Table of Contents

[Abstract 2](#_Toc493608952)

[Introduction 3](#_Toc493608953)

[Two Classification Problems Considered 3](#_Toc493608954)

[Freddie Mac mortgages loan-level performance data 3](#_Toc493608955)

[Blood donation prediction 4](#_Toc493608956)

[Training and Test Results 5](#_Toc493608957)

[Freddie Mac loan-level dataset 5](#_Toc493608958)

[Learning curve analysis 5](#_Toc493608959)

[Complexity analysis 6](#_Toc493608960)

[Blood donation prediction dataset 7](#_Toc493608961)

[Learning curve analysis 7](#_Toc493608962)

[Complexity analysis 8](#_Toc493608963)

[Analysis of models 9](#_Toc493608964)

[Freddie Mac loan-level dataset 9](#_Toc493608965)

[Blood donation prediction dataset 9](#_Toc493608966)

# Abstract

##TODO: There is no free lunch in data science. Flesh this out more.

# 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 and analyzed.

# Two Classification Problems Considered

## Freddie Mac mortgages loan-level performance data

Fannie Mae and 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.”

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, motivated in large part by losses on loans that they guaranteed, so I figured there must be something interesting in the loan-level data.

I decided to dig into this dataset – 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.

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 dataset, I am more interested in finding out which of the 5 classifiers work the best (without resorting to deep neural networks):

* Decision tree with pre-pruning
* Neural network
* Boosting
* SVM
* KNN

This is particularly feasible given the large size of my dataset (> 200,000 mortgages).

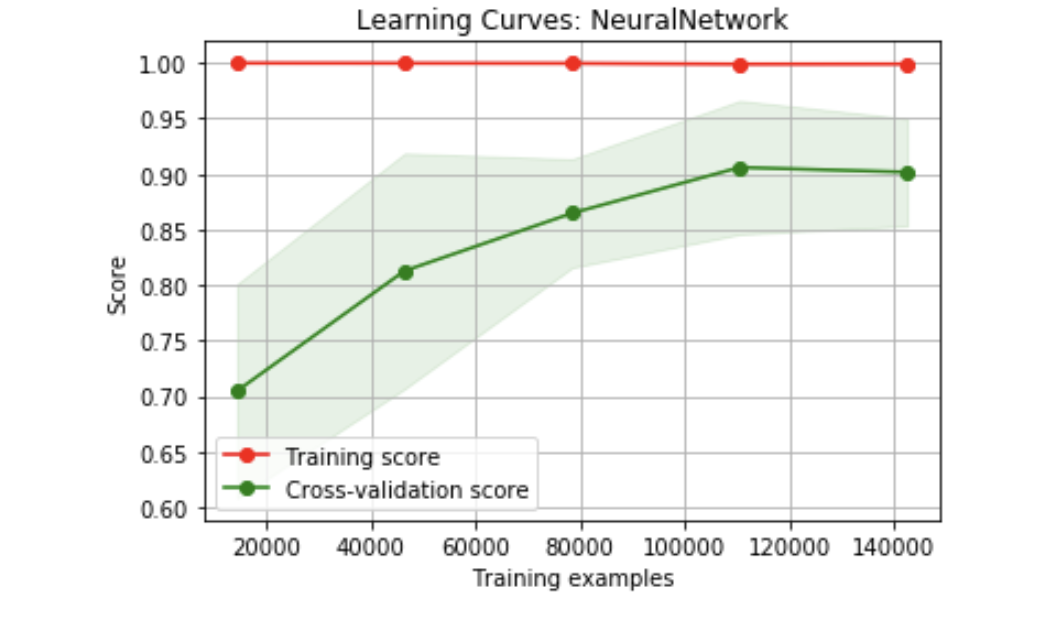
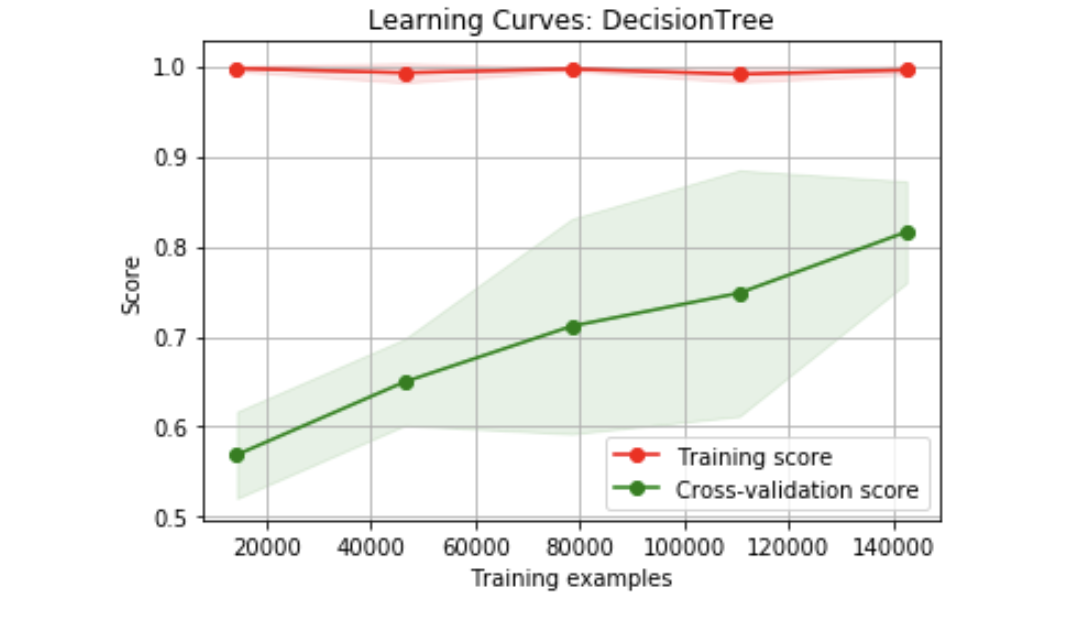
## Blood donation prediction

