CS7641- Assignment 1: Supervised learning

Tirthajyoti Sarkar (GT ID: tsarkar31)

Abstract: *This assignment deals with five supervised learning algorithms and their use for classification tasks. We choose two datasets which are 'interesting' in their own way. One is a real dataset about loan default and another a synthetic dataset from a well-known textbook. We analyze classification accuracy performance of Decision Tree, Boosting (AdaBoost), Support vector machine (SVM), k-Nearest Neighbor (kNN), and multi-layer perceptron (MLP) on these data sets. Learning curves and various types of model complexity curves were generated from this analysis to understand the features of these learning algorithms.*

Introduction

Classification is one of the most important machine learning tasks. Understanding 'concepts' about the world around us starts with classifying objects into categories. Before learning the concept of separate taste (sweet, sour, salty, and bitter) perhaps little babies inherently classify food in binary categories such as 'tasty' and 'not tasty'. In the grand journey to make machines truly intelligent and self-aware, the first step often starts with their classification ability.

In this assignment, we perform classification on two separate data sets using five (5) algorithms,

- Decision Tree with some pruning
- Boosting (ensemble of Decision Trees)
- *K*-Nearest neighbor
- Support Vector Machines (SVM)
- Multi-layer perceptron (MLP) aka feedforward/densely connected neural network

We strive to show prominent features of these classification algorithms by plotting,

- Model complexity/hyperparameter vs. model accuracy (on training and validation set)
- Training data set size vs. model accuracy (on training and validation set) – learning curve

Datasets

We chose two data sets for this assignment. The choice of the data set is a non-trivial task as these are supposed to 'showcase' the intricacies of the algorithms through the model complexity and learning curves. If all of the algorithms show nearly identical accuracy for the data sets over a wide range of parameter space and training set size, then the data is either extremely easily separable or have lot of irreducible bias/noise. In that case, they may be considered 'not interesting'. Therefore, after some deliberation I chose following two datasets for this assignment.

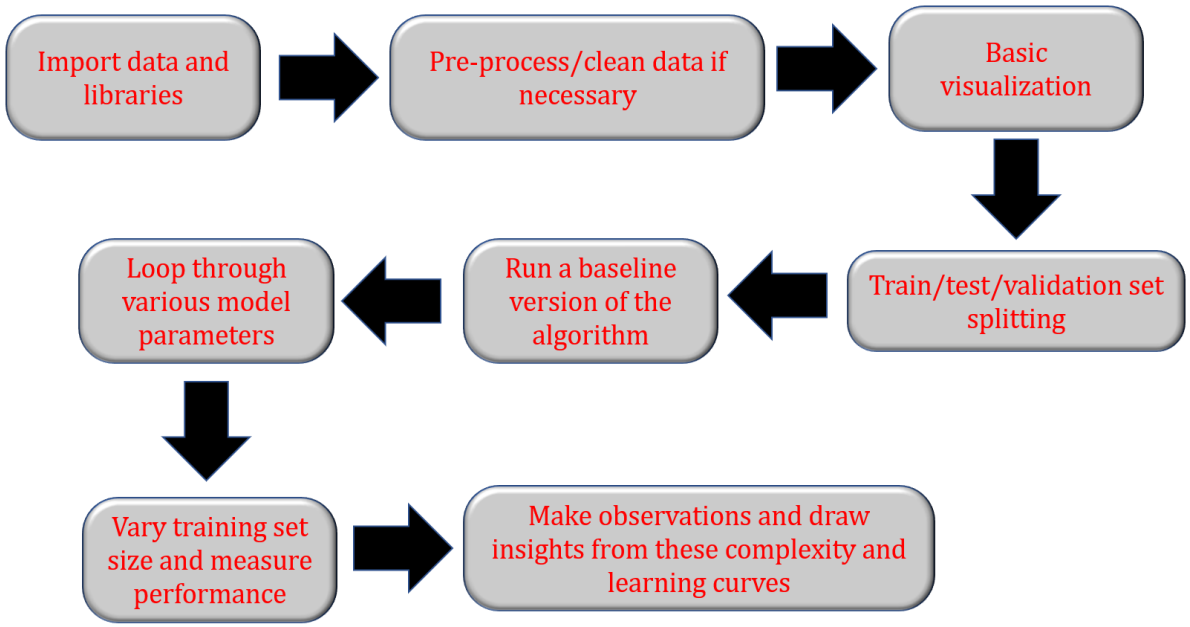
- Loan default prediction data set
- A synthetic data set generated by Python Scikit-learn library inspired by Trevor Hastie's book "*Elements of Statistical Learning*".

Here are some distinguishing features justifying the choice of these two data sets,

