Comparison of Supervised Learning Models:

Case Studies with Two Classification Problems

Jenny Hung

(Jhung34)

Table of Contents

[Introduction 2](#_Toc493687324)

[Two Classification Problems Considered 2](#_Toc493687325)

[Freddie Mac mortgages loan-level performance data 2](#_Toc493687326)

[Blood donation prediction Error! Bookmark not defined.](#_Toc493687327)

[Training and Test Results Error! Bookmark not defined.](#_Toc493687328)

[Freddie Mac loan-level dataset 5](#_Toc493687329)

[Learning curve analysis 5](#_Toc493687330)

[Complexity analysis 6](#_Toc493687331)

[Blood donation prediction dataset 9](#_Toc493687332)

[Learning curve analysis 9](#_Toc493687333)

[Complexity analysis 10](#_Toc493687334)

[Analysis of models Error! Bookmark not defined.](#_Toc493687335)

[Freddie Mac loan-level dataset Error! Bookmark not defined.](#_Toc493687336)

[Blood donation prediction dataset Error! Bookmark not defined.](#_Toc493687337)

# Introduction

We implemented the following five learning algorithms:

* Decision tree with pre-pruning
* Neural network
* Gradient Boosting
* Support Vector Machine
* K Nearest Neighbors

In addition, we introduced two separate classification problems as separate platforms to deploy the learning algorithms. Training and testing errors using each of the learning algorithms are then recorded (in terms of learning curve as well as complexity curve) and analyzed.

We showed that there is no free lunch in data science, in that the classifier that provides the best performance for one problems does not necessarily deliver the same level of performance for the other problem.

# Two Classification Problems Considered

## Freddie Mac mortgages loan-level performance data

Freddie Mac, one of the government-sponsored enterprises went through a nearly [$200 billion government bailout](http://en.wikipedia.org/wiki/Federal_takeover_of_Fannie_Mae_and_Freddie_Mac) during the financial crisis, caused in large part by losses on loans that it guaranteed. Further, Freddie Mac began reporting loan-level credit performance data in 2013 at the direction of their regulator, the Federal Housing Finance Agency. The stated purpose of releasing the data was to “increase transparency, which helps investors build more accurate credit performance models in support of potential risk-sharing initiatives.”

This financial melt-down and the subsequent bailout has motivated my research into the Freddie Mac loan-level performance dataset. This dataset lends naturally to the classification problem of mortgage defaults, and the goal of this classification problem is to analyze multiple-period mortgage risk at loan level using the Freddie Mac dataset prime and subprime mortgages originated in the United States in 2016, which includes the individual characteristics of each loan, and monthly updates on loan performances over life of a loan.

The entire Freddie Mac dataset is immensely rich, in that it encompasses a 10-year span and contains millions of loan-level mortgage performances and default information. Freddie Mac has created a smaller dataset, which is a random sample of 50,000 loans selected from each full vintage year (defined as the calendar year in which the loan was originated). Each vintage year has one origination data file and one monthly performance file, containing the same loan-level data fields as those included in the full dataset. In this implementation, we have located the sample\_2016.zip file from the full dataset package, and used this zip package as our data source for this iteration. The “2016” in the file name indicates that the loan information was recorded in the year 2016, but the loan could be originated in an earlier year (namely, the vintage year could be an earlier year). The dataset in this implementation has more than 203,000 mortgages performance records.

The 2016 zip packages has two files: sample\_orig\_2016.txt and sample\_svcg\_2016.txt. The .txt files do not come with headers but instead, we refer to the User Guide (<http://www.freddiemac.com/research/pdf/user_guide.pdf>) to grab the name of the columns. We then join the two data files together by the loan number.

It is expected that as we progressed further, we will be using larger and larger datasets. But for this first iteration, this is what we have chosen.

Missing values can be found in the dataset. Key features that are missing are more likely to be the result of reporting errors by the originator or the servicer, or incomplete information provided by the borrower. Similar to the Deep Learning paper we are reading, we have insisted that an observation must have no missing values in any of the following:

* FICO score
* LTV ratio
* Original interest rate
* Original balance

Samples missing one of the above variables are removed.

Compared to the first iteration, we have removed mas (Metropolitan Statistical Area) as a feature. This feature does not carry a lot of additional information as the geographical location can be identified through both the state and zipcode variables.

We also examined other variables where missing values exists. Good examples of these are Super Conforming flag (exceed\_conform) and First Time HomeBuyer Flag (first\_time). Our code would set any missing values to zero first. In cases of categorical variables like these, this action will yield 3 values: Y, N, and 00. These values will then be coded as dummy vairalbes / indicator variables.

In the case of a numerical variable with missing values, the missing values would still first be converted to zero. Columns of numerical variables will then be scaled while preserving the sparse structures in the next step.

To further process the data, we have taken the following steps:

* Get the delinquency status that is associated with the loans and last observed month,
* Remove the curr\_delinq from our feature space
* Use curr\_delinq as our taget
* For the categorical variables, we convert them into dummy/indicator variables

After processing the dataset, there are 203,642 loan performance observations and 109 variables.

This problem is interesting because there exists a highly nonlinear relationship between the variables and the default prediction. In particular, many of the existing academic research studied deep neural networks, which have multiple layers of hidden nodes. In this implementation, I am more interested in finding out the relative strengths, time complexity, sample complexity and mistake bounds associated with using each one of the classifiers in question.

## Blood donation prediction

The [UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/index.html) is a great repository of data science-related projects. This dataset was originated from a mobile blood donation vehicle in Taiwan. The Blood Transfusion Service Center drives to different universities and collect blood as part of the blood drive. In this dataset, we want to predict whether or not a donor will give blood the next time the vehicle comes to campus. Given my interest in discovery our mission, we're interested in predicting if a blood donor will donate within a given time window. This is considered a beginnier’s dataset.

After processing the dataset, there are 574 loan performance observations and 3 variables: number of donations, months since first donation, months since last donation.

The choice of having a beginner’s dataset is quite deliberate. Contrasting with the mortgage default prediction problem, the Blood Donation prediction problem is interesting because it motivates the comparison of classifier performances, and thus highlight the fact that different classifier have different computational complexity, sample complexity as well as mistake bounds: what worked for one problem does not necessarily transfer to the next problem.

