CS7641 Assignment 1: Supervised Learning

Jack Chai Jie Feng
jchai41@gatech.edu

DATASETS

The 2 classification datasets chosen are adult income data from UCI machine learning repository (link) and bank churn data from kaggle (link). Both datasets are binary classification problems.

Income Data

This is an old dataset that was made available on 4/30/1996. It contains 32k rows and 14 features, with 50% of the features being categorical and the other 50% numerical. The goal of this dataset is to predict whether an individual makes >50k annual income or not, with an imbalance class of ~24.1% making the former label and ~75.9% for the latter. This dataset is also interesting because the one hot encoding of the categorical features will create many more columns.

Bank Churn Data

This is a synthetic dataset which contains 175k rows and 24 features of both numerical and categorical data. The goal of this dataset is to predict whether a customer has churned with an imbalance class of ~21.1% make up the churned customers and ~78.9% not. This dataset is interesting because it has a good amount of features that makes it a complex problem.

EXPERIMENTS AND ANALYSIS

We explore 5 different supervised learning models, namely decision tree, boosted decision trees, neural networks (multilayer perceptron - MLP), support vector machine (SVM) and k-nearest neighbors (KNN).

All models are first split into a stratified training and testing set of 4:1 ratio. Only the training set is used with a 5 fold grid search CV for hyper parameter tuning. Due to the imbalance of class labels on both datasets, **F1 score** is a more suitable evaluation metric.

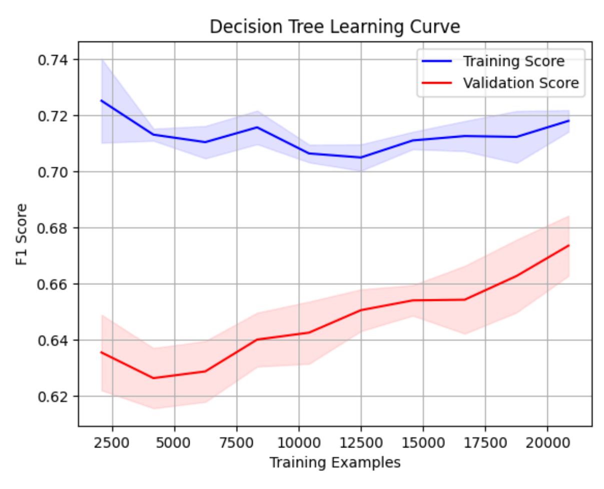
We will analyze the learning curve of the tuned model only on the training dataset, as well as how the model performs over iterations (only for applicable models). We will also look at how some hyper parameters affect the model performance on both training and testing datasets. In all the tables below, the left column refers to Income Data and right refers to Bank Churn Data.

Decision Tree (DT)

The library used for the DT algorithm is scikit-learn's `DecisionTreeClassifier`. Since DT are scale invariant, all features are left unscaled. The hyper parameters tree max depth and minimum samples at every leaf node will prevent overfitting.

Hyper parameters	{'criterion': 'gini', 'max_depth': 16, 'min_samples_leaf': 17}	{'criterion': 'entropy', 'max_depth': 9, 'min_samples_leaf': 34}
Train F1 score	0.719580359446212	0.6409122375201908
Test F1 score	0.6689678978253366	0.6244875860468714

Table 1.1: Income and Bank Churn data DT hyper-params & scores



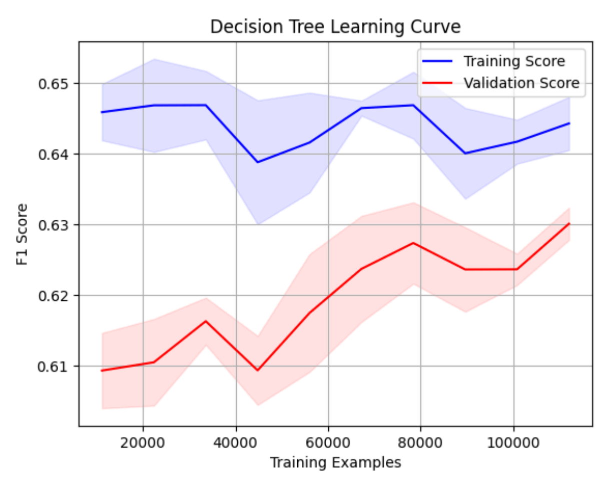


Table 1.2: Income and Bank Churn data DT learning curve (training sample)

In both instances, the validation score follows an upwards trend, suggesting that the model is suffering from high variance and more training data can potentially improve the model performance. The high standard deviation scores (lighter shade band around the mean) further confirms that it suffers from high variance.

