Introduction

In this paper, 5 different supervised learning techniques are explored. The five algorithms, including Decision Trees, Boosted Decision Trees, K-Nearest Neighbors, Support Vector Machines, and Neural Networks, are tuned and tested on two interesting classification problems. The goal of this paper is to provide a basic understanding of the algorithms, the rationale behind tuning their hyperparameters, the tradeoffs between bias and variance, and a comparison between the different techniques.

Data

The datasets illustrated in this report are the Diabetes dataset[1] and Breast Cancer dataset[2]. The Diabetes dataset contains diagnostic testing information for 15,000 patients who have tested either positive or negative for diabetes, as given in the "Diabetic" column. This is used as the label for classification. There are 8 features used in the classification, including Pregnancies, BMI, Age, etc. The Breast Cancer dataset contains clinical data for cells obtained from breast mass. The label for this classification problem is the "Class" column, which indicates if the cell sample is benign or malignant. The 9 features for this dataset include information about the breast cells, like cell thickness, uniformity, and smoothness. This data contains 683 rows of datapoints. The large difference in the chosen dataset sizes is intentional, and it will be interesting to compare model performance on two such varying datasets. For example, clear difference will be shown in model training and prediction times. Some questions that provoke deeper analysis include: How will model performance vary as a function of training size? Does more data necessarily result in better models? Will a smaller dataset require more simple or complex models to identify differences? Additional information about the datasets can be found in the README.txt.

Both datasets originally contained imbalanced target classes. The Diabetes dataset had 10,000 nondiabetic patients and 5,000 diabetic. The Breast Cancer dataset had 444 benign cases and 239 malignant. Both datasets were down sampled with random samples taken from the over-represented classes to match the under-represented classes. In the end, 10,000 rows of diabetes data and 478 rows of breast cancer data was used for this study.

Classification Accuracy is the primary metric used in this paper to evaluate model performance. Accuracy is a useful metric to use when target classes are well balanced. With imbalanced classes, accuracy may be misleading since models that are biased towards the dominant target class are preferred.

Modelling

Decision Trees

The Decision Tree classifier is a relatively simple algorithm and is widely used in machine learning for many reasons, including low computational cost and high interpretability. I first fit a decision tree with default parameters to establish the baseline performance for both datasets. The baseline performance for the Diabetes dataset was an accuracy score of 88.3%, and 93.05% for Breast Cancer.

Hypertuning

After establishing the baseline performance, a GridSearch was performed to fine-tune the hyperparameters. For DecisionTreeClassifiers, the 'criterion', 'max_depth', 'min_samples_split', 'min_samples_leaf', and 'max_features' were tuned, giving a CV score of 91.23% for Diabetes and 97.28% for Breast Cancer. The effects of the hyperparameters are explored next.

The 'criterion' supported by scikit-learn include "gini", "entropy", and "log_loss", each of which provide a method to measure the quality of a node split. The 'max_depth' param allows the user to prune decision trees by limiting the depth of the tree. Left to the default value of None, a decision tree will expand in depth until all leaves are pure. In general, the deeper a decision tree grows, the more complexity is added to the model. This is because the decision trees will include more splits, more nodes, and more leaves.

[1] https://github.com/MicrosoftDocs/ml-basics/blob/master/data/diabetes.csv
[2] https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)

While this helps capture more information about the training data, it is also the primary cause of overfitting with decision trees. Figure 1 illustrates the model accuracy with respect to 'max_depth' and the chosen 'criterion' in an experiment with 5-fold cross validation.

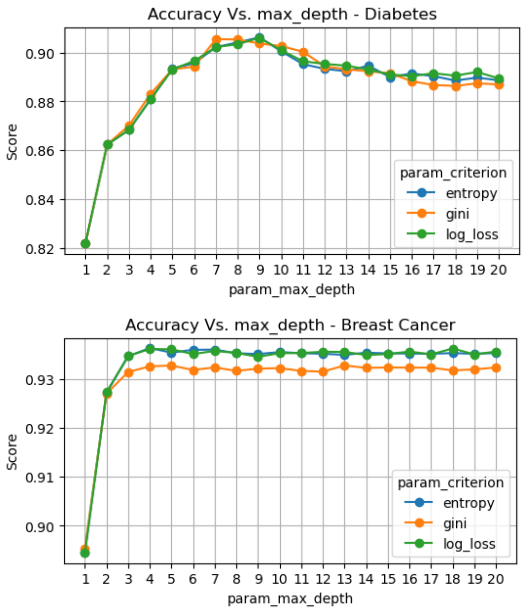


Figure 1

From Figure 1**Error! Reference source not found.**, there is no major difference in performance amongst the 3 criterion. This is expected, and Raileanu and Stoffel, authors of the paper "Theoretical comparison between the Gini Index and Information Gain criteria" conclude that only 2% of studied cases showed disagreement between the two functions[3]. There is generally no significant difference between the criteria, and "best" criteria should be determined empirically for a dataset.

When observing the increase of max_depth in Figure 1, there is a significant difference in model accuracy. At the lower range, the low accuracies signal underfitting. As the max_depth increases, so does the model accuracy until a certain point. This phenomenon is best explored using a validation curve in Figure 2.

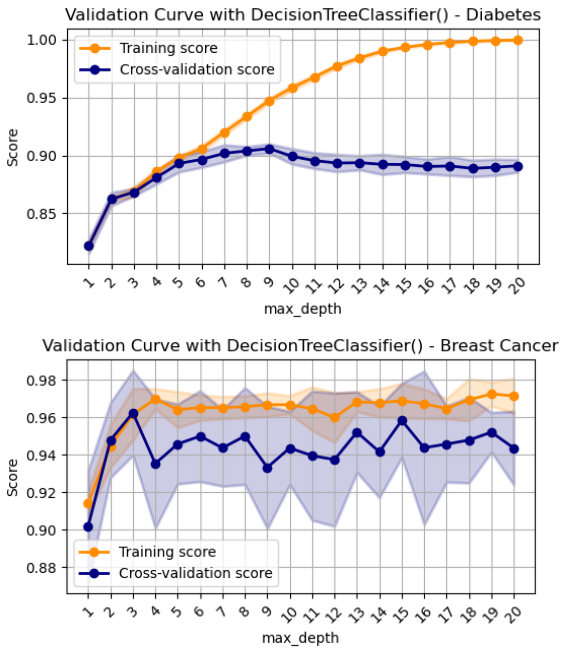


Figure 2

The Diabetes plot in Figure 2 shows the cross-validation score increasing until max_depth = 8, after which it decreases while the training score continues to rise. This split between the two curves illustrates the model overfitting to the training data as max_depth increases. Because the model overfits to the training data, it does not generalize well to the validation data. This observation is closely related to Occam's Razor, a principle that states a simpler hypothesis that achieves the same result should be preferred. In this case, a max_depth > 8 shows similar validation scores, so the smaller max_depths should be preferred. A similar phenomenon happens in the Cancer plot, but with much less disparity between the two curves. This suggests less complexity and noise between the classes. The validation curve shows that max_depth is a good parameter to tune for pruning, a term for removing deeper tree nodes to better generalize to new data.

