Assignment 1 Supervised Learning

Lujia Zhang
CS7641
Lzhang341@gatech.edu

1. Introduction

The purpose of this assignment is to explore two datasets

1. Hotel booking demand data
2. MNIST handwritten digit database

implement 5 learning algorithms

1. Decision trees
2. Neural Networks
3. Boosting
4. Support Vector Machines
5. K-nearest neighbors

to design two interesting classification problems and to show off my understanding of the techniques.
Implementation of the code can be find at: https://github.com/swagluke/CS7641-HW1

2. Hotel booking demand data

The Hotel booking demand data [1] is a combination of two data sets, one from the resort hotel demand data (40,060 observations), and the other from city hotel demand data (79,330 observations). Each observation represents a hotel booking record. Each record contains 31 different features/attributes. These features/attributes are useful information about the booking. Data set was collected the 1st of July of 2015 and the 31st of August 2017.

The classification problem for this data set is to use the features/attributes to predict whether each hotel booking will turn out to be cancel or not. In the data set, it does provide *is_canceled* as the correct label of this classification problem for each hotel booking demand record.

The classification label *is_canceled* is binary, 0 stands for not canceled, and 1 stands for canceled.

2. 1. Data Processing/Cleaning

It turns out the data set has 31994 duplicate rows of data records and some null values on country, agent and company features. I took the data cleaning approach of dropping duplicate data and filling 0 for both agent and company to clean the data.

Out of the 31 features that this data set provides, 19 of the features are numerical features and 12 of the features are categorical features.

I split the data set into 80% training set and 20% testing set using sklearn train_test_split function.

2.2. Exploratory Data Analysis

After cleaning up the data, I did exploratory data analysis to help me understand the data set better, this helps me to prepare for the classification problem.

Figure 1,2 below help me to understand the hotel booking break down into hotel type and monthly.

Figure 1: City Hotel Customer Booking compare to Resort Hotel Customer Booking

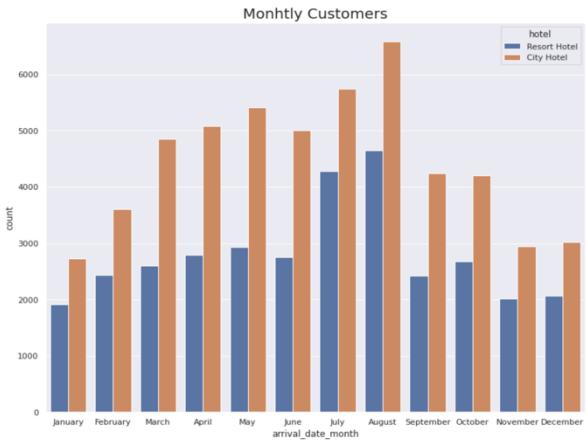

Figure 2: Monthly Customers Booking

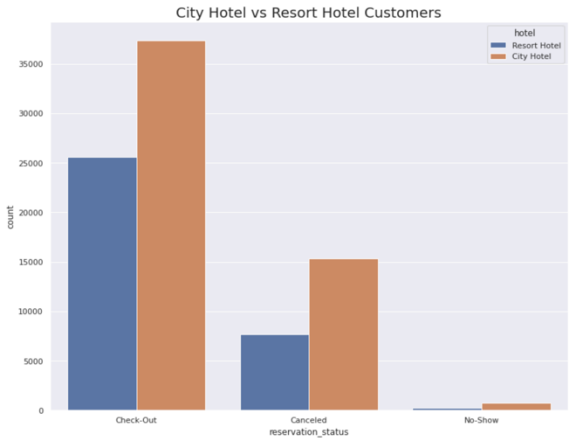

Figure 3: Customers Booking Reservation Type

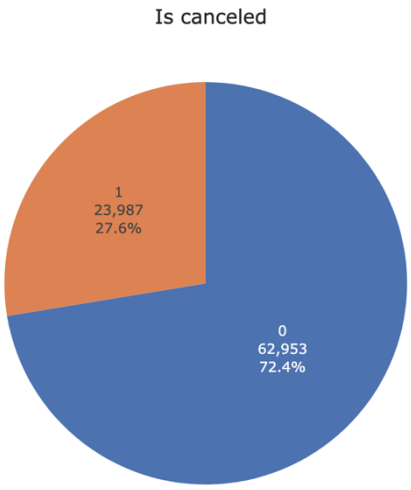

Figure 4: Booking Is Canceled Pie Chart

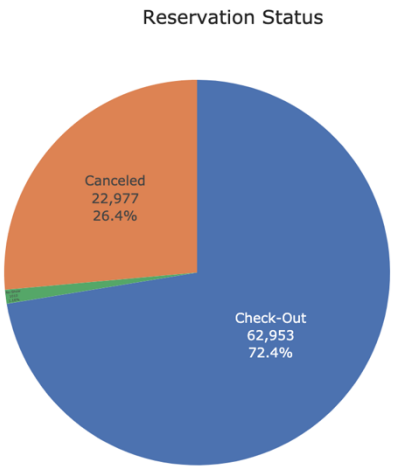

Figure 5: Booking Reservation Status Pie Chart

Figure 3,4,5 on left help me to understand the hotel booking reservation type and is canceled.

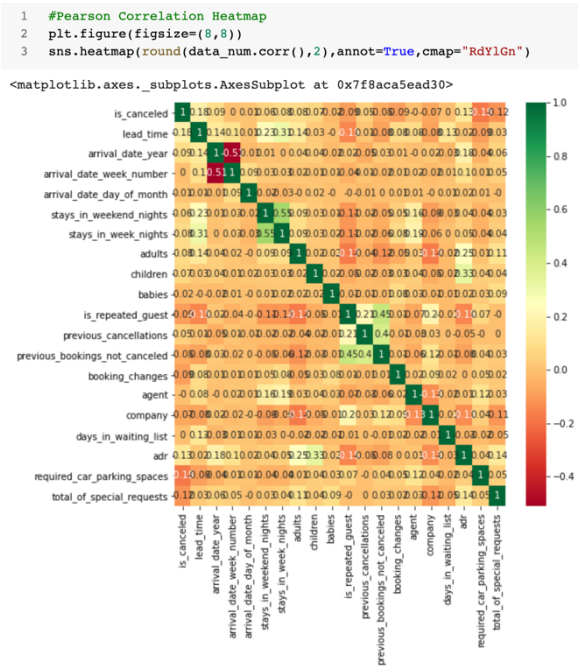

Figure 6: Pearson Correlation HeatMap

```
#Spearman Correalation Heatmap
plt.figure(figsize=(8,8))
corr = data_num.corr(method = 'spearman')
sns.heatmap(round(corr,2), annot = True,cmap="RdYlGn")
plt.show()
```