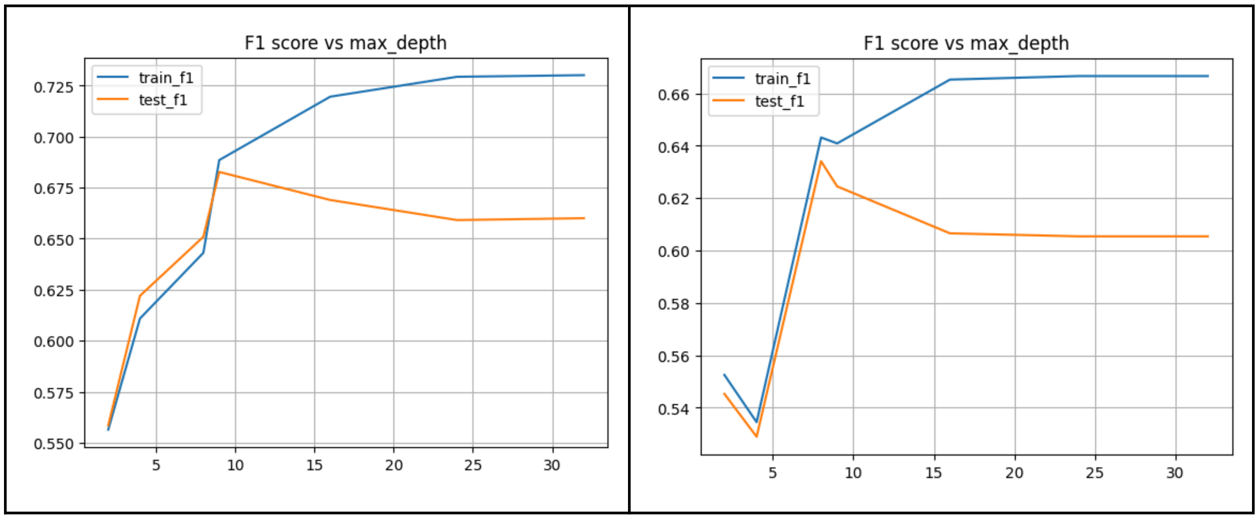


Table 1.3: Income and Bank Churn data DT max depth

As one would expect, increasing depth will cause the overfitting. However, for the income dataset, the max depth found from grid CV contradicts with the final testing set (16 vs 9). This means that a single DT built from the training set is not able to capture a good representation of the testing set, possibly due to a smaller dataset size. On the other hand, the bank dataset's CV suggestion is not that far off from the testing set (9 vs 8) due to a way larger dataset size.

Boosted Decision Trees

The library used for the boosted DT algorithm is xgboost's `XGBClassifier`.Due to scale invariant nature of trees, all features are left unscaled. The hyper parameters involved are tree max depth, regularization term alpha (L1) and lambda (L2).

Hyper parameters	{'max_depth': 3, 'reg_alpha': 0.1, 'reg_lambda': 10}	{'max_depth': 3, 'reg_alpha': 1, 'reg_lambda': 0.1}
Train F1 score	0.745364010989011	0.6700929342890817
Test F1 score	0.7096107475025836	0.6404170881643452

Table 2.1: Income and Bank Churn data Boosted DT hyper-params & scores

Observe that the pruning on a boosted DT model is more aggressive (max depth 3).