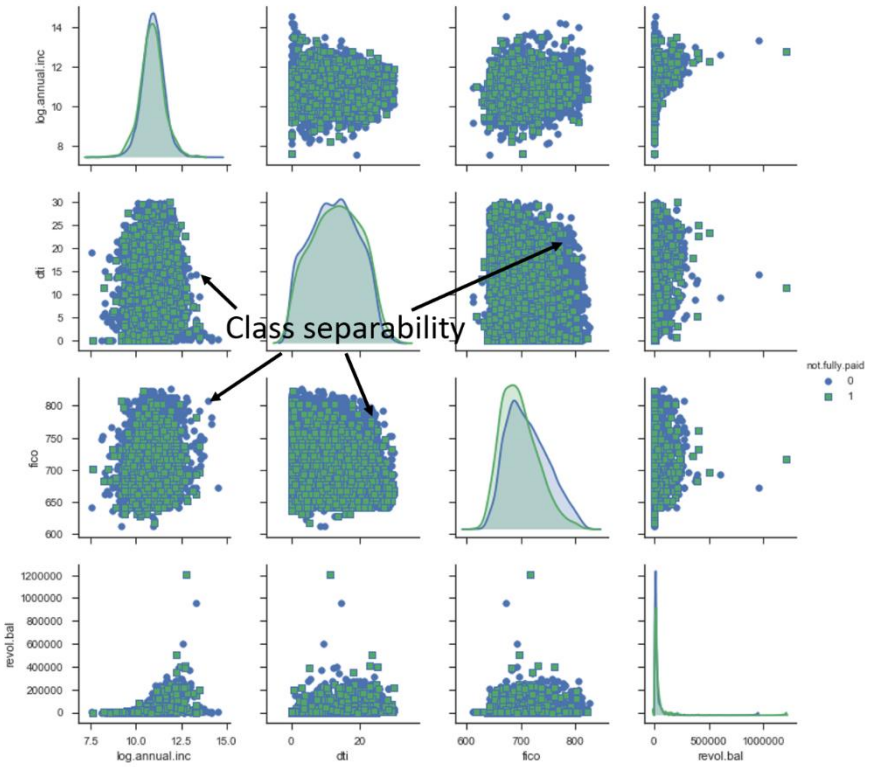
Features	Loan default data set	"Hastie" data set
Source	Real world	Synthetic
Irreducible error	Shows a fixed irreducible error i.e. a bias from data pattern/unknown noise, which will always introduce a relatively fixed amount of error rate for any validation set. Even ensemble method like Boosting cannot reduce this error.	Ensemble method like Boosting shows clear improvement over Decision Tree for this data set.
Neural Net is not the king	All the algorithms could achieve similar level of validation accuracy.	Neural network actually showed poor performance.
Complexity and learning curves	Shows its own distinct trends in the model complexity and learning curves.	Shows its own distinct trends in the model complexity and learning curves.
Learning rate of Boosting	N/A	Showed interesting inverted bell shaped accuracy for the learning rate indicating there is an optimum learning rate.
Impact of kernels in SVM	Both polynomial and RBF kernel showed similar max accuracy performance.	Clearly showed the advantage of RBF kernel over the simple linear separator. Also, helped estimating the impact of `gamma` parameter in `RBF` kernel.

General flow of the machine learning pipeline

The general flow of the machine learning pipeline we used, is as follows,



Experiments on the Dataset-1: Loan default prediction data



This is a dataset, publicly available from LendingClub.com. Lending Club connects people who need money (borrowers) with people who have money (investors). We try to create a model to predict the risk of lending money to someone given a wide range of credit related data. We will use lending data from 2007-2010 and be trying to classify and predict **whether or not the borrower paid back their loan in full.**

To get a feeling about the class separability of the data, we plot pairwise scatters and the corresponding kernel density estimates using Python `seaborn` package. We observe good overlap between the features i.e. they are not very easy to separate like some well-known datasets (e.g. Iris).

We do basic pre-processing like dropping the credit.policy feature as that is based on some internal model of LendingClub and should not be considered in an open-ended machine learning task. Also, we **one-hot-encode the 'purpose' variable** to multiple binary-valued dummy variables.

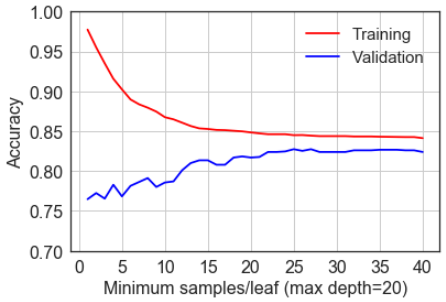
```
print("Shape of validation set:", X_val.shape)
print("Shape of test set:", X_test.shape)
print("Shape of training set:", X_train.shape)
```

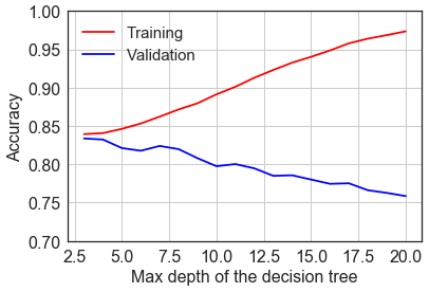
```
Shape of validation set: (1437, 17)
Shape of test set: (1437, 17)
Shape of training set: (6704, 17)
```

Thereafter, we split the data in 70/30 ratio, keeping the 70% for training. Between the 30%, we again split that in 50:50 ratio to create a Test set and a Validation set. In fact, for this particular exercise, the **test set data is not used till the very end as we do all modeling using the training and validation set only**.

Decision Tree algorithm

We build a decision tree by calling the `DecisionTree` classifier of Scikit-learn package. Default model returns: ***80% accuracy on the validation set***.



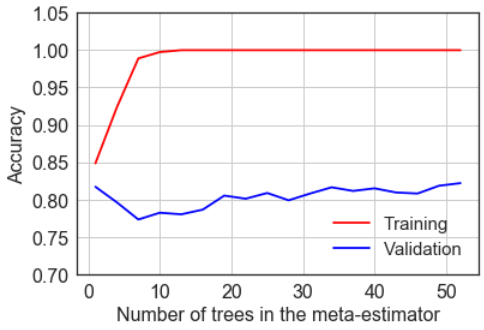


Pruning the Tree: Restricting *maximum depth* and tuning *minimum number of leaf nodes*

Pruning is important to increase the robustness of a decision tree classifier i.e. increase its chance of performing better with unseen test data. One way to prune the tree is to not let it grow indefinitely. We test this by setting the `max_depth` parameter. As expected, we see an increase in the validation set accuracy, accompanied by a decrease in training set accuracy for smaller trees. This means that for this dataset, a deeper tree clearly overfits the data and a shallow tree of depth 2/3 should be considered. Similarly, we tweak the `min_samples_leaf` parameter to limit overfitting. If we allow a single leaf per node then the algorithm will fit to the noise of the data down to a single data point (splitting data on as low as two points) thereby making the decision boundary extremely nonlinear and vulnarable to the nosie.