There are a few other hyperparameters worth mentioning, but are not fully analyzed in this paper because they are less illustrative. For example, "max_features" allows you to control the number of features used for finding the best split. For datasets with many features, decision trees tend to overfit[4]. In our datasets, the ratio of samples to features is

[3] https://www.unine.ch/files/live/sites/imi/files/shared/documents/papers/Gini_index_fulltext.pdf
[4] https://scikit-learn.org/stable/modules/tree.html#tree

sufficiently high, which helps mitigate this problem. "min_samples_split" helps control overfitting as higher values means less splits in a tree.

Using the best classifier from the GridSearch, an experiment was done to create learning curves and understand the effect of training size on model performance, as shown in Figure 3.

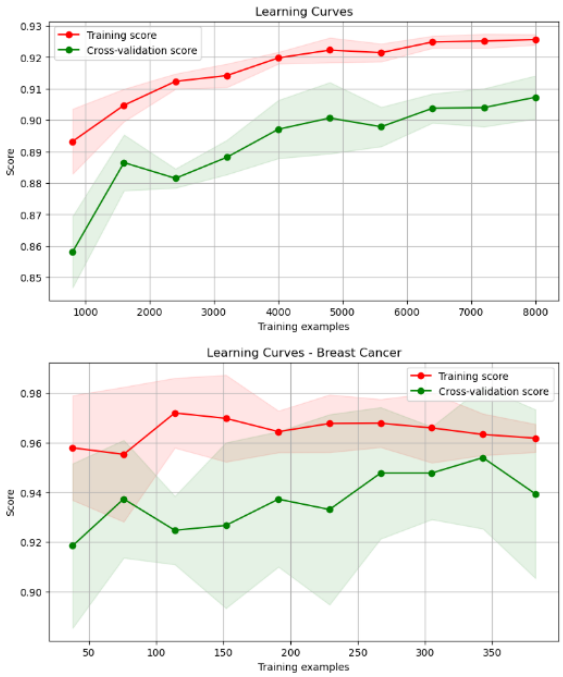


Figure 3

There are two main takeaways from this experiment: 1) The validation curve has not fully converged, meaning we could benefit from adding more training instances. 2) the hyperparameter-tuned DecisionTreeClassifier shows moderately low bias and variance.

In the Diabetes data, the validation curve has not fully converged and is still increasing, albeit slowing down, as we add more training data. In this case, adding more training instances would help decrease validation errors. It seems that 8000 data points is a decent amount for training, but a little more may be helpful. Similarly in the Cancer data, the validation curve appears to be increasing, suggesting more training could improve model accuracy.

Overall we have high validation and training accuracies (both >90%) when we train on 80% of the data.

Because of the low validation error, there is not an indication of a large bias problem. Instead, the correspondingly low training error indicates low bias, and the training data is fitted well by the model. Because of the bias-variance tradeoff, a lower bias comes with higher variance. This is represented by a gap between the training and validation learning curves. Because Figure 3 shows learning curves for an already hyperparameter-tuned model, the gap between the learning curves is not very wide. Generally, a wider gap signifies greater variance.

AdaBoost

AdaBoost is a popular ensemble boosting algorithm. As an ensemble method, it uses multiple weak estimators to build a strong one with better generalizability. As a boosting method, it focuses learning on areas where the model doesn't perform well. Again, I first fit AdaBoost with default parameters to establish the baseline performance. The baseline performance for the Diabetes dataset was an accuracy score of 91% and 94.44% for Breast Cancer, which is higher than the baselines for the DecisionTreeClassifer.

Hypertuning

With the baseline performance established, a GridSearch was performed to tune the hyperparameters. For AdaBoost, the 'n_estimators', 'learning_rate', and 'algorithm' were tuned, giving a CV score of 95.49% and 96.86% for Diabetes and Breast Cancer, respectively. The effects of the hyperparameters are explored next.

As the scikit-learn user guide suggests, the 'n_estimators' is the main parameter to tune for better results[5]. The 'n_estimators' controls the total number of weak learners, which in our case are DecisionTreeClassifiers with max_depth=1. With such small depths, the weak learners are prone to high bias and underfitting, but by combining many weak learners, bias can be reduced. This is illustrated in Figure 4, where we see low validation errors and even lower training errors with additional estimators, indicating low bias problems.

[5] https://scikit-learn.org/stable/modules/ensemble.html#adaboost

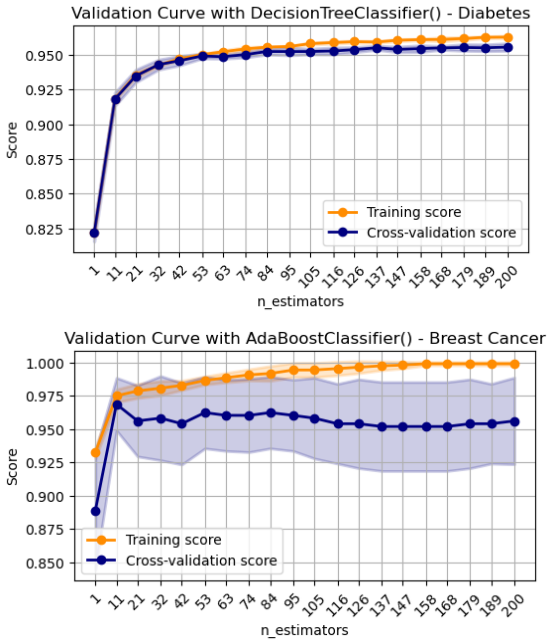


Figure 4

The 'learning_rate', which affects how much each iteration and successive classifier contributes to the ensemble, is also tuned. It is evident from Figure 5 that the learning_rate has less impact on improving model accuracy than n_estimators, bumping the score from 90.5% to 95.2%. There is a trade-off between n_estimators and learning_rate, where increasing one of them would require a lower the other. This can be tuned empirically from using GridSearch.