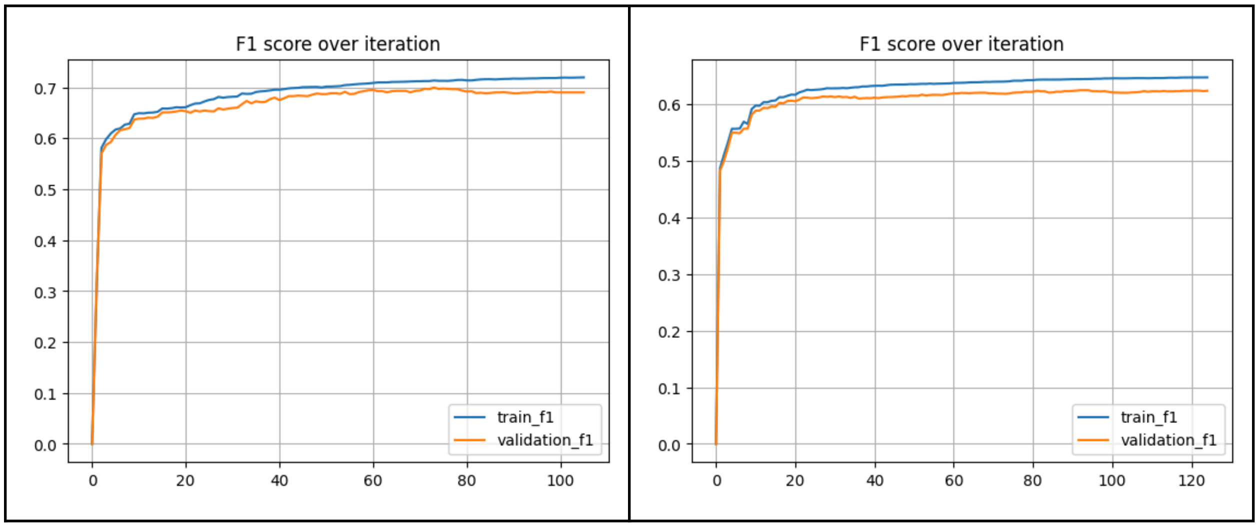


Table 2.2: Income and Bank Churn data Boosted DT learning curve (iteration)

Multiple boosting iterations means we can use early stopping mechanisms to decrease training time and prevent overfitting. For both datasets, the model will train up to 500 iterations and stop training once it stops seeing improvement in the 10% validation set for consecutive 32 iterations. In both cases, the model stopped training way before the max iterations.

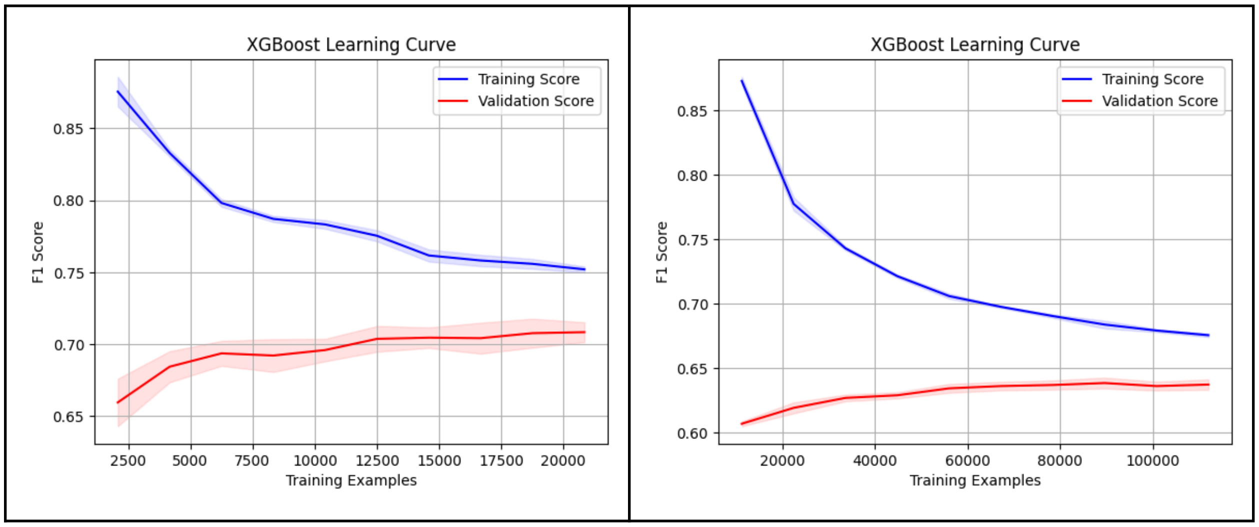
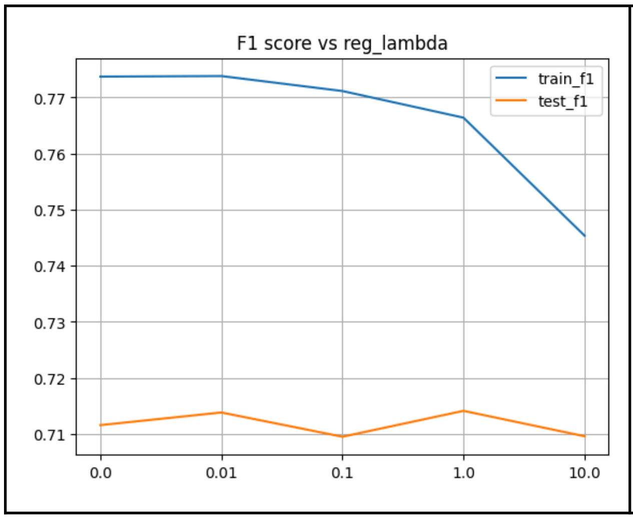