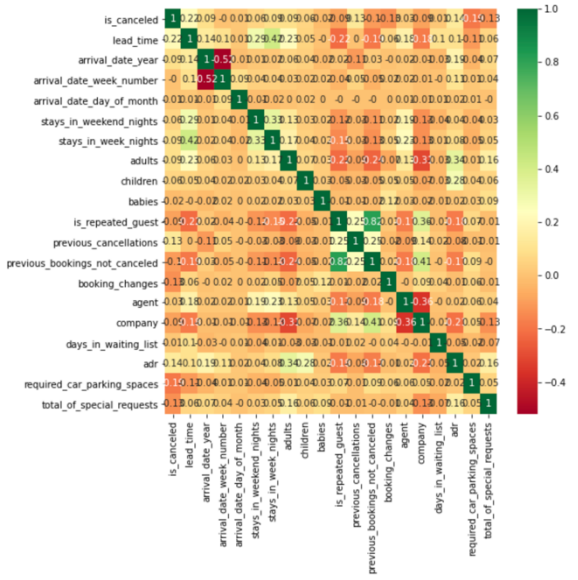

Figure 7: Spearman Correlation HeatMap

Figure 6,7 shows the person and spearman correlation for the numerical features.

2. 3. Decision Tree

I started with Decision Tree without pruning, see classification reports for training and testing and learning curve graph below (Figure 8,9,10). The classification report provides important metrics such as precision, recall and f1 score. With the training set, it achieves almost 100% accuracy, which is understandable since decision tree without pruning will end up with all leaves are pure (over fitting). The effect of training set over fitting is obvious since the classification report for the testing set only achieves 80% accuracy, not close to the 100% from the training set. With cross validation, with more training, the accuracy score does jump from 0.75 to 0.8.

```
DecisionTreeClassifier Train score: 0.9975270301357257
DecisionTreeClassifier Train Classification Report
precision {1: 0.9970109663882181, 0: 0.9988967111484711}
recall {1: 0.9995832341033579, 0: 0.9921206428720518}
f1 {1: 0.9982954432838483, 0: 0.995497146447458}
```

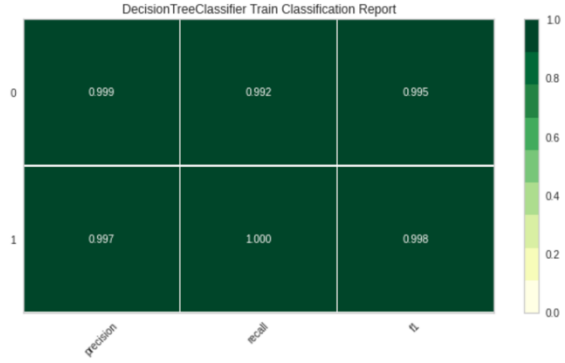

Figure 8: Decision Tree without Pruning Train Classification Report

```
DecisionTreeClassifier Test score: 0.800954681389464
DecisionTreeClassifier Test Classification Report
precision {1: 0.8625935956667198, 0: 0.6408771203971866}
recall {1: 0.8618384401114206, 0: 0.6423387932821895}
f1 {1: 0.8622158525419005, 0: 0.6416071243657451}
```

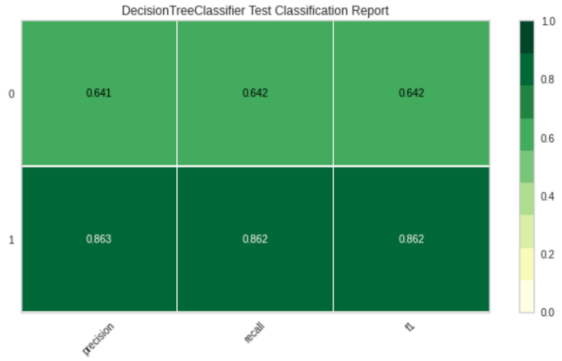

Figure 9: Decision Tree without Pruning Test Classification Report

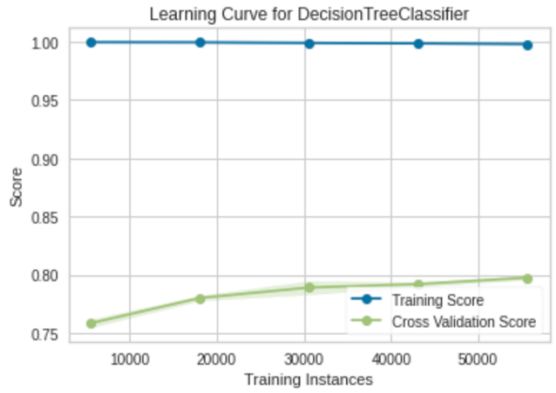

Figure 10: Decision Tree without Pruning Learning Curve

I change the max depth to 10 to conduct some pruning and results show below (Figure 11,12,13). Results aren't as good as the Decision Tree without Pruning, which is expected. But this does avoid over fitting. I found it super weird that with more training, the training score starts to decrease but the cross-validation score starts to increase. I assume the model is learning to generalize with normal data set instead of trying to over fit the training data.

```
DecisionTreeClassifier Train score: 0.8274097078444904
DecisionTreeClassifier Train Classification Report
precision {1: 0.8643681652490887, 0: 0.7120853080568721}
recall {1: 0.9035484639199809, 0: 0.6272176998538928}
f1 {1: 0.8835241606830972, 0: 0.6669626012651204}
```

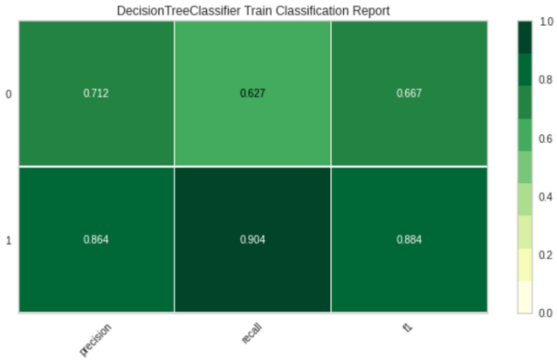

Figure 11: Decision Tree Max Depth 10 with Pruning Train Classification Report

```
DecisionTreeClassifier Test score: 0.8239015412928457
DecisionTreeClassifier Test Classification Report
precision {1: 0.858360359001433, 0: 0.713247759748123}
recall {1: 0.9057699960206924, 0: 0.6106157992950446}
f1 {1: 0.8814281288723668, 0: 0.6579535299374443}
```