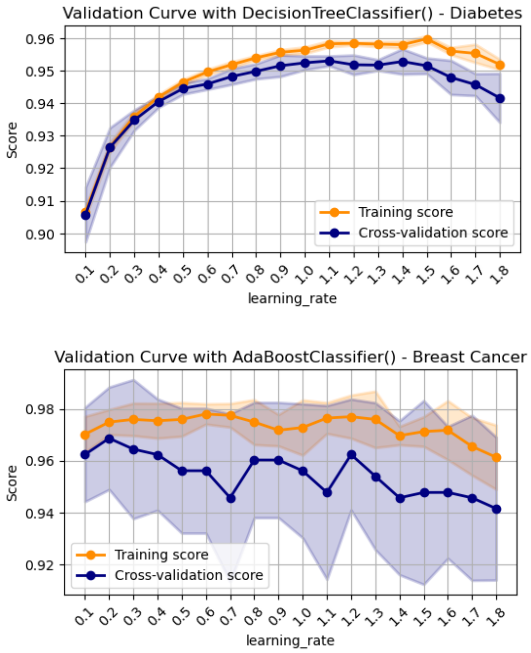


Figure 5

Lastly, the 'algorithm' of AdaBoost was tuned. As with the DecisionTreeClassifier, the change in algorithm didn't show major improvements to model score. After searching for the best classifier, learning curves are created for AdaBoost as shown in Figure 6.

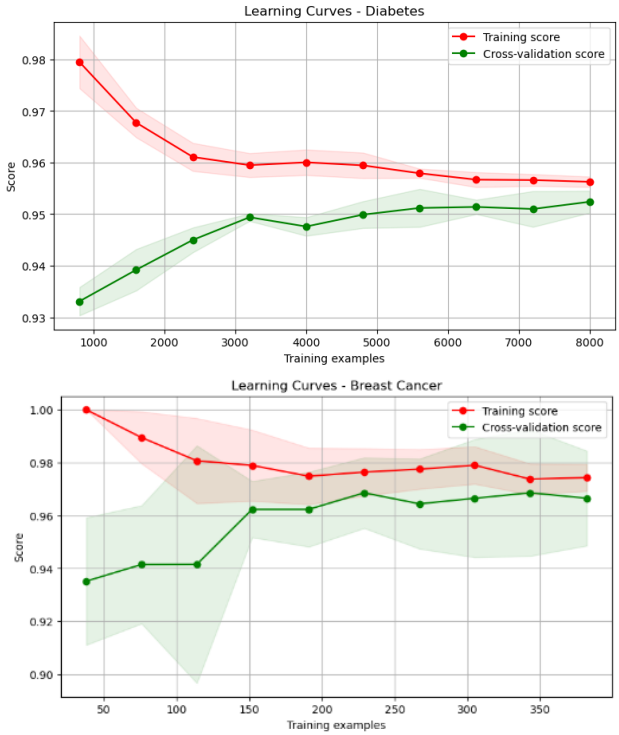


Figure 6

From the learning curves of the two datasets, two evident observations are that the training and cross-validation curves are converging, and both curves show small errors. The former observation suggests that there is no need to continue adding training instances, and even ~5000 Diabetes and ~300 Cancer training samples are sufficient. The latter observation is wonderful news, as a low training error suggests low bias and narrow gap indicates low variance.

In summary, the bias reduction from multiple weak estimators, in combination with the low variance from small trees, results in an ensemble method that shows both low bias and low variance.

k-Nearest Neighbors

The next model applied is scikit-learn's KNeighborsClassifier that implements a k-nearest neighbors (KNN) algorithm. With this model, the nearest neighboers of each data point vote on the target class and majority vote wins. Unlike the tree-based algorithms I've discussed so far, KNN is an instance-based model and is non-parametric. An advantage of this is that non-parametric models often

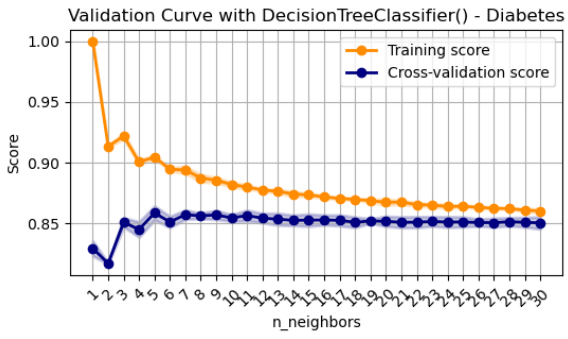
show success in classification where decision boundaries are very irregular[6]. I start by estimating the model performance using default parameters, yielding an accuracy score of ~85.07% for Diabetes and 98.61% for Cancer.

Up to this point, we haven't talked about any preprocessing or scaling steps. However, the KNN algorithm depends on calculating distances to the closest neighbors, so if the data is not scaled to the same scale, then the classifier could potentially be much more sensitive to some features than others. Therefore, the data is scaled before the model is fitted.

Hypertuning

Next, I tuned the hyperparameters to make improvements from the baseline performance. The hyperparameters tuned include "n_neighbors" and "p", which gave a best validation score of 89.66% and 96.86% for Diabetes and Cancer.

One of the main hyperparameters to tune for KNN models is the number of neighbors included in the majority vote. This parameter, n_neighbors, is highly dependent on the data so should be determined empirically. In general, smaller values of n_neighbors mean the model only considers closer neighbors that are the most similar. This leads to high variance, but low bias. Similarly, higher values of n_neighbors yield higher bias and lower variance, and suppresses the effects of noise. These concepts are corroborated by the validation curve shown in Figure 7.



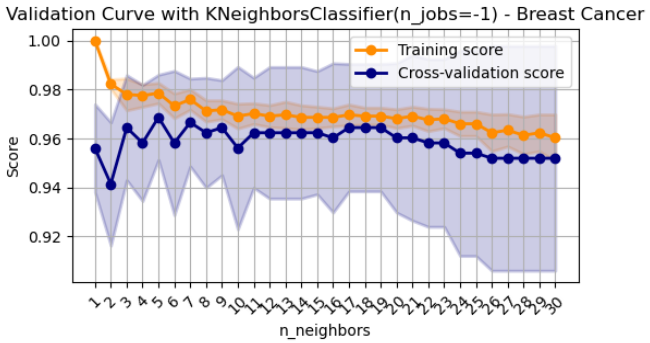


Figure 7

In Figure 7, as the n_neighbors increases, the training score decreases and the cross-validation score increases until both plateau. At the lower ranges of n_neighbors, the KNN model fits to the closest neighbors for each data point, allowing the model to be very flexible. This high flexibility and high variance (shown by large gap between the curves) fits the training extremely well, but doesn't generalize well to validation datasets. As n_neighbors increases, the model's variance decreases and is better able to generalize to unseen data.