Table 2.3: Income and Bank Churn data Boosted DT learning curve (training sample)

For both datasets, the training score and validation score looks to be converging, suggesting that the model is quite good at generalizing with the given amount of dataset size. The small standard deviation of training and testing CV scores means that the model has low variance.



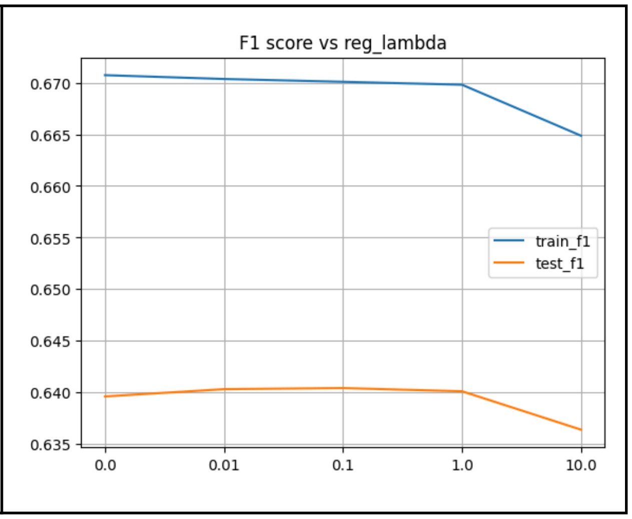


Table 2.4: Income and Bank Churn data Boosted DT lambda regularization term (L2)

We explore how boosted DT uses regularization term on objective function to prevent overfitting. Searching for an optimal regularization term is more nuanced as the value can be any real number. From the graphs, we can see that the testing set f1 score peaked at 1.0 and dropped at 10.0, so we can further search within this space with more computing / time resources. In both cases, L1(omitted) and L2 terms peak at 1.0 and drop at 10.0.

Neural Networks (Multilayer Perceptron - MLP)

The library used for the neural network is scikit-learn's `MLPClassifier`. MLP models are sensitive to feature scale so inputs are fit through a standard scaler. The MLP models have ReLU activation functions in all of their hidden layers and uses adam optimizer. Both models also use 3 hidden layers of 128, 64 and 16 neurons.

Hyper parameters	{'alpha': 1e-05, 'learning_rate_init': 0.001}	{'alpha': 0.01, 'learning_rate_init': 0.1}
Train F1 score	0.6966680378445085	0.6161906100082298
Test F1 score	0.6893172165958837	0.6155957378107846