# Analysis of Models: Training and Test Errors

In general, a learning curve demonstrates how accuracy of a learning algorithm changes as a function of training size. The rationale is as follows: given a train/test set partition which gives a given sample size , we randomly select data points from the training set, apply the learning algorithm to this set, and evaluate the model accuracy on the test set to determine the accuracy. This accuracy is then stored for each given train/test set partition, then a graph is plotted. In my implementation, I adopted the learning\_curve() function from the scikit-learn package, which determines cross-validated training and test scores for different training set sizes. In particular, a cross-validation generator splits the whole dataset k times in training and test data. Subsets of the training set with varying sizes will be used to train the estimator and a score for each training subset size and the test set will be computed. Afterwards, the scores will be averaged over all k runs for each training subset size.

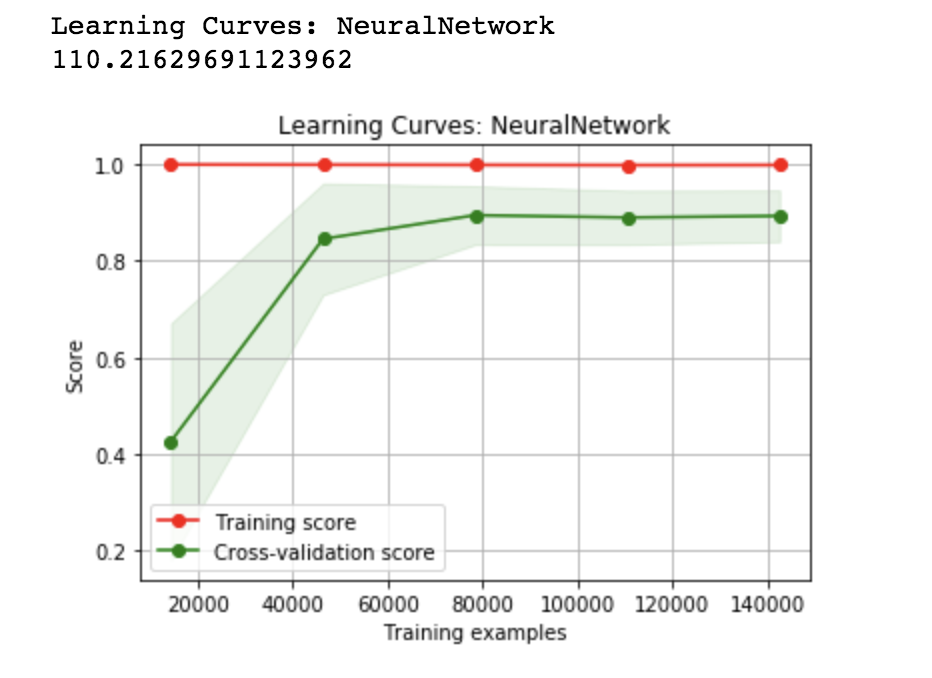
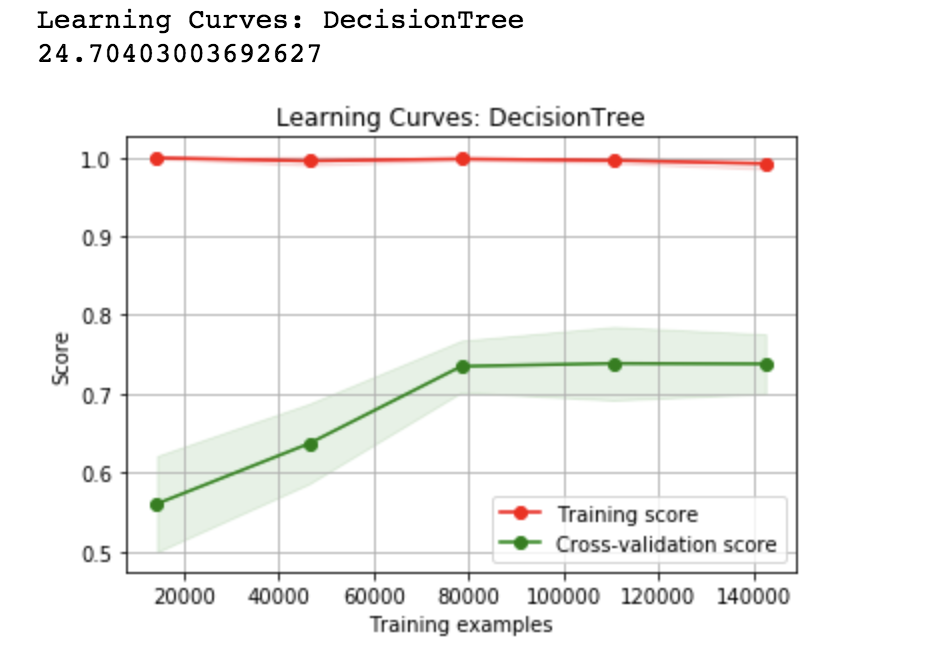
On the other hand, a complexity curve demonstrates how accuracy of a learning algorithm changes as a result of the tuning of hyper-parameters. In my implementation, I adopted the validation\_curve() method from the scikit-learn package, which determines training and test scores for varying parameter values, and compute scores for an estimator with different values of a specified parameter. This is similar to grid search with one parameter. However, my implementation will also compute training scores, in addition to being a plotting utility.