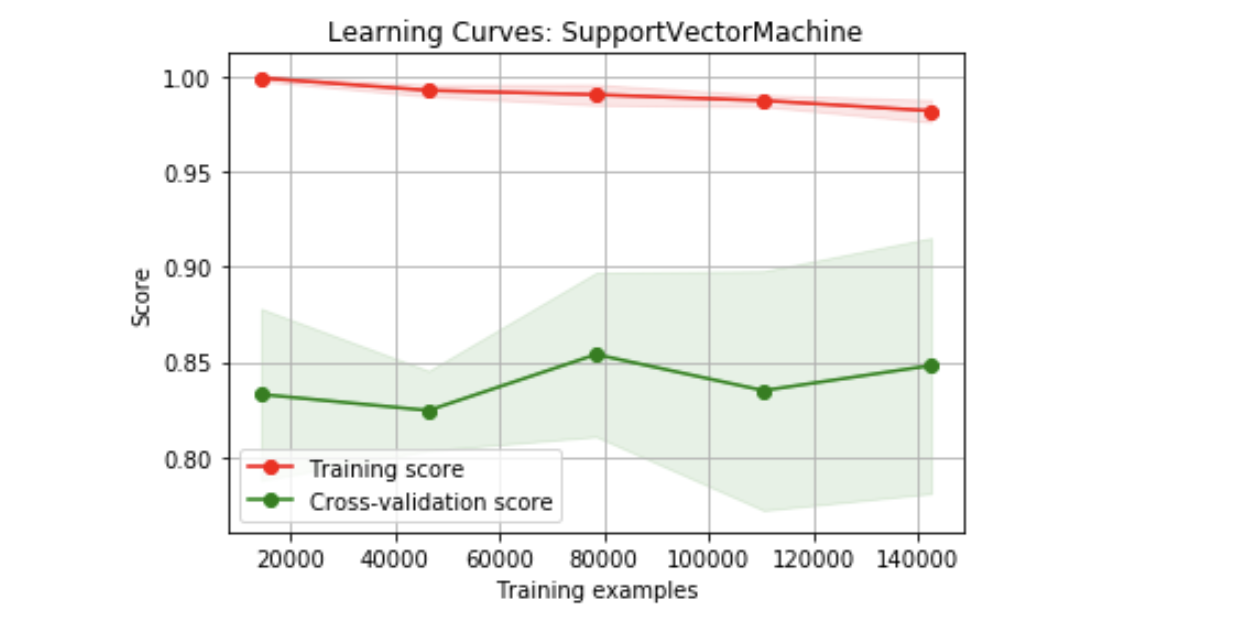
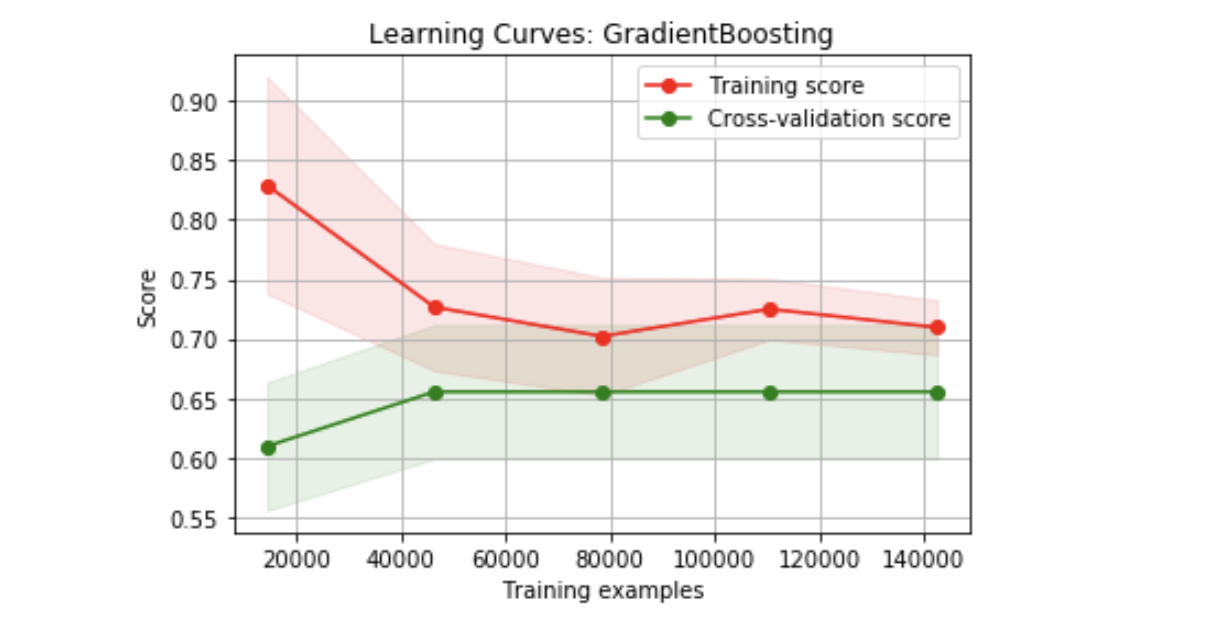
The [UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/index.html) is a great repository of data science-related projects. Given my interest in discovery our mission, we're interested in predicting if a blood donor will donate within a given time window.