Learning curve

We run a loop over a range of fractions (10% to 100%) and **sample random points up to that fraction from the original data set**. For each iteration, we run a decision tree model and **calculate prediction accuracy over the sampled training data and original validation set**. The plot shows that for this dataset and a sensible choice of hyperparameters, the accuracy is not a strong function of training data size down to small fraction of the original data. This means, in principle, **we can run the algorithm on a much smaller model than using all the data and still create decent predictions.**

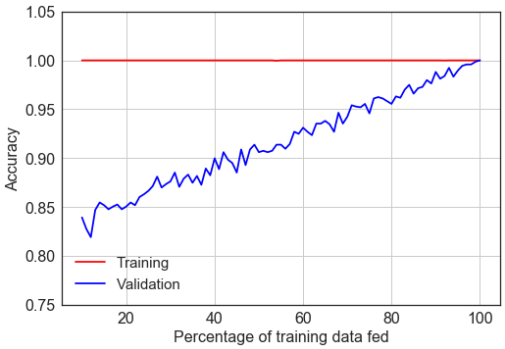


Boosting algorithm

We use `AdaBoostClassifier` from Scikit-learn for ensemble learning exercise. Adaboost is one of the earliest meta-learning algorithms, formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. Here is a concise article describing the concept.

Impact of the number of estimators

Adaboost works by combining multiple 'weak learners' and dynamically updating the weights of those learners and training data depending on whether in a given pass, those training data was correctly classified or not. Therefore, it is expected that **with increasing number of estimators, training accuracy should reach very high and the validation accuracy also should increase to the point where it hits the irreducible error rate**. We observe this trend in the result. Also, the

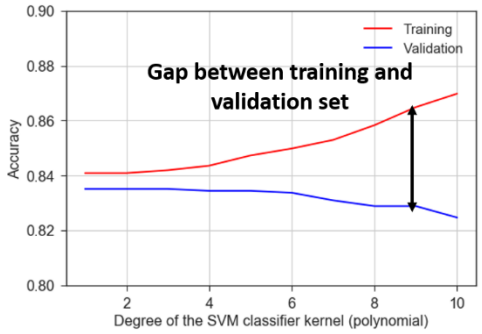


biggest different between a meta-learner like Adaboost and a single decision tree is that the **validation accuracy does not drop back sharply with increase in estimators i.e. model complexity**.

Learning curve

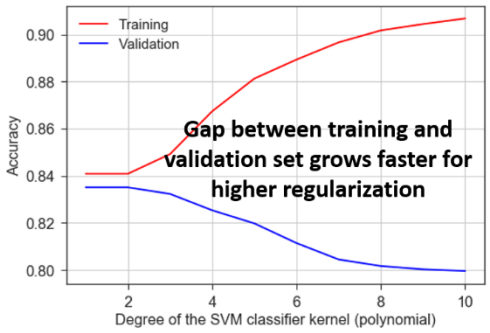
The learning curve is plotted following the similar process as described above with decision tree. Training accuracy is almost 1.0 for all data set size and **validation accuracy approaches training accuracy with increase in the training set size**. We used 20 tree estimators for this exercise.

Support Vector Machine (SVM) algorithm



SVM is one of the most widely used non-probabilistic binary classifiers. We use `svc` method from Scikit-learn for this exercise. But, **SVM requires normalizing the data to the same scale**. So, we use `StandardScaler` method to pre-process the data to the same scale and then apply SVM. Default model shows ~ 83% accuracy on the validation set.

Playing with kernels and regularization parameter



We play with `polynomial` and `rbf` (radial basis function i.e. Gaussian) kernels. The degree of polynomial does not seem to improve validation accuracy i.e. a lower degree kernel or even a linear classifier is good enough. In fact, the result shows the **familiar gap between training and validation accuracy as the model complexity (degree) increases.** A higher degree kernel surely overfits the training data. Further, we also examine the effect of regularization parameter C and observe that this gap grows faster for a large value of C. This means, the **increasing strength of regularization imposes more penalty on a complex, higher degree model in terms of validation accuracy while the training accuracy can saturate to 1.0 due to over-fitting and not depending on the regularization.**

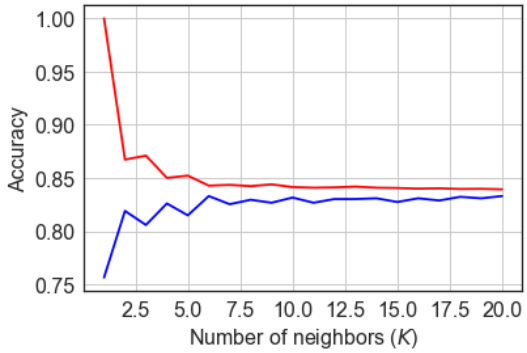
Learning curve

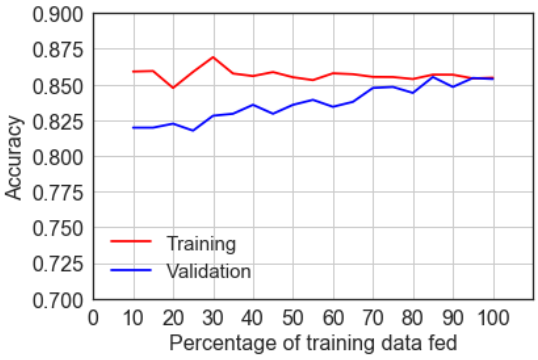
The learning curve of the SVM model turns out pretty uninteresting for this dataset. SVM model with 2nd degree polynomial kernel is used for this experiment, and just like the single decision tree model, it shows relatively little dependence on the amount of training data. For a machine learning engineer, **this is still a very useful result indicating that this dataset can be classified using relatively simple kernel and using**

small amount of data randomly sampled from the full dataset. Therefore, the whole **learning process can run with high speed**.