Table 3.1: Income and Bank Churn data MLP hyper-params & scores

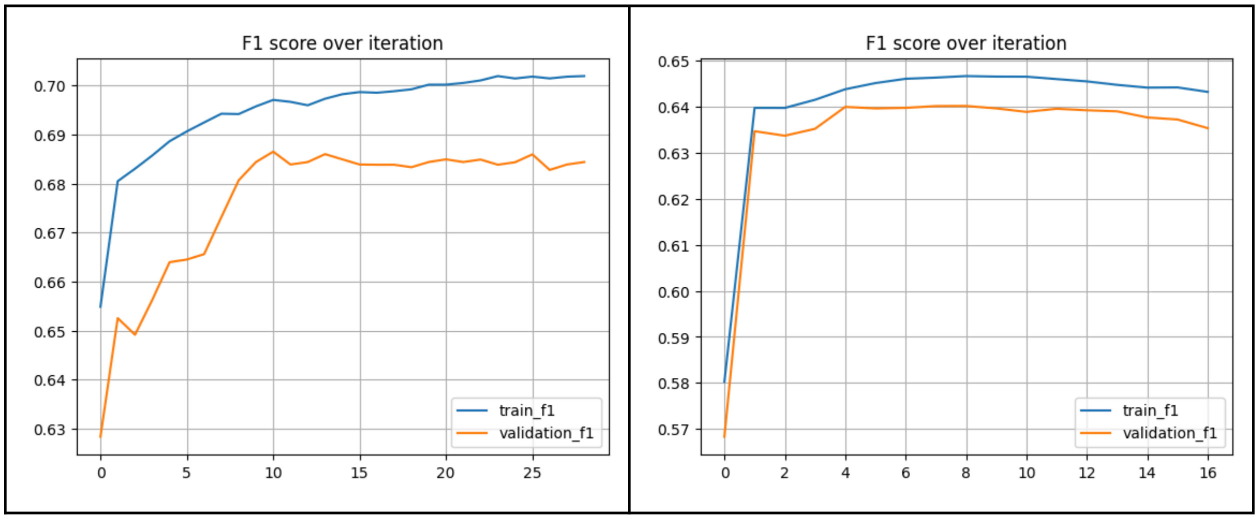


Table 3.2: Income and Bank Churn data MLP learning curve (iteration)

MLP are also trained on multiple iterations so we can also use early stopping mechanisms. For both datasets, the model will train up to 64 iterations and stop training once it stops seeing improvement in validation set accuracy for 10 iterations (scikit-learn package lacks customized early stopping metric). In both cases, the model stopped training way before the max iterations.

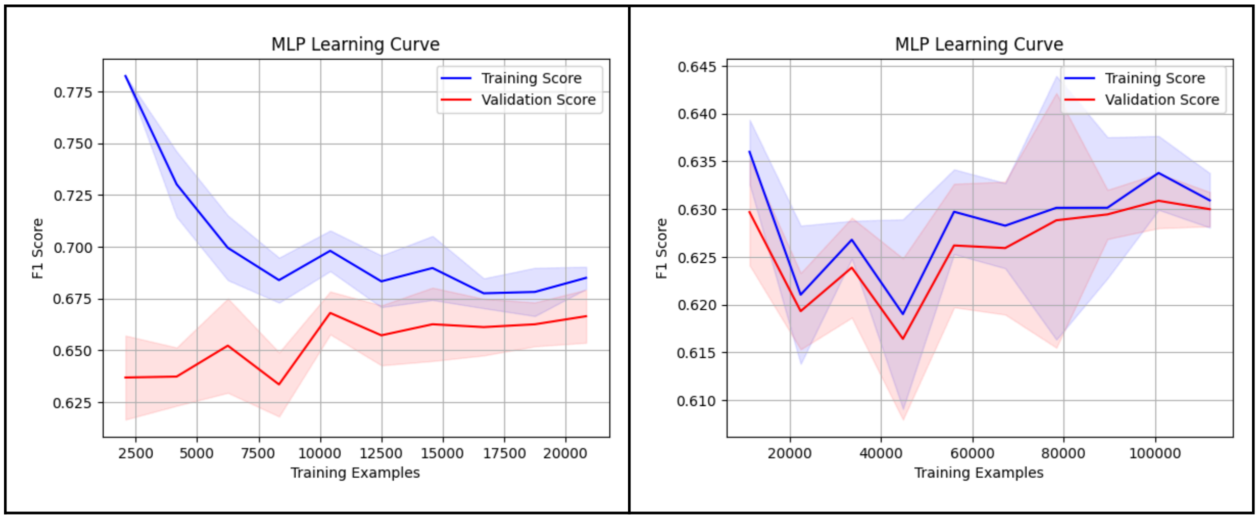
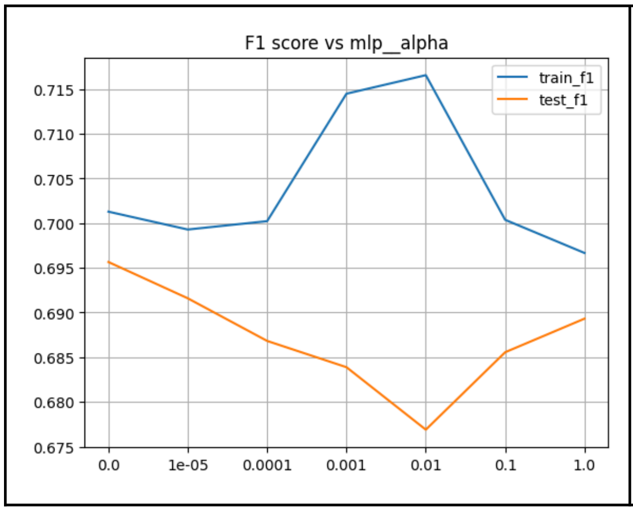


