Supervised Learning Analysis

CS7641: Machine Learning

Kim, Irene
skim3364@gatech.edu

I. Objective

The objective of this report is to achieve empirical and engineering aspects of supervised learning in machine learning by implementing different algorithms. Two different data sets will be used to compare various behaviours of different machine learning algorithms. Towards this goal, this paper will practice in-depth exploration, implementation, and analyzation of five different supervised learning algorithms. The remainder of the paper is structured as follows: Section 2 provides a theoretical framework for the chosen datasets. Section 3 describes the experimental setup and justification of the decisions. Section 4 demonstrates the result of different algorithms with detailed analysis. Finally, the work is summarized in section 5.

II. Theoretical Framework

In this experiment, five different machine learning algorithms will be implemented. This section will provide the theoretical framework of each algorithm. This part will serve as a crucial theoretical basis for further analysis.

A. *Decision Tree*

The decision tree approximates discrete-valued functions and produces a single output value. The final output value is the *decision*. This method is robust to noise in both the data and attributes. It also effectively handles missing values as well. This algorithm performs a greedy search to find optimal decisions. Therefore, it is not able to backtrack its search path. To perform the best decision, the decision tree uses split actions as its search method. This tree method finds the most important attribute by calculating a statistical property called information gain [1]. The tree splits when it finds the node with the highest information gain. Information gain measures how correctly a given feature splits the training instances according to their target classification. The greater the information gain is, the greater decrease in uncertainty. The uncertainty is measured by entropy. Entropy is the expected value of the surprise or expected value of the inverse of the probability. The entropy is calculated as follows:

$$Entropy = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{1}$$

The formula of information gain is as follows:

$$InfoGain = Entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2}$$

The first term refers to the entropy value before splitting the tree. The second term is the expected value of entropy after S is separated using the attribute A. Instead of using entropy, another property called Gini impurity is frequently used to determine which partitioning is the best decision. The Gini impurity score measures how impure the nodes are. Then it chooses attributes with the lowest impurity value. The equation for the Gini impurity score is as follows:

$$Gini = 1 - \sum_{i=1}^{n} p_i^2 \tag{3}$$

B. *Boosting*

Boosting generates a number of weak learners to build a strong learner at the end. The weight values are utilized to train weak learners. Its mechanism is to give higher weight on the misclassified samples and lower weight on the instances that were correctly classified. In doing so, it aims to focus on the difficult samples that the model is not confident at. The boosting algorithm that will be used in this experiment in AdaBoost. The AdaBoost initially imposes equal weight and successively generates hypotheses by re-weighting the training examples. In short, it pays a special attention to previous examples with higher training error. This procedure makes boosting mechanism more robust to outliers than original decision tree.

C. *Neural Networks*

An artificial neural network is a powerful algorithm that can address noisy and complex real-world datasets. The neural network uses a backpropagation algorithm as the learning technique. The network learns the best weight vectors that minimize the error between the output and the target example using backpropagation. Backpropagation is the process of modifying the weight vectors using gradient descent. Gradient descent identifies the optimal value for slope and intercepts that gives minimum loss. It takes a step toward the point where the slope is minimum or close to 0. The step size is determined by multiplying the slope with a small learning rate. Towards the bottom of the curve, the stepsize gets smaller to find the spot where it reaches the minimum loss. The gradient descent stops when the size of the step is almost 0. The network can be stacked with multiple layers as long as units whose output is a nonlinear function of its inputs and whose output is a differentiable function of its inputs [8]. The layers between input and output layers are called hidden layers. The representations of these hidden layers offer very expressive hypothesis space for backpropagation even with a limited number of depths [8]. The intriguing property of a neural network is its ability to capture essential representations at the hidden layers that were not explicit in the input representation. Also, increasing number of layers can also enable the model to discover and learn more complex features.

*D. **k-Nearest Neighbor***

The k-nearest neighbor(KNN) is a type of a non-parametric mode, which cannot be characterized by a bounded set of parameters. KNN measures the distance from a query point to an example point. Depending on the features used in the dataset, different distance measure such as Euclidean distance and Manhattan distance can be used. The word "nearest" refers to a distance metric. As the name suggests, KNN algorithm groups examples that are close enough together [1]. Finding the best value of k or best number of groups is a hyperparameter. The cross-validation can be used to find this value. One thing that needs to be kept in mind when implementing KNN is that it is sensitive to the differing dimensions. For example, the width dimension from meters to inches while keeping the other dimensions the same will produce different result. Hence, applying the normalization is vital for each dimensions. Thus, normalization was conducted on all of the dataset for the experiment. KNN is robust to noisy training data as they can smooth out the impact of isolated noisy samples. However, KNN is susceptible to the curse of dimensionality. To be more specific, the distance between features that are relevant can be far from each other. In contrast, other irrelevant attributes might be close to one another [8]. Another practical drawback of this algorithm is that, it postpones the learning process until the query time. Due to its lazy learning nature, significant computation may be required to process each new query [8].

*E. **Support Vector Machines***

The support vector machine (SVM) is a method that constructs a maximum margin separator, which is a decision boundary that creates the largest possible distance to example points called support vectors [1]. This maximum margin separator is drawn in the midpoint of example points closest to the boundary. These points are the margin support vectors. The algorithm first maps the non-linearly separable data to some higher dimensional feature space. The algorithm then searches for the optimal separating hyperplane or maximum margin separator that separates these non-separable data. Often, projecting data that are originally not separable can be separated in a higher dimension. This process is referred to as a kernel trick. The kernel method enables constructing an optimal separating hyperplane in the hypothesis space without computing the calculations in a higher dimension [9]. Rather than reducing empirical loss on the training dataset, SVMs attempt to minimize expected generalization loss. Finding a maximum margin space to split data involves making the probabilistic assumption that similar data will fall in similar areas. This is because of the assumption that both seen and unseen example points are drawn from the same distribution. This structural risk minimization principle that SVMs follow incorporates capacity control to prevent overfitting problems [9]. Thus, SVM generalizes well and also can deal with high-dimensional space.

