HW1: Supervised Learning

Name: Jiun-Yu Lee GTID: 903435223

Environment

In this assignment, I used python 3.7.2 with scikit-learn and matplotlib as my coding environment. To reproduce the result a proper version of python and a least the two packages listed above should be installed. Use

```
pip install sklearn
pip install matplotlib
```

to install the packages. Please see README.txt file in the submission to get detail information of how to run my code.

Algorithm

- **Decision Tree:** *DecisionTreeClassifier* module in sklearn is used to be my implementation of the decision tree algorithm. It support multiple pruning methods such as minimum number of samples required at a leaf node or the maximum depth of the tree. Here, the **maximum depth of the tree** is chosen to be my pruning method. It also support two criterion for splitting the node, gini index and entropy. We will search both of the methods to see which one is best suitable for the classification problem.
- **Support Vector Machine:** *SVC* module in sklearn is used to train my support vector machine model. It support various predefined kernel we can use, such as *linear, polynomial, sigmoid, RBF*, etc. To be specific, ***linear, polynomial*, and *RBF*** kernel as well as other parameter such as ***gamma, degree*** of the polynomial kernel, and regularization parameter ***C*** is included in my experiment.
- **K-nearest Neighbors:** I used *KNeighborsClassifier* module in sklearn to do my experiment in this assignment. Different k value will be experimented, as well as different distance metric (manhattan_distance (**l1**) and euclidean_distance (**l2**), specifically.) Also, in addition to treated all k-nearest neighbors equally important, I try to weight the vote by the inverse of the distance of a node so that a node will contribute more if it is closer to the test case.
- **Boosting:** *AdaBoostClassifier* module is used to implement my boosting version of decision tree. To be able to see the influence of boosting without interfering by the factor of decision tree, and reduce overfit bringing by a not weak enough classifier, I fixed the max depth of the decision tree to one (that is, **decision stump**.) Other parameter like the **number of estimators** and the **learning rate** will be explored to find the best performance for the following classification problem.
- **Neural Network:** There are many different structure and framework of neural network is emerging in deep learning field, such as CNN, RNN, GAN, and do on. However, in order to make a fair comparison between Neural Network and the other four algorithms in the above (e.g using CNN on

images or RNN on time series data is very much likely to perform better than the other four algorithm, which we already apply more domain knowledge on this particular algorithm), I only explored fully-connected network in this assignment. *MLPClassifier* module is used in our experiment to build the network. Here, I will explore different **network structure** (various combination layers) **activation functions** (*relu, tanh, logistic*), and **learning rate**. The solver is fixed to Adam optimizer since it is can converge more quickly than others (such as SGD).

Hyper-parameter Optimization:

Before applying each algorithm to classification problems, I will first do some hyper-parameter optimization for those algorithm on each dataset to gain a reasonable performance. My optimization algorithm is mainly based on **grid search**, which I used *GridSearchCV* module in sklearn to achieve it. The parameter search space for each algorithm is predefined, and the GridSearchCV will run exhaustedly search through all the parameter combination in the search space, perform cross validation, and return the best parameter set. For the cross validation, I used *StratifiedKFold* here to ensure the proportion of the classes in each folds is equal to the proportion in the entire dataset (since I actually have an imbalanced data set in the following classification problem) and divided the training data into 5 folds. After doing the grid search through the default parameter space. I will see the result (learning curve, final performance) and fine-tune it manually if necessary. The detail information about how the parameter space is defined can be seen in src/searchParameter.py

Problem 1: Credit Card Fraud Detection

For the first classification problem, I want to analyze a very common problem that may appear in the real-world dataset, imbalanced dataset. I choose the credit card fraud detection dataset on kaggle to analyze since it has about 285K samples but only about 500 of them are label as a fraud. This problem itself is a very interest because we can actual deal with the real world data and contribute to the real world. Moreover, Imbalanced dataset can happen in not only credit card fraud but also many other classification problem like cancer detection. Compare to balanced dataset, We will care much less about accuracy (we can simply classify all sample as negetive to achieve a high accuracy, but this is meaningless) but much more about the recall (if there is a credit card fraud or cancer, we don't want our classifier to miss it.) Therefore, I want to see how each of the five algorithm will perform on this kind of imbalanced dataset. My expectation is that AdaBoost my have a better result in this case since it has the ability to get the hard sample right by changing their weights, yet SVM or neural network may just treated lots of the positive as noise.