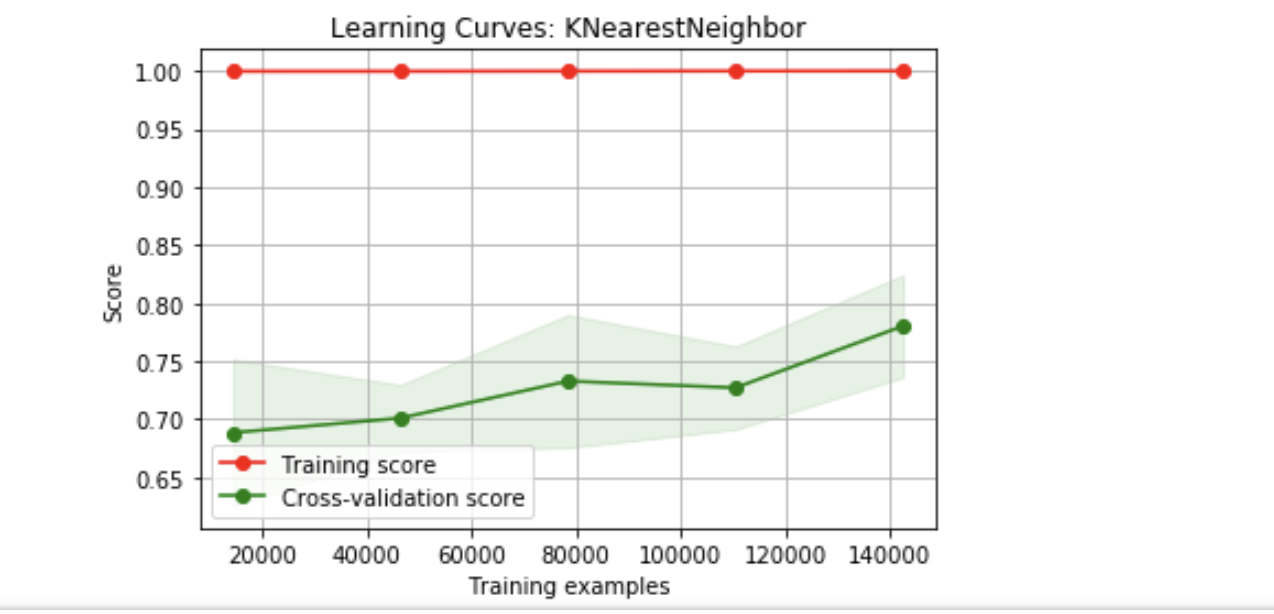
# Training and Test Results

## Freddie Mac loan-level dataset

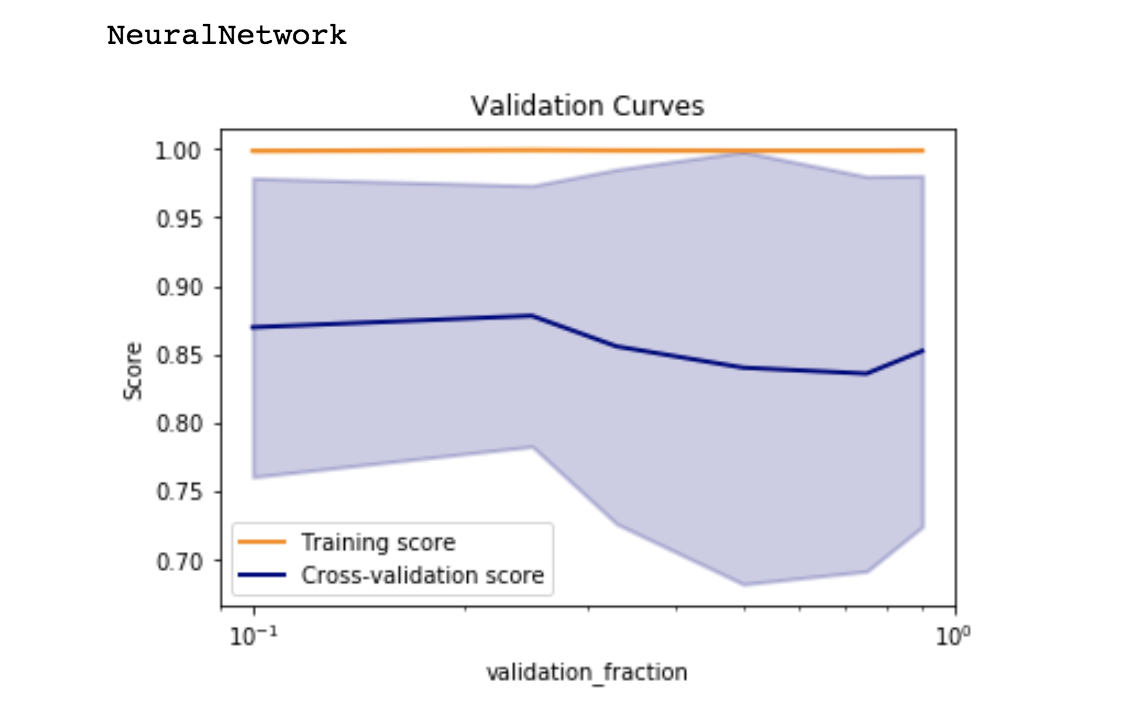
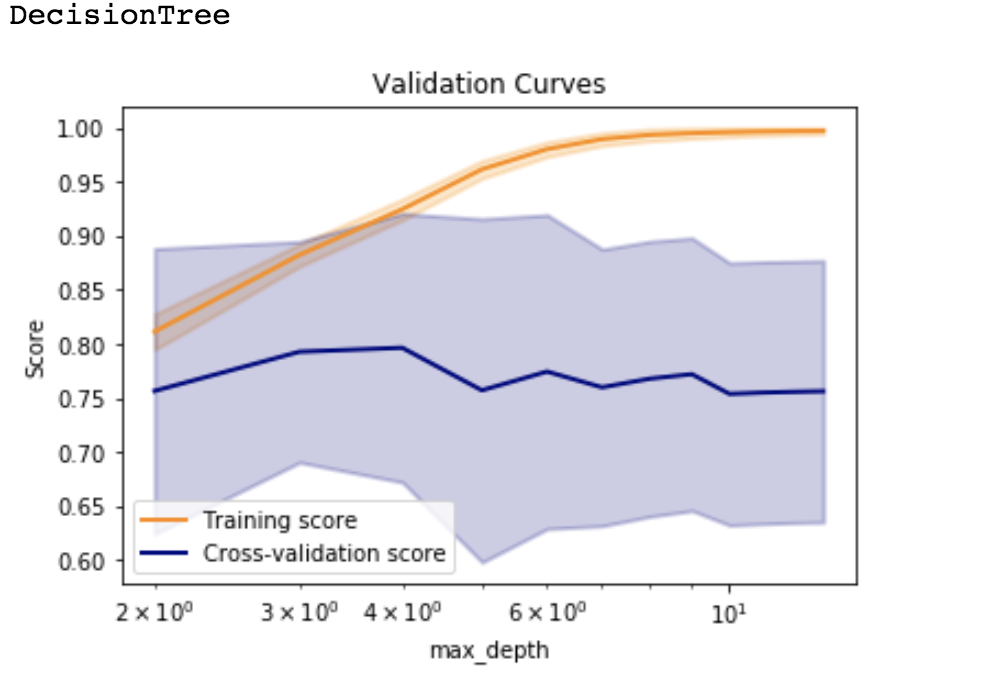
### Learning curve analysis