Table 3.3: Income and Bank Churn data MLP learning curve (training sample)

In both instances, the learning curves have low variance. Both datasets have a relatively high bias and low variance, so we could continue exploring a more complex model (increasing the number of hidden layers and number of neurons) to capture a better representation of the data.



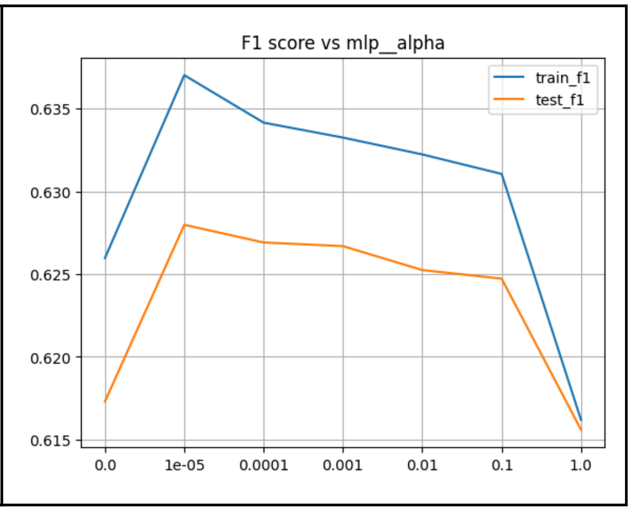


Table 3.4: Income and Bank Churn data MLP alpha regularization term (L2)

Apart from testing different numbers of neurons, the L2 regularization term can also be used to control the model complexity. Similar to the learning curve findings, the model preference for low alpha suggests that we can afford to make a more complex model without overfitting.

Support Vector Machine (SVM)

The library used for the SVM algorithm is scikit-learn's `SVC`. SVC models are sensitive to feature scale so inputs are put through a min-max scaler first. We set class weight to be balanced so the class is weighted based on its y distribution. We also use a polynomial kernel so we can observe how the model performs with various kernel degrees. The hyper parameters involved are kernel degree of polynomial and regularization C (L2).

Hyper parameters	{'C': 10, 'degree': 3}	{'C': 10, 'degree': 4}
Train F1 score	0.7041920636895136	0.6355340273503358
Test F1 score	0.6671630677587491	0.6228406909788867

