, 2022 [↑](#footnote-ref-20)