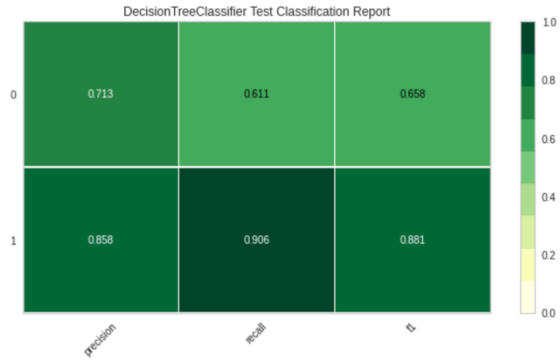

Figure 12: Decision Tree Max Depth 10 with Pruning Test Classification Report

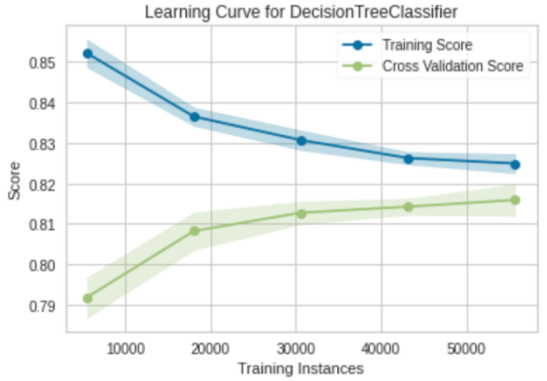

Figure 13: Decision Tree Max Depth 10 with Pruning Learning Curve

2. 4. Neural Networks

For Neural Network, I tried to use the MLPClassifier with lbfgs solver, hidden layer size 15 by 1 and relu activation. Classification reports for training and testing and learning curve graph below(Figure 14,15,16). The training score is still a lot higher than the cross-validation score.

```
MLPClassifier Train score: 0.8980475040257649
MLPClassifier Train Classification Report
precision {1: 0.9169507521041582, 0: 0.842379899041461}
recall {1: 0.9448479796777011, 0: 0.7749947818826968}
f1 {1: 0.9306903595969073, 0: 0.8072835983149885}
```

Figure 14: Neural Networks Train Classification Report

```
Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.

MLPClassifier Test score: 0.8225212790430182
MLPClassifier Test Classification Report
precision {1: 0.8949912492707726, 0: 0.6611616255334941}
recall {1: 0.8546756864305611, 0: 0.7387518142235123}
f1 {1: 0.874368995277642, 0: 0.6978065021543282}
```

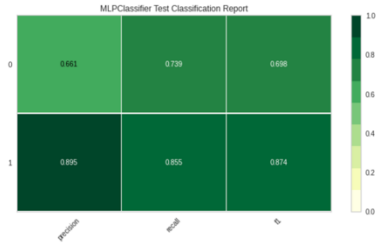

Figure 15: Neural Networks with Pruning Test Classification Report

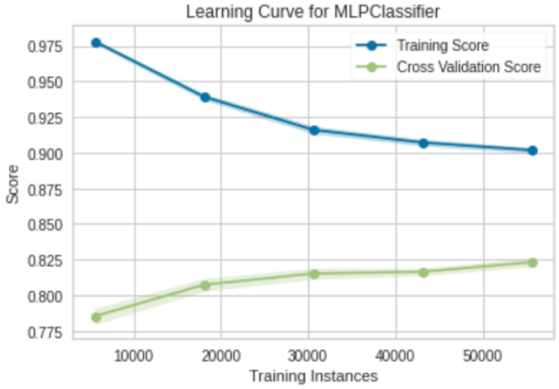

Figure 16: Neural Networks with Pruning Learning Curve

2. 5. Boosting

Implemented AdaBoostClassifier, Classification reports for training and testing and learning curve graph below(Figure 17,18,19). Boosting's result is really similar to Decision Tree with Pruning. I found it interesting that the learning curve show different characteristic, the score hit the highest peak then dropped off.

```
AdaBoostClassifier Train score: 0.8111916264090178
AdaBoostClassifier Train Classification Report
precision {1: 0.836999782907591, 0: 0.711263659288316}
recall {1: 0.9181948082876875, 0: 0.5298476309747443}
f1 {1: 0.8757192610539069, 0: 0.6072966507177033}
```

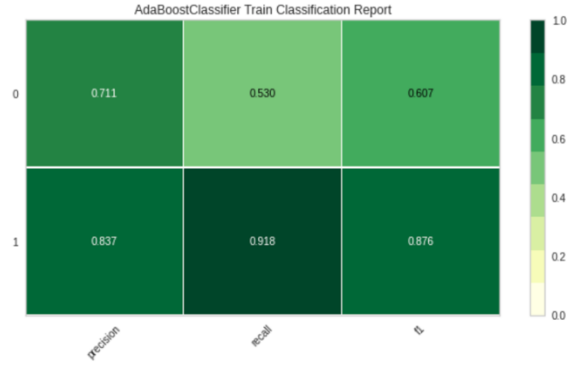

Figure 17: AdaBoost Train Classification Report

```
AdaBoostClassifier Test score: 0.8132620197837589
AdaBoostClassifier Test Classification Report
precision {1: 0.8355661192739845, 0: 0.7248858447488584}
recall {1: 0.9232789494627934, 0: 0.5266431681526021}
f1 {1: 0.877235434231918, 0: 0.6100636483727634}
```