The next hyperparameter tuned was "p", which is the power parameter for the Minkowski metric. In other words, it changes the distance calculation between data points, where p=1 is the Manhattan distance and p=2 is the Euclidian distance. "p" can be any arbitrary integer. From the experiment shown in Figure 8, it is evident that increasing p values reduces the accuracy of the model in both training and validation datasets. Although the Euclidian distance is the default parameter, it seems the Manhattan distance provides a better distance metric.

Lastly, for the KNeighborsClassifier, the 'algorithm' is not tuned because the model already automatically decides the most appropriate algorithm based on the input data.

[6] https://scikit-learn.org/stable/modules/neighbors.html#classification

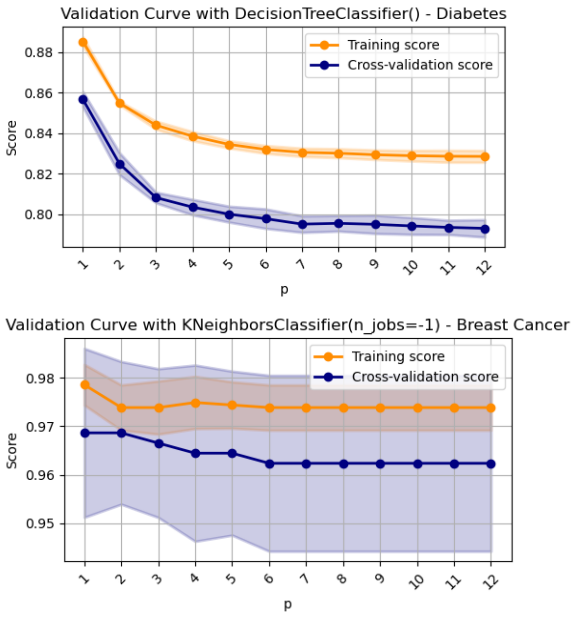


Figure 8

Using the best classifier from GridSearch, learning curves were created as shown in Figure 9.

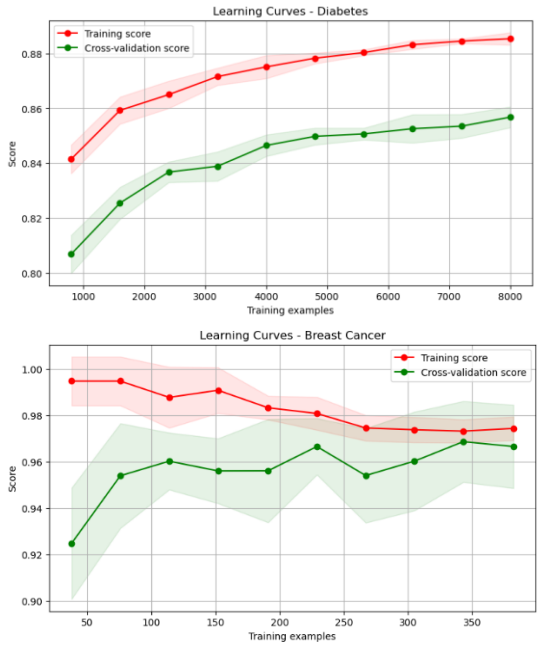


Figure 9

From Figure 9, we see that both training and validation learning curves are still increasing at the max number of training examples. This indicates that additional training samples would help improve the model performance for both datasets. Without further domain expertise regarding Diabetes and Cancer, it is difficult to make a judgement of if the validation errors, although small, are acceptable. Assuming that the errors are considered small, this helps conclude that there is not strong bias in the models. There is a gap of ~3% between the two curves, which signifies insignificant variance. Overall, by tuning the hyperparameters of the KNN model, there seems to be a good balance between bias and variance.

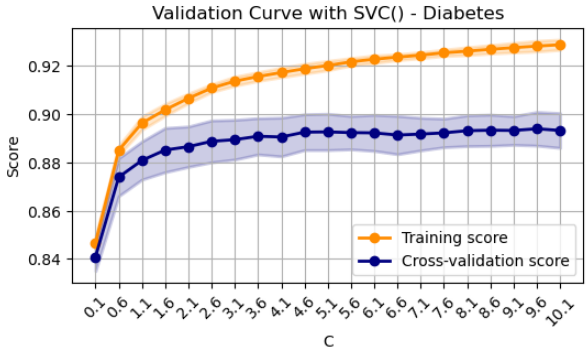
Support Vector Machines

The next model to consider is Support Vector Machines (SVM), which is an algorithm that creates hyperplanes to separate data into classes. As with KNN, Support Vector Machines are sensitive to scaling. This is because the SVM algorithm is calculating distances between support vectors and maximizing it to create a separating plane. Again, if not scaled to the same interval, the SVM classifier could unequally give importance to some features over others. Therefore, the data use for the following experiments are scaled. Using the default parameters for SVM, the baseline accuracy score is 87.4% for Diabetes and 98.61% for Cancer.

Hypertuning

To improve the baseline performance, I tried to tune the hyperparameters "C" and "kernel". The best estimator had an accuracy score of 89.39% for Diabetes and 97.69% for Cancer.

One of the main hyperparameters to tune is ‘C’, the regularization parameter. Regularization prevents models from overfitting to training data points. By increasing the ‘C’ value, the regularization decreases, and the model will fit training points better, bias will decrease, and variance will increase. We see evidence of this in the validation curves in Figure 10. As the ‘C’ value increases, the training accuracy score continues to increase even as the validation score plateaus. This signals overfitting beyond approximately C=5 for the Diabetes dataset.



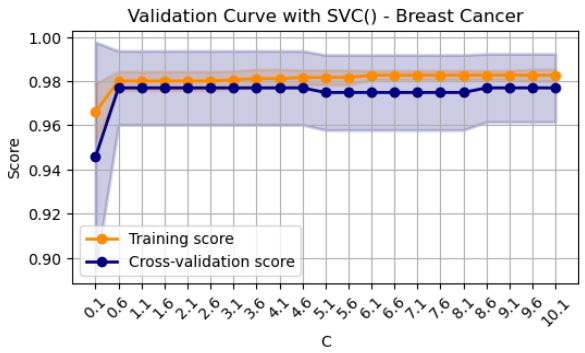


Figure 10

The other important hyperparameter to tune is the 'kernel', which is the transformation done to the data to more easily separate the target classes. For example, linear data can easily be separated by Linear SVM. When the data is not linear, the linear SVM is no longer ideal, and other kernels would be chosen, which increases model complexity. In Figure 11, the validation curve shows that the rbf kernel, or radial basis function, performs the best for both training and validation for the Diabetes dataset, while polynomial kernel is best for the Cancer data. This implies that the target classes for Diabetes are not linearly separable given the features used, while the Cancer data is.

