CS7641 A1

Supervised Learning

Wei En Chen

***Abstract* – This paper demonstrates different supervised machine learning methods and how parameters affect performance. In addition, it also shows how the characteristics of datasets can affect ML training and results.**

I. INTRODUCTION

In this assignment, five ML (machine learning) models are compared and contrasted in various ways with two different datasets. The five ML models are decision tree, boosting, support vector machine (SVM), k-nearest neighbours, and neural network. The various analyses include some prediction accuracy metrics, learning curves, wall clock time of model training, and hyperparameter sensitivities. The similarities and differences under these analyses are discussed in details. The coding language in this assignment is python with some popular packages such as Numpy [1], Pandas [2], and Openml [3].

II. DATASETS

The two datasets I picked for this assignment are bank loan credit [4] and EEG-eye state predictions [5]. First, the bank loan credit prediction has one thousand instances. Each instance takes 21 attributes (or features), some are continuous and some are categorical. The output labels are good/bad, indicating if the customers have good or bad credit. This dataset is imbalanced, with the good to bad ratio of 7:3 shown in Figure 1[6]. This dataset is oversampled to balance to 1:1 with SMOTE at neighbours = 6.

Figure 1 - Imbalanced dataset for "credit prediction".

Some of the categories in the attributes can be converted to numerical if they are ordinal in nature, such as employment: [unemployed, '<1', '1<=X<4', '4<=X<7', '>=7']. Any other attributes with yes/no or similar binary choices are converted to 1/0. For the rest of the attributes they are pre-processed to one hot vectors.

Second, the EEG-eye state prediction has 14,980 instances. Each instance takes 14 attributes (or features), and all of them are real numbers, measurements of electrical activities of the brains. The output labels are binary (eyes closed or open). This dataset is very close to balanced, with ratio of 11:9, shown in Figure 2[6]. I find these two datasets interesting because the first one has a lot of attributes while the number of instances is quite low.

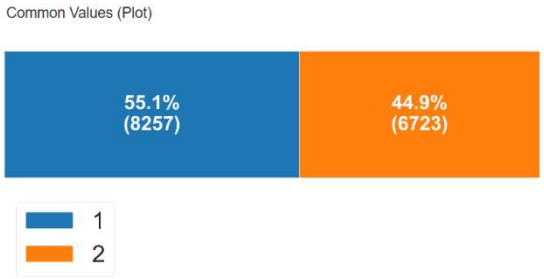


Figure 2 - Very close to balanced dataset for "eeg-eye state".

The most of the attributes in the first dataset are categorical. Some ML models may be able to shine (or not) with these characteristics. For the second dataset, all attributes are real numbers, and it is more or less balanced, compared to the first dataset. There are also a lot more instances than the first dataset. It is possible some ML models such as neural nets that would do better because of the real number attributes and large amount of instances while decision tree based models prefers categorical attributes. With these contrasts, they should provide different results among the five ML models for in-depth discussions.

Before the datasets are fed into the models for learning, they are all preprocessed by applying one-hot vectorization (if applicable), split into training and test sets with ratio of 4:1, and then apply standardization to the datasets with sklearn [7]. What it does is to rescale the input attributes such the mean is zero and standard deviation of one. Some ML models don't benefit from standardization such as decision trees, but other models like neural nets and SVM would benefit greatly with attributes values that are small across all attributes. When a dataset is standardized, it is based on the scale of the training set only. This is to prevent "peeking" at the test set, which is supposed to be evaluated only at the very end, after tuning is done.

III. GENERAL ANALYSIS PARAMETERS

When sensitivity analyses are done on each of the hyperparameters of each ML model, cross validation of 5 folds and 10 folds are utilized for "credit prediction" and "eeg-eye state" datasets, respectively. Due to the nature of smaller "credit prediction" dataset, fewer folds is used to avoid higher sampling error in the validation sets. The same k-fold strategy is utilized for final tuning and plotting learning curves and wall clock time curves.

Since both datasets are balanced, a simple accuracy score can be utilized instead of f1 [9] or some other metrics to consider false positive and false negative.

Untuned and tuned learning curves and wall clock times are generated and analyzed in the analysis section.

IV. DECISION TREE AND ANALYSIS

SPLIT CRITERION

A decision tree model learns by choosing one attribute at a time to split its decision. The decision to choose a attribute can be random, or by information gain. With information gain, there are two popular types: gini impurity and entropy. The gini impurity of each attribute calculates for each attribute value (categorical) with the gini impurity formula (Eq 2) [8], and sum them with weights (occurrences). The gini impurity value ranges from 0 to 0.5. Lower means the split is better.

$$Gini = 1 - \sum_i p_i^2, \qquad (p_i: probability\ of\ class\ i) \tag{2}$$

The entropy is a measure of randomness in the data after split with an attribute and calculated with the formula below (Eq 3) [8].

$$Entropy = -\sum_i p_i \times \log_2 p_i, (p_i: probability\ of\ class\ i) \tag{3}$$

When all data is split properly by an attribute, that is, all output labels are grouped on one side, the "randomness or entropy" is zero, while the opposite will result in entropy of one.

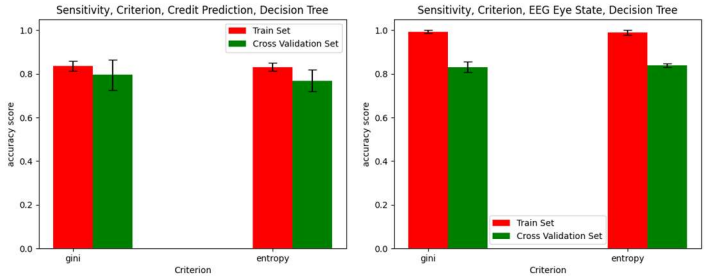


Figure 3 - Results of splitting criterion on two datasets. Left: "credit prediction", right: "eeg-eye state".

The plots in Figure 3 show the accuracy scores difference between the two splitting criteria and datasets. It can be observed that there is no noticeable difference in cross validation score performance between gini impurity and entropy. Other hyperparameters are varied to see if the two criteria made any differences in performance scores. The results are all quite similar. Since there is no difference in scores, it is better to choose gini impurity instead of entropy because entropy involves log calculation which is more computational expensive.

TREE MAX DEPTH (PRE-PRUNING)

In a decision tree model, it can potentially split data with many depths until each leaf node is left with one instance. This can overfit the training data so it will be interesting to see what maximum tree depth do to the scores in the two datasets. Figure 4 show some very interesting behaviour. For both datasets, there is some level of interaction between two hyperparameters, tree max depth and minimum samples leaf (min number of samples required to be at leaf node). From contrasting figures within a dataset, one can see the change in min samples leaf does affect max tree depthaccuracy scores (bias & variance). The variance increases as max depth increases, and also when min samples leaf decreases. This makes sense as both hyperparameters are considered pre-pruning tools for decision tree models. Lower tree depth and higher samples leaf will under fit the data while greater tree depth and fewer samples leaf will overfit the data.

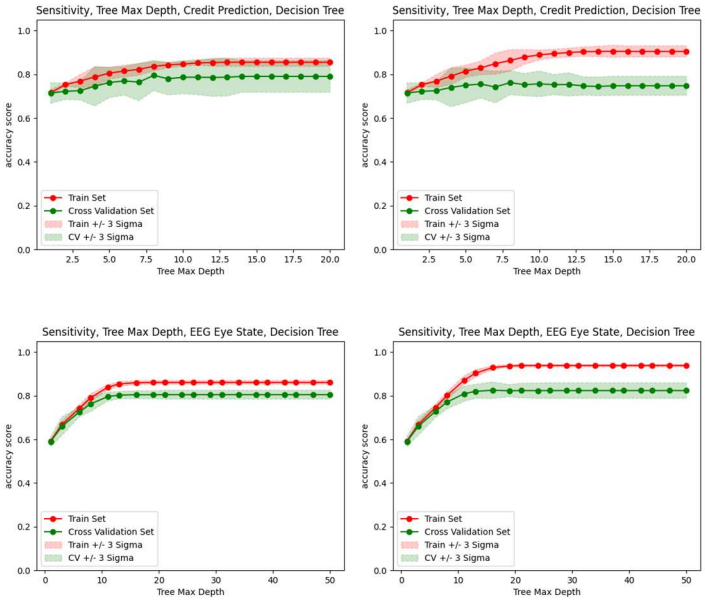


Figure 4 – Sensitivity of tree max depth on bias & variance. Top left: min_samples_leaf=12, top right: min_samples_leaf=5, bottom left: min_samples_leaf=20, last one: min_samples_leaf=5.