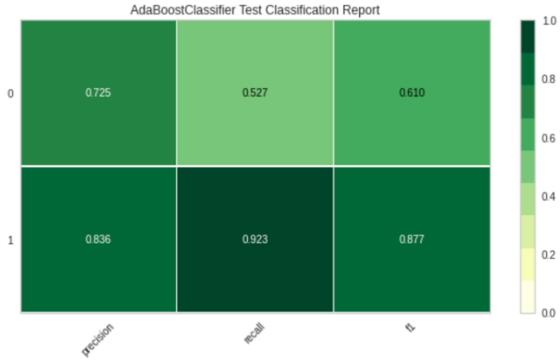

Figure 18: AdaBoost Test Classification Report

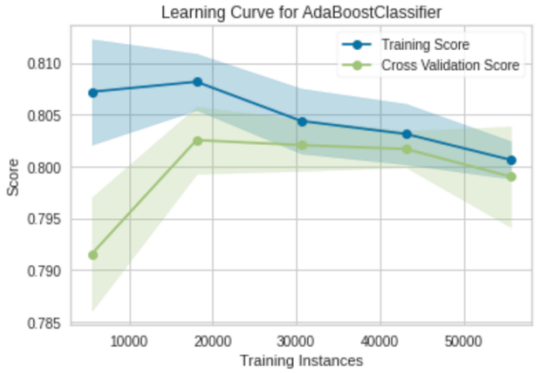

Figure 19: AdaBoost Test Learning Curve

2.6. Support Vector Machines

With SVC, I tried two different kernel (rbf and linear), results. With both training and testing data, SVC with rbf kernel return better results.

```
SVC Train score: 0.8426357257878997
SVC Train Classification Report
precision {1: 0.8576558277869462, 0: 0.7851641107487336}
recall {1: 0.9385567992379138, 0: 0.59042997286579}
f1 {1: 0.8962844336627847, 0: 0.6740134028294862}
```

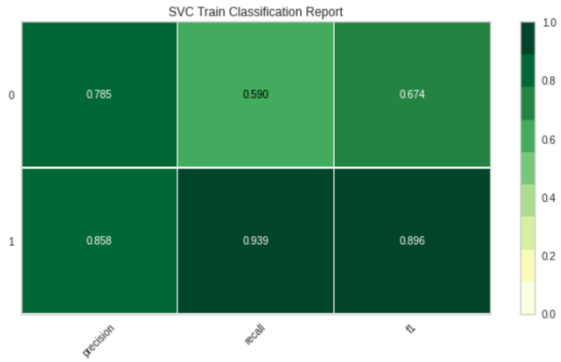

Figure 20: SVC rbf kernel Train Classification Report

```
SVC Test score: 0.8341384863123994
SVC Test Classification Report
precision {1: 0.8495198209256986, 0: 0.7739474427804465}
recall {1: 0.9363310783923597, 0: 0.5679037943188887}
f1 {1: 0.8908154766411751, 0: 0.6551064338674959}
```

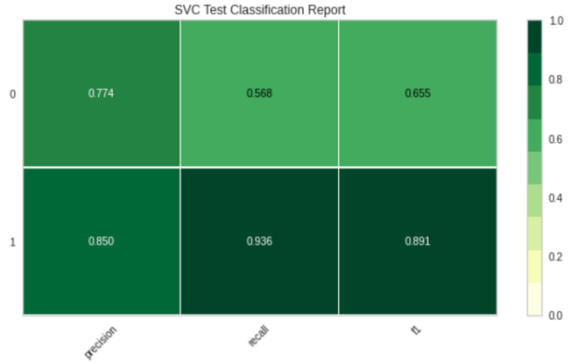

Figure 21: SVC rbf kernel Test Classification Report

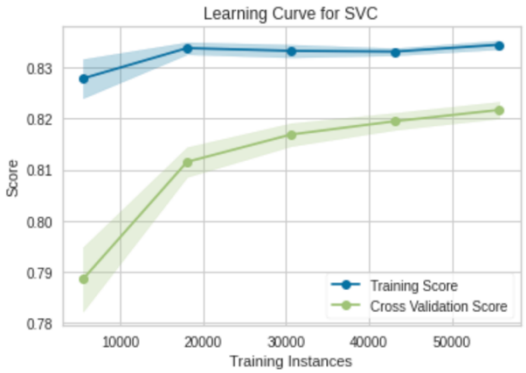

Figure 22: SVC rbf kernel Learning Curve

```
SVC Train score: 0.800149528410398
SVC Train Classification Report
precision {1: 0.8212084932568048, 0: 0.7063666300768386}
recall {1: 0.9256767484321664, 0: 0.4701001878522229}
f1 {1: 0.8703188849290019, 0: 0.5645090544520333}
```

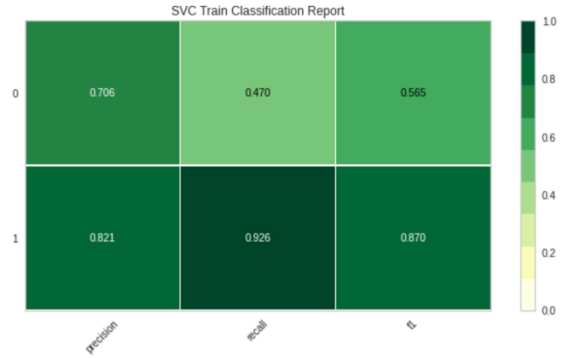

Figure 23: SVC linear kernel Train Classification Report

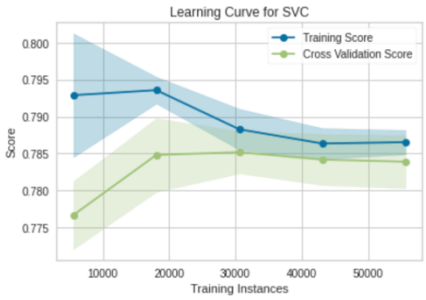

Figure 24: SVC linear kernel Learning Curve

2.7. K-Nearest Neighbors

With KNN, I tried two different k values (3 and 4), for both train and test set, k = 3 report better results. With more training, the score gets better.

```
KNeighborsClassifier Train score: 0.8884719346675869
KNeighborsClassifier Train Classification Report
precision {1: 0.9171412356406192, 0: 0.8090825340053108}
recall {1: 0.9300825593395253, 0: 0.7790649133792528}
f1 {1: 0.9235665651758355, 0: 0.7937900417364491}
```

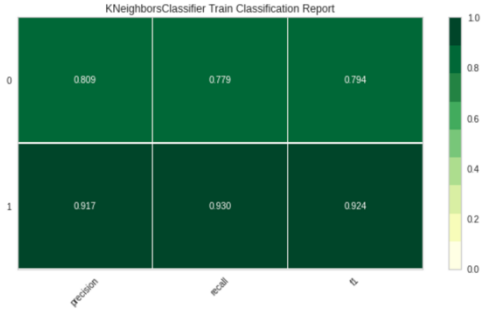

Figure 25: KNN K-3 Train Classification Report

```
KNeighborsClassifier Test score: 0.7824361628709455
KNeighborsClassifier Test Classification Report
precision {1: 0.8479949278808052, 0: 0.6090146750524109}
recall {1: 0.8515718265021887, 0: 0.602322206095791}
f1 {1: 0.849779613231148, 0: 0.6056499530907954}
```