III. Data

The data used for this experiment are *Wisconsin Breast Cancer* dataset(Data1) and *Pima Indians Diabetes* dataset(Data2). The source of each dataset and details of their classes are provided in Table 1. There are three interesting aspects of these datasets that contributed to the selection. First, these two datasets are both dealing with a real-world problem. Specifically, machine learning based disease diagnosis enables early identification of numerous diseases and develops effective treatment [4]. Thus, it is worth conducting experiments with socially-meaningful datasets with practical implications. Another interesting structural aspect of these datasets is that we can explore how the performance of the models changes depending on the attributes. Data1 contains a total of 569 entries with 32 features. Data2 contains 768 entries with 9 features and they both are are imbalanced datasets. Data1 is comprised of approximately 62.8% of positive class and 37.2% of negative class. The Data2 is composed of approximately 65% of positive samples and 35% of negative samples. Both Data1 and Data2 were all continuous numerical value and Data1 contains 32 features whereas Data2 only has 9 attributes. Although having a similar properties, the performance of the algorithms on each dataset differ greatly. It would be worth investigating this phenomenon. The final reason that makes these datasets quite interesting is that it shows different behaviour of each model based on their structural nature.

TABLE I
DATA DETAILS

Data	**Instances**	**Positives**	**Negatives**
Data1 [2]	569	357	212
Data2 [3]	768	500	268

IV. Experimental Setup

A. Data Preprocessing

Wisconsin Breast Cancer Dataset

The dataset does not have any missing values that need to be handled. Among 32 attributes, the ID attribute has been dropped since it does not have any meaningful information but just a personal identification. The problem with this dataset is that the scale of 30 features excluding the label varies widely as can be seen in Appendix A (Fig 26). Containing features of various dimensions and scales may lead to poor modelling results. This is due to some algorithms making biased decisions on features with maximum variance. In addition, scaling the features accelerate the speed of convergence [5]. Nevertheless, it does not always guarantee better performance. To scale the features, standardization was performed. This scaling method normalizes the data by zero mean and unit variance. The standardization formula is as follows:

$$x' = x - \mu / \sigma \tag{4}$$

Pima Indians Diabetes Dataset Similar to Data1, the Pima Indians Diabetes dataset does not contain any missing values. It has 8 features, excluding the label category. The features of this dataset also have wide scale ranges between each feature. The scale of each feature can be found in Appendix A (Fig 27). For the same reason mentioned above, the same normalization method has been applied to this dataset. Although rescaling does not always lead to a better result, it would be helpful for model training, especially for some of the algorithms used in this experiment.

B. *Implementation Details*

This report compares five different machine learning models mentioned above with two different datasets: one is a binary classification task on Wisconsin breast cancer, and the other is a binary classification task on the Pima Indians diabetes dataset. All networks were implemented using the scikit-learn machine learning framework[6]. For reproducibility, the fixed number 123 was used for the seed value. Training and testing sets were split in an 80:20 ratio in a stratified fashion. The stratified method ensures that relative class frequencies are preserved, which is necessary as we are dealing with an imbalanced dataset. Having only 569 and 768 samples for each dataset, cross-validation was conducted to address the overfitting problem. When using cross-validation, it iteratively selects a different slice of the samples as the validation set and keeps the rest as the training set. The average error over all the folds on the validation set determines which hyperparameter value is the best. This method is especially effective when there is a limited number of examples. The majority of networks except for neural network and AdaBoost used 5-fold cross-validation. Due to limited computational resources, 4-fold cross-validation was executed on AdaBoost and neural networks. The grid search was used with the cross-validation for the hyperparameter tuning search to find the best hyperparameter value. The grid search tries all combinations of values and determines the best-performing value on the validation data. It is advantageous over the random search, which uniformly samples from all possible hyperparameter settings, in terms of its exhaustive searching nature. The grid search will be less likely to miss the best parameter choice but at the cost of more computational time. Since the size of the dataset is small, grid search would be fast enough. Approximately 20% of the dataset was preserved as testing set since the size of the dataset is small. Keeping too much dataset for testing could hurt model training quality. The evaluation metrics used to measure model performance are accuracy, f-1 score, and computing the area under the receiver operating characteristic curve(ROCAUC) score. The f-1 score is the harmonic mean of precision and recall. It can give a more fair evaluation of the model result, especially when using an imbalanced dataset. Finally, the ROCAUC score outputs values that represent how well the model correctly classifies true positives and false negatives. When the score is 1, it means the model classified all examples perfectly. It is a robust metric to assess the model performance with noise in the dataset, imbalanced dataset, and outliers.

V. Result and Analysis

All left-side plots are the properties of Data1 and right-side plots are the result of Data2 for all figures in this section.

A. *Dataset Comparison*

Throughout the entire experiment, Data1 continually recorded better results compared to the algorithms trained on Data2. The primary factor that resulted in lower model performance with Data2 is because it is more difficult problem to solve. There are multiple criteria that defines what the harder learning problem is. One possible reason that contributes in hard-problem is the nature of the problem itself. For example, cancer may have more straightforward causal-relationship compared to the diabetes in terms of medical aspect. However, finding the exact reason why diabetes is naturally more harder to solve cannot be scientifically justified in this analysis. Inspecting from machine learning point of view, it could be due to the representation of the dataset. If the Data1 is more easier and thus can be represented linearly, algorithms will easily classify different label examples. However, if the representation of the dataset is not linear, the models need to create non-linear decision boundary while keeping its generalization power. Solving non-linearly separable data is intrinsically more difficult problem for algorithms. Another reason may be the quality of the dataset. It is possible that Data1 may be more well processed with less noise, whereas Data2 might contain more noise and outliers.

TABLE II
Accuracy on Different Dataset

Data	**DT**	**AdaBoost**	**NN**	**KNN**	**SVM**
Data1	0.974	0.982	**0.983**	0.965	0.974
Data2	0.721	0.740	**0.786**	0.766	0.773

B. *Decision Tree*

The result of applying the decision tree before pruning on Data1 gave a quite promising result with an accuracy of 0.956 and an f1-score of 0.965. On the other hand, the accuracy and f1-score for Data2 are 0.695 and 0.577 respectively. The ROCAUC score for Data1 is 0.951 and 0.667 for Data2.