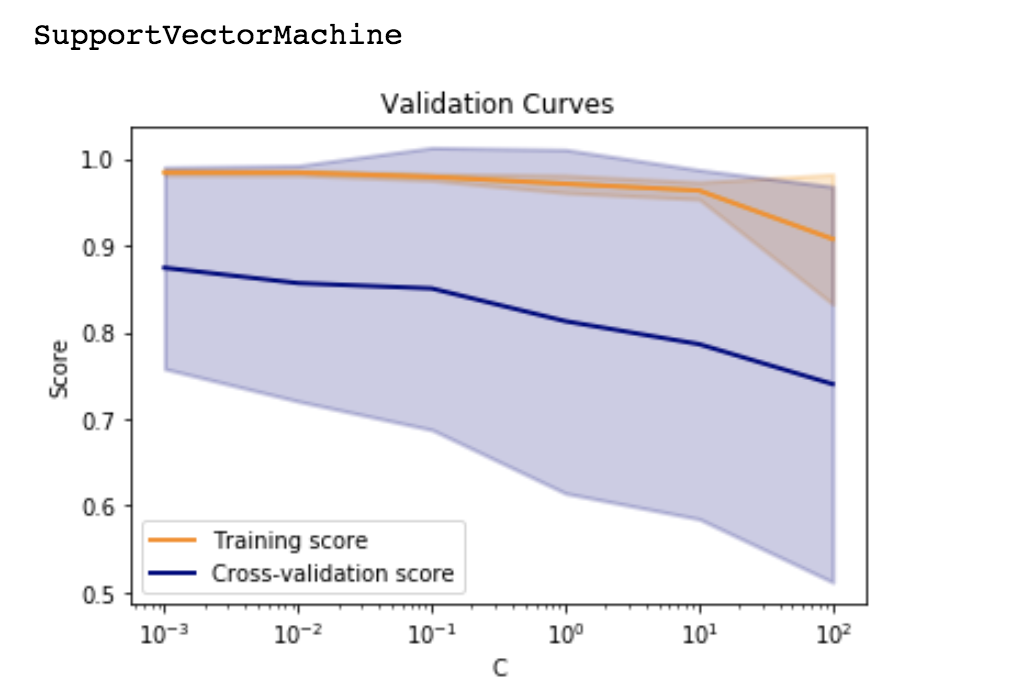
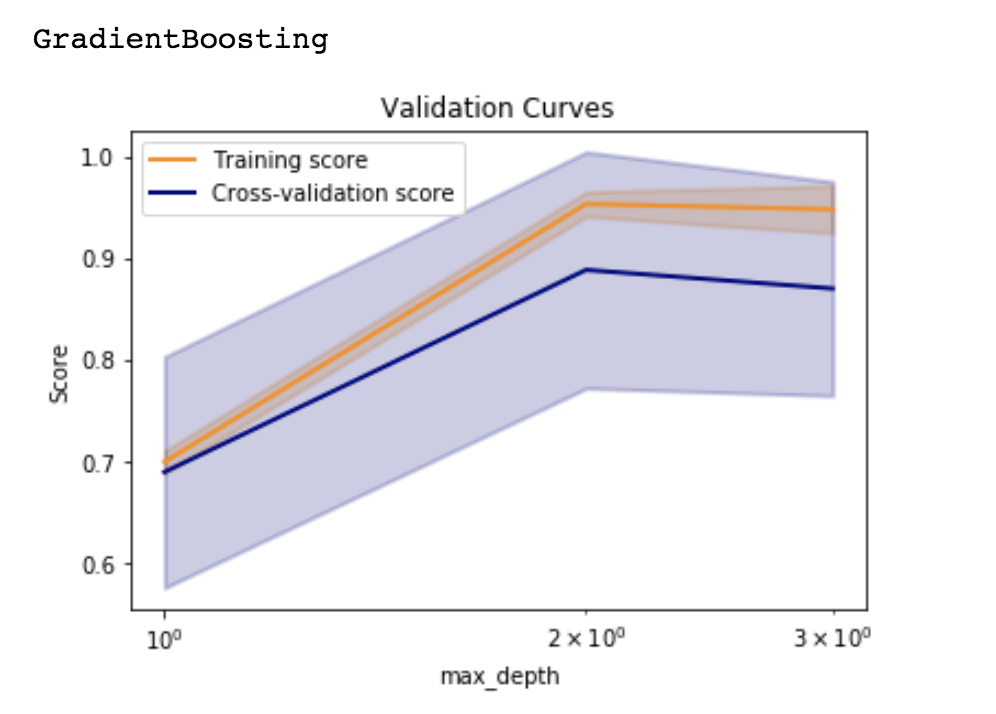