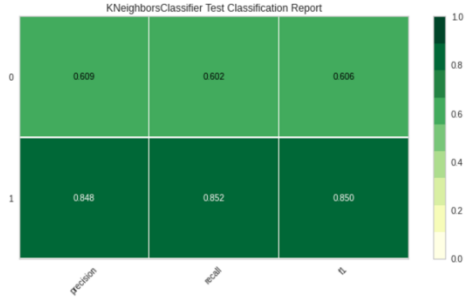

Figure 26: KNN K-3 Test Classification Report

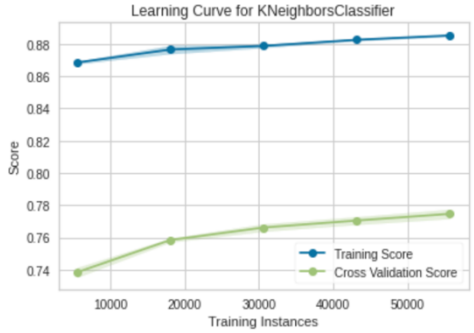

Figure 27: KNN K-3 Learning Curve

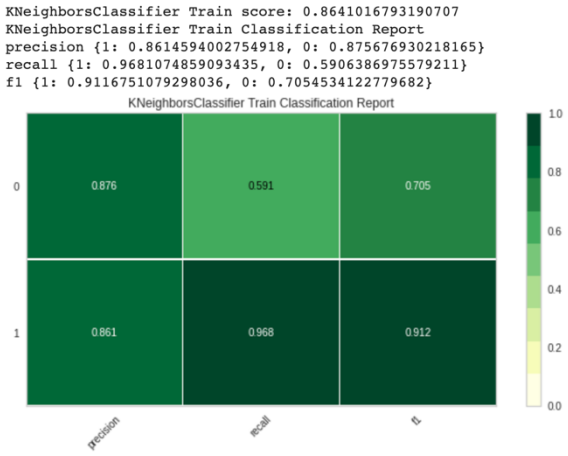

Figure 28: KNN K-4 Test Classification Report

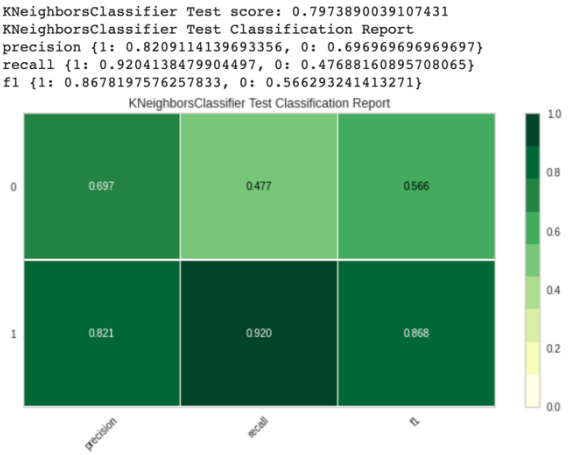

Figure 29: KNN K-4 Test Classification Report

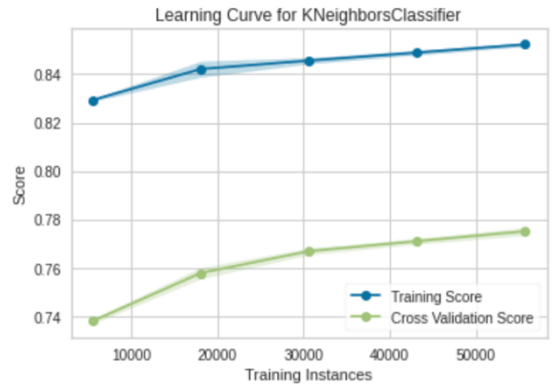

Figure 30: KNN K-3 Test Learning Curve

3. MNIST Handwritten Digit Database

The MNIST database (Modified National Institute of Standards and Technology database)[2] is a large database of handwritten digits that is commonly used for training various image processing systems.

It has a training set of 60,000 examples, and a test set of 10,000 examples with 789 features/attributes.

The classification problem for this data set is to use the features/attributes to predict the handwritten digits. In the data set, it does provide the correct label of this classification problem for record. The data set is well organized and cleaned, so there is no need to do any extra data preprocessing.

3. 1. Decision Tree

Decision Tree without pruning has generated similar results just like the Hotel booking data set. It's over fitting since I didn't set the max depth. After setting the max depth as 10 to have the Decision Tree with Pruning, it has generated similar result as Decision Tree without Pruning.