The result after pruning with the best hyperparameter values, the accuracy increased to 0.974 for Data 1 and 0.721 for Data2. The f1-score was 0.979 and 0.494 for Data2. The ROCAUC score for Data1 is 0.970 and 0.705 for Data2. Post-pruning(Figure 1), results in a shorter tree compared to a pre-pruning tree(Figure 2). The Gini index was used to split the tree, which is described in the theoretical framework section. Once it finds the best attribute with the Gini index, the tree keeps splitting until it classifies all the samples correct. However, this can lead to overfitting. Thus, pruning can mitigate this problem by pruning nodes until further pruning is harmful which can decrease the accuracy of the tree over the validation set. Therefore, the accuracy of trees on all datasets increased with slight a decrease in f1-score only on Data2. It may seem odd that the f1-score is reduced after hyperparameter tuning for Data2. As mentioned in the experimental setup section, f1-score is the harmonic mean of precision and recall that shows how well the model performs on both labels with an unequal number of instances. The recall and precision values prior to pruning were 0.561 and 0.593 respectively. The recall and precision post-pruning were 0.677 and 0.389 respectively. Although the accuracy improved on Data2, the precision was decreased whereas recall was increased resulting in a lower f1-score. However, the overall correctly classified samples were greater with the post-pruned tree, highlighting the fact that the pruning tree can improve the performance by eliminating nodes that are not clearly relevant until it hurts the accuracy. There are many other hyperparameters that can be tuned for the decision tree that can further improve the result, which is out of scope for this analysis due to time constraints and the main motivation is to compare the pruning result.

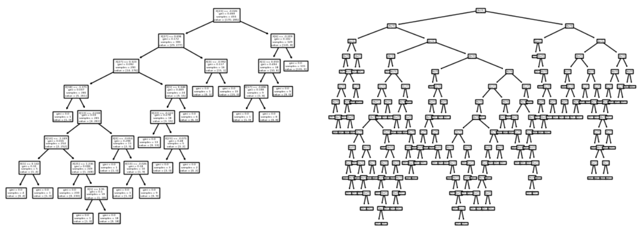

Fig. 1. Pre-Pruning Decision Tree

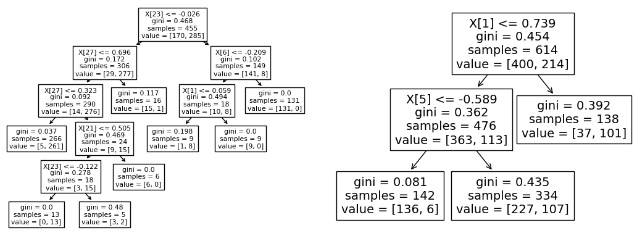

Fig. 2. Post-Pruning Decision Tree

The pruning was done by setting the maximum depth of the tree and a cost-complexity value. Setting the maximum depth of the tree halts the growth of the tree once it reaches the maximum depth. The cost-complexity parameter is the alpha value that controls the size of the tree. It finds the node with the weakest link that is defined with an effective alpha value. The nodes with the smallest effective alpha value are pruned first. Then it recursively prunes each node. Increasing the effective alpha value means more nodes will be pruned. This iterative process terminates once the pruned tree's minimal effective alpha value is greater than the cost-complexity alpha value. Finding the optimal maximum depth and cost-complexity alpha value was done via grid search with 5-fold cross-validation. Each definition and justification of its choices are explained in the experimental setup part. Figure 3 and figure 4 display the best value for a given space. The space searched for maximum depth was 1 to 30. As grid search exhaustively searches all the possible values with a defined range, I limited to 30 for a maximum depth for both datasets. In addition, the whole point of pruning is to reduce the size of the tree to get rid of unnecessary splits. Therefore, setting the maximum depth too large is could be a waste of computation resources. The best depth for Data1 was 5 and 3 for Data2(Figure 3). The best cost-complexity alpha value was explored between 0 and 0.05. Setting alpha value too high may undesirably reduce the size of a tree too much. The best value for Data1 was 0.00555556 and 0.02222222 for Data2(Figure 4). The learning curve for both models trained with Data1 and Data2 respectively are shown in figure 5. The learning curve illustrates performance on both training data and validation data as a function of training size, showing the generalization ability of the model. Both decision trees with Data1 and Data2 shows that as the size of the training set increase, the better the cross-validation accuracy score is. The pattern of the learning curve gap between the cross-validation score and training set score narrowing down its gap as the training set size increases demonstrates the generalization ability of the model. It is clear from figure 5 that the validation score could be increased with more training for Data1 as the trend of the graph is going upwards for the cross-validation accuracy score. Figure 6 shows the times required by the models to train with different sizes of the training set. Generally, more time is consumed with more training set size.