Data preprocessing:

The original dataset have a size of (285k, 30), in which it feature space is already reduced using PCA. Only around 500 of the samples is positive (0.17%), which is a very hard imbalanced data. Apply algorithm directly on this dataset is not likely to perform well anyway. In order to deal with this problem, I create another dataset but keeping all the positive data and randomly sample negative data from the original dataset until it have 5000 samples. With the new (5000, 30) dataset, the positive class proportion is now 10%, still an Imbalanced dataset, but more easily to deal with.

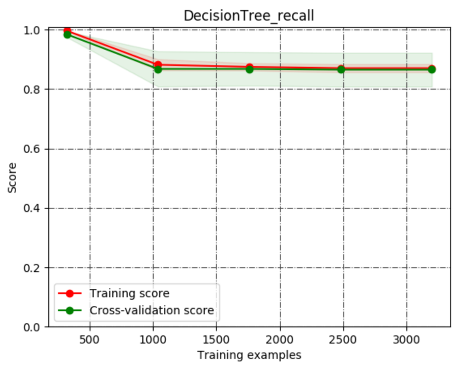
This dataset is then separated to training and testing data with 4:1 ratio. The training data is used to train and validate and the testing data is used only in the final test.

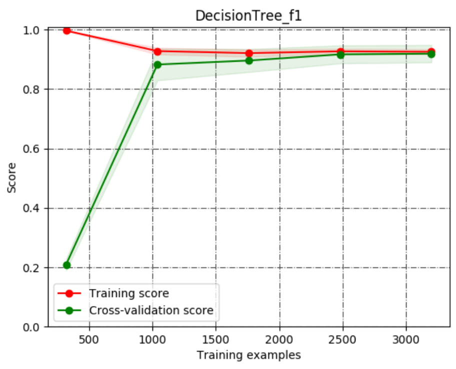
Metrics:

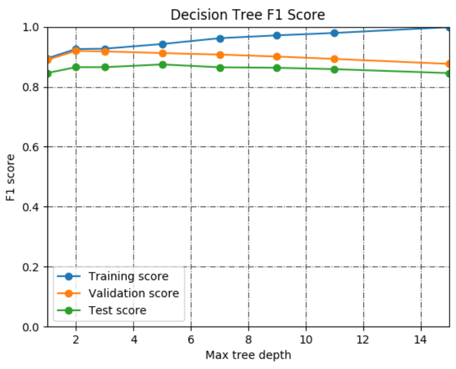
In this problem, I use F1 score to be the main metric to measure the performance of the model. As I mention above, the recall of the results is the most important thing I should consider in this kind of problem since I don't want my problem to misclassify any thing that may be positive. However, a model with a high false positive rate is not a good model, either. In other words, I still want to consider the precision. Therefore, F-1 score become the ideal metric to measure the performance, since it is a harmonic average between precision and recall, which can reflect both of them.

Result & Analysis:

- **Decision Tree:**





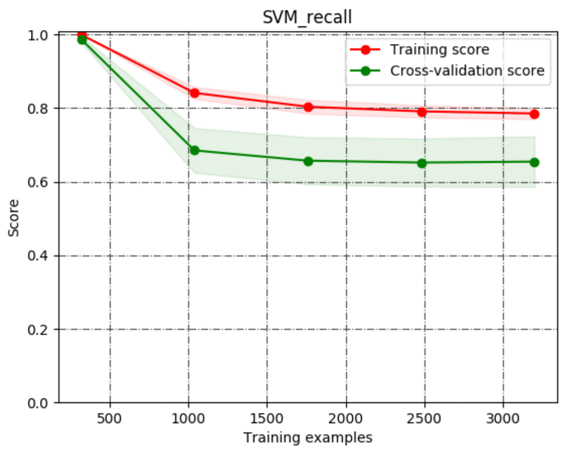


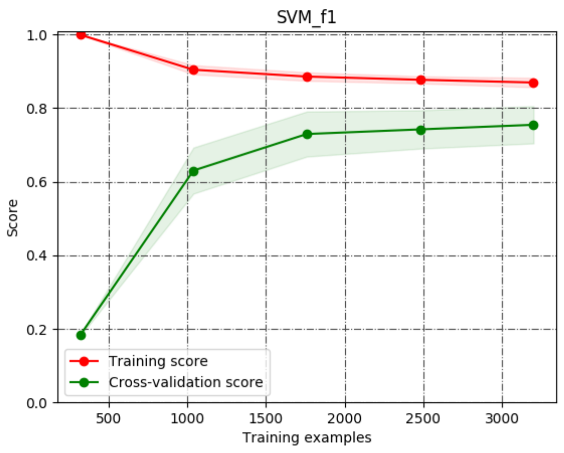
best parameter - criterion: "gini",
max_depth": 2