k-Nearest Neighbor algorithm

k-Nearest Neighbor or *k*NN, is one of the simplest algorithms for classification. It is extremely fast and **works through lazy evaluation** i.e. it **does not build any explicit model** but just stores the input training data and predicts the label of a test point by calculating various distances of other points from the test point. We use `KNeighborsClassifier` method from scikit-learn stable. For default model, we use *k*=3 and obtain ~80% accuracy. Just like SVM, **we use the scaled data for this algorithm.**





How does the choice of number of neighbors impact algorithm performance?

Increasing the number of neighbors have a 'smoothing effect' on the decision boundary created by *k*NN i.e. noise and irregularities in the data get 'averaged over' when distances from multiple neighbors are considered. Quite expectedly, **this should increase the validation accuracy**. We observe exactly this trend in the plot. **Interestingly, when *k*=1, the training accuracy is always 1.0 because the model just stores the value of that point itself** (rote learning, just remembers the position). Training and validation accuracy approach one another to a plateau as k is increased. For this dataset, *k*=5 should be good enough.

Learning curve

Learning curve for *k*NN is also not super interesting for this dataset. We extract the learning curve for *k*=5. Validation accuracy does exhibit a little dip for very small fraction of data. Anything above 50% should be good for a decent learner.

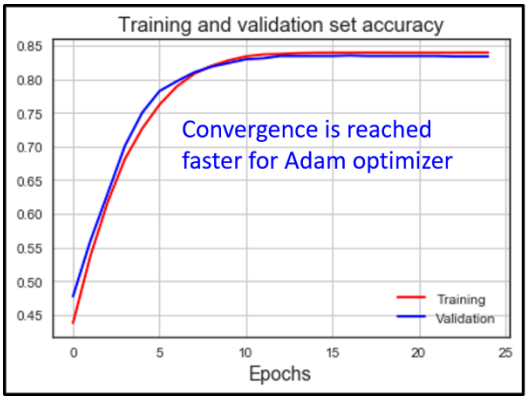
Multi-layer perceptron (MLP) *aka denseley connected deep network*

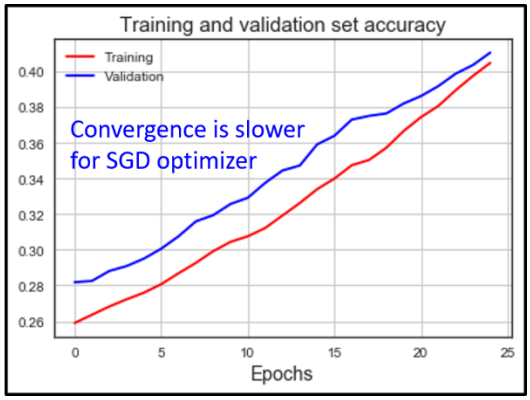
In this report, we don't spend any time discussing the vast amount of information that can be said about such powerful models. Interested readers must read Deep Learning Book by Goodfellow et al. We use **keras** high-level API (based on **TensorFlow backend**) to build our MLP. For this dataset, we chose a **2-hidden-layer network** with **10 neurons/5 neurons/0.2 dropout/binary cross-entropy error**. Because neural network has a vast number of hyperparameters, we built a couple of helper functions (see attached Jupyter notebook) to easily tune them.

Does the choice of optimizer matter?

There are many choices of optimizers which perform backpropagation for MLP. We try two of the most common optimizers to see their impact on the speed of convergence. The results show that for this dataset, **Adam optimizer** reaches convergence faster i.e. in lesser number of epochs than

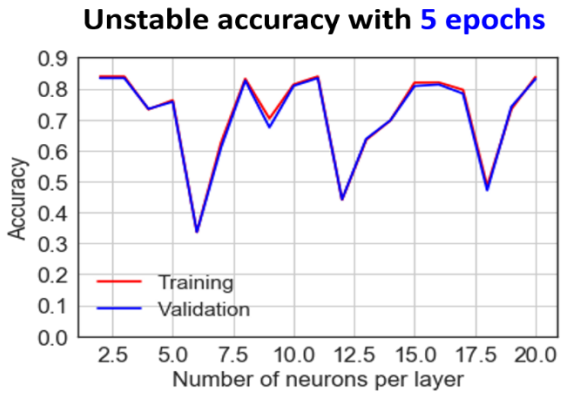
conventional **stochastic gradient descent (SGD)**. Another distinct feature of this set of results is that they show that in case of MLP, **the validation accuracy tracks the training accuracy closely.** This kind of parallel trend is not seen in other type of algorithms. One of the major contributors to this type of generalizing power of MLP is **dropout technique which randomly prunes some nodes and therefore make the network robust against overfitting**.

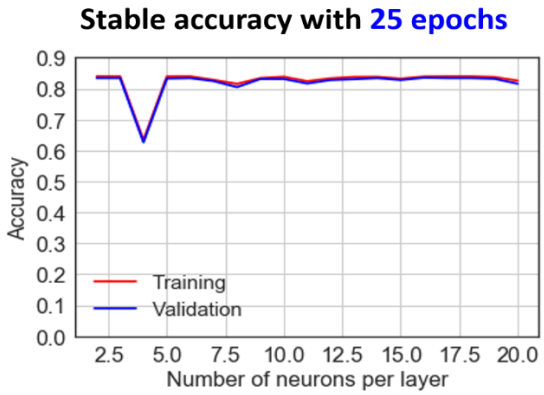




How many neurons do we need? How many epochs should we run for?