For both datasets, as the number of tree depth increases, the cross validation curves do increase inaccuracy scores up to some depths. The tree depth hyperparameter range will likely need to be wide enough during final tuning of the model.

MIN SAMPLES LEAF (PRE-PRUNING)

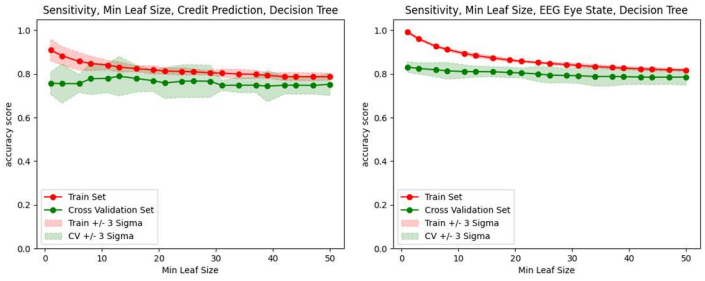


Figure 5 - Sensitivity of min leaf size on bias & variance. Left: "credit prediction", right: "eeg-eye state".

As mentioned before, this is part of the pre-pruning tool and it interacts with the max tree depth hyperparameter. This hyperparameter sets a minimum samples required in a leaf node. In Figure 5, both show the variance starts high because small leaf size node indicates overfitting. As the leaf size increases, variance decreases as seen with closing gaps between training and cross validation set curves. The difference between 2 datasets is, for "credit prediction", the cross validation curve peaks around leaf size=15 while for "eeg-eye state", the validation curve peaks around leaf size=1. Different leaf size ranges will be set accordingly to conduct the final tuning of hyperparameters.

For reasons why "eeg-eye state" dataset needs small leaf size is because splitting branches while all attributes are real numbers is not considered effective for decision trees. As mentioned earlier, decision trees are more suitable for attributes with categorical values.

OPTIMIZED LEARNING CURVES & WALL CLOCK TIME

The tuned hyperparameters:

Table 1 - Optimized hyperparameters on "credit prediction".

HyperParam	criterion	max depth	min leaf size
Value	gini	8	12

Table 2 - Optimized hyperparameters on "eeg-eye state".

HyperParam	criterion	max depth	min leaf size
Value	entropy	19	1

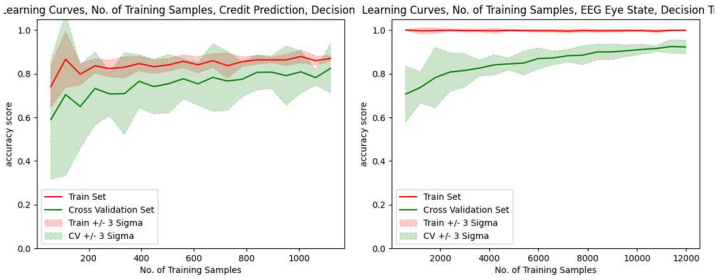


Figure 6 - Learning curves for two datasetsbias. Left: "credit prediction", right: "eeg-eye state".

The general trend of decision tree learn curves is as number of instances increases, the cross validation score should become closer to the training score because at fewer training samples, it essentially means the variance is large. The +/-3 sigmas in "credit prediction" dataset is also larger likely due to lower number of samples and folds, shown in Figure 6.

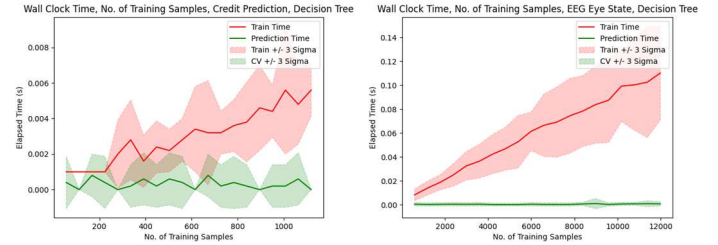


Figure 7 - Wall clock time of decision tree fitting/prediction on: left, credit prediction; right, eeg-eye state. For prediction curves, no. of samples are x0.2 for "credit prediction" and x0.1 for "eeg-eye state".

Both wall clock time plots show a linear relationship to number of samples in Figure 7. The "credit prediction" displays more uncertainty likely due to the small number of samples. During prediction, the time is constant because the big O is O(depth) which is optimized.

V. ADABOOST

Boosting works by using a weak classifier such as shallow decision tree, and in each training estimator, it puts more weights on misclassified samples in the previous estimator. It keeps iterating until a specified number of estimators is reached or all samples are perfectly classified. The first step is establish a weak learner, which has accuracy slightly better than guessing. Both datasets are balanced at this point so if the training accuracy is just slightly higher than ~0.5, it is considered as a weak learner. Table below summarizes the hyperparameters to establish weak learners in both datasets.

Table 3 - Weak learners hyperparameters

	max depth	min leaf size
Credit prediction	≥3	≤100
Eeg-eye state	≥2	≤1000

NO OF ESTIMATORS

As described above, number of estimators will repeat learning the same data while weights are more emphasized on misclassified samples from previous estimator. The distribution of weights is calculated [10]:

$$D_{t+1}(i) = \frac{D_t(i) \times e^{-\alpha_t \times y_i \times h_t(x_i)}}{Z_t} \quad (4)$$

$D_t(i)$: previous estimator at sample i

$h_t(x_i)$: returns predicted label of sample i

y_i: ground truth label of sample i

α_t: always positive

When predicted and ground truth labels agree, the product is always positive while if they don't agree, the product is always negative. With the negative sign in front of the α_t, the signs are flipped. The correct classification sample will have a lower weight because e to a power of negative value is small while misclassified sample will have e to a power of positive value, hence larger weight.

With this in mind, having more estimators means it will increase the accuracy of the model prediction. It should not overfit as long as the learners are weak. Figure 8 shows the accuracy scores with increasing number of estimators.

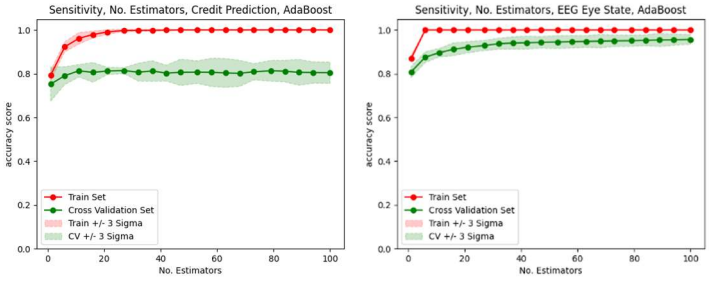


Figure 8 - Sensitivity of No. of estimators on bias & variance. Left: "credit prediction", right: "eeg-eye state".

The left plot shows increasing variance, which is due to stronger learner we picked. The learners shown in Figure 8 are optimized so they are stronger than the weakest learners shown in Table 3, but weaker than optimized learners in decision tree

section. To further understand the effect of weak learners on bias & variance, additional plots are shown in Figure 9. The top two figures show if the learners take the hyperparameters from Table 3, the train and validation scores are very close to each other meaning the variance (overfit) is very low. The bias (underfit) on the other hand are high since the accuracies are just slightly better than random guess. The accuracy scores do improve slowly with increasing No. of estimators and there is no sign of overfitting at 100 estimators. The middle two figures show that if the learners perform worse than random guess, boosting stops learning with any number of estimators. The bottom two figures show that if the learners take the hyperparameters from optimized decision tree tuning in the previous section, the learners essentially become strong learners and the variances start to widen as the number of estimators increases.