As we can see in the F-1 learning curve, decision tree work well in this dataset. the f1 score of both training and validation is above 0.9, and there are barely no overfitting happening is this setting when you see . I am surprised that we only need a tree of max depth of 2 to achieve this performance. This may due to the reason that there are two many negative samples so increase the max_depth will just overfit them. So what the model do may just choose two to three feature that can best separate the positive sample out and leave the other as negative. In fact, if you see the right graph which shows the relationship between tree depth the performance, overfitting does happening when the max_depth is 3 or above.

- **Support Vector Machine:**

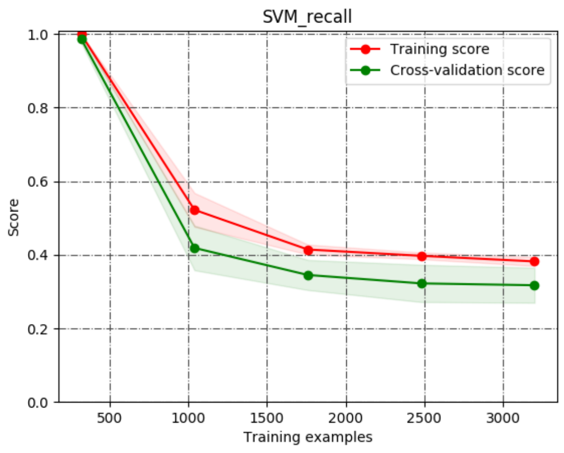
best_parameter - C: 10000,
gamma: 1e-07,
kernel": rbf,
max_iter: 100000

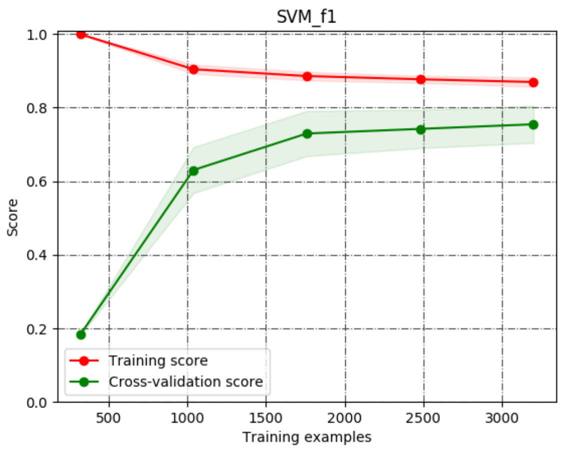




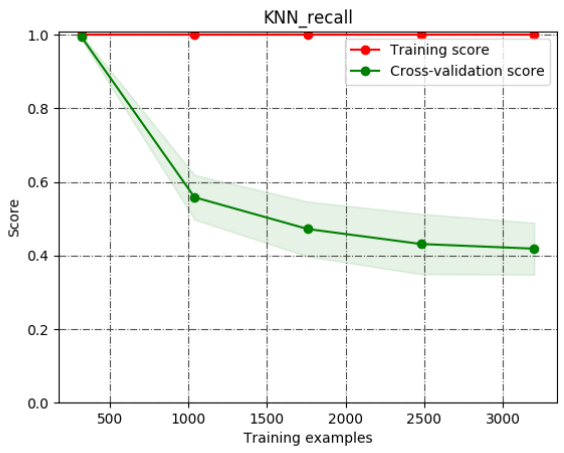
In the result of support vector machine, we can see there is gap between training and cross-validation score for both F1 and recall, indicating there is an overfitting happening. From the recall graph, we can see it only has about 0.8 even in training score. I think this information support the expectation that support vector machine will likely to treated the positive sample in an imbalanced dataset as noise, since it is even not able to get the positive data in it dataset correct.