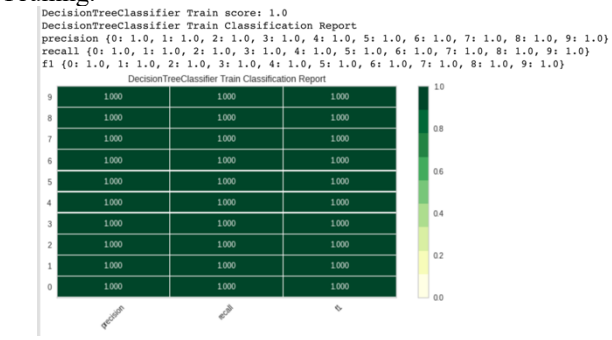

Figure 31: Decision Tree without Pruning Train classification report

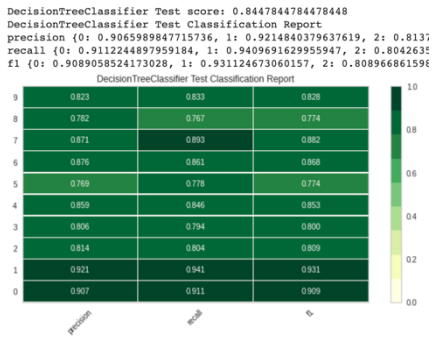

Figure 31: Decision Tree without Pruning Test classification report

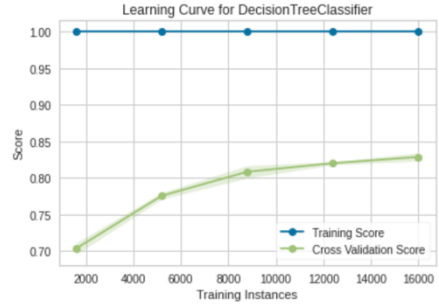

Figure 32: Decision Tree without Pruning Learning Curve

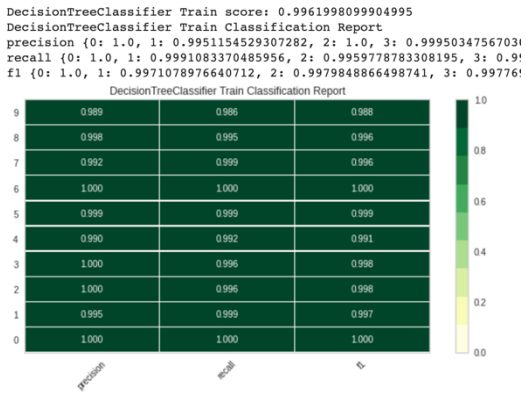

Figure 33: Decision Tree with Pruning Train Classification Report

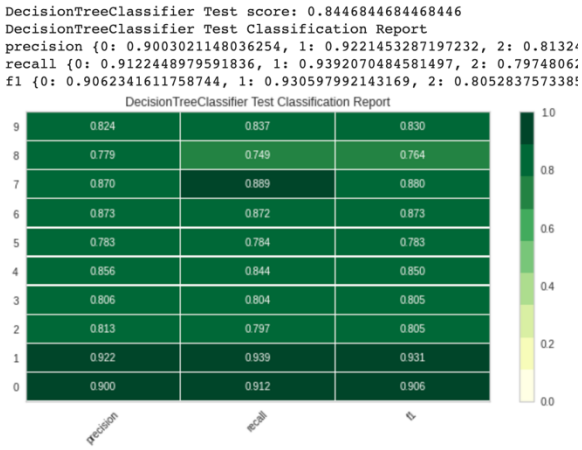

Figure 34: Decision Tree with Pruning Test Classification Report

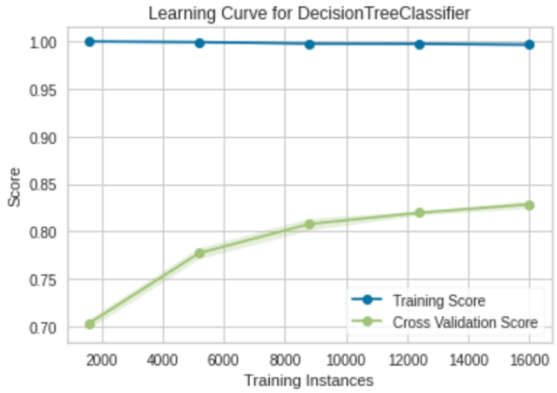

Figure 35: Decision Tree with Pruning Learning Curve

3. 2. Neural Networks

Neural Network has generated great good results just like the Decision Tree, which is concerning in a way since I know in fact Decision Tree without Pruning created overfitting. But it can be Neural Network performing super fantastic.

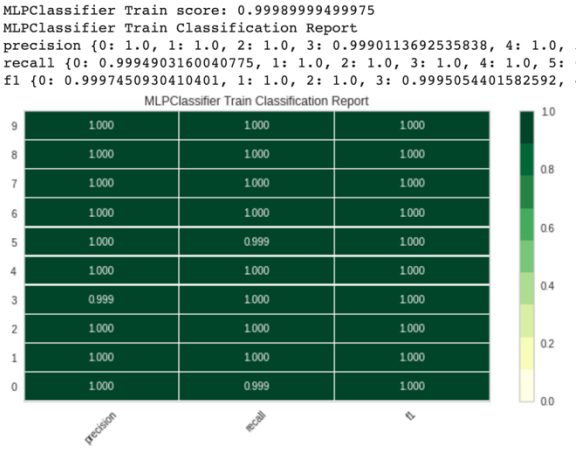

Figure 36: Neural Networks Train Classification Report

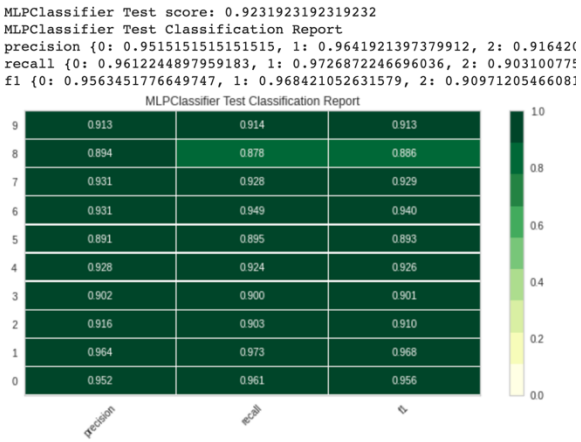

Figure 37: Neural Networks Test Classification Report

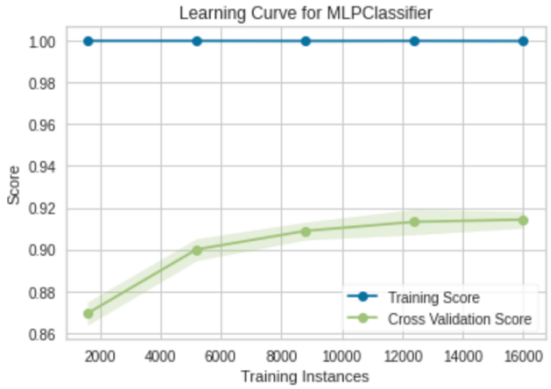

Figure 38: Neural Networks Learning Curve

3. 3. Boosting

AdaBoosting didn't generate good results at all. But surprising, the train score is always the same as the cross-validation score.

```
AdaBoostClassifier Train score: 0.7534876743837192
AdaBoostClassifier Train Classification Report
precision {0: 0.876074860900354, 1: 0.8324690865576386, 2: 0.7199
recall {0: 0.8827726809378186, 1: 0.9304502897904592, 2: 0.652589
f1 {0: 0.8794110180248794, 1: 0.8787368421052633, 2: 0.6845991561
```

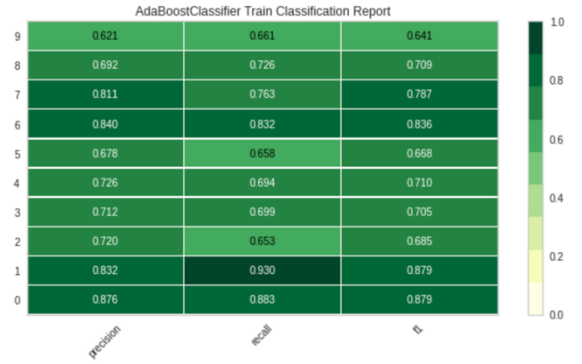

Figure 39: Boosting Train Classification Report

```
AdaBoostClassifier Test score: 0.7548754875487549
AdaBoostClassifier Test Classification Report
precision {0: 0.8627838104639685, 1: 0.8396150761828388, 2: 0.727
recall {0: 0.8918367346938776, 1: 0.9224669603524229, 2: 0.6375968
f1 {0: 0.8770697441043653, 1: 0.8790931989924433, 2: 0.6794011357
```