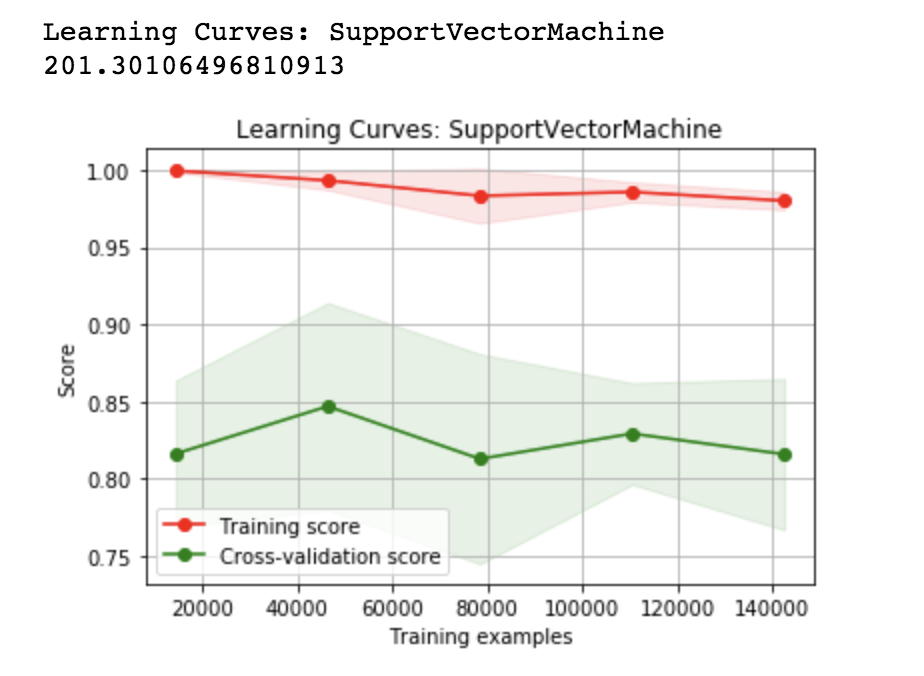
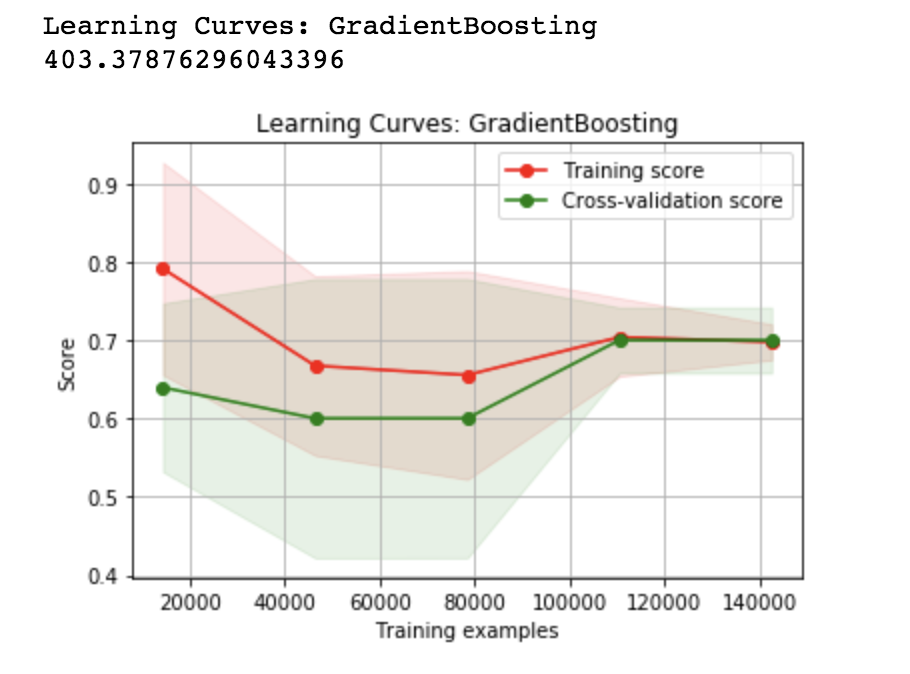
## Freddie Mac loan-level dataset

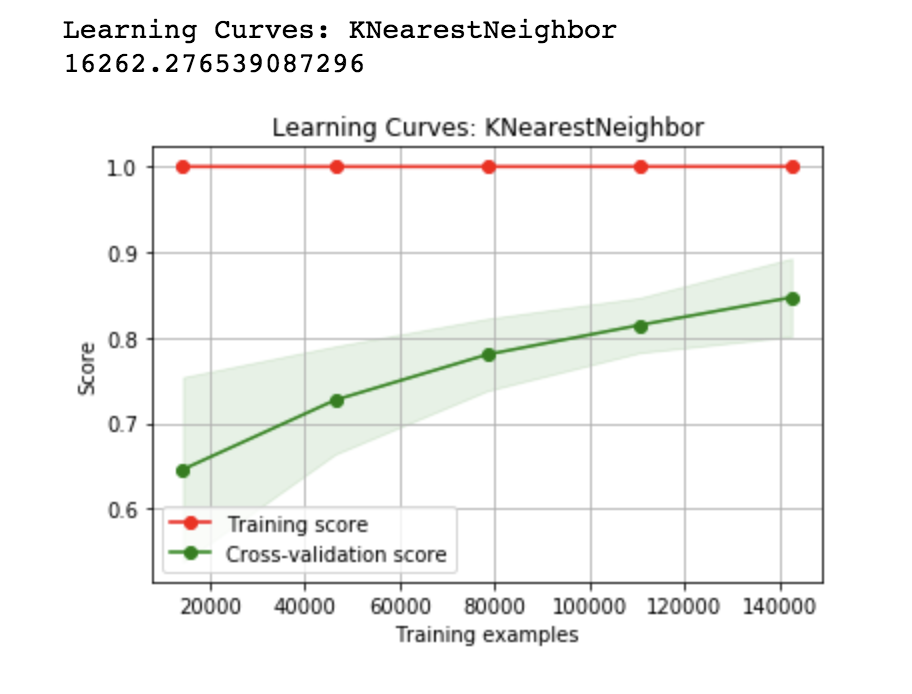
A significant challenge of the Freddie Mac dataset is that it is severely imbalanced, as it is much more commonplace that a mortgage loan is serviced on time as opposed to be in default. In my implementation, I had balanced the positive and negative classes whenever possible. Without balancing, the skew between positive and negative classes might have been so severe as to pose a challenge to any learning algorithm.

The graph is listed below:

### Learning curve analysis





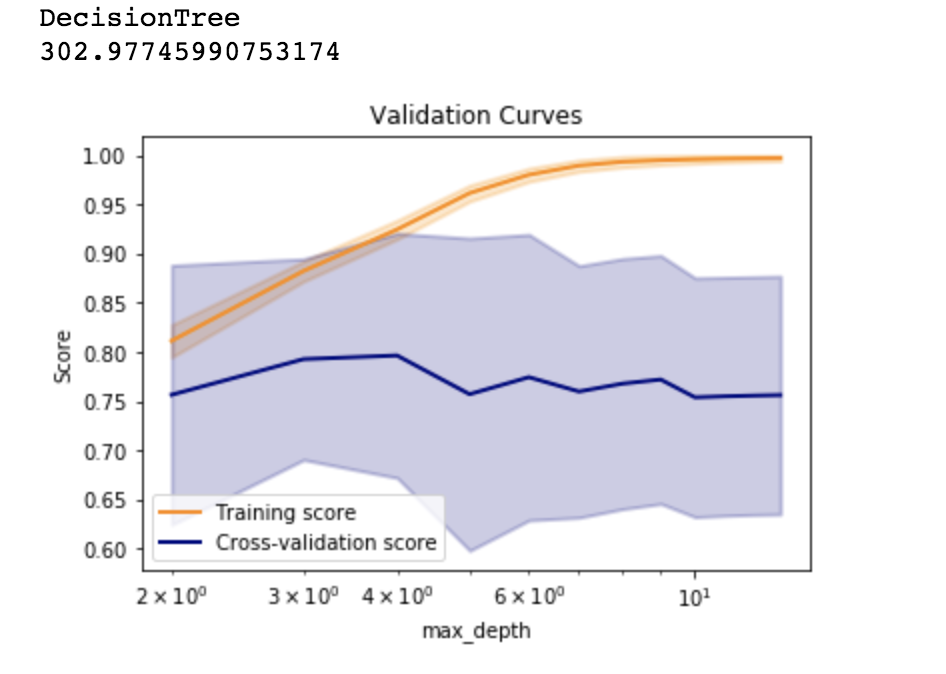
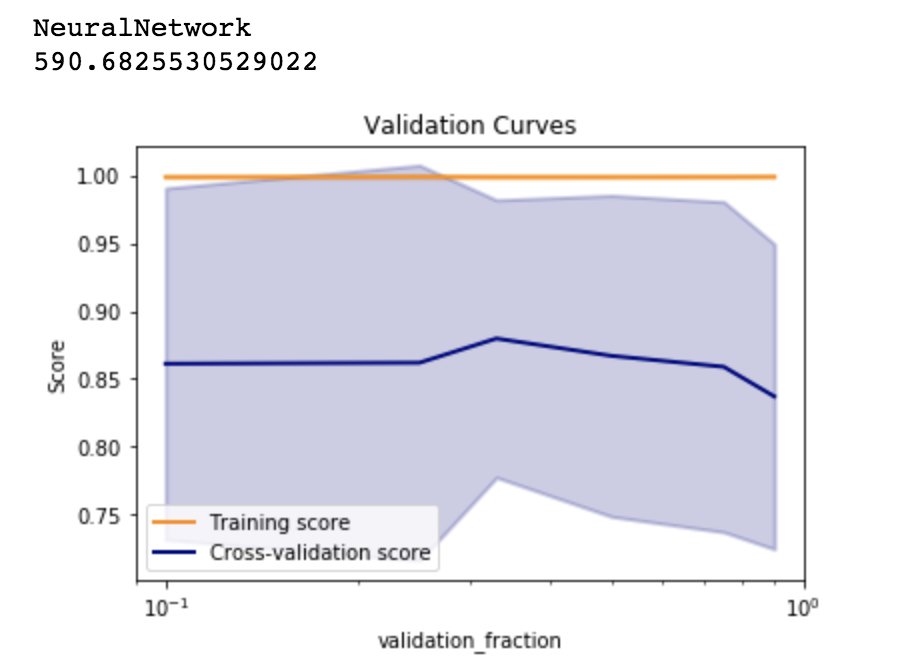


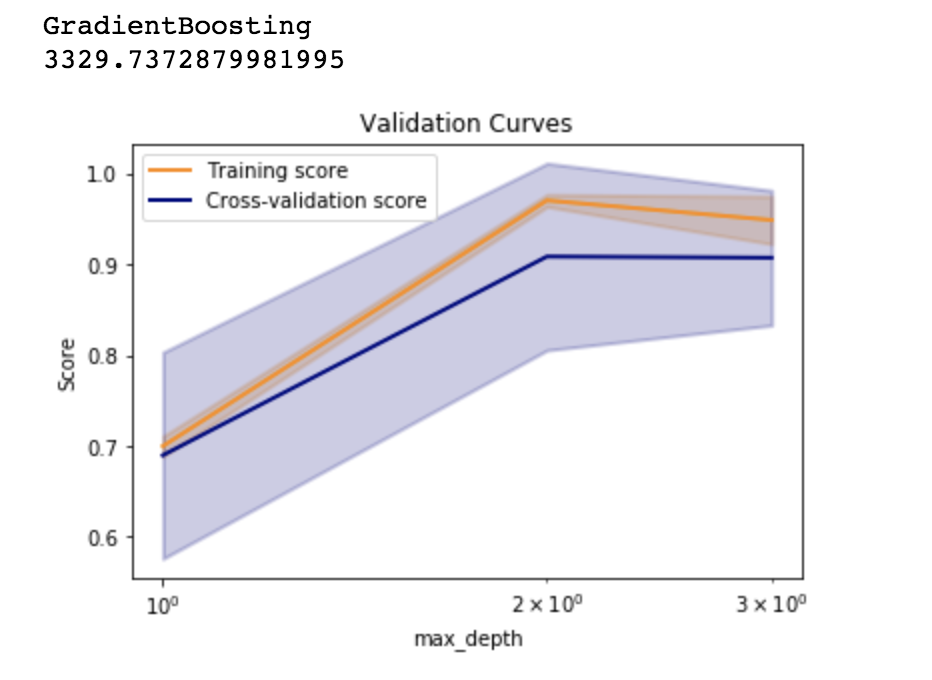
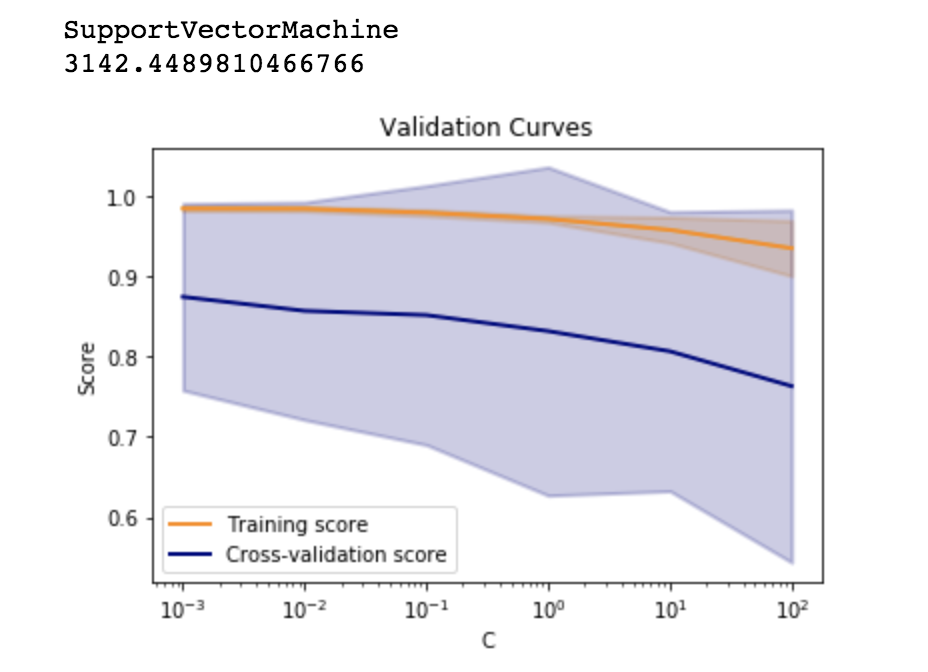
The various classifiers have dramatically different time costs when performing the learning curve and complexity curve valuation. Indeed, different classifiers have different training time – for example, KNN models would have very little time devoted in training, whereas all the other classifiers considered in this implementation have non-nominal training time. More concretely, in terms of comparing the learning curve analysis, the required wall time is increasing in the order of Decision Tree < 5 x Neural Network < 2 X Support Vector Machine < 2 Gradient Boosting < 40 x KNN.