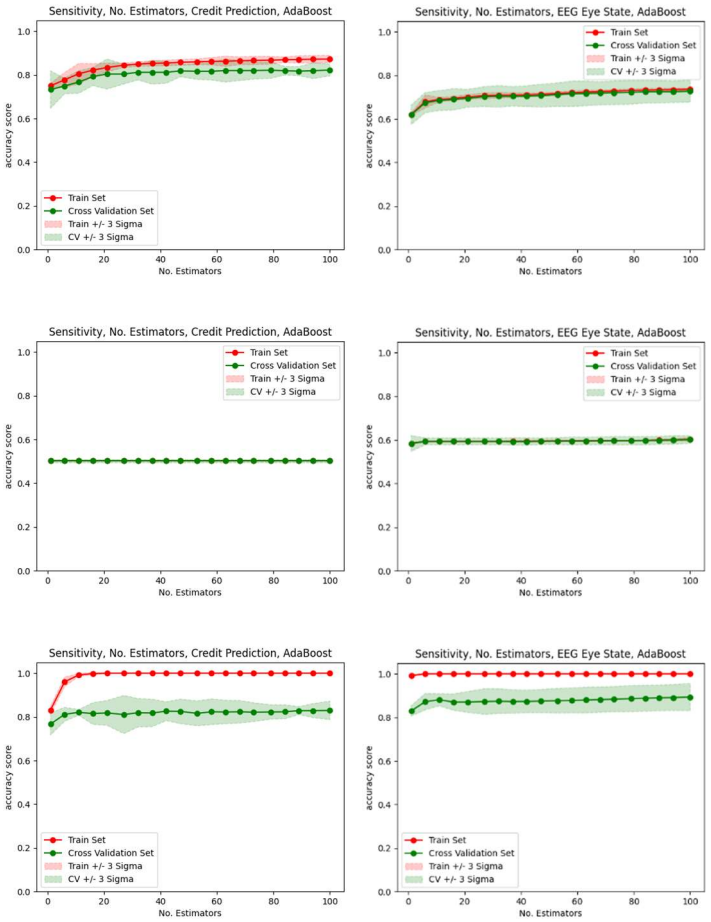


Figure 9 - Effect of learner strengths on bias & variance. Top row: weakest learners with hyperparameters from Table 3. Middle row: weaker than the weakest learners (left: max depth=1, min leaf size=500; right: max depth=1, min leaf size=5000). Bottom row: strong learners with hyperparameters from Table 1 & Table 2.

MAX DEPTH & MIN SAMPLES LEAF (PRE-PRUNING)

Since the base estimator of Adaboost is a decision tree, the sensitivity of tree max depth and min leaf in Adaboost will be similar to that of the decision tree.

LEARNING RATE

The learning rate in Adaboost is the α_t term in Equation 4. If learning rate is higher, it adds more weights to samples, exponentially more on misclassified samples. Increasing the rate can increase the effectiveness of boosting, likely reduce the number of estimators. However, if the learning rate is too high, the learners will likely completely ignore the low weight samples and become unable to learn anything. Figure 10 shows that as the learning rate increases up to around 1, the accuracy scores are generally increasing. As the learning rate goes over 1, there is complete meltdown on accuracy scores.

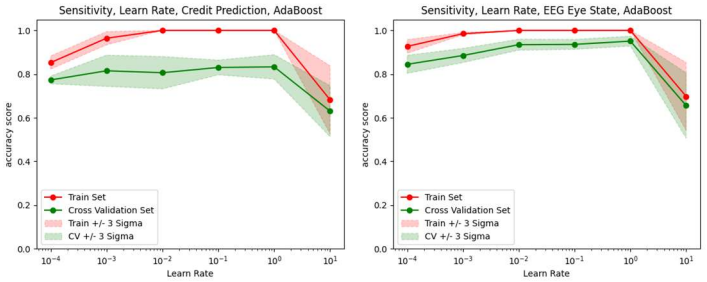


Figure 10 - Sensitivity of learning rate on bias & variance. Left: "credit prediction", right: "eeg-eye state".

OPTIMIZED LEARNING CURVES & WALL CLOCK TIME

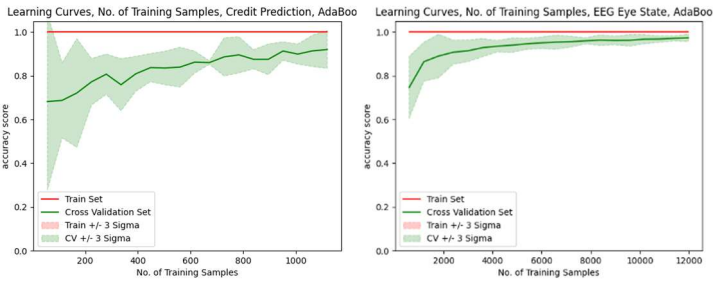
The tuned hyperparameters:

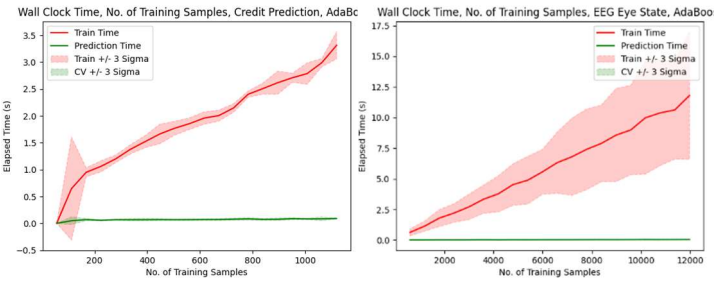
Table 4 - Optimized hyperparameters on "credit prediction".

H.Param	criterion	max depth	min leaf	lr	estimators
Value	entropy	5	1	0.0775	500

Table 5 - Optimized hyperparameters on "eeg-eye state".

H.Param	criterion	max depth	min leaf	lr	estimators
Value	entropy	13	13	0.5	100





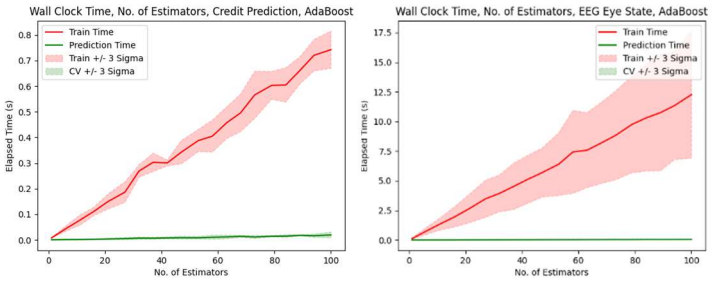


Figure 11 – Learning curves and wall clock time for two datasets. Left: "credit prediction", right: "eeg-eye state". Top row: learning curves on No. of training samples. Middle row: wall clock time on No. of training samples. Bottom row: wall clock time on No. of estimators. For prediction curves, no. of samples are x0.2 for "credit prediction" and x0.1 for "eeg-eye state".

The optimized hyperparameters are listed in Table 4 and Table 5. It is not surprising that the shapes of the learning curves in Figure 11 are not far off from the decision tree learning curves in Figure 6. Both training and validation curves are higher in adaboost due to the number of estimators used. For wall clock time, if the number of estimators are 100, the time it should take to train is likely 100 times longer, assuming max depth are not too different. This is exactly what is observed. Adaboost with 100 estimators takes about 100 times longer to train compared to a single optimized decision tree. The train time is still in linear relationship to the number of training samples. Two plots are added to show the wall clock time on number of estimators. Since the number of estimators are simply computing the same decision tree in series, the relationship is linear, as expected, for both training and validation curves. From the learning curves and wall clock times, 100 estimators is a good balance between speed and accuracy performance.

VI. SVM

SVM (support vector machine) attempts to draw a boundary on hyper-dimensional space (assuming more than 3 attributes, each attribute is one dimension) to separate data with maximum margin while classifying samples correctly. Concretely, it is a quadratic programming equation to be maximized [10]:

$$W(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j \quad (5)$$

Most α_i can be zeros meaning the associated samples don't contribute to the construction of the boundary (vector). The last two terms $x_i^T x_j$ can be rewritten as $k(x_i, x_j)$, representing a kernel function which controls the shape of the boundary (vector). This is also called the kernel trick. The subscripts i and j are each sample in the dataset [10].