Above are some of the natural questions that arise for a MLP. With our helper function, we run a loop over a wide range of neurons/hidden layer, extract training and validation accuracy, and show that the accuracy varies wildly with the model complexity i.e. number of neurons when we train it for small number of epochs. But this variation is smoothed out with large number of epochs.



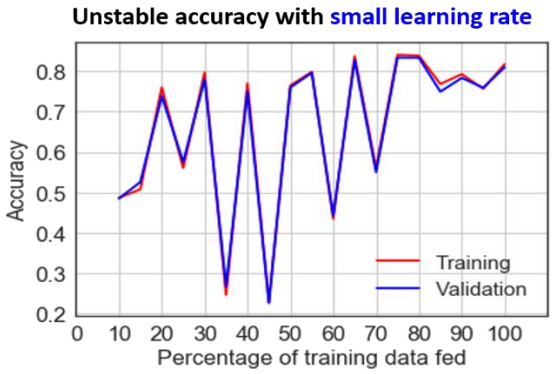


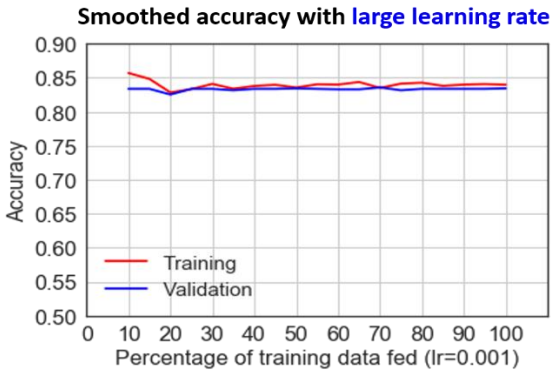
Learning curve (and how it is impacted by the *learning rate* of the network)

Neural networks are known for their insatiable hunger for data! We plot the learning curve to see the expected variation of performance for smaller training set size. For small fraction of data, the network cannot learn up to the required degree of complexity (nonlinearity) present in the data set. Therefore, the performance oscillates. Effectively, the objective function **wanders around some local minima**.

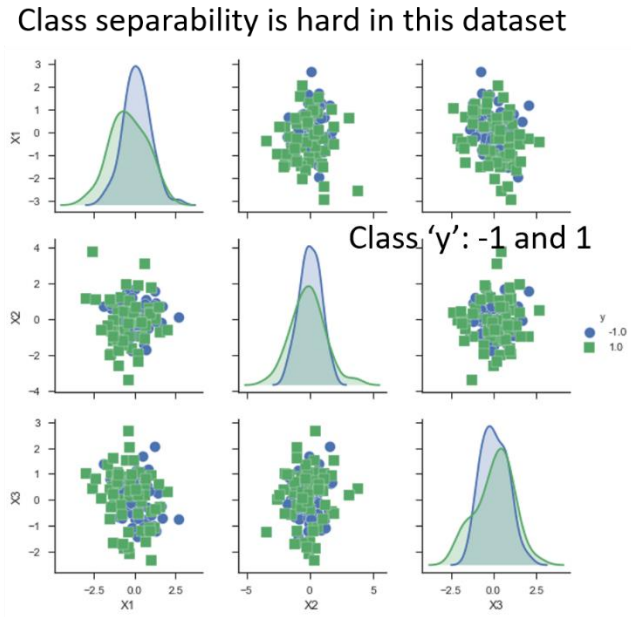
However, for this dataset, we also found that by dialing the learning rate parameter higher, we can mostly smooth out this variation i.e. make the network learn on a smaller subset of training data. With a **high learning rate, even with a small data, the backpropagation makes larger jumps and therefore, on average, converges closer to a good minimum.**

It is not recommended for all types of network and tuning the learning rate is an art. But it worked for this dataset.



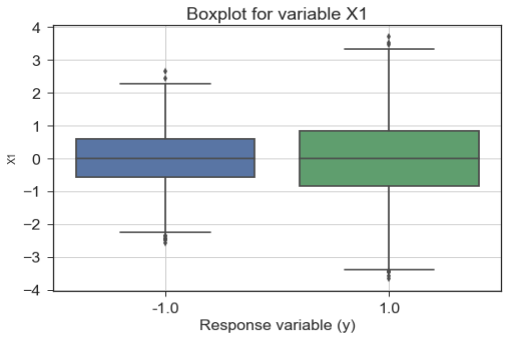


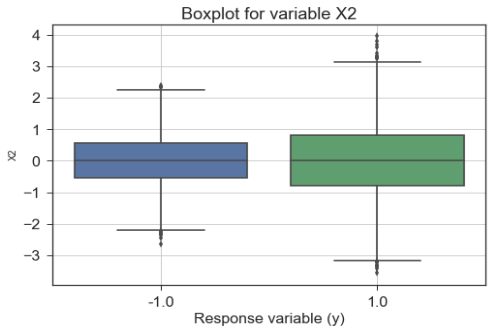
Experiments on the Dataset-2: Synthetic 'Hastie' dataset (scikit-learn)



Now we turn to the 2nd dataset for this assignment. In the loan data dataset, we did not observe a markedly improved performance with Boosting algorithm over the single decision tree. Therefore, we choose to use a synthetic dataset to clearly demonstrate this advantage. We use `datasets.make_hastie_10_2` method from scikit-learn to generate 12,000 samples with 10 feature variables and a binary class variable.

Apart from pairwise scatter plots, to illustrate the class separation, we show boxplots grouped by response classes. It is clear from these plots that **there is lot of overlap between classes for every individual feature**. Therefore, it is be expected that **a single feature or a single decision tree may not be an effective classifier but an ensemble meta-learner could prove to be effective**.