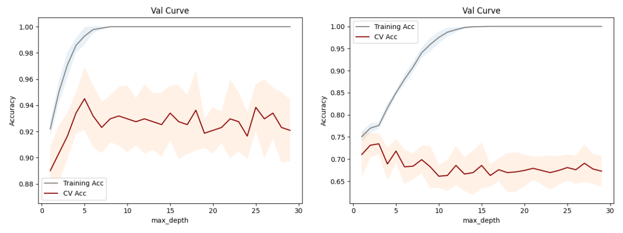

Fig. 3. Best Maximum Number of Depth

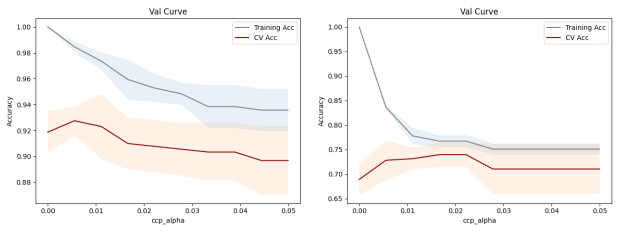

Fig. 4. Best Cost-Complexity Alpha Value

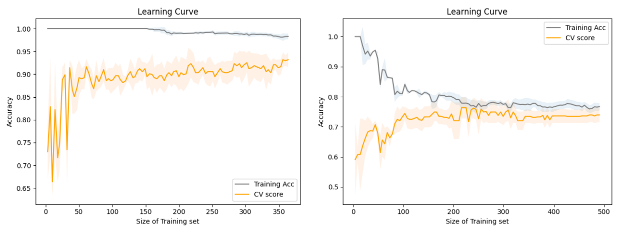

Fig. 5. Learning Curve of Decision Tree

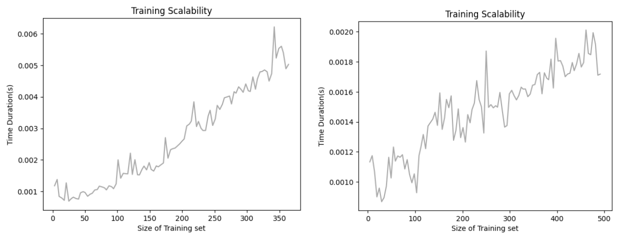

Fig. 6. Training Scalability of Decision Tree

C. *Boosting*

Among many boosting algorithms, adaptive boosting(AdaBoost) was selected as it can be easily implemented with the scientific framework scikit-learn [7]. As the name suggests, the AdaBoost model adaptively adjusts to the errors of the weak hypotheses. With a base estimator as a decision tree, there are two hyperparameters to optimize, which is the number of weak learners to fit the dataset. Although the default value for the number of weak learners is 50, the search space was from 1 to 1500. The reason for the large search space was to check whether increasing the number of an estimator would be redundant once it hits the golden spot. Figure 7 represents that increasing the number of base estimators doesn't improve the result after it finds the best value. The optimal value for the base estimator was 101 for Data1 and 401 for Data2 (Figure 7). Another hyperparameter experimented on is the learning rate. The learning rate assigns weights to determine the amount of contribution of each classifier. This parameter shrinks the effect of each classifier. To not overfit, the learning rate search space was limited to ranges 0 and 1. The best learning rate was 0.9 for Data1 and 0.05 for Data2 (Figure 8). The accuracy of the model on each dataset is 0.982 for Data1 and 0.740 for Data2. The f-1 score is 0.986 for Data1 and 0.5 for Data 2. The ROC ACU score is 0.981 for Data1 and 0.752 for Data2. The AdaBoost algorithm gives an improved model performance when compared with the pruned decision tree. This is because the AdaBoost pays special attention to the examples that were previously misclassified. Therefore, the model learns to classify more complicated examples. Both models on Data1 and Data2 seem to generalize well with less overfitting compared to the decision tree. A major drawback, however, is the training time for AdaBoost. As depicted in figure 10 the fit time for AdaBoost increases drastically as the size of the training set increases, taking a much longer time in contrast to the decision tree. The AdaBoost combines multiple weak learners to reach the final strong learners, which is a very time-intensive model itself. When the training time and computation resources are limited, boosting may not be the wisest option. When the training time and computation resource are limited, boosting may not be the wisest option.