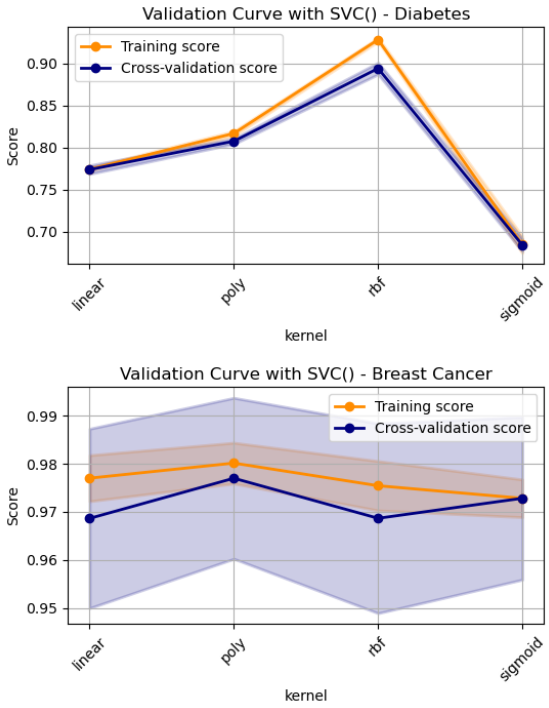


Figure 11

With the best model provided by hypertuning using GridSearch, the learning curves in Figure 12 are generated.

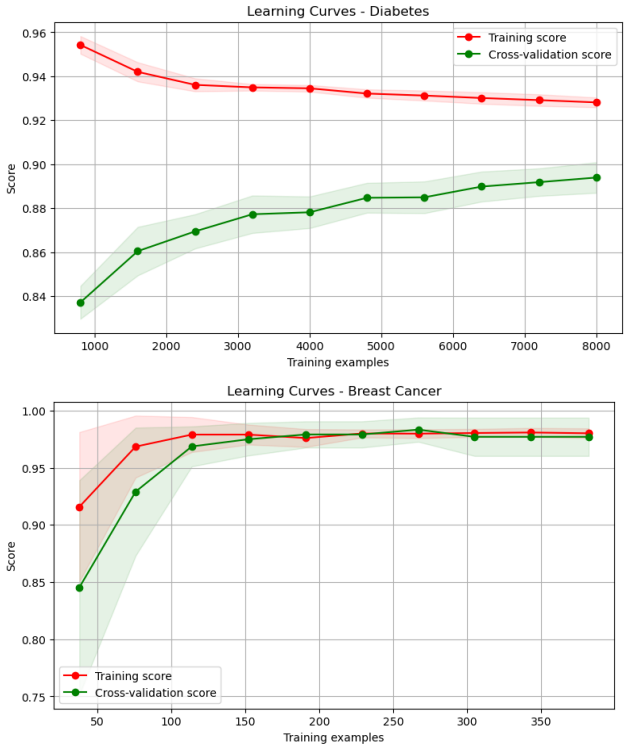


Figure 12

From Figure 12, it's evident for Diabetes that the models would continue to improve with additional training data since the validation curve has not fully plateaued yet. Extrapolating the curve, it would be approximately sufficient to include a few thousand more training examples. The Breast Cancer curves show that the models have sufficient data to learn from, and more examples would not help. Furthermore, regarding variance in Diabetes, the gap between the two curves shows moderate variance in the model. To decrease the variance, one possibility would be to simplify the model by reducing the number of features.

Neural Network

The last learning algorithm used in this study is a Multi-layer Perceptron (MLP), a neural network model. While an incredibly powerful method to approximate any function, Neural Networks are also often seen as "black-box" models as it's very difficult to understand how individual neurons create the final output. Like several of the previous algorithms discussed, Multi-layer Perceptron models are also sensitive to feature scaling so data is scaled prior to model training. Using the default parameters of the MLPClassifier model provided by scikit-learn, a baseline accuracy score of

91.97% for Diabetes and 98.61% for Cancer was achieved. This is the highest baseline score of all the models considered.

Hypertuning

There are a few hyperparameters, chosen out of many, that were tuned to optimize the MLP, including the number of hidden layers, number of neurons per hidden layer, and iterations. With a GridSearch, the following hyperparameters were tuned: 'alpha', 'hidden_layer_sizes', and 'max_iter'. The best estimator yielded an accuracy score of 92.41% for Diabetes and 97.49% for Cancer.

The parameter 'hidden_layer_sizes' allows the user to add hidden layers with an arbitrary amount of neurons. It is interesting to understand the effects of both (adding more layers and adding more neurons per layer) on model performance. Figure 13 shows the validation curve for increasing neurons per layer for a single hidden layer.

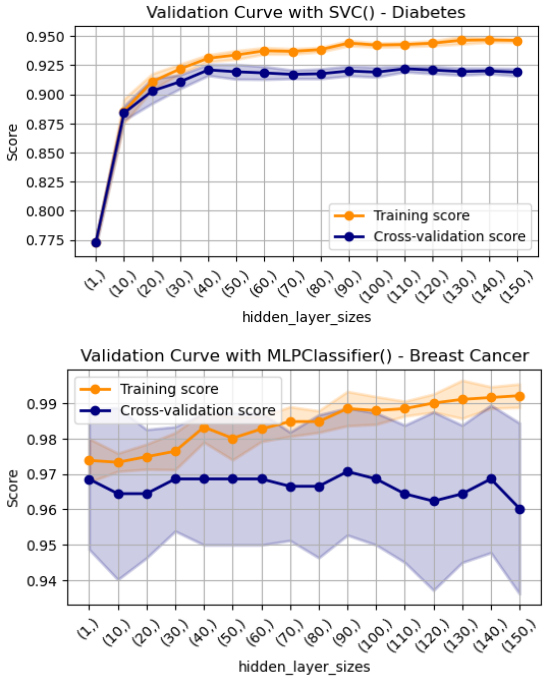


Figure 13

Interestingly, Figure 13 shows for Diabetes that adding neurons to the hidden layer increases model performance initially, but the improvement plateaus at higher values, while there is not visible improvement for Cancer data. At approximately 40 neurons in the hidden layer, the cross-validation score shows insignificant improvement while the training score continues to improve, signaling overfitting. Again, this illustrates Occam's Razor concept, and having a simpler model (one with less hidden_layers) is sufficient. For the Cancer data, adding more neurons and thus more complexity to the model does not help models increase above 97% accuracy. This may suggest that models have already learned everything it can from the data.