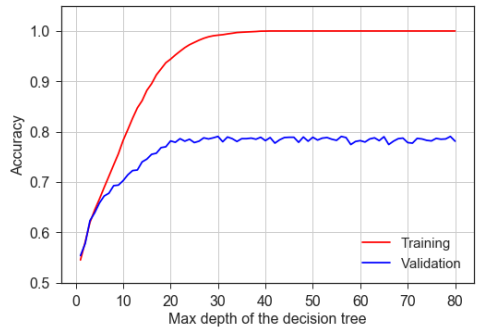
Note

For the sake of brevity, we neither repeat the procedure/parameters for generating model complexity or learning curves nor discuss details of the classification algorithm. We just show the

results for each algorithm and explain the observation, especially if any result appears significantly different from the other dataset. We divide the data in training/ test/ validation sets following the exact same procedure as described for the loan dataset. Next, we show the model complexity, learning curves, and running time plots for various algorithms.

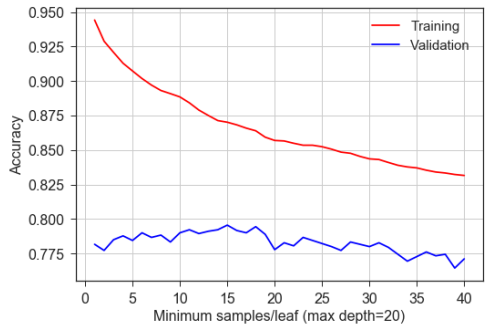
Decision Tree algorithm

Accuracy with default `DecisionTreeClassifier` model is only 70%. This is because of the harder class separability as can be seen from the scatter and boxplots.



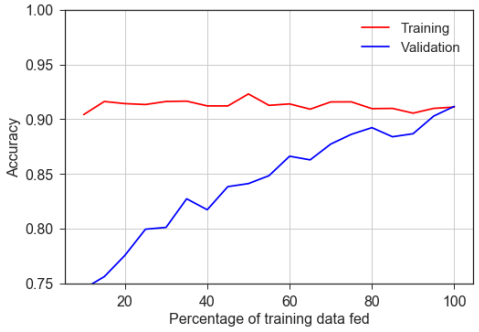
Pruning the Tree: Restricting *maximum depth* and tuning *minimum number of leaf nodes*

For this dataset, the pruning impacts the training and validation accuracy in a similar manner. **A shallow tree cannot create flexible enough decision boundary** to classify test points correctly, so the accuracy increases with `max_depth` parameter and saturates after a level.



When we tweak the `min_sample_leaf` parameter, we observe that **training accuracy decreases with higher number of leaves because this forces the tree not to split the node with arbitrary degree of nonlinearity**. The training accuracy approaches to that of validation accuracy with increasing number of minimum leaves condition. However, the validation accuracy remains relatively steady over this parameter. It is, therefore, advisable to use a high number of minimum leaves restriction to create a robust, generalizable model.

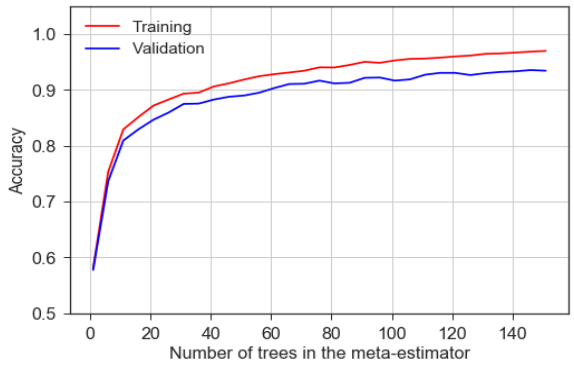
Learning curve

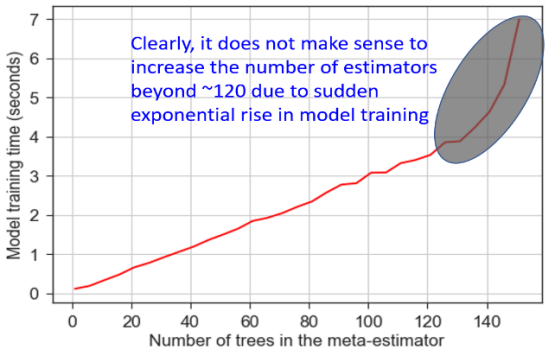


We observe classic learning curve behavior for this dataset with decision tree. The training accuracy remains relatively steady but validation accuracy rises gradually with training data size. **With highly nonlinear decision boundary required, this model generalizes better as more data is fed.**

Boosting algorithm

We use the same AdaBoost classifier and observe markedly improved accuracy with the ensemble learner. We note that individual tree showed baseline accuracy of about 70%, which can easily be classified as a 'weak learner'. To drive this point home, we set the `max_depth` to only 2 and `min_sample_leaf` to 20 for the individual trees in the meta-learner i.e. **make the pruning aggressive**. With a small depth and many leaves in the node the individual trees cannot overfit but they will have high bias and therefore weak learners by themselves. But the Boosting technique makes a powerful ensemble learner from a set this weak learning trees.

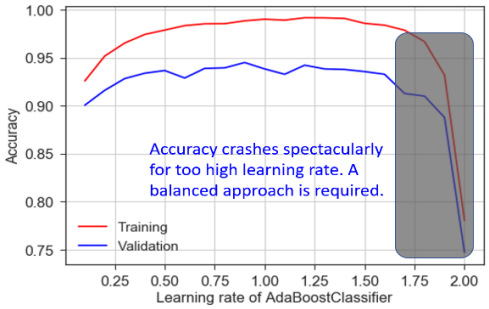




Accuracy and training time vs. # of estimators

The accuracy vs. number of estimators showed the expected trend rising from a lowly 70% to above 95%. However, we need to use a rather high number of trees > 150 to achieve > 95% accuracy. Overall, this **slowed down the model training speed considerably**. When we plot the training time vs. estimators, it becomes clear that the **marginal return of accuracy beyond ~ 120 estimators is very low** due to exponential increase in training time.