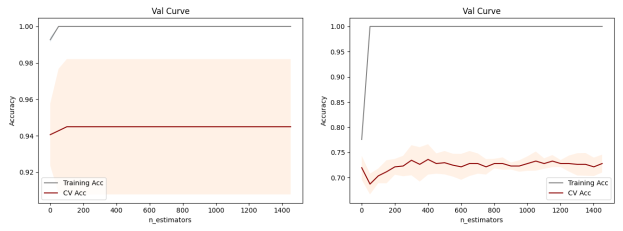

Fig. 7. Best Number of Estimator

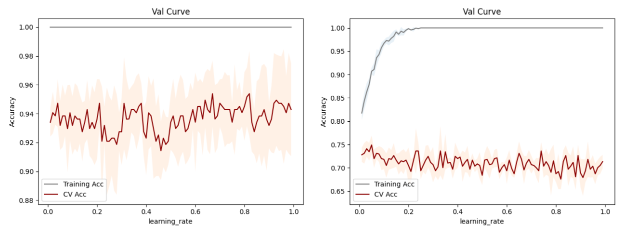

Fig. 8. Best Learning Rate

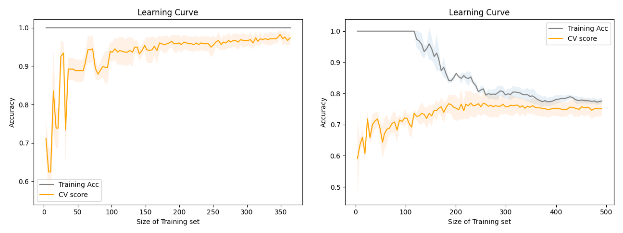

Fig. 9. Learning Curve of AdaBoost

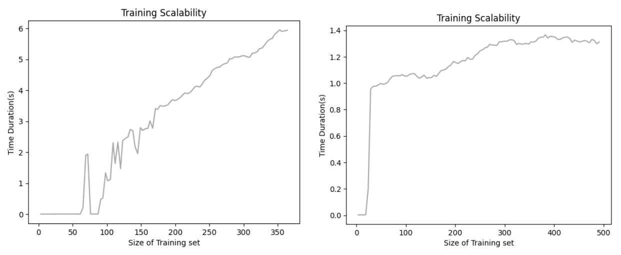

Fig. 10. Training Scalability of AdaBoost

D. Neural Networks

The result of the neural network on Data1 achieved the generalization accuracy of the network of 0.983, f1-score of 0.986, and ROCAUC score of 0.981. The result of the neural network on Data2 obtained an accuracy of 0.786, an f1-score of 0.673, and a ROCAUC score of 0.75. Compared to the above two algorithms, the neural network recorded the highest model performance. There are a couple of hyperparameter choices that were made when building neural networks for both datasets. First, the size of the hidden layer was 7 for both models on Data1 and Data2. Figure 11 presents the hyperparameter search space for hidden layer size. As described in the theoretical framework section, a small size of layers such as 2 or 3 layers can express rich representation hypotheses space for backpropagation. However, a deeper layered model can sometimes capture useful information for the given task. Therefore, I wanted to explore how the model performs as the number of layers increases. The layer size 7 was optimal for both datasets. At the same time, I limited the search space to 10 since unnecessarily deep layers for simple tasks can lead to underfitting and increase computation time. Second, the learning rate was explored. Learning rate moderates the degree to which weights are changed every step. It usually starts at a small value (e.g., 0.1 or 0.001). As written in the theoretical framework, this learning rate controls how big or small the step size should be during the backpropagation. If this value is too high, then the gradient might explode. If the learning rate is too small, it can take a long time to converge. Given the small size of the dataset, setting a small learning rate might be helpful to learn the representation of the data. Thus, the learning rate search space was limited to the interval $(10^{-5}, 10^{-1})$. The best value for the learning rate found by the grid search was 0.00035565 for Data1 and 0.00075431 for Data2 (Fig 12). Additionally, the regularization parameter alpha was searhed. In this network, it used the default L2 regularization. The L2 regularization minimizes the sum of squares and penalizes complex hypotheses. The alpha value is a hyperparameter value that serves as a conversion rate between loss and the complexity of the hypotheses. The purpose of introducing L2 regularization is to deal with the common overfitting problem in neural networks. Utilizing cross-validation can help find the sweet stopping spot for gradient descent search before the network starts to overfit the training dataset [8]. However, L2 regularization can also nicely balance the empirical loss of simple functions against more complex and the tendency to overfit in complicated functions [1]. As we have seven layers of units in the network, it may suffer from overfitting. Thus, I believed that the regularization technique would be helpful in our case. Alpha value search space was in the interval $(10^{-5}, 10^{2})$. I tried to give as wide a range as possible so that data would speak for itself. Figure 13 suggests that the alpha value increases until 5 and drastically decreases the accuracy of the model afterwards on Data 1 and around 18 for Data2. The best value found for alpha was 2.6826958 for Data1 and 0.19306977 for Data2. The training loss for both models keeps decreasing as the number of training iterations(epochs) increases as expected from the network. The training and validation score curve for Data1 depicted in Figure 15 showcases the good network performance throughout the entire iterations. The loss curve continues to decrease for the network trained with Data2 and the score curve also increases as the training precedes. However, compared to the model trained on Data1, the model trained and tested on Data2 shows higher loss and lower scores. This entails that Data2 was harder problem to solve compared to Data1. Interestingly, the score curve(Fig 15) for Data1(left-side plot) displays that the validation accuracy documents higher accuracy than the training accuracy. This seems counter-intuitive at first because as the model trains longer and longer, algorithms has tendency to overfit on the training dataset. However, the validation dataset can sometimes reach a higher accuracy result than the training dataset due to regularization effect. In our case, the L2 regularization was implemented on the neural network. The regularizing is only applied in the training phase but not in the validation step, often leading the validation results better than training results. This experiment could not include all the hyperparameters relevant to the neural network. There are many hyperparameters such as activation functions, early stopping, batch size, loss function, and many more that could be tuned to improve the model performance and reveal more interesting behaviours.