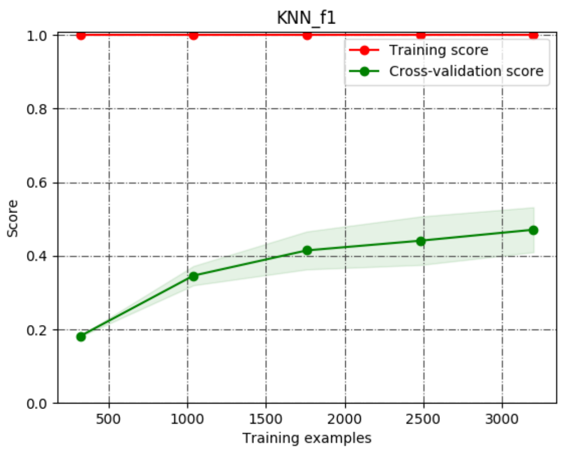
Another Interesting finding is that the C parameter is very sensitive in this classification problem, and it has a tendency toward a bigger C. When I changed C to 1000, the performance decreased significantly as the following graph shows. I think the reason is that if we use bigger C, SVM will try to get everything right no matter how small the margin is. On the contrary, SVM will try to find a bigger margin even there will be some classification error. Such a Imbalanced dataset like this one will make the positive sample more ‘sparse’ to each other, which require SVM use a bigger C and train a more complex model to get it right. Smaller C will just treat lots of positive sample as noise. However, high will make the model more likely to overfit.





- **K-nearest Neighbors:**



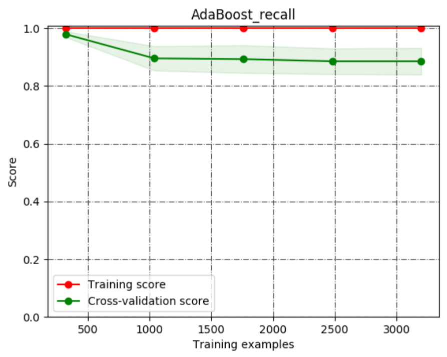


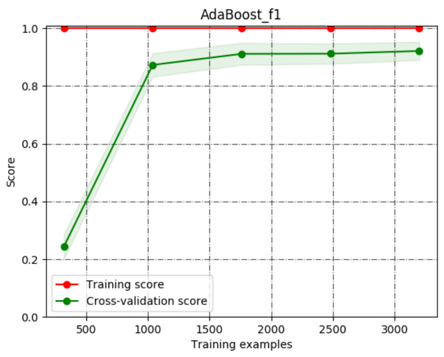
best parameters - n_neighbors: 1,
weights: 'uniform',
p: 1

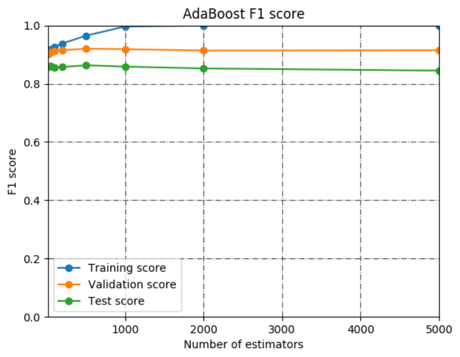
The best parameter I found using grid search is very surprising in the beginning. KNN just entirely overfit to the data by choosing the the n_neighbors = 1. Indeed, no matter how I try to increase the n_neighbors, the result is not increasing. However, it make sense when you consider the factor that data is imbalanced. Since the positive data are far more away from each other, taking more point into consideration will just let the result become negative (since nearly all your surrounding are negative.) Thus, choosing one will maximize the validation score even though it is still low.

One of the improvement for this case could be that if we see one positive sample in k neighbors, we label it as positive. However, there is a chance the precision will become very low since a negative sample will become positive if it is too close to a positive one.

- **Boosting:**





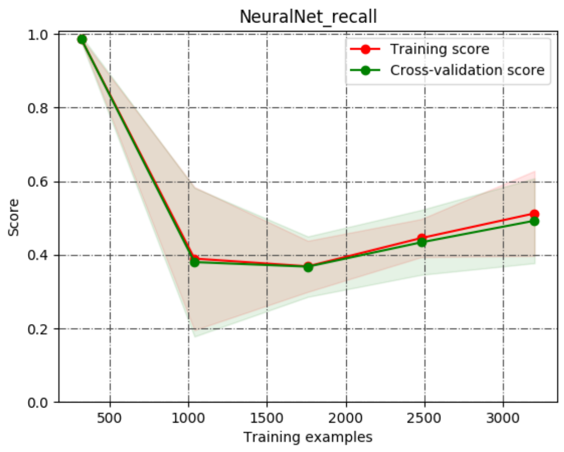


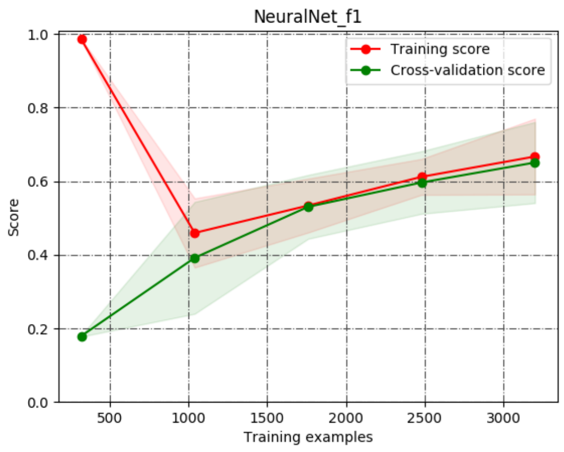
best parameter - learning_rate: 0.1,
n_estimators: 2000