It is also true that in general, the larger the size of the training set, the higher the area under the ROC curve is for the cross-validation set. This holds true for all classifiers. However, the degree to which the area under the ROC curve is maximized varies depending on the classifier in question. Across many runs, the classifiers that yield the largest area under the ROC curve is listed (in decreasing order): Neural network > KNN (k = 5) > Support Vector Machine > Decision Tree > Gradient Boosting Classifier (k=100).

### Complexity analysis

The various classifiers have dramatically different time costs when performing the learning curve and complexity curve valuation. In terms of comparing the complexity curve analysis, the required wall time is increasing in the order of Decision Tree time < 2 x Neural Network < 5 x Support Vector Machine < 1 x Gradient Boosting < 3 x Support Vector Machine < 17 x KNN.



Remarkably, every one of the five classifiers that we had deployed received a tremendous amount of gain in performance from tuning the model hyper-parameter. The gain in performance is defined as the increase in area under the Receiver Operating Characteristic curve. More specifically, the performance gain was due to tuning the following hyper-parameters:

* Approx. 5% gain based on tuning the maximum depth of decision tree on a decision tree classifier: a pre-pruning method, as opposed to rule post-pruning. The optimal depth of tree for this implementation is 4. The deeper the decision tree is fit, the more likely that training set has been over-fitted and the generalization power therefore decreases.
* Approx. 5% gain on tuning the validation fraction on a neural network classifier
* Approx. 20% gain on tuning the maximum depth of gradient boosting classifier
* Approx. 10% gain on tuning the C on the support vector machine classifier. The value of C controls the smoothness of decision boundary while making correct classifications as much as possible.
* Approx. 5% gain on tuning the value of k in KNN classifier

Among the five classifiers being considered, neural network classifier results in the best performance, defined as the largest area under the ROC curve, at approx. 90%. Considering the required training time (compared above), neural network classifier has the optimal combination of performance and training time – not only it has the highest performance based on our valuation metric, it is consistently the second least model in terms of training time complexity of all five classifiers in question.

## 

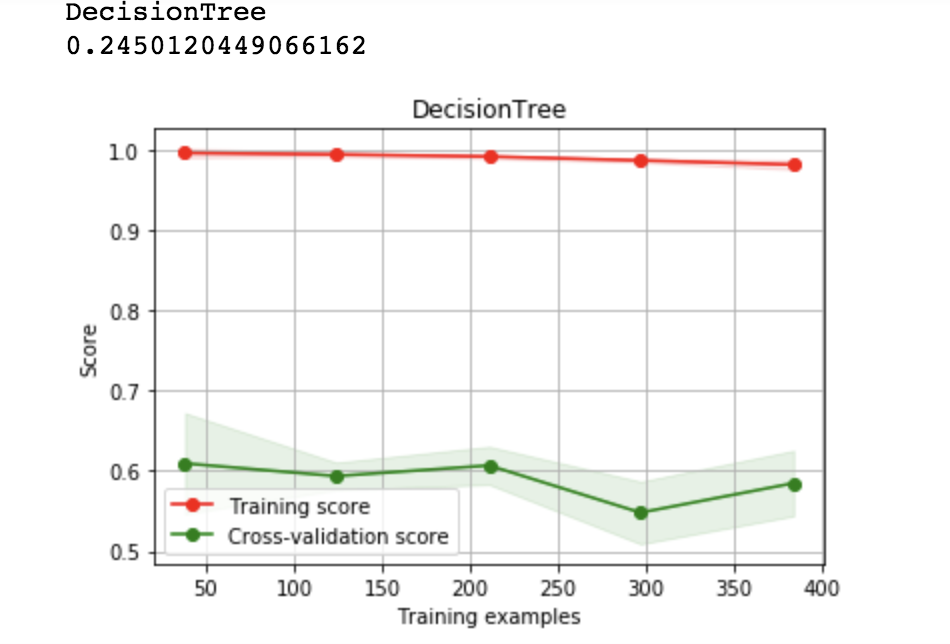
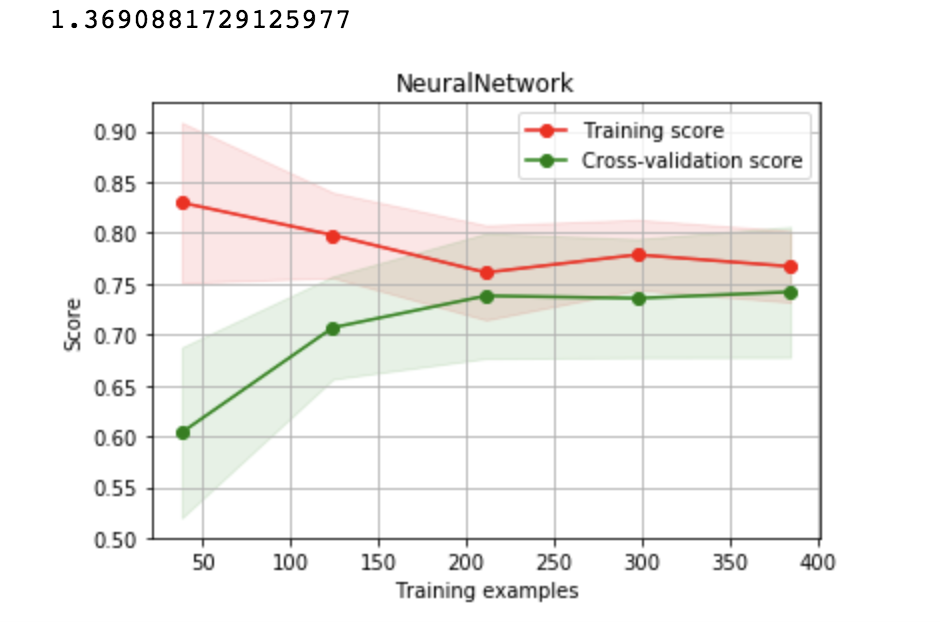
## Blood donation prediction dataset