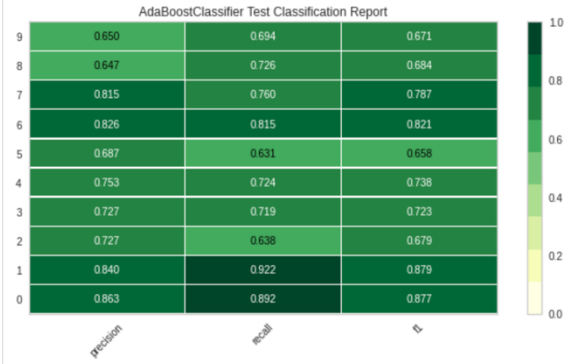

Figure 40: Boosting Test Classification Report

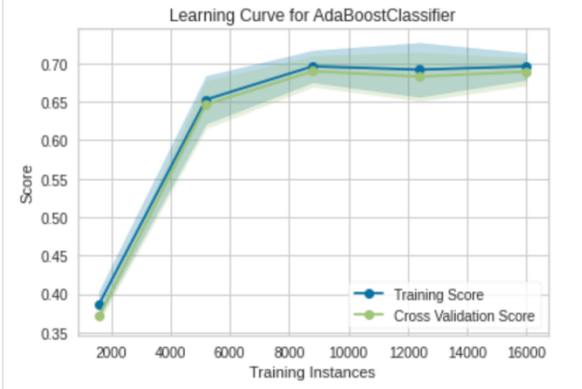

Figure 41: Boosting Learning Curve

3. 4. Support Vector Machines

I tried rbf and linear kernel for Support Vector Machines and again the linear kernel SVM work out much better. SVM works super great compare to other models.

```
SVC Train score: 0.9831991599579979
SVC Train Classification Report
precision {0: 0.9949057564951604, 1: 0.9840283939662822, 2: 0.982
recall {0: 0.9954128440366973, 1: 0.9888542131074454, 2: 0.987933
f1 {0: 0.9951592356687898, 1: 0.986435401378697, 2: 0.98496240601
```

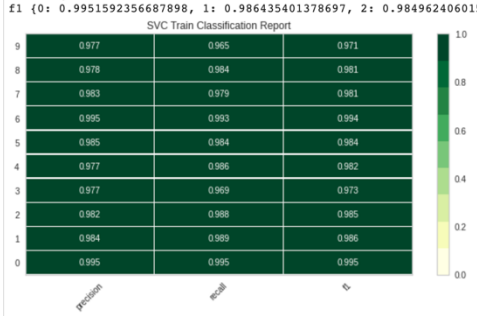

Figure 42: SVM rbf Train Classification Report

```
SVC Test score: 0.9532953295329533
SVC Test Classification Report
precision {0: 0.9726443768996961, 1: 0.9816433566433567, 2: 0.936
recall {0: 0.9795918367346939, 1: 0.9894273127753304, 2: 0.950581
f1 {0: 0.9761057447890189, 1: 0.9855199648968846, 2: 0.9437229437
```

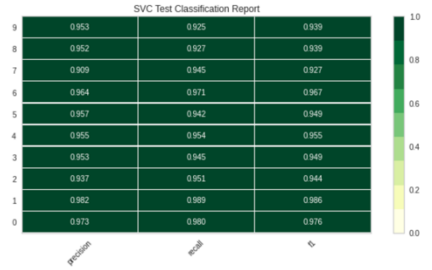

Figure 43: SVM rvf Test Classification Report

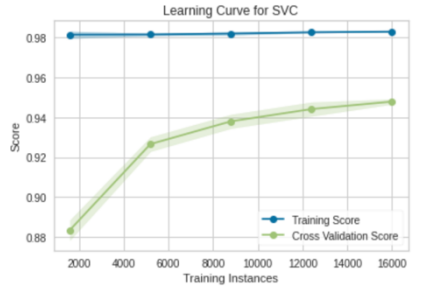

Figure 44: SVM rbf Learning Curve

```
SVC Train score: 0.9991499574978749
SVC Train Classification Report
precision {0: 1.0, 1: 1.0, 2: 1.0, 3: 0.998019801980198, 4: 0.999
recall {0: 1.0, 1: 1.0, 2: 1.0, 3: 0.9975259772389906, 4: 1.0, 5:
f1 {0: 1.0, 1: 1.0, 2: 1.0, 3: 0.9977728285077795, 4: 0.9997401922
```

Figure 45: SVM Linear Train Classification Report

```
SVC Test score: 0.9177917791779178
SVC Test Classification Report
precision {0: 0.9426310583580614, 1: 0.959656652360515, 2: 0.89463
recall {0: 0.9724489795918367, 1: 0.9850220264317181, 2: 0.90503871
f1 {0: 0.9573078854846812, 1: 0.9721739130434782, 2: 0.89980732177
```

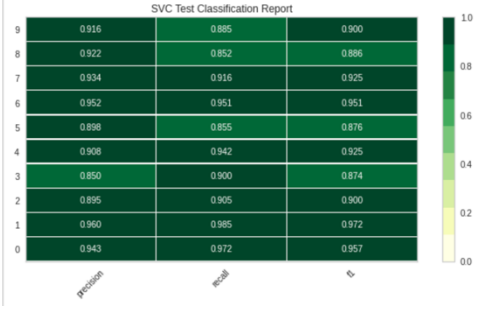

Figure 46: SVM Linear Test Classification Report

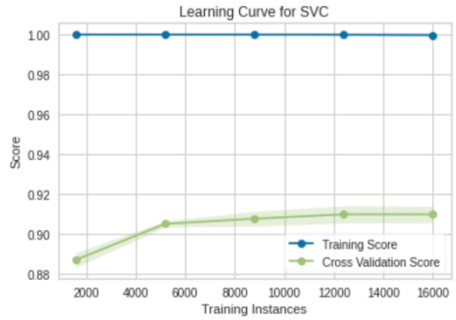

Figure 47: SVM Linear Learning Curve

3. 5. K-Nearest Neighbors

KNN turns out to be the best model in terms of both training and test data set. I tried both K= 3 and K = 4 and both results are fantastic.

```
KNeighborsClassifier Train score: 0.9615480774038702
KNeighborsClassifier Train Classification Report
precision {0: 0.9647992067426872, 1: 0.9611231101511879, 2: 0.9584599
recall {0: 0.9918450560652395, 1: 0.9919750334373607, 2: 0.95123177474
f1 {0: 0.9781352098517214, 1: 0.976305397103993, 2: 0.9548321978299268
```