KERNELS

There are several types of kernels and each with its own pros and cons. Formulae are given below[11]:

1. Polynomial: $k(x_i, x_j) = (x_i \cdot x_j + c)^d$ (6)
   - If d=1, it is identical to linear kernel.
2. Sigmoid: $k(x_i, x_j) = \tanh(\gamma(x_i \cdot x_j) + \theta)$ (7)
3. RBF (radial basis function):

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \quad (8)$$

1st order polynomial, or linear, as the name suggests, is a linear boundary. The computation will be fast since it is easy to compute. On the other hand, if the data is highly nonlinear in nature, this kernel may not perform well. 2nd order or above polynomial kernels can fit nonlinear data better, at the cost of computation time. Not only more calculation between a pair of data points, but also for each pair of data points in the dataset. For the most of the time, only non-zero α_i points are required in the calculations, but the complexity is still $O(n_{attributes} {n_{used\ points}}^2)$.

For sigmoid kernel, it behaves similarly to the sigmoid activation in neural networks. For RBF kernel, it is commonly used when there is no domain knowledge on the dataset prior to any in-depth analysis[11]. Figure 12 below demonstrates the effect of different kernels on accuracy scores. On "credit prediction" dataset, rbf is the best among all kernels. There is no huge advantage between linear/poly 1 and other nonlinear kernels. This can be contributed by the fact most of the attributes are categorical so linear/poly 1 kernels performs about the same as higher order kernels. There is a tiny difference in scores between linear and poly 1, likely due to the coefficient gamma applied to poly, but not linear, under sklearn package, which will be discussed in the next hyperparameter.

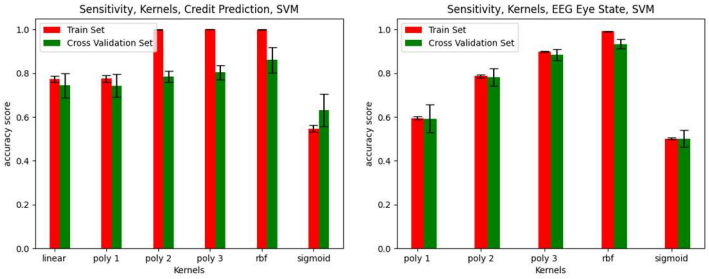


Figure 12 - Sensitivity of kernels on bias & variance. Left: "credit prediction", right: "eeg-eye state".

On the "eeg-eye state" dataset, we can clearly see non-linear kernels fair better, except for sigmoid. All attributes are numerical, unlike the other dataset, which shows it behaves very different. Sigmoid performs worse than linear/poly 1 kernel. Upon inspecting Eq. 6 to 8, sigmoid is the only kernel that limits return value of [-1, 1], if θ is set to zero. This characteristic likely limits how flexible the model can be compared to others.

GAMMA

Gamma, γ, is a scaling factor on the kernel. It is applied to polynomial, sigmoid, and RBF kernels. Gamma is not included in Eq. 6, but it's likely γ is multiplied to the term $x_i \cdot x_j$. Figure 13 shows how γ affects accuracy scores in both datasets. According to [12], low γ indicates smoother boundaries while high γ will introduce boundaries with ridges, which in other words, overfitting. We observe the same behaviour here. For both datasets, as γ increases over a certain value (which can be different, depending on datasets), train and validation curves start to diverge. The "eeg-

eye state" dataset peaked in accuracy at γ around 10 while "credit prediction" peaked in accuracy at γ around 0.1.

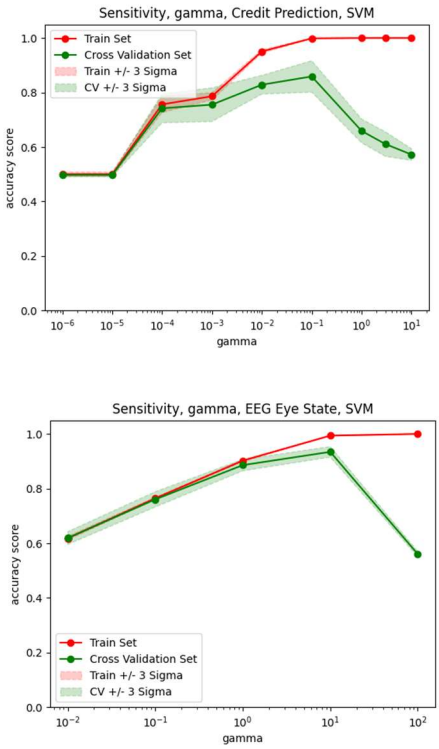


Figure 13 - Sensitivity of gamma on bias & variance. Top: "credit prediction", bottom: "eeg-eye state". "Credit prediction": poly 1 kernel. "Eeg-eye state": RBF kernel.

C, REGULARIZATION

Another important hyperparameter is the regularization which acts as overfitting control. It reduces variance while not impacting too much on bias. As the sklearn defines it [13], C is a L2 penalty, and the regularization strength is inversely proportional to the value of C. Unlike L2 penalty in neural networks, the strength is proportional to the value of the penalty.

Figure 14 plots the accuracy scores in response to different C values. When C is higher, regularization strength is lower, and overfitting can occur. We do see overfitting in both datasets when C's are higher. It is more pronounced in "credit prediction" dataset could be due to the limited instances of data. There is subtle overfit in the "eeg-eye state" dataset when C is higher 1, the train curve approaches 1.0 accuracy score while validation curve drops very slightly. In other words, regularization indeed reduces variances in both datasets and the bias is negatively impacted in both datasets when they are heavily regularized. SVM starts to shift to a "constant" model where it performs as good as random guess. For both datasets, the tuning range of C should be under 10 to shorten the computation time on searching optimal values.

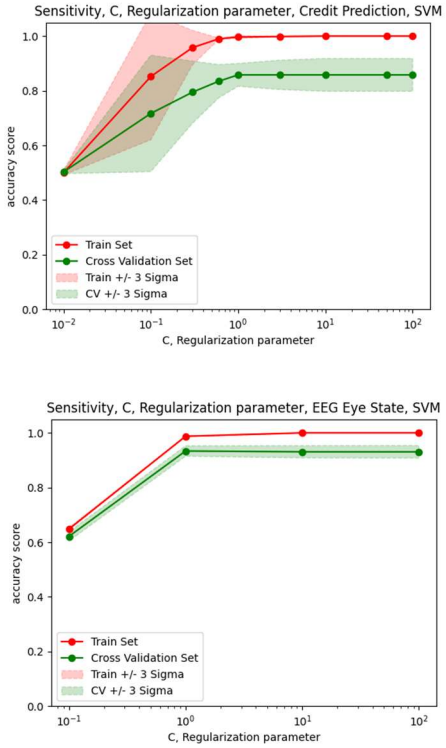


Figure 14 - Sensitivity of C, regularization, on bias & variance. Top: "credit prediction", bottom: "eeg-eye state". "Credit prediction": poly 1 kernel. "Eeg-eye state": RBF kernel.

OPTIMIZED LEARNING CURVES & WALL CLOCK TIME

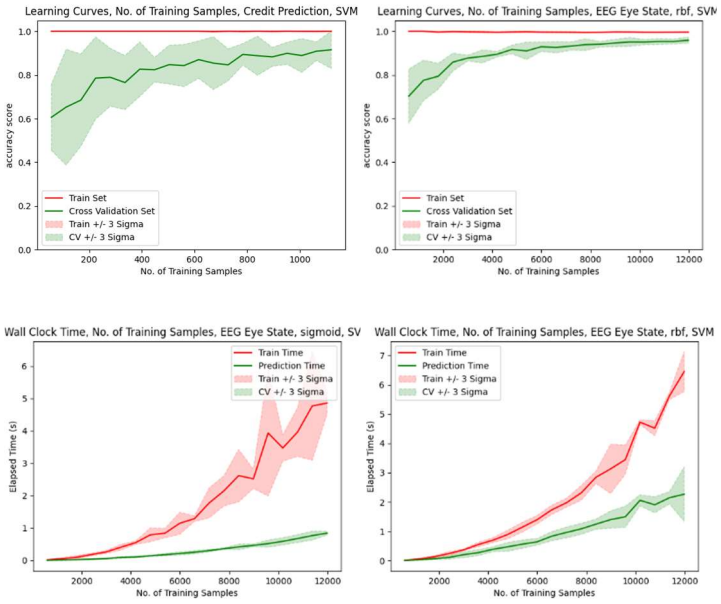
The tuned hyperparameters:

Table 6 - Optimized hyperparameters on "credit prediction".