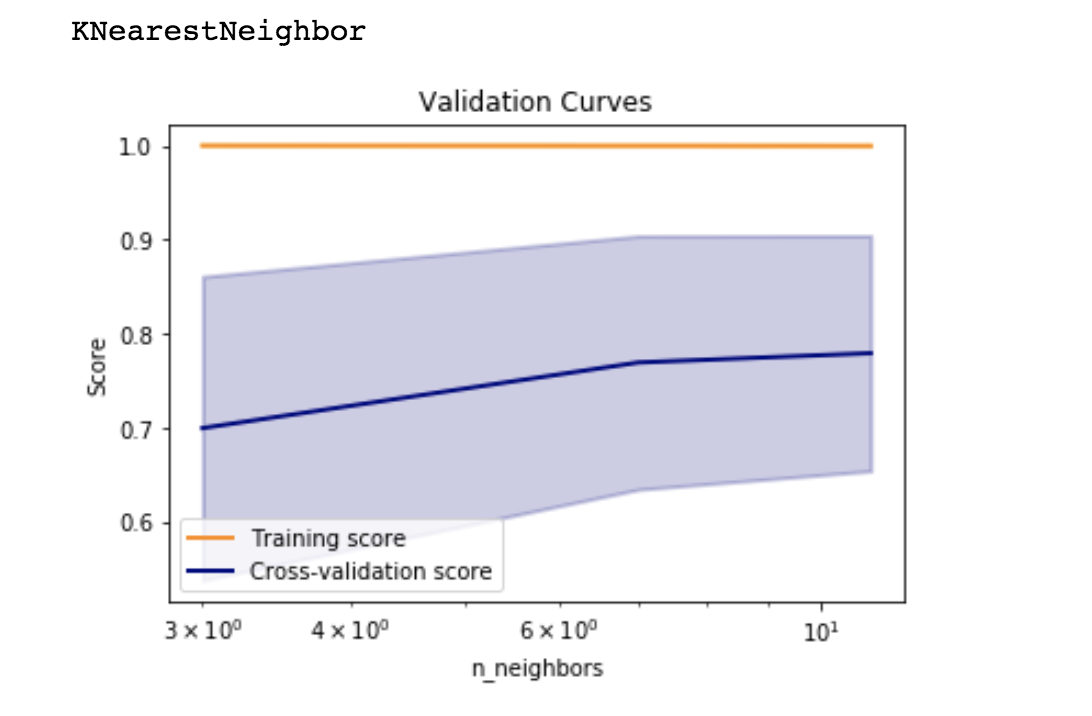




### Complexity analysis

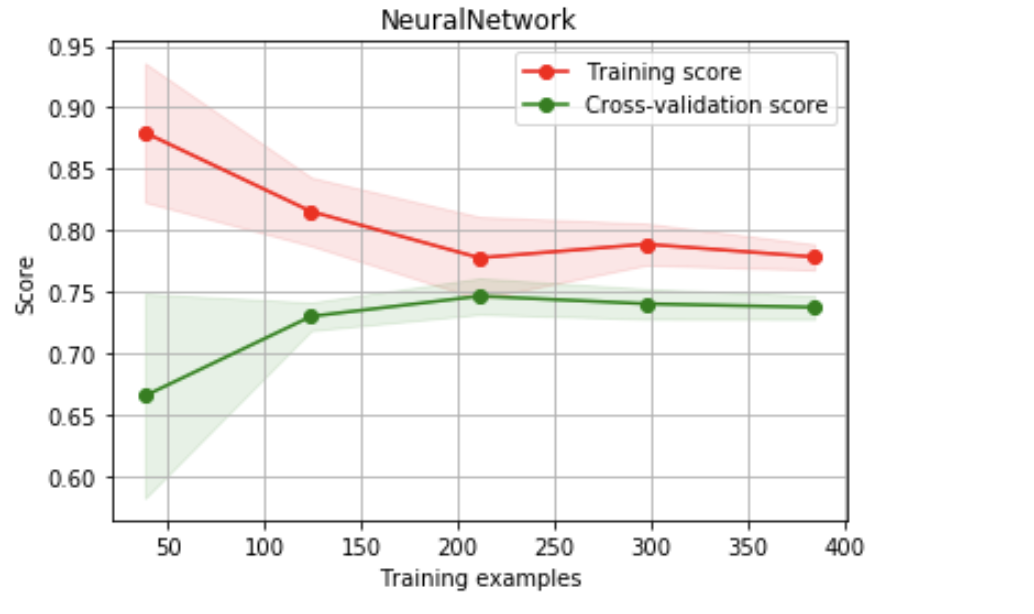
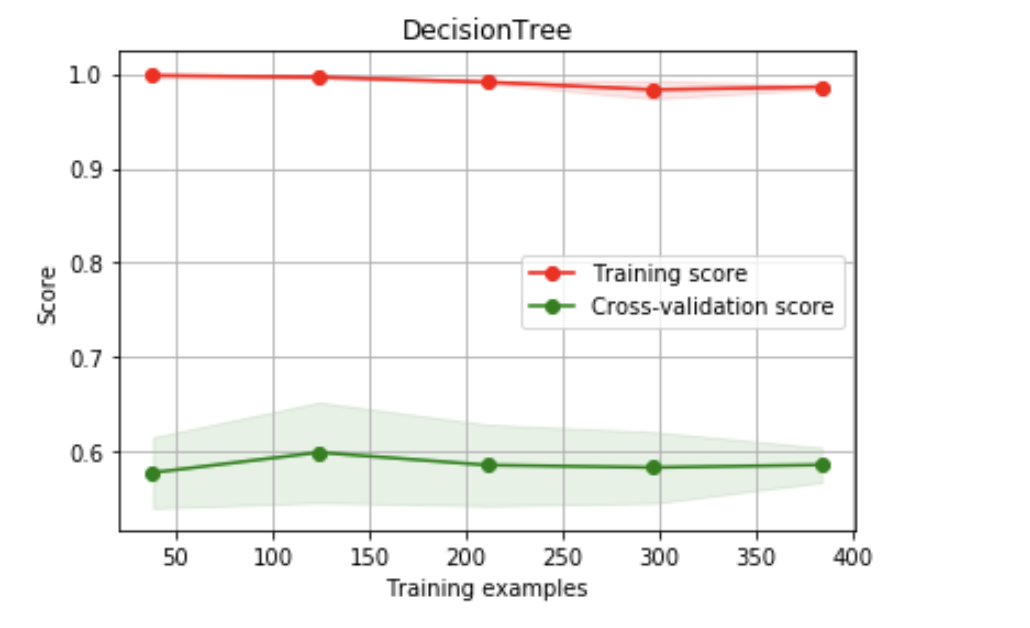