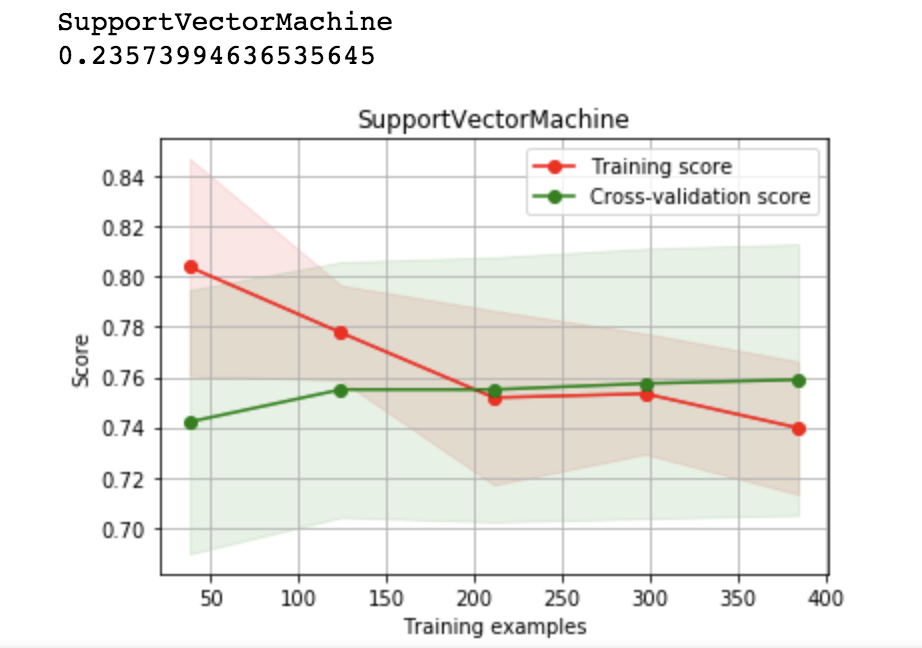
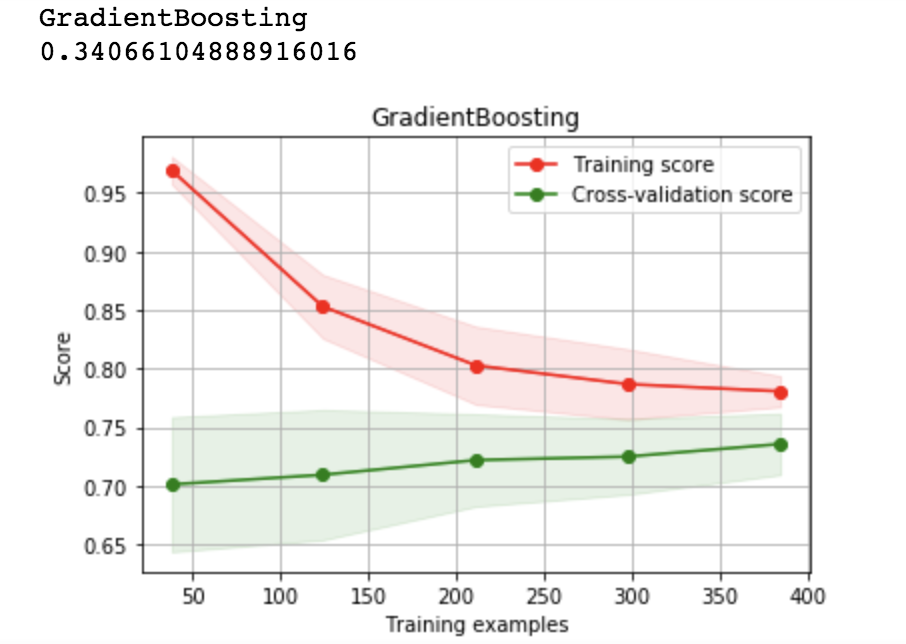
Contrary to the Freddie Mac dataset, the challenge with the Blood Donation dataset is its relative small size and limited feature space.

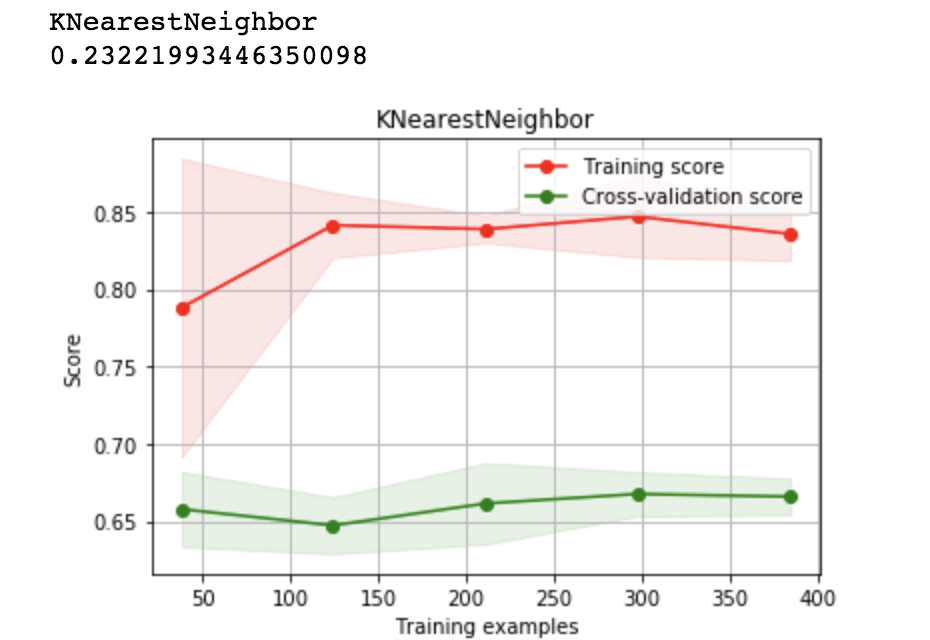
### Learning curve analysis

Due to its relative small size, the various classifiers have very similar time costs when performing the learning curve and complexity curve valuation. More concretely, in terms of comparing the learning curve analysis, the required wall time is increasing in the order of KNN < Decision Tree < Gradient Boosting < Support Vector Machine < Decision Tree < Gradient Boosting < Gradient Boosting < 4 x Neural Network.