H.Param	C	degree	gamma	kernel
Value	5	N/A	0.08	rbf

Table 7 - Optimized hyperparameters on "eeg-eye state".

H.Param	C	degree	gamma	kernel
Value	1.3	N/A	8.7	rbf



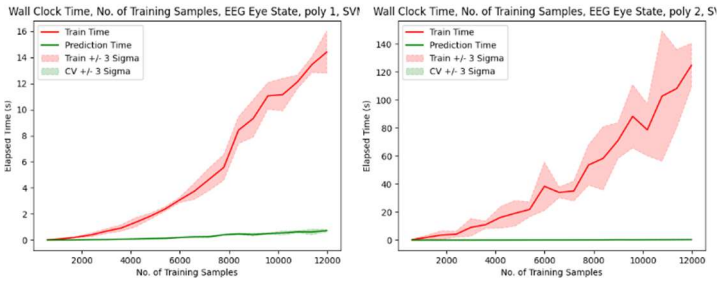


Figure 15 - Learning curves and wall clock time for two datasets. Top Left: "credit prediction", top right and rest: "eeg-eye state". Middle left: sigmoid kernel. Middle right: RBF kernel. Bottom left: 1st order polynomial. Bottom right: 2nd order polynomial. For prediction curves, no. of samples are x0.2 for "credit prediction" and x0.1 for "eeg-eye state".

From the optimized hyperparameter learning curves, Figure 15, we can observe that both curves behave similarly that when number of samples is low, it is very easy for the SVM to overfit since the hyperparameters are tuned for full sample size. As sample size increases, the validation curves move close to the training curves, and the variances drop. The training curve remain at 100% accuracy at all training samples for "credit prediction" dataset due to relatively small sample size while the training curve slightly drops in "eeg-eye state" dataset.

For wall clock times, we can observe that it takes relatively the same time to train a SVM with sigmoid and RBF kernels. It takes somewhat more time (in this case 2x for 12k samples) to train a SVM with 1st order polynomial kernel. It take about 10 times longer to train a SVM with 2nd order polynomial kernel. Generally the complexity is $O(n_{attributes}{n_{used\ points}}^2)$, as mentioned earlier. However, calculating higher order polynomial will require more computation time. If we move to 3rd order or higher, the cost of computation will dramatically outweigh the performance gain, if there is any. The prediction wall clock time among different kernels are more or less the same since the computation is as simple as figuring out which side is the data point based on the learned boundary.

VII. NEURAL NETWORK

Neural networks utilize a perceptron's ability to take multiple input values, multiply by a scalar (learnable weights) and superimpose onto another bias scalar (learnable bias), then the output is usually transformed by nonlinear function. When a number of perceptrons are in parallel acting as a layer, processing input attributes, and then pass down the processed values to another layer of perceptrons, and so on, this creates a network with high capability of processing nonlinear information. Figure 16 below visualizes how a network of perceptrons are structured [14]:

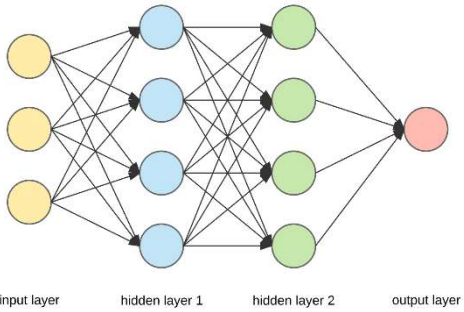


Figure 16 - Neural network sample visualization

To update weights in each perceptron, an optimization algorithm, like gradient descent, is needed to conduct backward propagation on the network. It is essentially trying to find out the partial derivative of the weights, meaning how sensitive the outputs are with respect to the weights. With the partial derivatives then weights can be updated accordingly.

HIDDEN LAYER 1

The number of perceptrons determines how much information a neural network can process information. Generally when the number of perceptrons increase, the network should be able to digest more non-linear inputs. However, greater number of perceptrons will require bigger dataset to not overfit. Figure 17 shows the relationship between hidden layer 1 perceptrons and accuracy scores on both datasets.

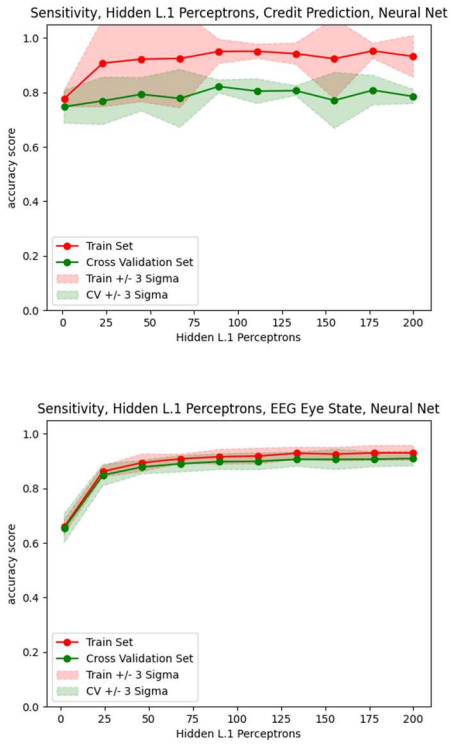


Figure 17 - Sensitivity of hidden layer 1 perceptrons on bias & variance. Top: "credit prediction", bottom: "eeg-eye state".

It is interesting that the number of perceptrons in hidden layer 1 has minor impact on accuracy score in "credit prediction" dataset. Also the variance is quite high. This is due to the small training size, and already optimized hyperparameters besides hidden layer 1.

For "eeg-eye state" dataset, we can clearly see the normal trend that increase in perceptrons does greatly improve prediction accuracy. The variance in the right plots is low. This is due to the number of perceptrons is not very high and other hyperparameters are tuned.

HIDDEN LAYER 2

When more layers are added to a neural network, it increases the ability to process higher degree of nonlinear/complex data. Figure 18 indicates that hidden layer 2 does not improve model prediction on "credit prediction" dataset, while the prediction ability is very slightly improved with the addition of hidden layer 2.

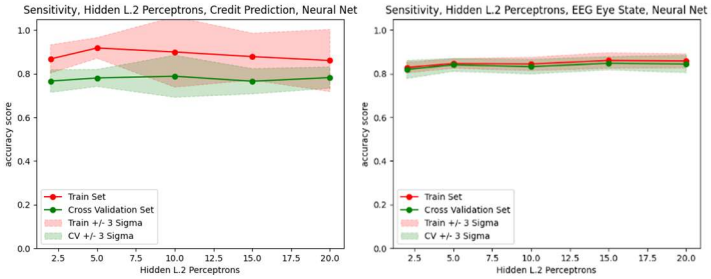


Figure 18 - Sensitivity of hidden layer 2 perceptrons on bias & variance. Left: "credit prediction", right: "eeg-eye state".

Although it is not necessary to add a second hidden layer, based on the sensitivity analysis, it doesn't hurt to add some headroom so the network is slightly larger than needed, and it is always easier to tame the variance by adding regularization.

L2 REGULARIZATION

It is generally very difficult to get exactly the size of the neural network for training. It is typically oversized in case more training data is available, it is very easy to train without modifying the neural network, and also it is easier to use some regularization to reduce variance (overfit).

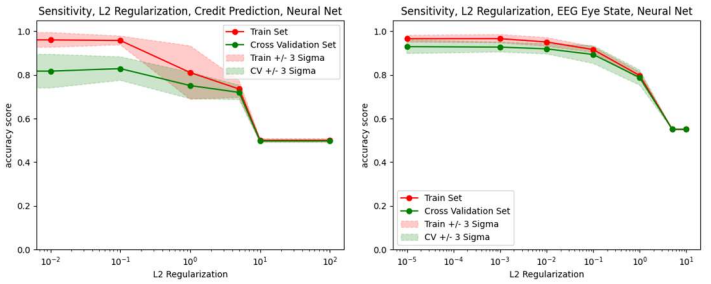


Figure 19 - Sensitivity of α, coeff of L2 on bias & variance. Left: "credit prediction", right: "eeg-eye state".