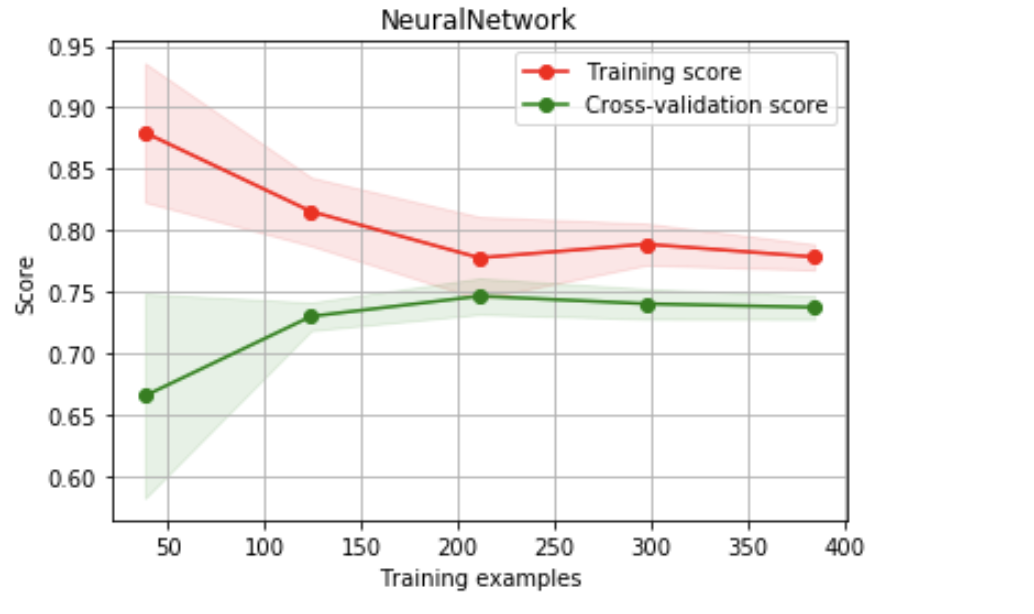
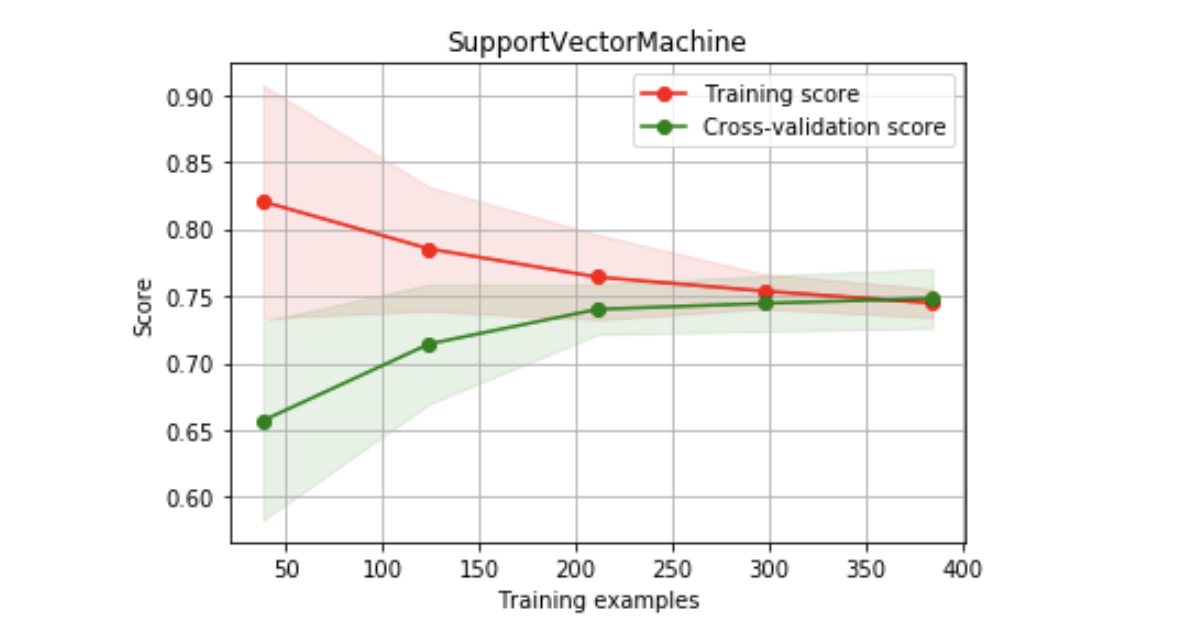