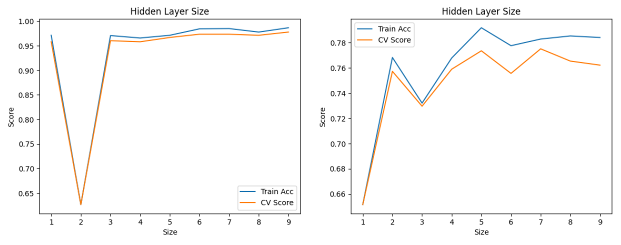

Fig. 11. Best Hidden Layer Size

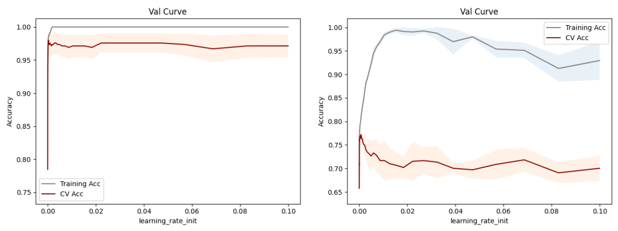

Fig. 12. Best Learning Rate

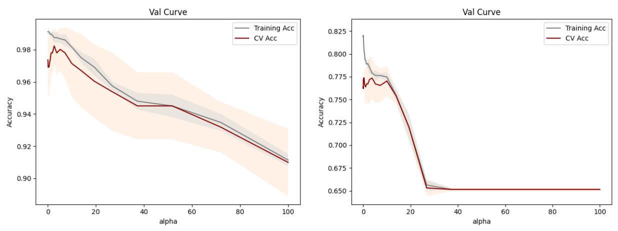

Fig. 13. Best Alpha for Regularization

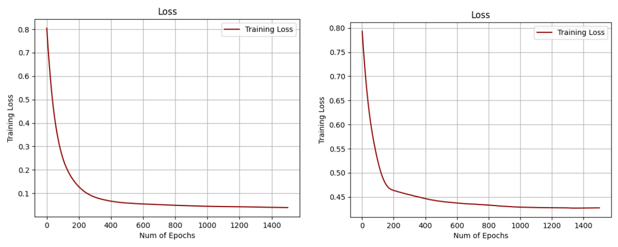

Fig. 14. Loss Curve of Neural Networks

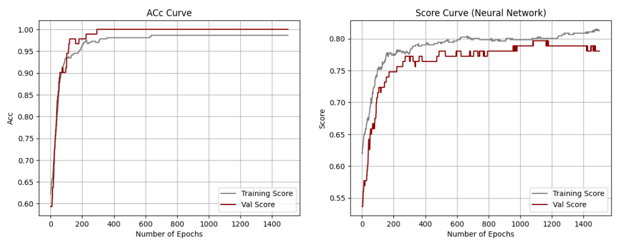

Fig. 15. Score Curve for Neural Networks

E. *k-Nearest Neighbor*

The accuracy for Data1 is approximately 0.965 and the f1-score is 0.975. The accuracy for Data2 is 0.766 and 0.633 for the f1-score. It is an interesting result that the performance of the model for Data1 is lower than simple algorithms like the decision tree. However, the model accuracy was higher on Data2. The KNN algorithm on Data2 was better than the result of the decision tree and AdaBoost, although the performance was worse on Data1. This phenomenon could be explained by the way KNN works. KNNs are effective at handling noisy training data as they smooth out the impact of isolated noisy samples. However, the practical issues of KNNs are that as it computes the distance between the query point and the training instances if the relevant features are far from each other and irrelevant attributes are close to one another the result could be quite misleading. The distance between neighbours can easily be dominated by a large number of irrelevant attributes. Therefore, the result is sensitive to the similarity metric used by the KNNs. This curse of dimensionality problem is also described in the theoretical framework section. In our case, the relevant attributes for Data1 might have been distant from each other, leading to poorer results compared to other algorithms. In contrast, the relevant features might be close to each other for Data2 and the smoothing noisy samples effect of KNN might have boosted the performance of the model. Two hyperparameters were explored for this algorithm. First, the number of neighbours determines where the query point belongs. If it's set too small, it can lead to overfitting. If it's too big, then the result may not be trustworthy. As the size of the dataset for both Data1 and Data2 is not large, the search space was set in the range between 1 to 50. The best number of neighbours received from cross-validation was 4 for Data1 and 15 for Data2(Fig 16). Secondly, the power parameter for the distance metric was searched. As covered throughout the analysis, the KNNs distance metric can have a large impact on the model performance. To find the most optimal similarity metric, various types of distance metrics were explored, including Manhattan distance, Euclidean distance, and Minkowski distance power of 2, 3, and 4. The final value returned was Manhattan distance for both Data1 and Data2(Fig 17). The learning curve plot (Fig 18) for KNNs for Data1(left-side) and Data2(right-side) portrays how its accuracy increases as the size of the training example increases. Again, this shows that the selected hyperparameters are generalizing well. The training time increases as the training set size increases (Fig 19)

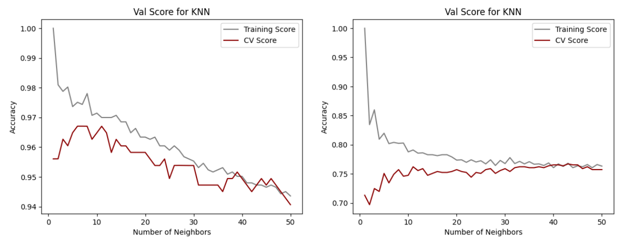

Fig. 16. Best Number of Neighbors

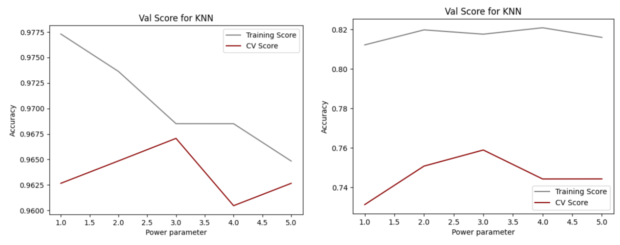

Fig. 17. Best Power Parameter

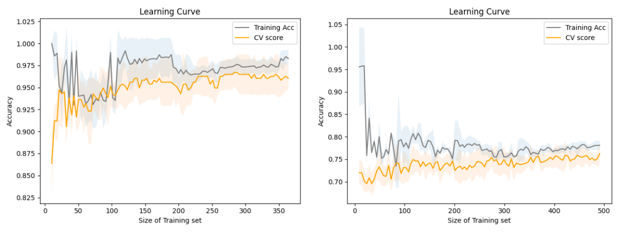
Fig. 18. Learning Curve of KNN

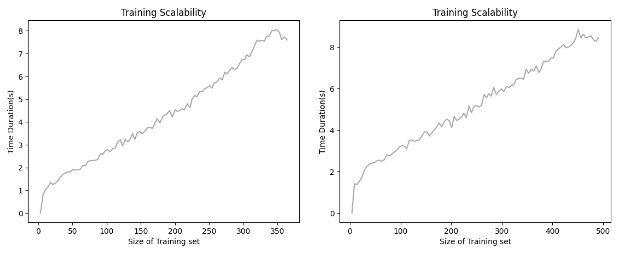
Fig. 19. Training Scalability of KNN

F. Support Vector Machines

The result with the best performing hyperparameter values for the SVM was 0.974 for Data1 and 0.773 for Data2 in terms of accuracy. The f1-score for Data1 was approximately 0.98 on Data1 and 0.60 on Data2. Compared to other algorithms, SVM was the third highest among four other algorithms. It has same accruracy with the decision tree for Data1. However, SVM trained with Data2 performed better than other algorithms except for the neural network. The reason why SVM would perform well on both datasets and especially on a harder dataset is due to their structural principle illustrated in the theoretical framework part. The SVMs form maximum margin separators that maximize the distance between support vectors, thereby, minimizing the risk of overfitting. At the same time, it projects the data into a higher dimensional space to separate non-linearly separable data to separate them linearly. Moreover, the kernel method used in SVMs is flexible that they can adopt to various kinds of kernel functions such as radial basis function and polynomial function to find the optimal hyperplane. Therefore, the SVM algorithms may be more flexible at creating decision boundaries compared to simple decision trees, AdaBoost, and KNN when data are non-linearly separable. By observing how other algorithms perform on Data1, we can assume that Data1 can easily be separated linearly. We can also assume that Data2 is more difficult to separate linearly as some algorithms perform worse than other algorithms that has the capacity to handle non-linear data. The ability to embed the data into a higher dimensional space and split non-linearly separable data would have resulted in a better result for SVM on Data2.