As I expected in the beginning, boosting gave me a high performance in terms of F1 and recall score. In fact it has a highest validation among five algorithm. As we can see in the recall graph the algorithm can fit all the positive training sample without decreasing validation score too much, proving that ada boost is able to get the hard sample right without suffering from significant overfitting. Nevertheless, both of the two graphs still have gaps between training and validation curve, indicating that there is still some form of overfitting happening. My interpretation is that, since a decision tree with max_depth = 2 is strong enough to get a good performance, decision stump may not be weak enough for boosting in this case. As a result, a 'not weak enough' estimator result in overfitting after perform boosting.

If you see the right figure, you will found out that the amount of overfitting are barely change when the model complexity (n_estimators) increase compared with decision tree. Thus, we can say that adaboost still have some ability to resist overfitting.

- **Neural Network:**





best parameter - "activation": "relu",
"alpha": 0.0001,
"hidden_layer_sizes": [64, 32, 16],
"learning_rate": "adaptive",
"learning_rate_init": 1e-05,
"max_iter": 1000,
"solver": "adam"

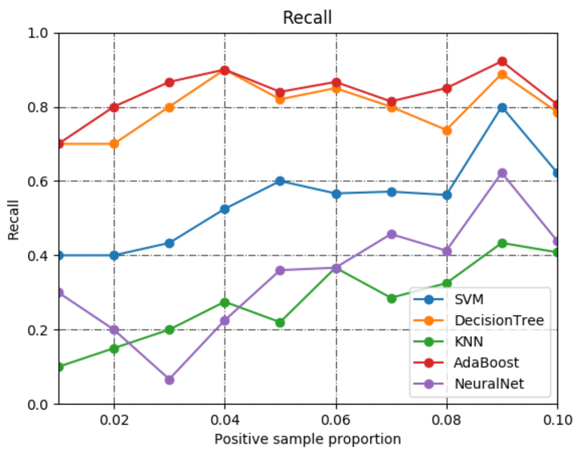
The learning curve of the Neural Network shows that it may underfit the train data. However, no matter how change the parameter., including increase the size and number of layer in the network, and the learning rate, it still give me the similar or even worse result, with train score equally lower with validation score. Nevertheless, if we see the testing result, we will find out that there is nearly no false positive. My interpretation of this situation is that neural treat lots of the positive sample as noise and therefore even the training recall score are very low.

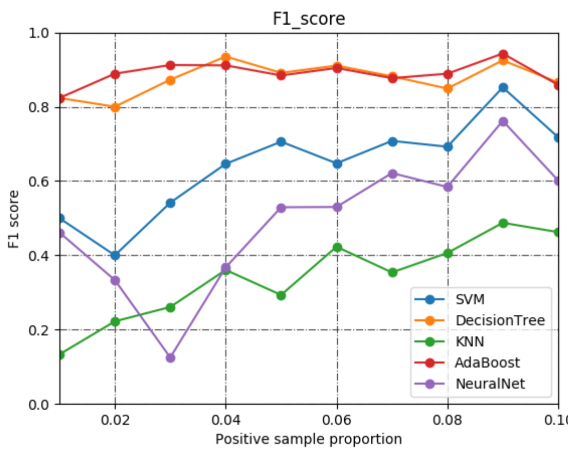
- **Conclusion:**

the following table show the overall result of five algorithm on test dataset.

	validation F1 score	Test F1 score	True positive	True negative	False positive	False negative
Decision Tree	0.919	**0.865**	77	898	3	**21**
SVM	0.752	0.718	61	890	11	37
KNN	0.471	0.462	40	866	35	58
AdaBoost	0.921	**0.859**	79	894	7	**19**
Neural Network	0.488	0.601	43	899	2	55

And this two figures are the recall and F1 score for five algorithm running on a fixed size (5000) dataset with different proportion of data.