Can we tweak the learning rate?



One possible way to train model faster is perhaps tweaking the learning rate of the model. However, when we sweep the learning rate over a wide range, we see that for high learning rate, the validation accuracy crashes sharply. This means the meta-learner **settles for a local optimum or oscillates around such a point** and never generalizes enough to score high. Therefore, **careful balancing of learning rate** is desirable.

Support Vector Machine (SVM) algorithm

Impact of regularization and gaussian spread parameter on classification accuracy

For SVM algorithm, we determine that the results with RBF kernel are much more interesting than linear or polynomial kernels. Therefore, we show those results in the following figures.

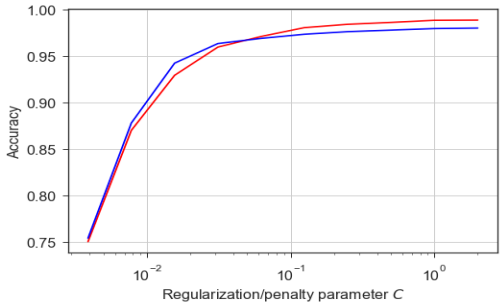


Fig: Impact of the penalty/regularization on accuracy for this dataset. **Higher penalty makes the SVM decision boundary more robust and generalizable.**

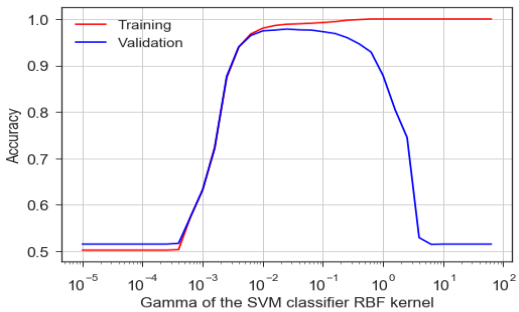
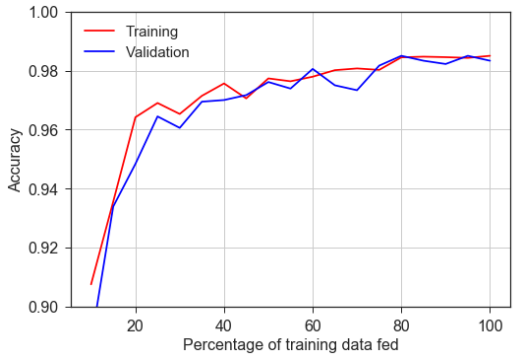


Fig: Impact of the ***gamma*** parameter (RBF kernel) on accuracy. Higher gamma means a higher spread of the Gaussian. Therefore, it allows **softer classification** and increases training accuracy. But for validation accuracy, there is a **sweet spot** of gamma and beyond that range, the classification boundary is either too hard or too soft.



Learning curve

The learning curve of the SVM model comes out as expected. For very low fraction of data, it starts with a poor classification accuracy, but rises quickly as more data are fed in and reaches a plateau. The great thing with this SVM model (with RBF kernel) is that **training and validation accuracy goes hand in hand (much like the flexible neural network model in earlier dataset).** We can say that SVM is probably the best choice of algorithm for this particular dataset.

k-Nearest Neigbor

We find that *k*NN is not an effective algorithm to perform classification on this dataset. This is probably because of the close overlap between classes. Although training data can be classified with high accuracy for small neighborhood (low values of *k*), it fails to generalize for validation set and higher value of *k*. The accuracy vs. k and learning curve are shown below.

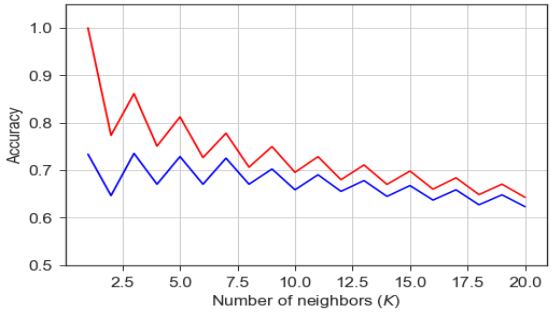


Fig: Accuracy as a function of *k*

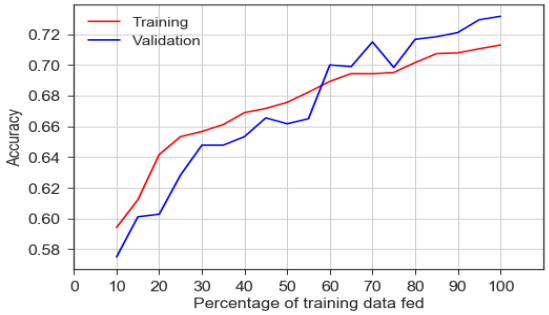


Fig: Learning curve of the *k*NN model.

Multi-layer perceptron (MLP)

MLP shows surprisingly poor performance for this dataset. We try many combinations of learning rate, architectures (layers and nodes), and optimizers but the accuracy never goes above 40-45%.

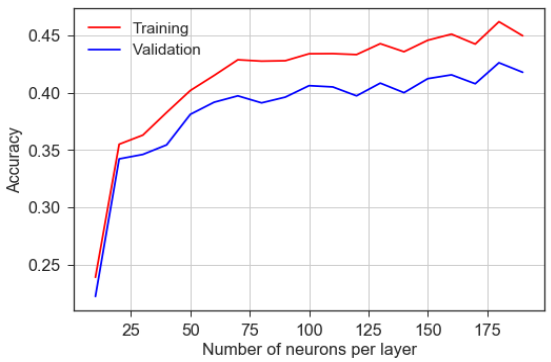


Fig: MLP Accuracy as a function of the number of neurons/hidden layer. Network performance saturates after a level of model complexity.