Figure 14 illustrates the effect of increasing the number of hidden layers while keeping the number of neurons static. Up to four hidden layers are tested to see model performance.

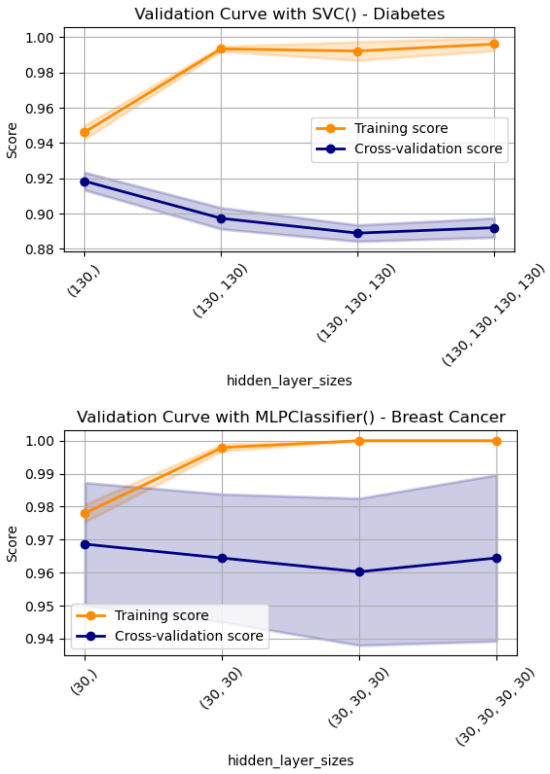


Figure 14

While the training score increases with increasing layers, the validation scores show continuous decrease for both datasets. This shows that the model is adding needless complexity, which fits to the training data very well, but is unable to generalize to previously unseen validation data.

Changing the 'max_iter' parameter also helps improve the model to certain extents. The validation curve for varying max iterations is shown in Figure 15.

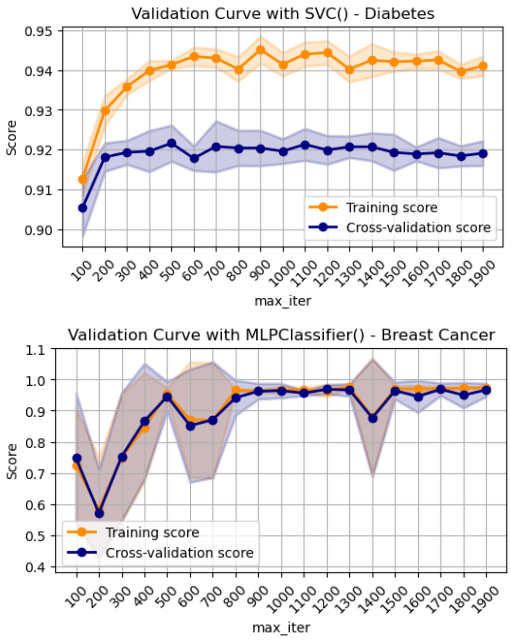


Figure 15

Increasing the iterations helps improve the model accuracy score initially, but quickly show limited returns as both training and validation curves flatten. This chart illustrates the importance of using early-stopping when training neural networks, since more iterations may not lead to better performance.

Finally as recommended by Scikit-learn, finding a reasonable regularization parameter alpha is also important for neural networks. As a reminder, more regularization means the model is less likely to overfit the data.

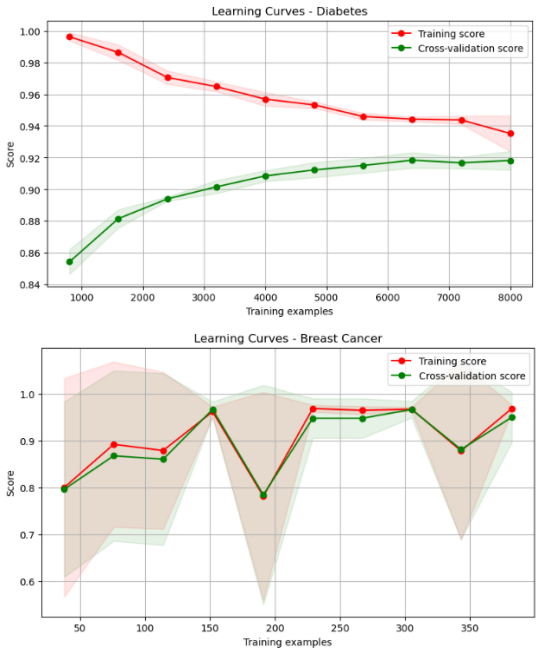


Figure 16

In Figure 16, the learning curves for GridSearch's best estimator are shown. For Diabetes, with low number of training samples, the neural network is easily able to overfit to the training data, but does not estimate validation data well. With more training examples, the learning curves quickly converge towards 92% with very small gap in between the training and validation scores. This shows two important observations. First, the quick convergence means that there is sufficient training data for the Multi-layer perceptron model. Secondly, the neural network is able to strike a good balance between bias and variance, since the model both fits the training set well and generalizes to unseen data well.

However, the learning curves look drastically different for the Breast Cancer dataset. There is neither continuous increase for the training or validation scores. Instead, both increase sporadically with large error bands. Due to these observations, the neural networks look unstable for this small dataset. It begs the question, is a neural network - with all its bells and whistles - too complex for such a dataset? As usual in the data science world, there is no definitive answer as it depends on the dataset. However, that does appear to be the case here, where simpler models may be preferred.

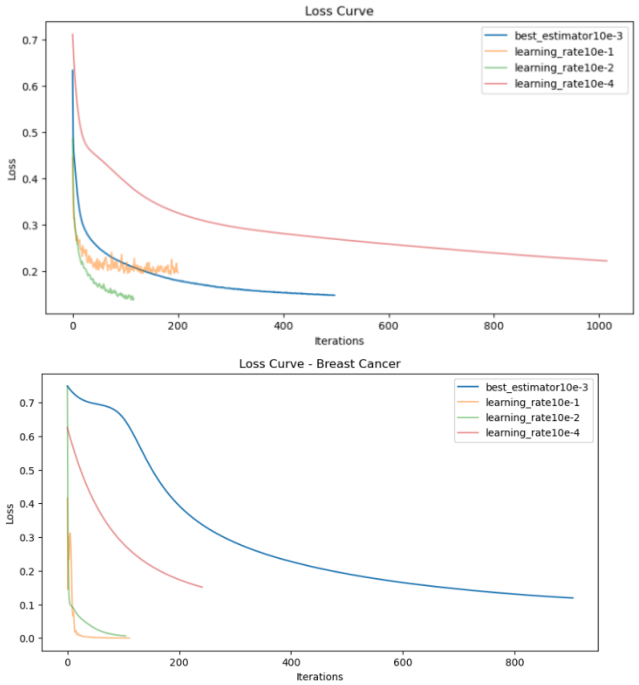


Figure 17

Figure 17 shows the loss curve for the best estimator in the training set, as well as a few other loss curves generated by changing the 'learning_rate_init' parameter for comparison. The blue curve shows the tuned model's performance. With increasing iterations, the log-loss sharply decreases and then flattens out. The shape of the curve is intuitive, since the Neural Network is minimizing the loss function with each iteration. The curve ends when the validation score no longer improves for 10 consecutive epochs. This curve helps us identify a good number of iterations when the model is well trained, rather than stopping model training too early and performance can be improved, or overtraining the model.