In conclusion, we can say that generally, Decision tree and Adaboost can tackle an imbalanced dataset better, and Ada boost can also reduce overfitting when model complexity increase. Nevertheless, SVM, KNN Neural Network are more likely to be influenced and treated many positive sample as noise.

Problem 2: High Dimensional (MNIST) Classification

High dimensional data is another problem that become more and more crucial in today's machine learning and data science. A 50x50 grayscale image will have 2500 dimensions. In today's world of big data, it is very common to have high dimensional feature space in our dataset. However, "The Curse of Dimensionality" theory says that when the number of dimension increase, the number of samples needed grow exponentially. MNIST dataset contain digit images with 784 features which is an ideal dataset for experiment the performance under curse of dimensionality. In this problem, I want to explore the performance of each algorithms under the dataset with high dimension limited of data.

Data preprocessing:

The original MNIST dataset has 10 class (0 ~ 9) with each class around 6000 ~ 7000 samples. To simplify the problem, I select data from 4 and 9, which is relative hard to distinguish among all pairs of digits, to form a binary classification problem. 1000 instance is randomly sample from each class to get a total size of (2000, 784) dataset so that the dataset is suffer more from curse of dimensionality. Again, this dataset is then separated to training and testing data with 4:1 ratio. The training data is used to train and validate and the testing data is used only in the final test

Metrics:

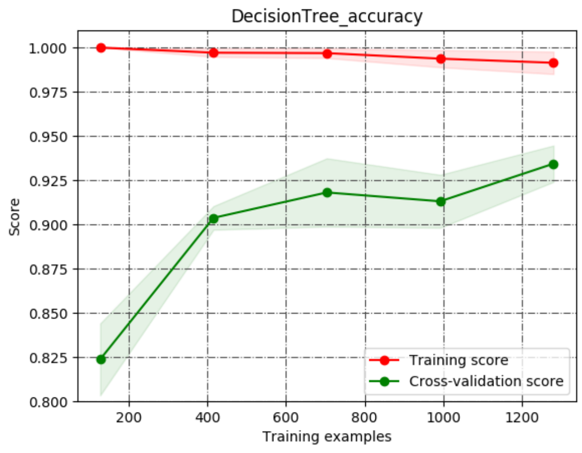
Since it is balanced binary classification problem, and I care both of the classes equally. I simply choose accuracy as my metric to evaluate the performance.

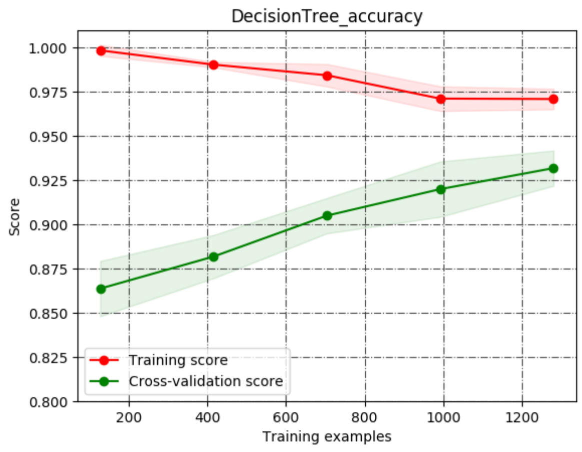
Result & Analysis:

Although the performance of the five algorithms seems to have a good performance on this dataset, we can still see difference after we dive into the result. I will discuss the results one by one.

- **Decision Tree:**

best parameter -
criterion: “gini”,
max_depth: 8





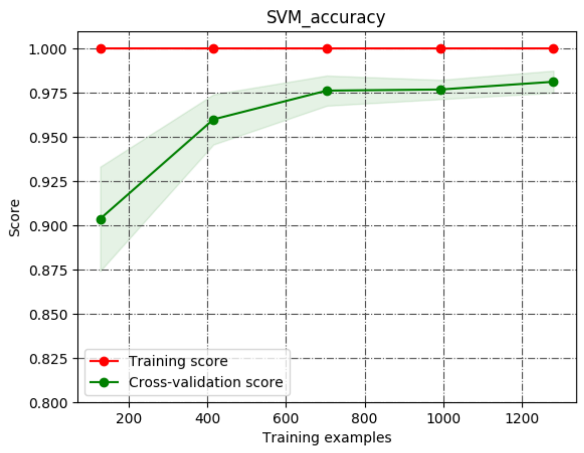
As we can see in the top learning curve on the left, decision tree has a huge gap between training and validation curve. In fact, the amount of overfitting in Decision Tree is the largest among all the algorithm. I decrease the max_depth to 5 and result in the bottom learning curve. Even though the gap between to score line is decreased, the validation score did not go higher. This result demonstrate that decision tree is more vulnerable with respect to the high dimensional data. The reason for this is that when dimension increase, the number of choice we can use increase as well. With limit samples, it is hard for the algorithm to choice one feature over another. Also, as the sample space increases, the distances between data points increases, which makes it much harder to find a "good" split.