In Figure 19, we can see that on "credit prediction" dataset, α (coeff of L2) reduces the variance as it increases until around 0.1, where validation curve peaks. The bias starts to increase as α goes beyond 0.1 due to over regularization. The wider gap between train and validation curves at low α means the number of perceptrons are oversized for this dataset. For "eeg-eye state", the variance is small to start with, so α does not need to be big (1e-4) before the bias starts to increase due to over regularization.

BATCH SIZE

This is used to bundle the desired number of training samples to go through back propagation (optimizations like gradient descent) to learn the weight updates. Typically smaller batch sizes means the model learns faster (at least in early epochs), but there would be more noise in the learning curves. For larger batch sizes, the model learns slower but more stable. The reason is during weight update, more samples would provide lower sampling error, but the magnitude may be small which is harder to jump through undesired local minima.

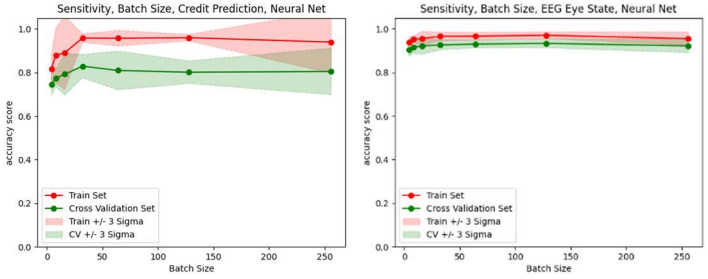


Figure 20 - Sensitivity of batch size on bias & variance. Left: "credit prediction", right: "eeg-eye state".

Figure 20 shows how batch sizes affect the bias and variance. For "credit prediction", the accuracy peaked at batch size of 32 and the variance increases with batch size higher than 32. For "eeg-eye state", the variance is not sensitive to batch sizes, but the accuracy does peak at around 128.

LEARNING RATE

Learning rate determines how big of a step the learning algorithm is taking towards a local minima at every weight update iteration. If rate is too high, the optimization may simply dance around the desired local minima. If the rate is too low, it takes longer to learn, and it may not be able to overcome undesired local minima. There are ways to get around this issue by using momentum or other optimization algorithm like adam. For this paper the optimization used is adam and it is fixed.

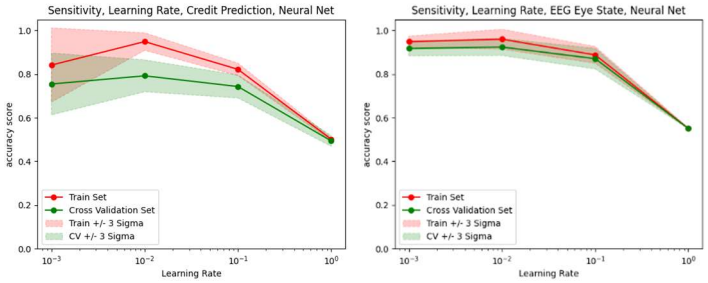


Figure 21 - Sensitivity of learning rate on bias & variance. Left: "credit prediction", right: "eeg-eye state".

Figure 21 shows the sensitivity of learning rate with the two datasets. For both datasets, the accuracy peaked (lowest bias) around 0.01. The wider variance in "credit prediction" is due to larger hidden layer perceptrons.

ACTIVATION FUNCTION

At each perceptron, after weights dot product the input signals, it goes through an activation function which transforms the value for the next hidden layer. There are several activation functions under neural network models and I will mention four of them [15].

Identity: $A(x) = x$. This is simply passing the output. Usually this is not very useful if we need the nonlinearity capability of neural networks.

Logistic (sigmoid): $A(x) = 1/(1 + e^{-x})$. This output value is bounded [0,1] and differentiable.

Tanh: $A(x) = \tanh(x)$. This output value is bounded [-1, 1] and differentiable.

Relu: $A(x) = \max(0, x)$. This output value ranges [0,x] and is differentiable.

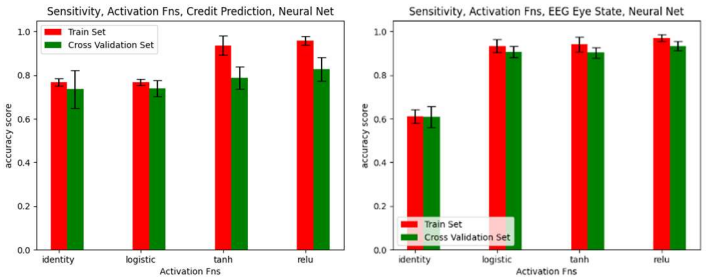


Figure 22 - Sensitivity of Activation Fns on bias & variance. Left: "credit prediction", right: "eeg-eye state".

Figure 22 demonstrates the difference in bias with four activation functions. For "credit prediction" dataset, identity is the lowest, which is not surprising. Relu is the best in terms of bias. For "eeg-eye state", identity function definitely is not generating enough nonlinearity in the model to fit the data. All the rest of the functions did decent with relu at the lowest bias. All of the functions don't impact variance which makes sense. Variance is usually associated with having bigger models trying to fit a small dataset. The bigger variance in tanh and relu in "credit prediction" is from the bigger size of the perceptrons.

OPTIMIZED LEARNING CURVES & WALL CLOCK TIME

The tuned hyperparameters:

Table 8 - Optimized hyperparameters on "credit prediction".

H.Param	α	batch	hidden L	lr	actvation
Value	0.1	32	(60, 20)	0.008	relu

Table 9 - Optimized hyperparameters on "eeg-eye state".

H.Param	α	batch	hidden L	lr	actvation
Value	5e-5	128	(250, 40)	0.005	relu

Table 10 – Early stopping hyperparameters on both datasets.

H.Param	Early stopping	Validation fraction
Value	True	0.2

For the plots with epochs as x-axis in Figure 23, there is no easy way of getting real time information of the accuracy scores and elapsed time during training with sklearn package, so a loop is generated to incrementally increase epochs for training.

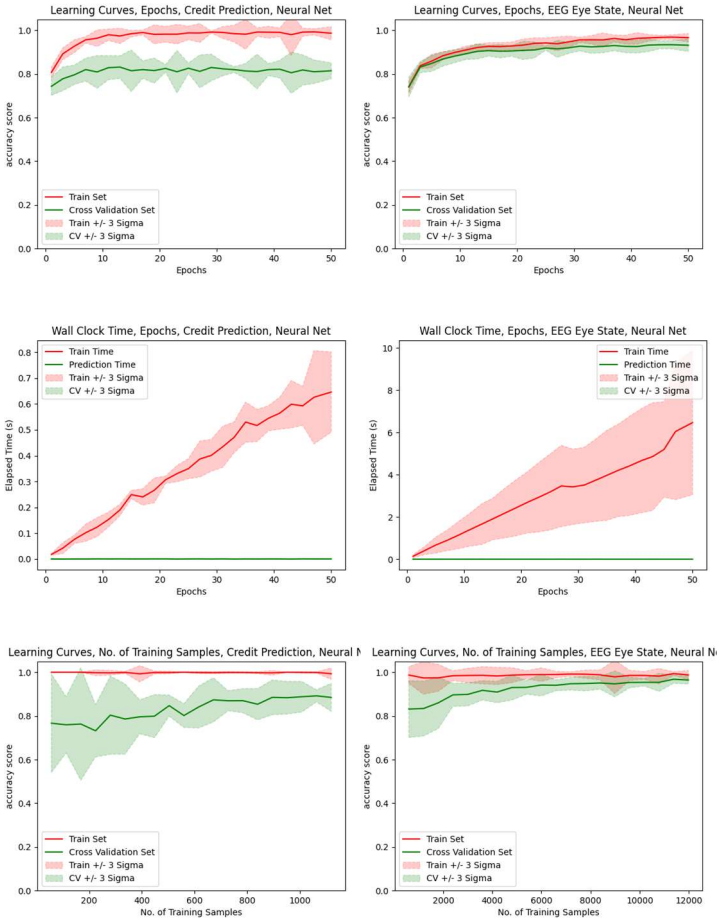


Figure 23 - Learning curves and wall clock time for two datasets. Left: "credit prediction", right: "eeg-eye state". Top row: lr curve vs epochs. Middle row: wall clock time vs epochs. Bottom row: lr curve vs train samples. For prediction curves, no. of samples are x0.2 for "credit prediction" and x0.1 for "eeg-eye state".