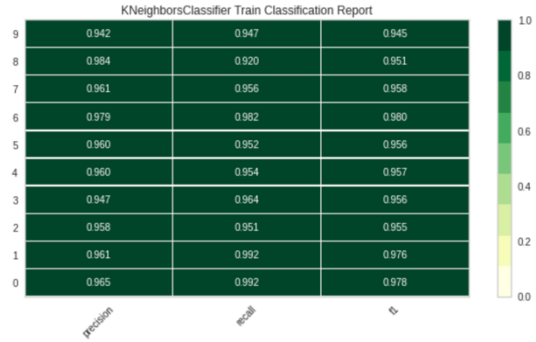

Figure 48: KNN k = 3 Train Classification Report

```
KNeighborsClassifier Test score: 0.930993099309931
KNeighborsClassifier Test Classification Report
precision {0: 0.9358600583090378, 1: 0.9423076923076923, 2: 0.9452
recall {0: 0.9826530612244898, 1: 0.9929515418502203, 2: 0.9031007
f1 {0: 0.9586859133897462, 1: 0.966966966966967, 2: 0.923686818632
```

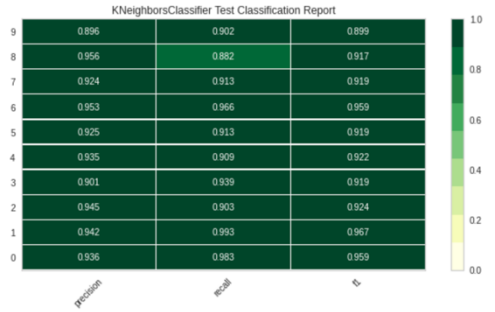

Figure 49: KNN k = 3 Test Classification Report

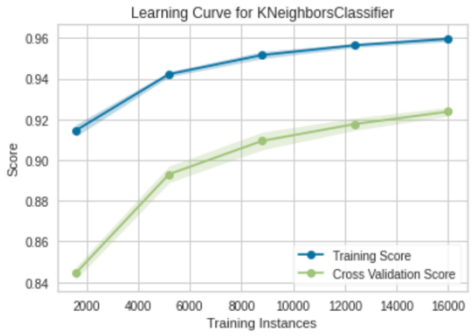

Figure 50: KNN k = 3 Learning Curve

```
KNeighborsClassifier Train score: 0.9541977098854942
KNeighborsClassifier Train Classification Report
precision {0: 0.9577187807276303, 1: 0.947726306842329, 2: 0.9582
recall {0: 0.9928644240570846, 1: 0.9942041908158716, 2: 0.946706
f1 {0: 0.974974974974975, 1: 0.9704090513489991, 2: 0.95245321193
```

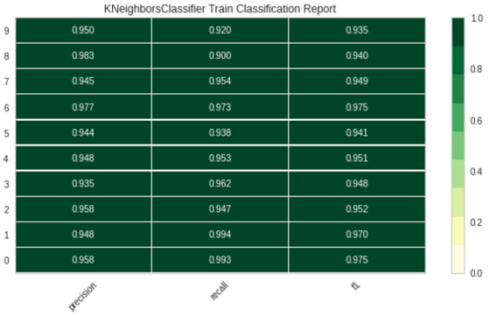

Figure 51: KNN k = 4 Train Classification Report

```
MLPClassifier Test score: 0.9231923192319232
MLPClassifier Test Classification Report
precision {0: 0.9515151515151515, 1: 0.9641921397379912, 2: 0.91642
recall {0: 0.9612244897959183, 1: 0.9726872246696036, 2: 0.90310077
f1 {0: 0.9563451776649747, 1: 0.968421052631579, 2: 0.9097120546608
```

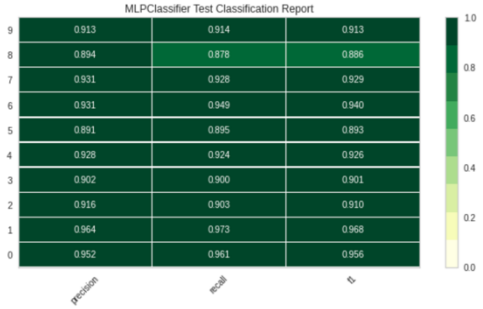

Figure 52: KNN k = 4 Test Classification Report

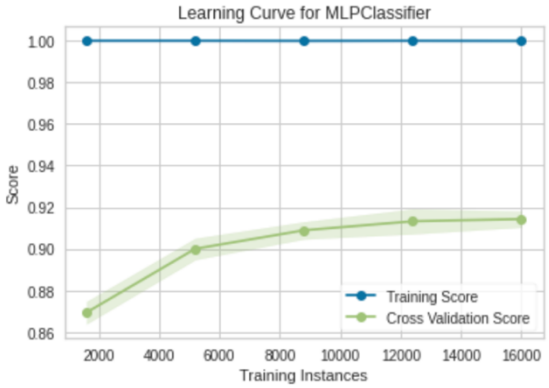

Figure 52: KNN k = 4 Learning Curve

4. Conclusion

I learned so much from this homework by implementing these 5 machine learning algorithms. Decision Tree is the algorithm that runs the fast, SVM and Neural Network both took very long time to train, which is totally reasonable since Neural Network has tons of hidden layers to process. KNN takes the most time during the predict since it's a weak algorithm. KNN has pretty decent results even though it's not the fanciest algorithm.

5. References

[1] "Hotel Booking Demand Datasets," Nuno Antonio, Ana de Almeida,LuisNunes.[Online].Available: https://www.sciencedirect.com/science/article/pii/S2352340918315191\#f0010. [Accessed: 19-Sep-2020].

[2] "MNIST-data", THE MNIST DATABASE of handwritten digits, Yann LeCun, Corinna Cortes, Christopher J.C. Burges 11-Dec-2018.[Online].Available: http://yann.lecun.com/exdb/mnist /ufcdataset. [Accessed: 19-Sep-2020].