The loss curve can also be used to diagnose optimization issues with the model. By studying the different curves in Figure 17 with different learning rates, we can glean some insights. The orange and green curves both have higher learning rates than the best estimator, while the red curve has a lower learning rate. For the Diabetes data, the orange curve with the largest learning rate decreases very quickly at first, but it plateaus at a higher level then the blue curve. This means the steps in learning are too large and the model is unable to find the minima in the loss. Also note the increased instability/noise in the orange and green curves which have higher learning rates. Again, this instability happens because the high learning rate makes the optimization function bounce around the minima, and the function is unable to descend without a smaller learning rate. On the other hand, the red curve has a smaller learning rate so the loss decreases slowly and it takes many iterations to improve the model. By studying this chart, a healthy learning rate would be somewhere in between the blue and green curves where the loss can be optimized quickly, yet still be minimized.

The Breast Cancer dataset provides an interesting contrast to the loss curves from the Diabetes datset. Here, the blue curve converges more slowly than the orange and green curves too, but the red curve that has the smallest learning rate converges faster than the blue curve. However, the red curve also ends with a higher loss than the blue curve. This suggests that the red curve with a smaller learning rate may have converged to a local minima rather than the global minimum. Again, tuning the neural network for the Breast Cancer data seems more difficult.

Model Comparison

There is a lot to be learned in the comparison of all 5 supervised learning techniques. The comparisons posed in this section is for all 5 algorithms that have been hyperparameter-tuned. I will discuss the model performances in terms of accuracy score, and model scalability in terms of training and prediction times. Figure 18 shows a side-by-side comparison of the 5 model performances. In worst-to-best order of cross-validation performance for Diabetes, the algorithms are ranked as such: SVC, KNeighborsClassifier, DecisionTreeClassifier, MLPClassifier, AdaBoostClassifier.

First, let's analyze the model performances on the training sets. Of notable importance is the fact that the DecisionTreeClassifier and KNeighborsClassifer models have an increasing training curve, whereas the other models have decreasing training curves. This may suggest that these two models have higher bias than the others, since the models are less flexible to fit the training data well. This is aligned with our intuition regarding the two models since they are relatively simpler ML models. Note this is not saying that decision trees are high bias models; It is merely a reflection of this specific decision tree that was fit on the training datasets. Next, the relatively larger gaps between the training and validation curves for the SVC, DecisionTreeClassifier, and KNeighborsClassifer also indicate that these three models have higher variance than the MLPClassifier and AdaBoostClassifier. These larger gaps are created by high validation errors and low training errors, meaning the models fit well to the training set, but are not as good at generalizing to the validation set. In terms of accuracy, the AdaBoostClassifier scored the highest. This is not surprising since AdaBoost is an ensemble method, and ensembles often reduce prediction errors due to variance as compared to single estimators. Boosting methods also often reduce the bias of the ensemble.