For this reason the wall clock time curves may have local zig-zag but the overall trends are the same, linearly proportional to number of epochs. For the same 50 epochs, it takes about 20 times longer to train due to the difference in number of training samples and the size of the networks. With smaller training samples, it also means training curves converges faster, as shown in top left corner of Figure 23. The downside of neural networks is it needs more training data to do well compared to other models, so the bias improvement is not very apparent. For larger dataset like "eeg-eye state", we can see initial learning is fast, typical of neural networks. As the epochs increase, the bias decreases while the variance starts to increase. An early stopping mechanism is implemented when tuning the neural networks to prevent overfitting. This is very important as neural network is the only ML model here which can overfit when training samples are repeatedly fed too many times for training. See Table 10 for related hyperparameter values.

It is also interesting to include the learning curves vs training samples. The bottom row plots of Figure 23 show such curves. Both of them start with high variance which makes a lot of sense. The model sizes are tuned for specific datasets, so when only a

fraction of the whole training samples are fed to train, it is bound to overfit. As more training samples are available, the bias and variance will both decrease.

VIII. KNN

K-nearest neighbours is a memory-based learning instead of instance-based learning like the other four models. This means all training data is required to be present for KNN to predict new samples. It does predictions by going through the training dataset and find closest k neighbours (to the new input) in the hyper dimension space.

POWER, OF DISTANCE

Measure of distance is required to find k nearest neighbours. To define distance, we can use Manhattan (power of 1), Euclidean (power of 2) distance, so on and so forth. The Figure 24 shows how the order of power affects bias in the two datasets.

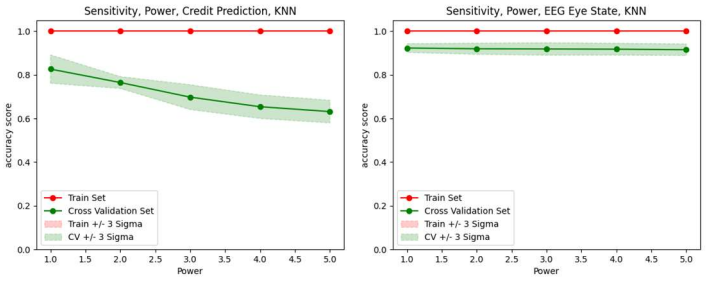


Figure 24 - Sensitivity of power on bias & variance. Left: "credit prediction", right: "eeg-eye state".

Surprisingly Manhattan distance works the best for "credit prediction" dataset. Upon more research, I found a paper [17] describing that data with high dimensionality could benefit from lower order distance measurement. The number of attributes to number of samples is indeed very high. On the other hand, there is no noticeable difference among powers on the "eeg-eye state" dataset. There are only 14 attributes while there are 15k samples.

K, NO OF NEIGHBOURS

K neighbours are used to determine how the new datapoints should be classified. When k is low, the closest points are considered which can provide more accurate answer, but if nearby points have noises then the prediction can be off. When k is high, it takes more points for average, but it can sometimes include undesired influence from less related data points.

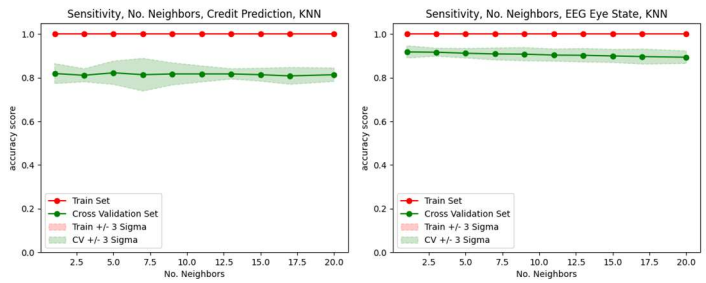


Figure 25 - Sensitivity of k on bias & variance. Left: "credit prediction", right: "eeg-eye state".

As seen in Figure 25, k value has little impact on "credit prediction" dataset bias and variance. Reason being at high dimensionality, the usual assumption of "similar" data points being close together may not be true anymore. In other words, the curse of dimensionality. For "egg-eye state" dataset, the lowest bias is at k=1~3. Since the output is a binary classification, it does not rely heavily on "averaging" of neighbours, which leads to finding the closest few data points will suffice. The validation bias increases as k increases. This is expected as more data points are included, less related data points will have influence on the output label.

DISTANCE WEIGHTS

The KNN learner can either treat all k neighbours the same weights, or by distance. Figure 26 shows how uniform or distance weights affect the accuracy. Since the optimized hyperparameter k is not one for both datasets, we can expect some difference. In both datasets, the distance weights performed slightly better than uniform weights.

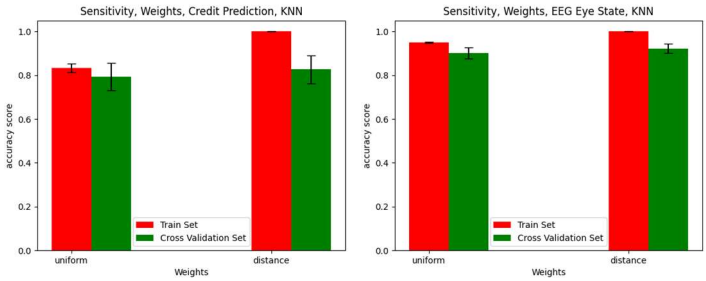


Figure 26 - Sensitivity of distance weights on bias & variance. Left: "credit prediction", right: "eeg-eye state".

OPTIMIZED LEARNING CURVES & WALL CLOCK TIME

The tuned hyperparameters:

Table 11 - Optimized hyperparameters on "credit prediction".

H.Param	k	p	weight
Value	8	1	distance

Table 12 - Optimized hyperparameters on "eeg-eye state".

H.Param	k	p	weight
Value	4	1	distance

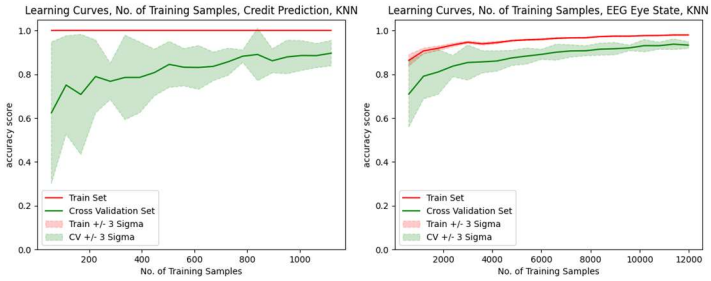


Figure 27 - Learning curves for two datasets. Left: "credit prediction", right: "eeg-eye state".

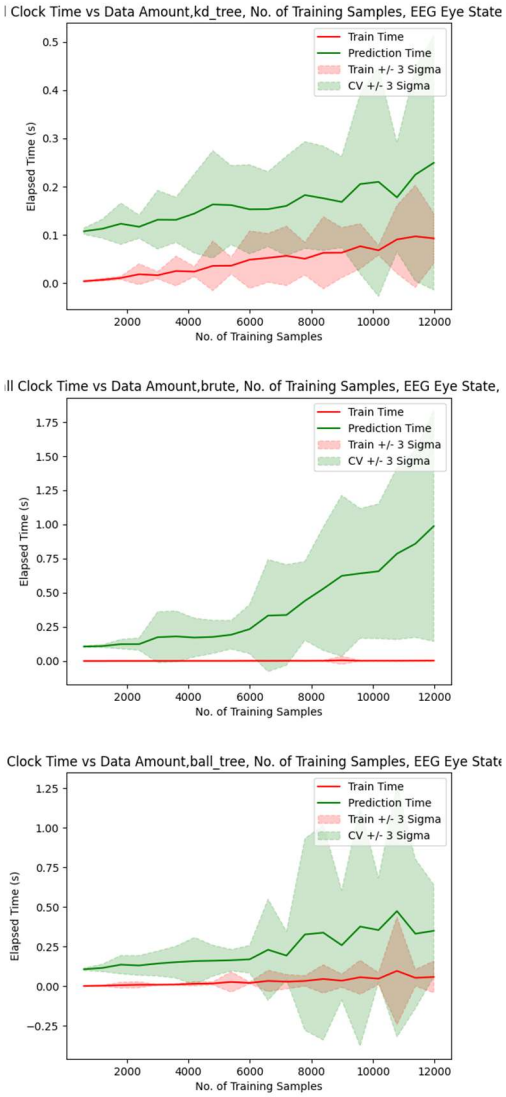


Figure 28 - Wall clock time for two datasets. Top: kd_tree, middle: brute. Bottom: ball_tree. For prediction curves, no. of samples are x0.2 for "credit prediction" and x0.1 for "eeg-eye state".