However, it is noteworthy that even though every classifier in question had very little training time, the performance was not great. This result can be attributed to the small size of data, and the fact that due to the small size, the classifiers encountered notable difficulties in distinguishing signal from noise. In particular, the classifiers that yield the highest area under the ROC curve is list in decreasing order: Support Vector Machine > Neural Network > Gradient Boosting > KNN (k=5) > Decision Tree.

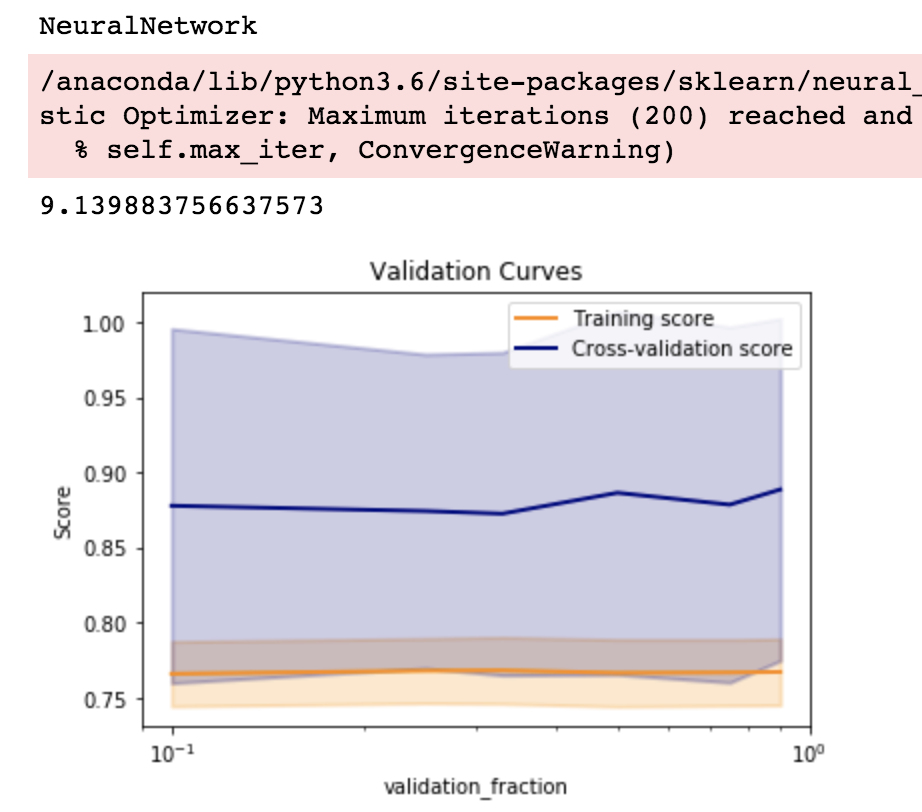
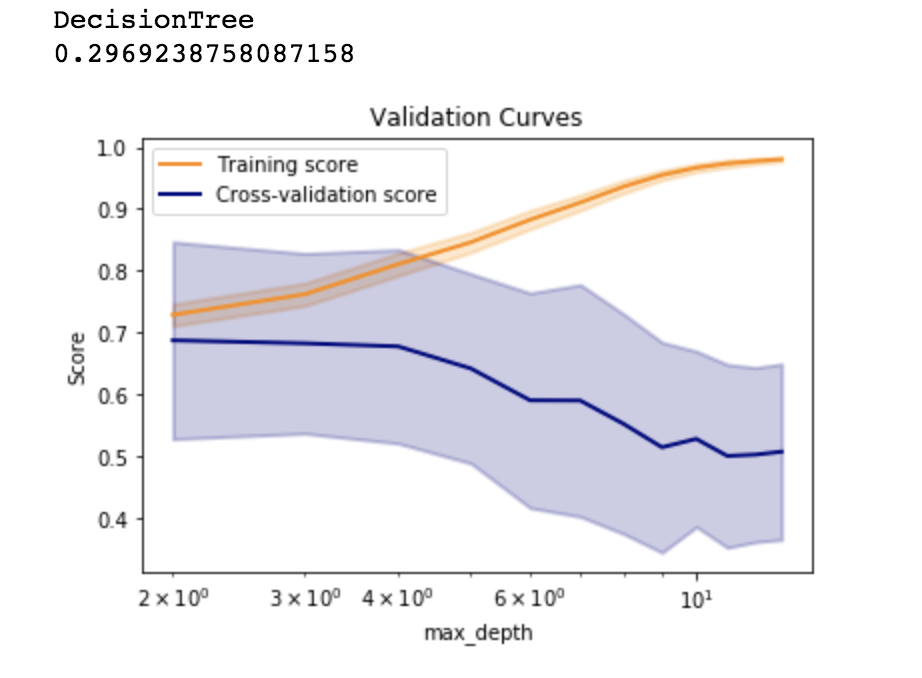
../../../../../../../Desktop/Screen%20Shot%202017-09-24%20at%206.