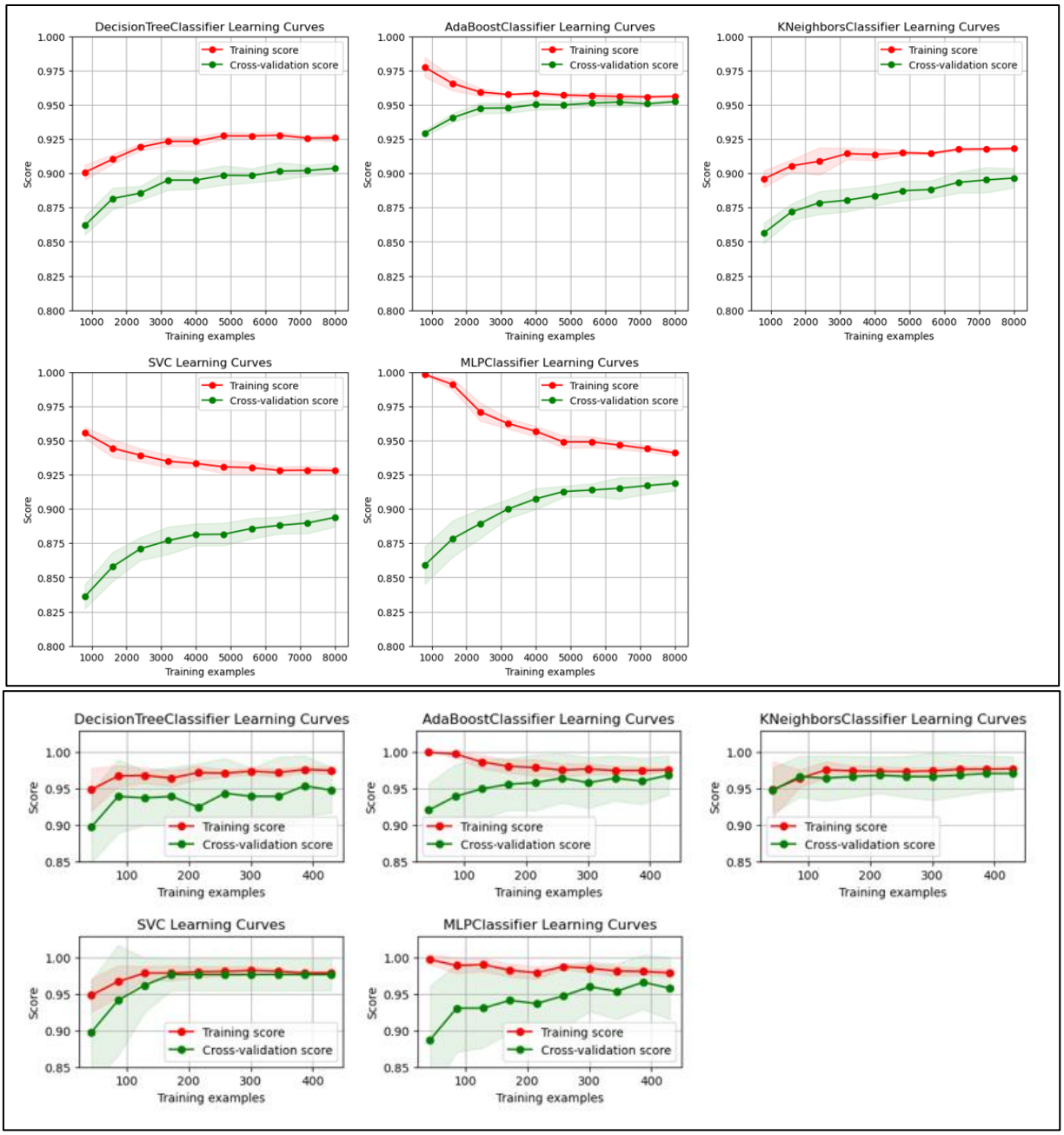


Figure 18

We see both effects in the high and tight learning curves.

For the Breast Cancer data, the best model is SVC, followed by KNN, AdaBoost, MLP and Decision Tree. Notice a very different ranking of algorithms as compared to the Diabetes dataset. Model performance for SVC and KNN seem drastically more stable than for more complicated models such as the neural network and AdaBoost, suggesting that for simpler datasets, complicated models can be overkill.

At this point, it is too early and erroneous to say AdaBoost is the best classifier for Diabetes, or SVC is the best for breast cancer. We also must compare the models' scalability. **Error! Reference source not found.** shows the scalability of all the models as measured by time (in milliseconds) for model training and predictions.

For Diabetes data, it's clear to see that MLPClassifier takes the longest to train by an order of magnitude, followed by a quadratic SVC curve, and linear curves for AdaBoost, DecisionTree, and KNeighbors. Because

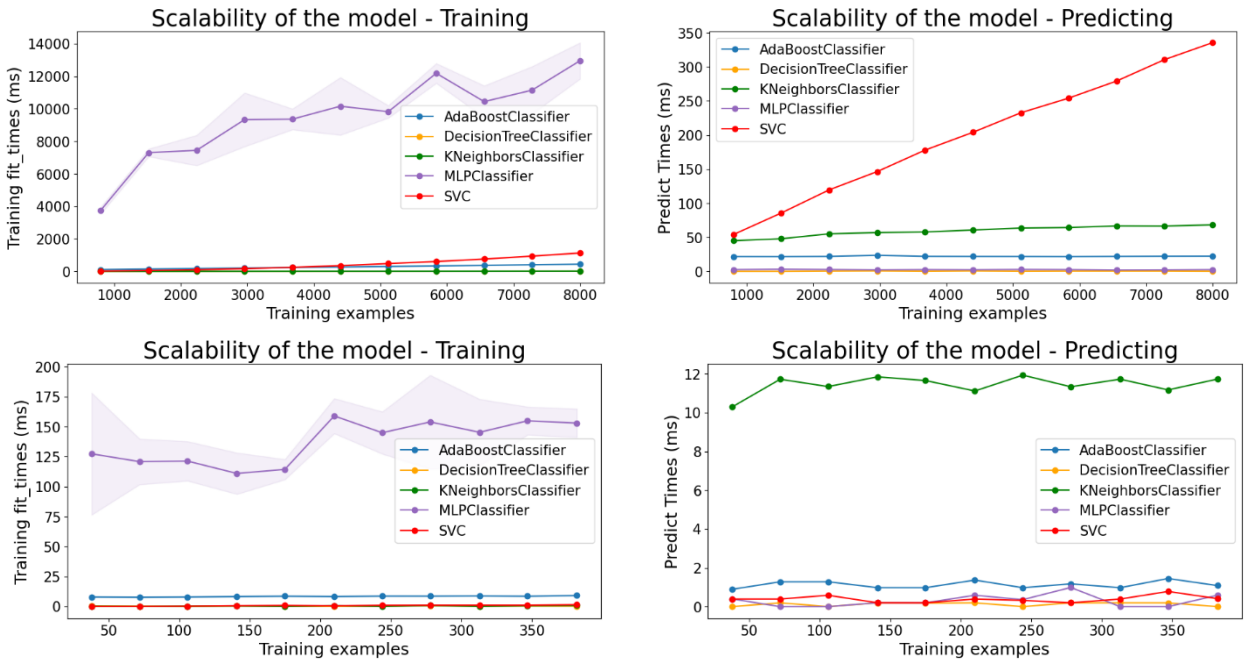


Figure 19

the neural network has to iteratively learn the weights for 130 neurons in the hidden layer, it makes sense it takes so long to train. The SVC model takes longer than other models because of the complexity of the RBF kernel. The quadratic nature of the learning curve can be explained by the fact that the complexity of projecting the data points into higher dimensional spaces increases with the size of the training set[7]. As an ensemble model, AdaBoost takes a significant amount of time to train multiple models rather than a single model, which explains the magnitudes higher training time as compared to a single DecisionTreeClassifier. Similar explanations can be offered for the Cancer data, but of noticeable importance is the shape of the SVC curve; For Breast Cancer, the SVC curve is not quadratic. Unlike the RBF kernel used for Diabetes, a linear kernel is used. The linear kernel is a parametric model and reduces the complexity of training as data sizes grow[7].

For Diabetes model predictions, SVC takes the most time, followed by KNeighbors, AdaBoost, MLP, and DecisionTree. Unlike a Linear SVM, a support vector machine with an RBF kernel is non-parametric meaning more training data needs to be stored as part of the model. Therefore, with increasing training examples, the model needs to reference more data before making predictions. The KNeighbors model, also non-parametric, requires more time for predictions because the model keeps and references all the training data for each query. On the other hand, the MLP and DecisionTree make very fast predictions. MLP is a parametric model. In general, non-parametric models require more space in memory to store data points, whereas parametric models are very space efficient.

Conclusion

This paper has provided an overview of 5 different supervised learning techniques: Decision Trees, Boosted Decision Trees, K-Nearest Neighbors, Support Vector Machines, and Neural Networks. Each model was tested on two datasets to explore the effects of hypertuning the parameters. It's difficult to determine which model was the best overall, since we need to take into account multiple factors such as propensity to overfit, training times, prediction times, complexity, explainability, etc. However, relevant tradeoffs are studied so that you, a future supervised learning practitioner, will have the knowledge to make those decisions!

[7] https://www.kdnuggets.com/2016/06/select-support-vector-machine-kernels.html