There are three hyperparameters explored in this experiment: regularization parameter, kernel, and the kernel coefficient for the chosen kernel type. First, the regularization parameter, often denoted as C, is an important variable that controls the trade-off between maximizing the margin and minimizing the training error term [9]. The SVM can overfit if this parameter is too large. It may place insufficient stress on fitting the training instances if it is too small. The value for C must be positive number. Thus, I observed how the accuracy changes as this value increases by setting them to be greater than 0(Fig 20). The best value for SVM with Data1 was 0.1 and 1.0 for Data2. However, increasing the value led to worse results, which is a sign of overfitting. Next, the kernel types tested were linear kernel, sigmoid kernel, and radial basis function kernel. The optimal kernel type was a linear function for Data1 and a radial basis function for Data2. The fact that the best kernel type was a linear function for Data1 delivers that they are linearly separable datasets. This entails that most algorithms can do quite well on this dataset since it is considered as easy problem. In contrast, the radial basis function was the best option for Data2. The result stresses that they are more difficult dataset due to their non-linear nature. This affected all models to perform worse compared to Data1. Finally, the coefficient for linear function for the algorithm with Data1 was 0.001 and roughly 0.0316 for Data2 (Fig 21). The coefficient for the kernel also controls the decision boundary created by the SVMs. This parameter referred to as gamma defines the magnitude of a single training example's influence. For instance, low value means the influence of a single training example is far. The high gamma value means the influence of a single training example is close. The lower value of gamma results in poorer accuracy for the model and the opposite for the higher gamma values. The regularization parameter and kernel coefficient combinations for models trained with Data1 and Data2 seem to generalize well on an unseen test set (Fig 22). The learning curve figure result presents the generalization power of the SVM. An additional information about the time duration as training example increase is presented in figure 23.

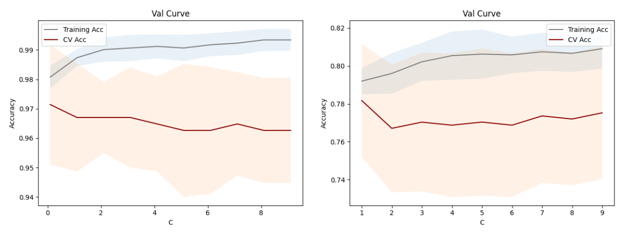
Fig. 20. Best Regularization Parameter

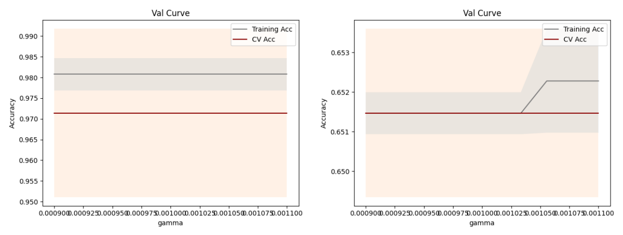
Fig. 21. Best Kernel Coefficient

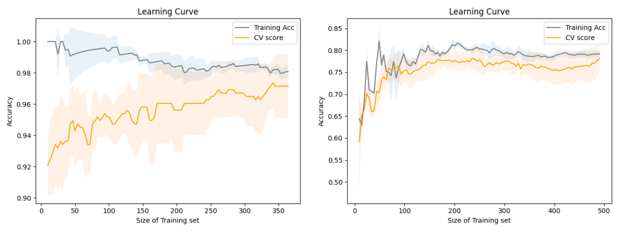

Fig. 22. Learning Curve of SVM

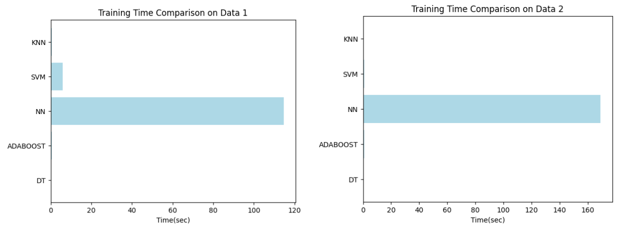

Fig. 24. Training Time Comparison

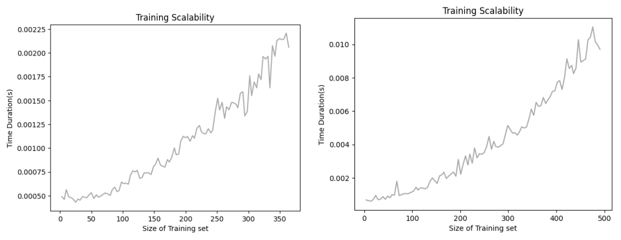

Fig. 23. Training Scalability of SVM

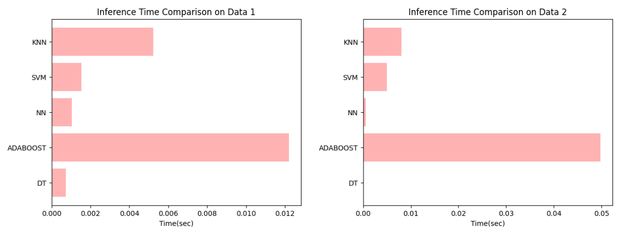

Fig. 25. Inference Time Comparison

G. *Wall Clock Time Comparison*

In this section, the wall clock time for training the model(Fig 24) and inference time(Fig 25) for each models are documented and compared. The training time or the fit time was incomparably higher for the neural network than any other models followed by SVM on both dataset. The fit time for the neural network is quite long as it performs forward propagation to compute the loss and then the back-propagation to correct the weights to find the optimal parameter values to fit data. Despite the amazing performance of the neural network, long training time is necessary. However, evaluating the learned network is typically very fast as depicted in figure 25. On the other hand, AdaBoost takes relatively long inference time although its fit time is very fast. As the number of weak learners increases, the inference time will also increase linearly. This is because AdaBoost will complete the prediction for each weak learner. Long inference time is also required for the KNN algorithm. The KNN algorithm is a lazy learner that postpones the learning process until it encounters new query instance. Such nature allows very short the training time but a long evaluation time. Although SVM took more time to train compared to other models on Data1, both fitting and inference time is fast. However, as SVM projects data into some higher dimensional space, having too much data and attributes can result in slow learning process. Finally, the decision tree takes the least time to train and inference for both datasets. Although simple, decision tree performed quite well on Data1. Fast training and inference time is a huge advantage in time-restricted setting. Nevertheless, the decision tree algorithm performed worst on Data2. This is because decision tree cannot build decision boundary that partitions not linearly separable dataset. Therefore, decision tree will perform well with fast speed as long as the dataset in linearly separable but poor on non-linearly separable dataset.