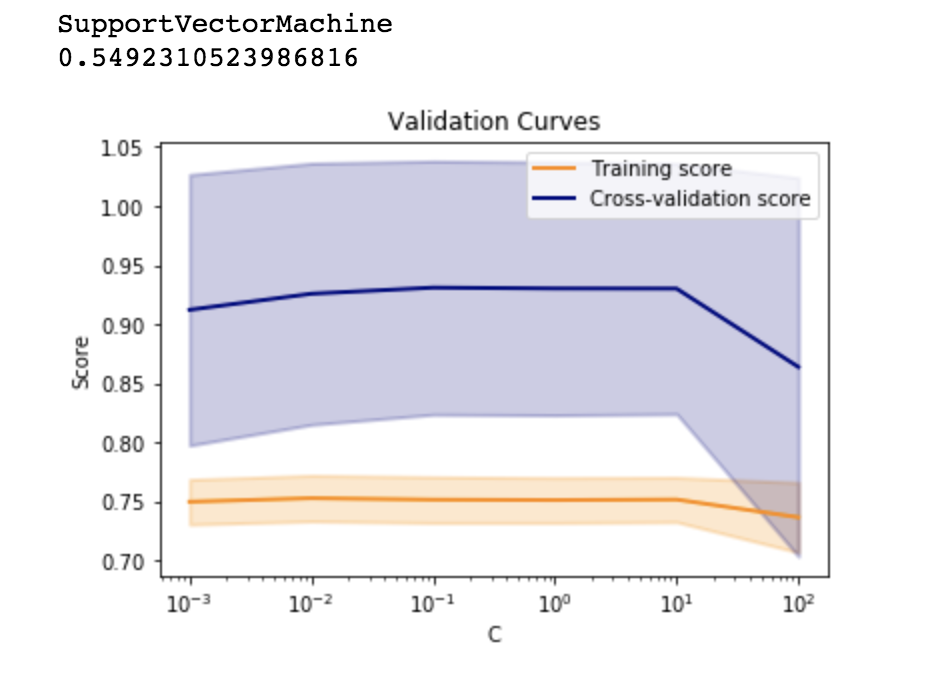
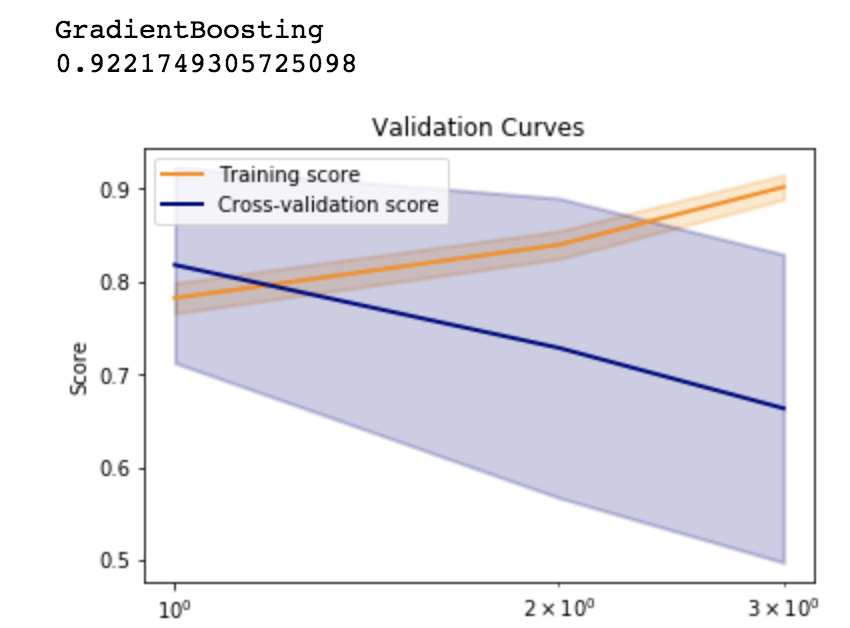


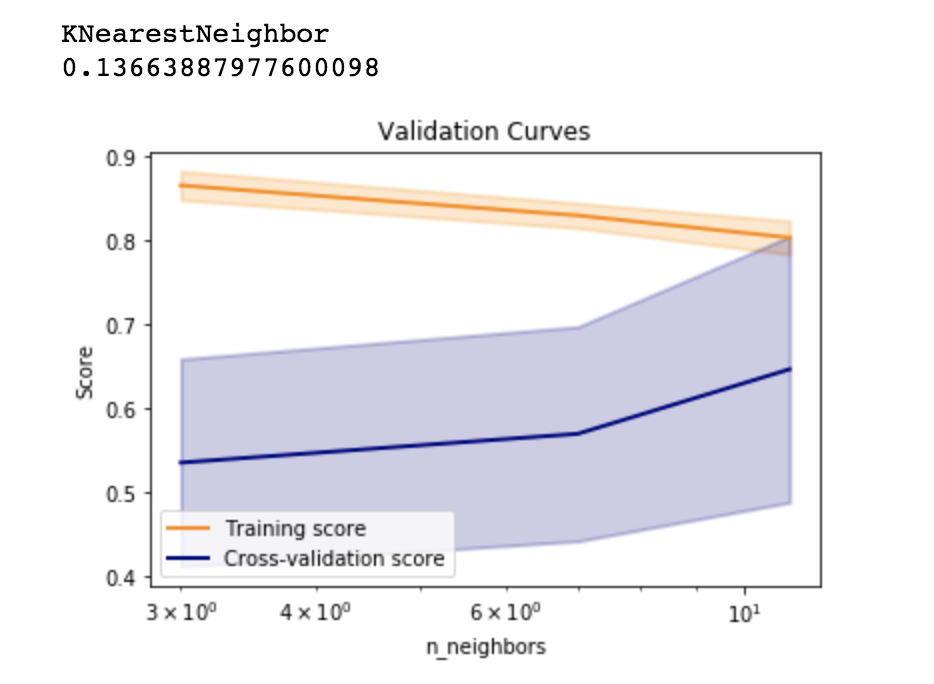


### Complexity analysis

Due to its relative small size, the various classifiers have very similar time costs when performing the learning curve and complexity curve valuation. In terms of comparing the complexity curve analysis, the required wall time is increasing in the order of KNN < 3 x Decision Tree < Gradient Boosting < Support Vector Machine < 4 x Neural Network.







Surprisingly, given the small set of data, with the exception of Neural Network, every one of the classifiers in questions have dramatic gain in performance from tuning the model hyper-parameter. The gain in performance is defined as the increase in area under the Receiver Operating Characteristic curve. More specifically, the performance gain was due to tuning the following hyper-parameters:

* Approx. 20% gain based on tuning the maximum depth of decision tree on a decision tree classifier: a pre-pruning method, as opposed to rule post-pruning. The optimal depth of tree for this implementation is 4. The deeper the decision tree is fit, the more likely that training set has been over-fitted and the generalization power therefore decreases.
* No gain on tuning the validation fraction on a neural network classifier
* Approx. 20% gain on tuning the maximum depth of gradient boosting classifier
* Approx. 5% gain on tuning the C on the support vector machine classifier. The value of C controls the smoothness of decision boundary while making correct classifications as much as possible.
* ApproX.15% gain on tuning the value of k in KNN classifier

Among the five classifiers being considered, Support Vector Machine and Gradient Boosting Classifiers result in the best performance, defined as the largest area under the ROC curve, at approx. 75%. Considering the required training time (compared above), Gradient Boosting has shorter required training time compared to Support Vector Machine (although both are comparable due to the small dataset).