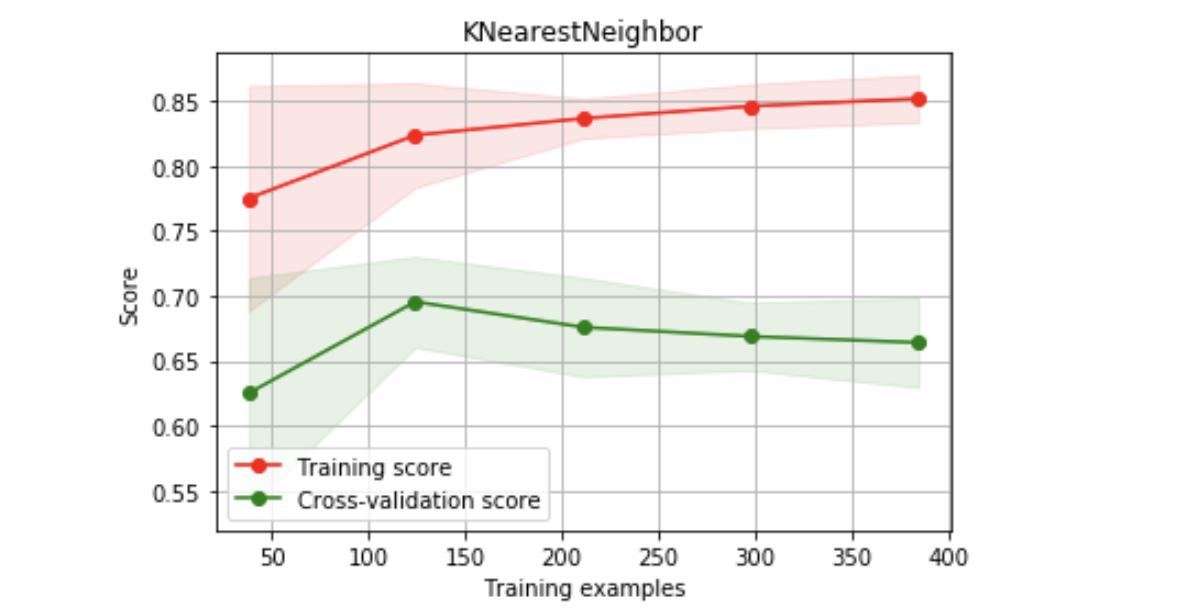


## Blood donation prediction dataset

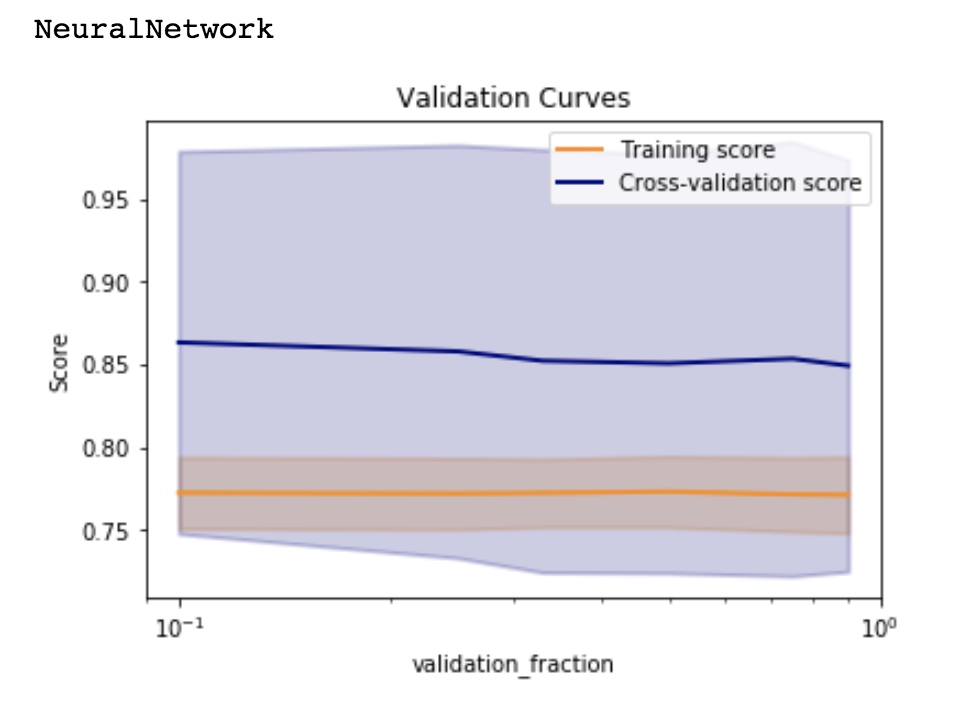
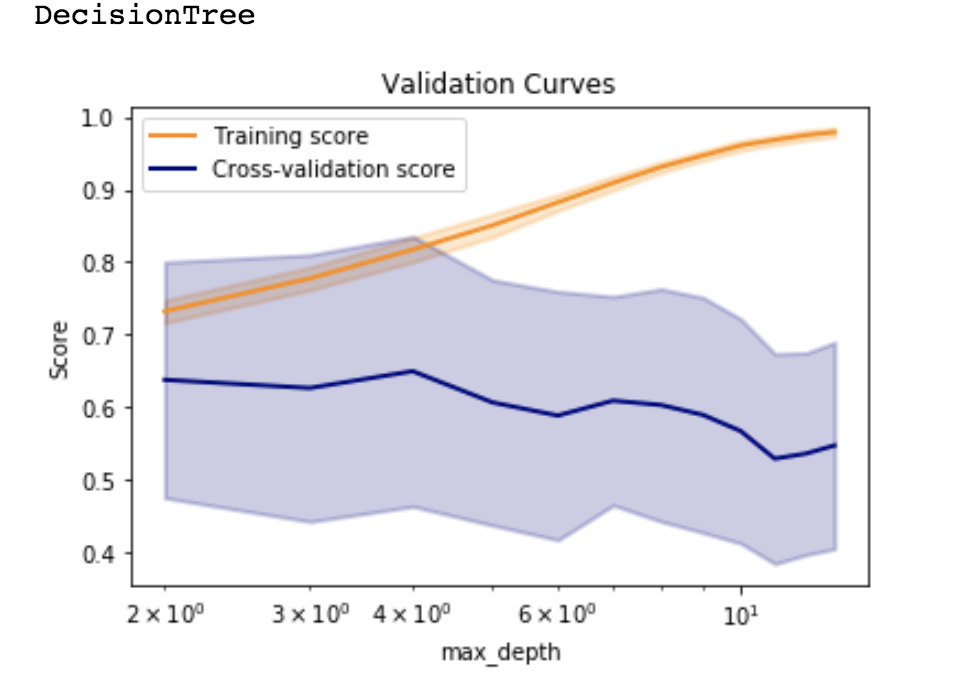
### Learning curve analysis