VI. Conclusion

In this experiment, I conducted analysis and comparison of 5 different machine learning models. Among these five models, neural network showed the highest performance. The neural network is particularly well-performing as it finds useful statistical representations of the training examples then continues to minimize the loss using backpropagation. Moreover, neural network is robust to errors in datasets, which might have also affected positively on classifying examples better compared to other models. However, the neural network took the longest training time among other models. These leads to conclusion that if the sufficient computational resource and time are provided, neural networks can outperform many traditional machine learning models. This also explains why deep neural networks are replacing traditional models in many areas including real-world practical applications as more powerful GPUs are provided along with massive amount of digital data. This report also compared each models on two different datasets. The results were quite interesting that all models performed well on Data1 but relatively poorer on Data2. Taking some models that can handle non-linear data representation performing better on Data2 describes that Data2 was more difficult problem set for models. In addition, different behaviors on different dataset were compared across all models. As carrying on the experiment, I realized that hyperparameter tuning process, although tedious, was quite important step. Depending on the hyperparameter, the performance of the model changed to 2 to even 9 percentage points. It suggests that picking the right hyperparmeter value can significantly boost the model performance. To summarize, this report provides theoretical framework for this experiment, explanation of the dataset, detailed experimental setup and its justification, results of all the models and hyperparameter selection steps, and wall clock time for each algorithms. More exhaustive hyperparameter tuning with larger dataset could bring more interesting and holistic result.

REFERENCES

[1] Russell, S., Norvig, P. (2020). Artificial Intelligence: A Modern Approach (Pearson Series in Artifical Intelligence) (4th ed.). Pearson.

[2] UCI Machine Learning Repository: Breast Cancer wisconsin (diagnostic) data set. (n.d.). Retrieved August 28, 2022, from https://archive.ics.uci.edu/ml/datasets/

[3] Pima Indians Diabetes Database. (2016, October 6). Kaggle. Retrieved August 28, 2022, from https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

[4] Ahsan, M. M., Luna, S. A., Siddique, Z. (2022). Machine-Learning-Based Disease Diagnosis: A Comprehensive Review. Healthcare, 10(3), 541. https://doi.org/10.3390/healthcare10030541

[5] Wan, X. (2019). Influence of feature scaling on convergence of gradient iterative algorithm. Journal of Physics: Conference Series, 1213(3), 032021. https://doi.org/10.1088/1742-6596/1213/3/032021

[6] scikit-learn: machine learning in Python — scikit-learn 1.1.2 documentation. (n.d.). Scikit-Learn.Org. Retrieved August 28, 2022, from https://scikit-learn.org/stable/

[7] sklearn.ensemble.AdaBoostClassifier. (n.d.). Scikit-Learn. Retrieved August 30, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[8] Machine Learning. (1997). McGraw-Hill Science/Engineering/Math. http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/mlbook.htm

[9] Burbidge, R., Buxton, B. (2001). An introduction to support vector machines for data mining.

VII. APPENDIX

A. Dataset Details

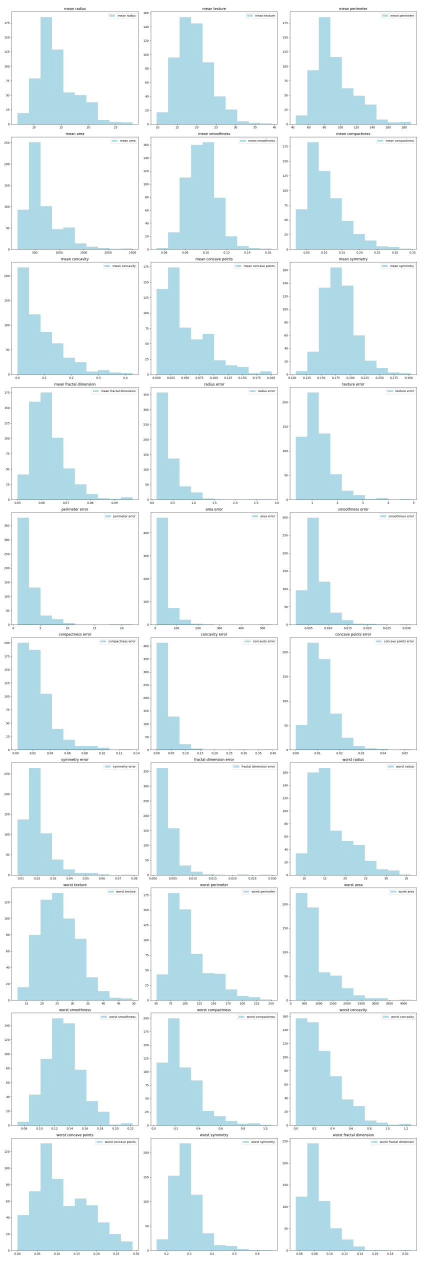

Fig. 26. Data1 Feature Scales

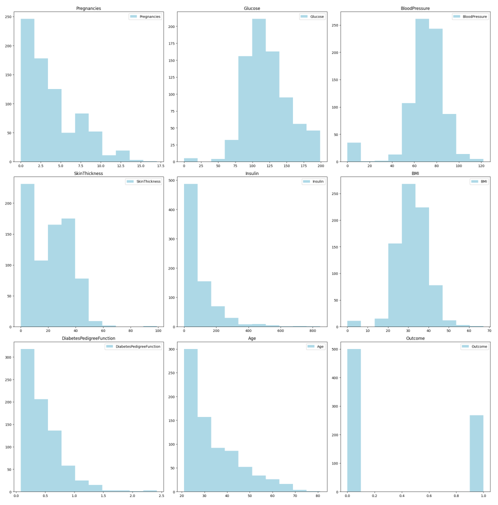

Fig. 27. Data1 Feature Scales