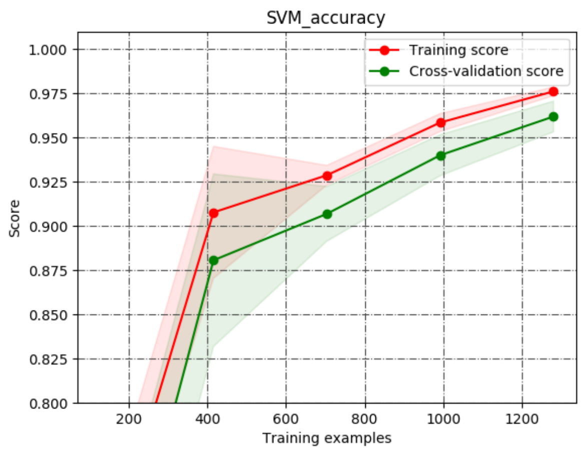
One way to tackle this problem is using random forest. Each tree in the random forest use a subset of features instead of all of them. Therefore, it reduce the space each tree is optimizing and can help to combat with curse of dimensionality.

- **Support Vector Machine:**

  best parameter -

  kernel:'poly',
  C: 0.0001,
  degree: 3,
  gamma: 1e-5



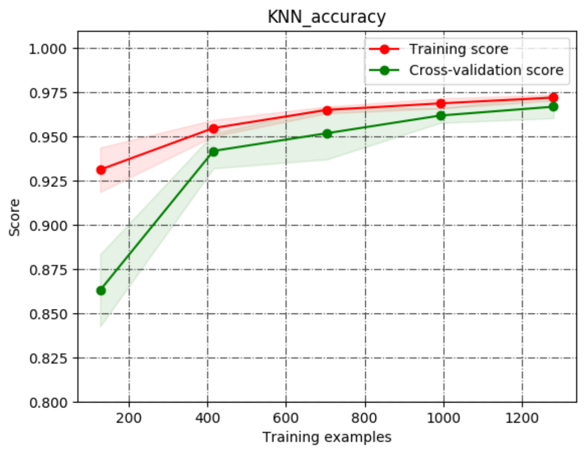


The top left learning curve shows the performance of SVM on MNIST dataset. In contrast to Decision Tree, the overfitting is less significant. After searching for why SVM can have the ability to resist curse of dimensionality, I found that the reason is the excessive regularization is can achieved. In another aspect, we can use kernel trick to transform our feature to some high dimension (even infinite) dimension space but still achieved good performance. Therefore, SVM should be independent to the dimension of feature space theoretically. However, in practice, SVM is highly sensitive to regularization parameter C and the kernel we used. For example, change C to 1e-6 will result in a very different results as the bottom left figure shows.

- **K-nearest Neighbors:**

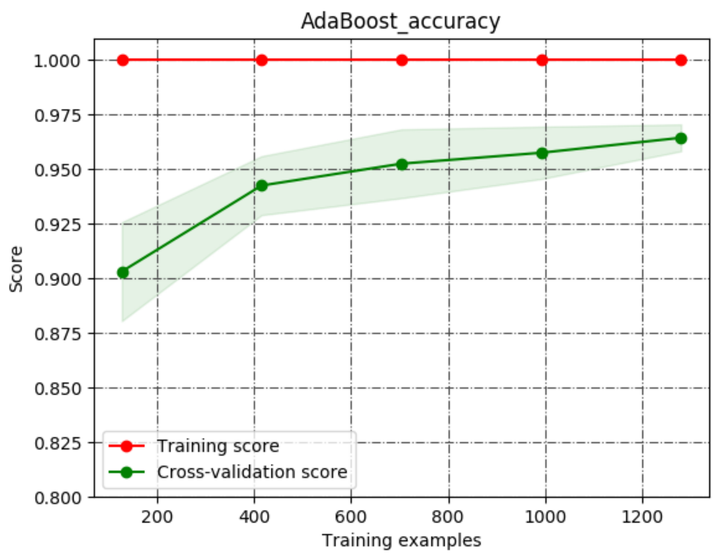
  best parameter -

  n_neighbors: 9,
  p: 2,
  weights: “uniform”



In the beginning, I thought the result of K-nearest Neighbor will suffer greatly from the curse of dimensionality since it use l2 as distance metric. As the dimension goes up, the distance between each sample increase significantly. Therefore, we need much more data to cover the sample space so that the newly-coming data point will have closer enough points to descript it.