In Figure 27, the training curves show typical behaviour where when the number of samples increases, KNN can better predict the new data labels. The "training" curves are not very meaningful as there is no training involved in KNN.

For wall clock times, if different types of trees are constructed, it affects the prediction time. As seen in Figure 28, kd_tree and ball_tree are faster in terms prediction comparing to brute. This is however data specific so in other datasets this can behave differently.

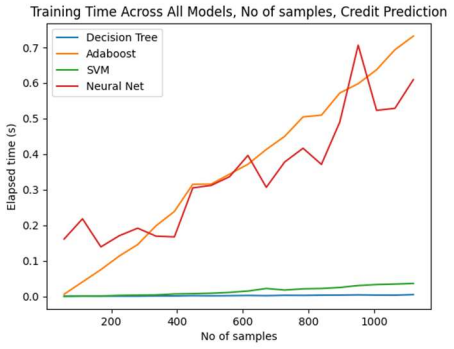
IX. TEST SCORES & OVERALL COMPARISON

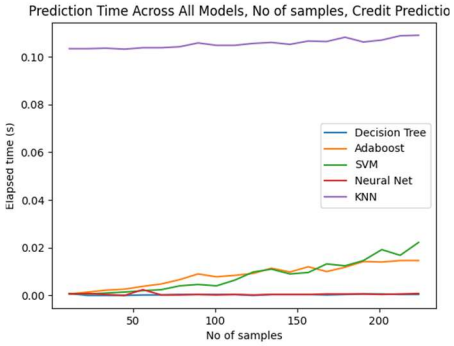
The final step is to compare test score results among 5 learners. Table 13 lists the cross validation and test scores of all five ML models on both datasets. Best on test scores, It shows that on the "credit prediction" dataset, SVM performs the best. Adaboost comes in second and neural network performs the worst. SVM wins in this case because it can typically perform quite well with small dataset and with high dimensionality. A simple decision tree suffers since it is not really capable of handling nonlinear data. KNN suffers because of the curse of dimensionality. Neural network has the lowest score because it does need a lot of data to do well. It may be possible get better scores with tuning the hyperparameters more in details by adding a lot more choices in network sizes, but it will increase the tuning time exponentially. The NN training time is already one of the worst according Figure 29. On "eeg-eye state" dataset, adaboost really shines here as it takes the top spot. Adaboost also does well in the first dataset. One of the biggest advantage of boosting is it can be tuned to not overfit, and that's why we can "boost" its performance from weak learners. Its downside is of course the training time required, according to Figure 29. SVM scores 2nd place and neural network is a close 3rd. Since the 2nd dataset has a lot more data compared to the attributes, neural network is able to improve a lot. Decision performed the worst likely due to attributes being numerical, which is less suitable for tree-like learning models.

Table 13 – Cross validation and test scores (Accuracy).

	"credit prediction" data		"eeg-eye state" data	
	CV score	Test score	CV score	Test score
DT	0.7929	0.7893	0.8384	0.8374
Adaboost	0.8437	0.8250	0.9565	0.9626
SVM	0.8607	0.8750	0.9359	0.9436
NN	0.8286	0.7786	0.9332	0.9366
KNN	0.8259	0.7893	0.9171	0.9156

According to Figure 29 - "credit prediction" dataset, in terms of training speed, decision tree is the fastest, and SVM does pretty well because the amount of data is small. Adaboost and neural net are the slowest to train because adaboost has 500 estimators and the neural network size is not very small. For prediction speed, neural network and decision tree are the fastest and KNN is the worst, which is expected because it needs to go through the entire dataset to find k nearest neighbours.





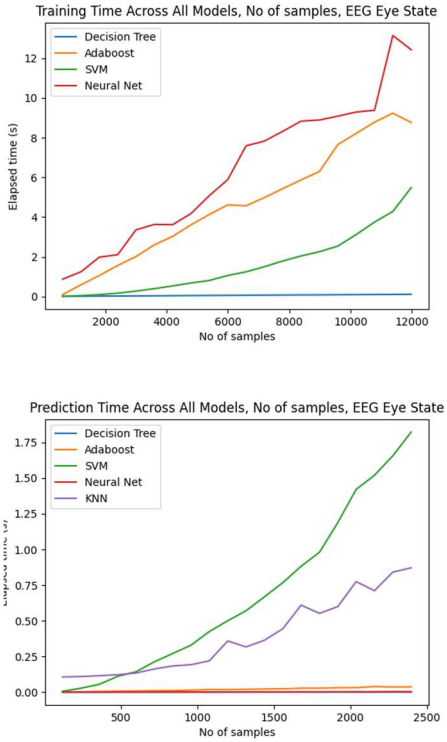


Figure 29 – Training and prediction time on both datasets from all models. Top 2: "credit prediction", bottom 2: "eeg-eye state".

For "eeg-eye state", neural network and adaboost have the worst training time likely due to neural network's iterative training where the same data is fed multiple times before convergence. For adaboost, the high number of estimators is needed to decrease bias, but at the cost of training. We can also clearly see the exponential curve of SVM in both training and prediction. If the number of samples go up to millions, it will likely not be favourable at all unless subsampling is done for training. For neural network and adaboost at prediction, they are very fast which scales up really well with big data.

To pick the most sensitive hyperparameters for every model, here are my choices, based on qualitative comparison in the sensitivity plots:

Model	Hyperparameter(s)
Decision Tree	Max depth
Adaboost	No. of estimators
SVM	Gamma and kernels
Neural Network	Learning rate and L2
KNN	distance power

X. REFERENCES

1. Numpy Usage, URL: https://numpy.org/
2. Pandas Usage, URL: https://pandas.pydata.org/
3. OpenML, URL: https://www.openml.org/apis
4. OpenML Credit Prediction Dataset: URL: https://www.openml.org/search?type=data&sort=runs&status=active&id=31
5. OpenML EEG to Eye State Prediction Dataset: URL: https://www.openml.org/search?type=data&status=active&id=1471
6. Pandas Profiling, URL: https://pypi.org/project/pandas-profiling/
7. Scikit Learn Standard Scalar Usage, URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
8. Aznar, Pablo, "Decision Trees: Gini vs Entropy", URL: https://quantdare.com/decision-trees-gini-vs-entropy/
9. Korstanje, Joos, "The F1 Score", URL: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6
10. GATECH CS7641 Machine Learning, URL: https://gatech.instructure.com/courses/299858
11. Kernel Functions-Introduction to SVM Kernel & Examples, URL: https://data-flair.training/blogs/svm-kernel-functions/
12. C and Gamma in SVM, URL: https://medium.com/@myselfaman12345/c-and-gamma-in-svm-e6cee48626be
13. Scikit Learn SVM Usage, URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
14. Dertat, Arden, "Applied Deep Learning – Part 1: Artificial Neural Networks", URL: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6
15. Scikit Learn MLPClassifier Usage, URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
16. Imbalanced Learn SMOTE Usage, URL: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html
17. Charu C, et. al., "On the Surprising Behavior of Distance Metrics in High Dimensional Space", institute of Computer Science, University of Halle Kurt-Mothes-Str.1, 06120 Halle (Saale), Germany, URL: https://bib.dbvis.de/uploadedFiles/155.pdf