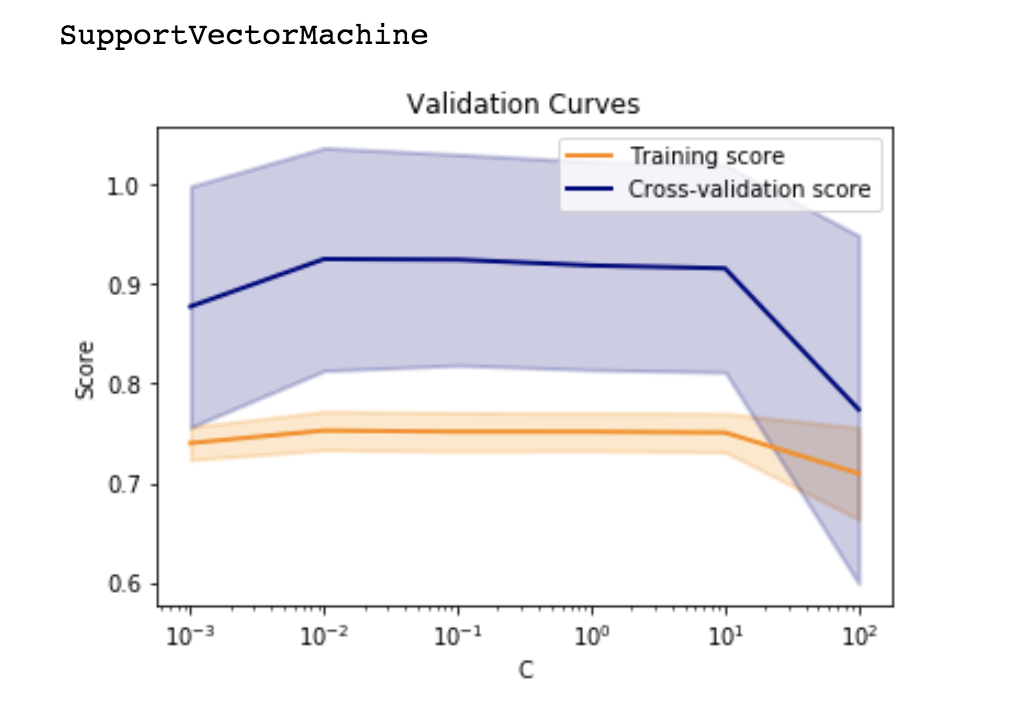
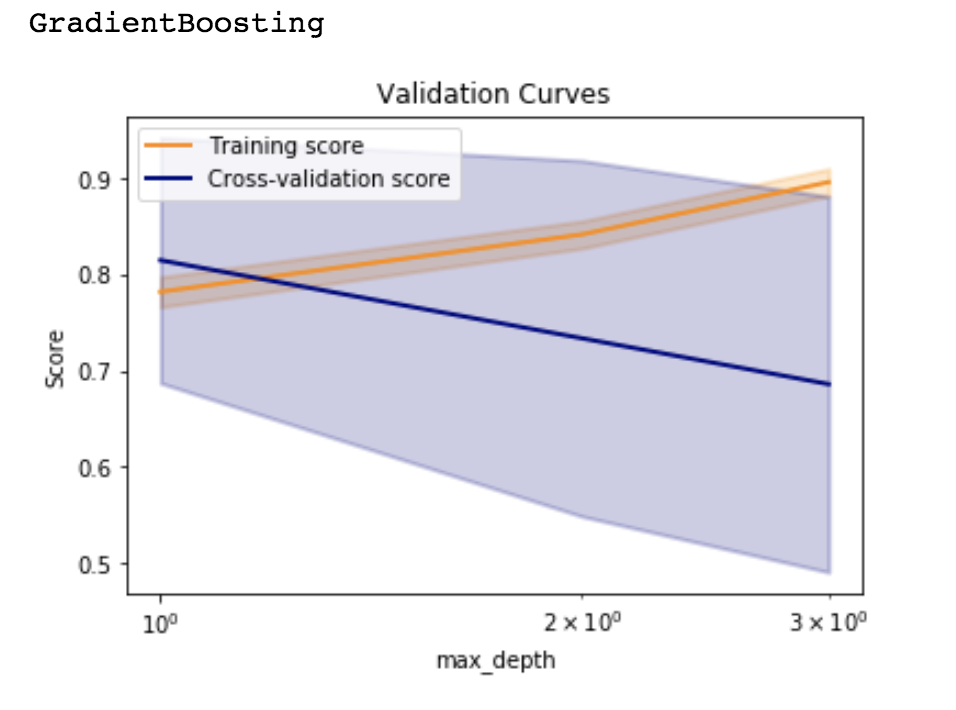


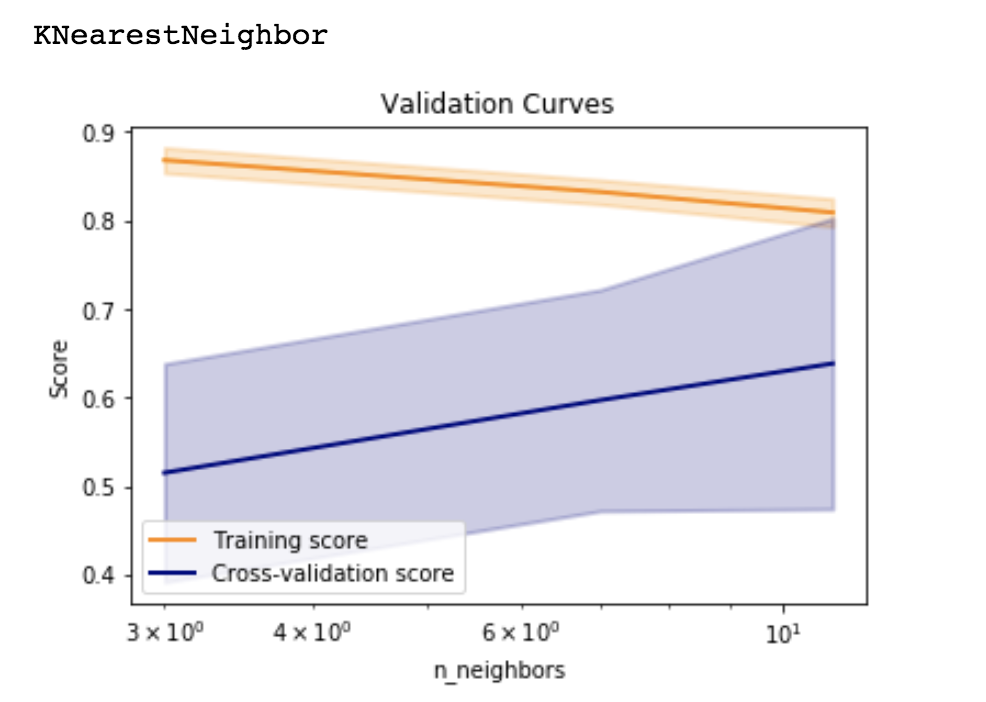




### Complexity analysis







# Analysis of models

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

My analysis showed that as the training size increases, so is the roc auc.

In addition, my analysis showed that …

The Freddie Mac dataset 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.

## TODO: how fast are the learning algorithms in terms of wall time?

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….. 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 time < 2.5 x Gradient Boosting < 2 x support vector machine < 10 x KNN.

Remarkably, with the exception of support vector machine, every one of the five classifiers that we have deployed, an approximate 5% gain in performance can be obtained from tuning the model hyper-parameter. For Support vector machine, the performance gain is much greater than 5%. 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:

* 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 overfit and the generalization power therefore decreases.
* 5% gain on tuning the validation fraction on a neural network classifier
* 15% gain on tuning the maximum depth of gradient boosting classifier
* 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.
* 5% gain on tuning the value of k in KNN classifier

## TODO: which algorithm are the best? What is best?

## Blood donation prediction dataset

My analysis showed that as the training size increases, so is the roc auc.

In addition, my analysis showed that

* why did you get the results you did? Compare and contrast the different algorithms. What sort of changes might you make to each of those algorithms to improve performance? How fast were they in terms of wall clock time? Iterations? Would cross validation help (and if it would, why didn't you implement it?)? How much performance was due to the problems you chose? How about the values you chose for learning rates, stopping criteria, pruning methods, and so forth (and why doesn't your analysis show results for the different values you chose?)? Which algorithm performed best? How do you define best? Be creative and think of as many questions you can, and as many answers as you can.