However, KNN perform better as I expect, and is even better than decision tree. After diving into the dataset, I found that those pixels who have value will concentrate in the same area (center). So doing KNN on this dataset may be approximate to KNN on those concentrated areas, which reduce the dimension indirectly.

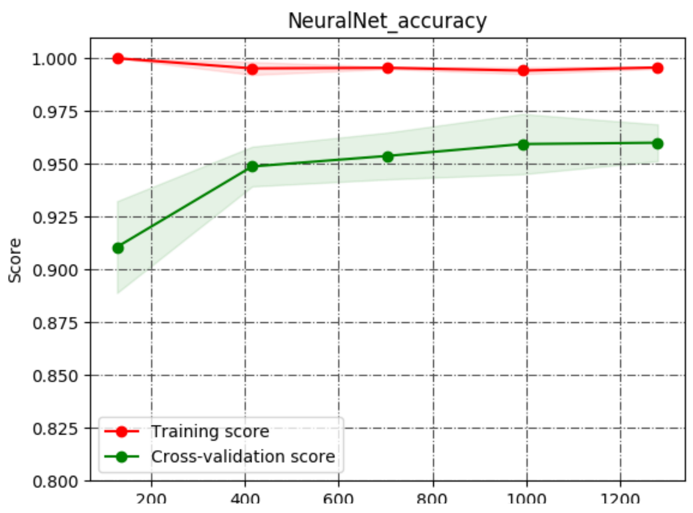


- **Boosting:**

  best parameter -

  learning_rate: 0.8,
  n_estimators: 500

Although not as much as decision tree, Adaboost is also suffer from high dimensionality. It make sense because adaboost is just linear combination of multiple decision tree, which itself still suffer from curse of dimensionality.

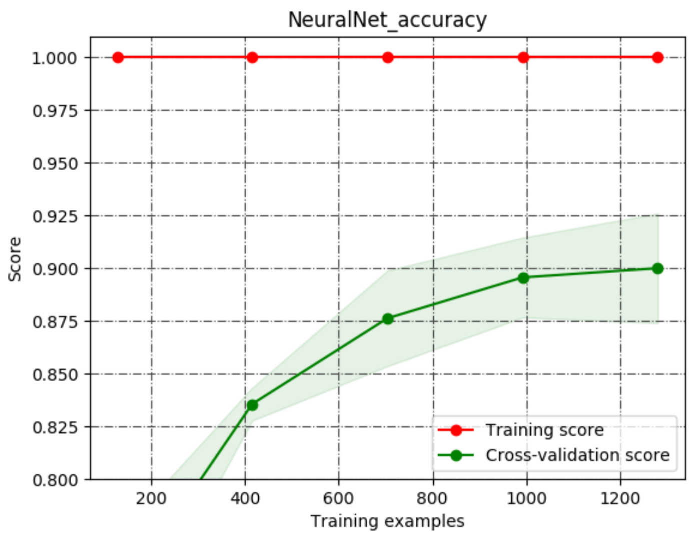


- **Neural Network:**

  best parameter -

  activation: "logistic",
  alpha: 1e-05,
  hidden_layer_sizes:
  (64, 64, 32, 16),
  learning_rate:
  "adaptive",
  learning_rate_init:
  0.0001,
  max_iter: 1000,
  solver: “adam"

The top left figure also shows neural network can also suffer from overfitting. The interest finding is that if I change the activation function to relu, the effect become much more significant (as bottom figure shows.)



- **Conclusion:**

  In conclusion, SVM is more likely to counter curse of dimensionality better using its regularization parameter C. Nevertheless, C and kernel function need to be choose carefully, and the non-optimal hyper-parameter for SVM can still lead to overfitting. On the other hand, Decision Tree, KNN, and Adaboost with decision stump are algorithms that work directly on samlpe space which is much more likely to suffer from the curse of dimensionality. Although the result of KNN did not demonstrate overfitting, I believe that overfitting will happen if we use other high dimension data without so many features contain nearly only zero.

Resources:

- scikit-learn https://scikit-learn.org/stable/index.html
- kaggle - Credit Card Fraud Detection https://www.kaggle.com/mlg-ulb/creditcardfraud
-