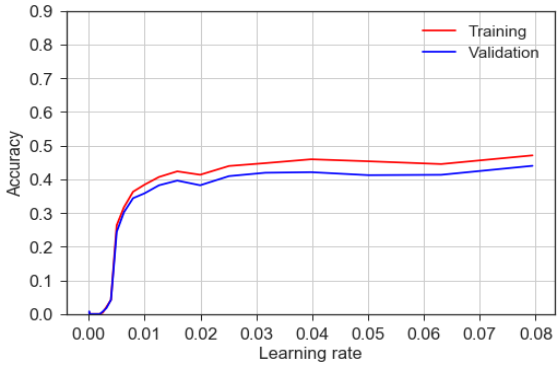
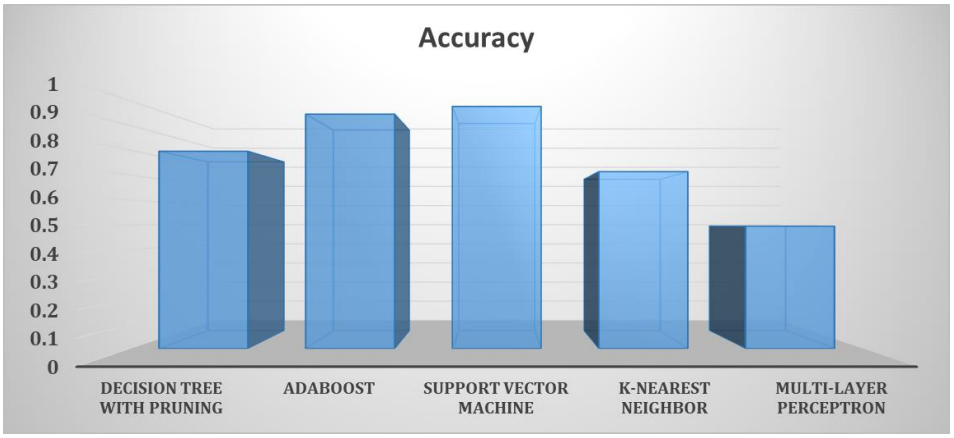
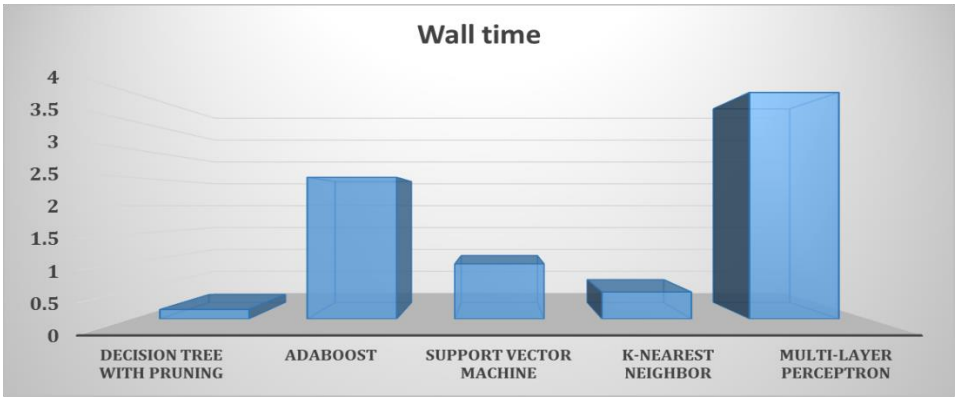


Fig: Accuracy as a function of the learning rate of the MLP. Very small learning rate is ineffective, but after a certain value, the accuracy does not improve.

Comparison among algorithms w.r.t. accuracy and wall time

We compare 5 algorithms on the **test set (untouched so far),** w.r.t. classification accuracy and wall time. We use Jupyter magic command %%time to measure this wall time, so there may be some error in this. Surprisingly, MLP fails for this data set but in general deep neural nets are an extremely powerful class of learners. **SVM is the best choice of this dataset** as it has high accuracy with reasonably low running time. Boosting can give very high accuracy but at the cost of running time as many estimator trees are required for that. KNN is not a good choice because of highly overlapped data points.





Summary and Conclusions

This was a fun learning, coding, and analysis experience. Key observations are summarized follows,

- The chosen datasets ***show different preference for classification algorithm***. First dataset had some irreducible bias, which was the limit for most algorithms. But it clearly demonstrated the ***impact of various model complexity/ hyperparameters*** on the training and validation accuracy and where they converge/diverge and why.
- Second dataset clearly ***demonstrated the power of ensemble meta-learning*** by combining multiple weak-learners. However, we also showed why ***too high a learning rate could be dangerous for the stability*** and why a balanced learning rate should always be pursued.
- The chosen datasets also ***brought out the difference between various algorithms with regard to the learning curve*** i.e. how much data they really need for a high validation accuracy. This has implication for speed of the machine learning task as ***some algorithms are better suited to work with smaller amount training data than others*** and therefore can complete the modeling task faster.

- While one dataset showed the power of densely connected neural network, other one showed it may not be the best choice for all kind of dataset. We also demonstrate the ***impact of neural network architectural choice, learning rate, and choice of optimizer*** on the classification accuracy.

Additional materials attached

Apart from this analysis document, Jupyter Notebooks with complete code (using Python 3.6+) are included in the ZIP file. README.txt explain how to get data and run the code.