Table 4.1: Income and Bank Churn data SVM hyper-params & scores

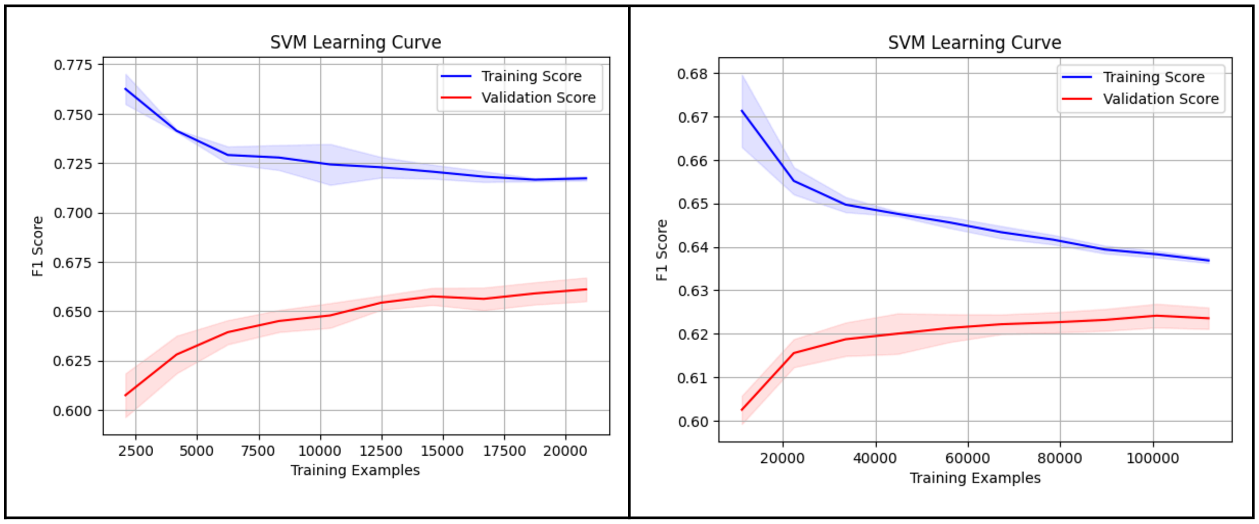


Table 4.2: Income and Bank Churn data SVM learning curve (training sample)

In both instances, the training score standard deviation tightens with increasing samples, suggesting that the model is able to fit the observed training data. A higher variance is observed from the income dataset and an increasing trend of validation score means that the model can benefit from more training data.

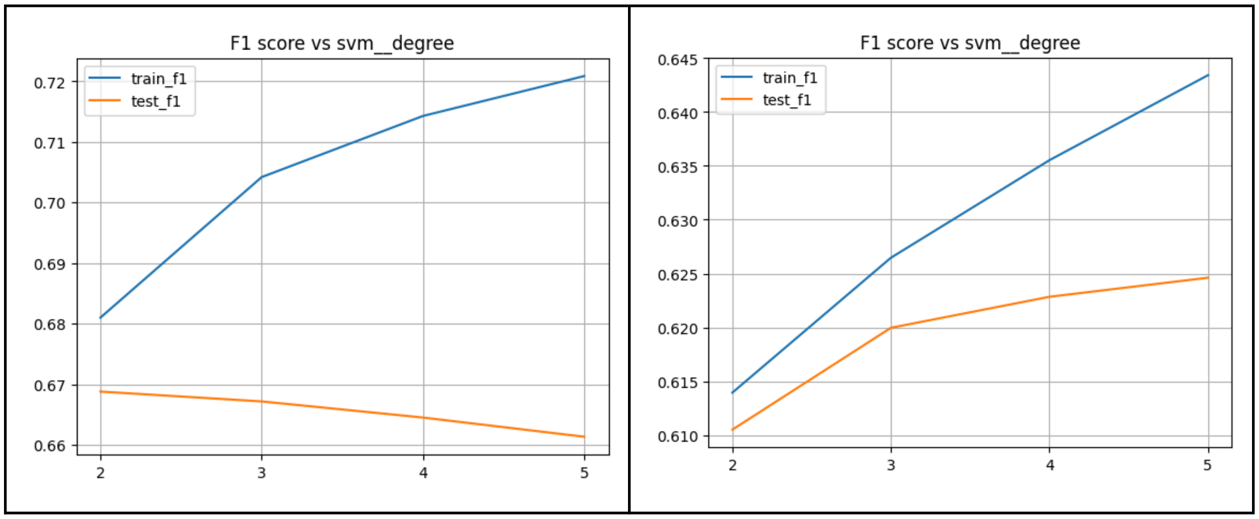


Table 4.3: Income and Bank Churn data SVM kernel polynomial

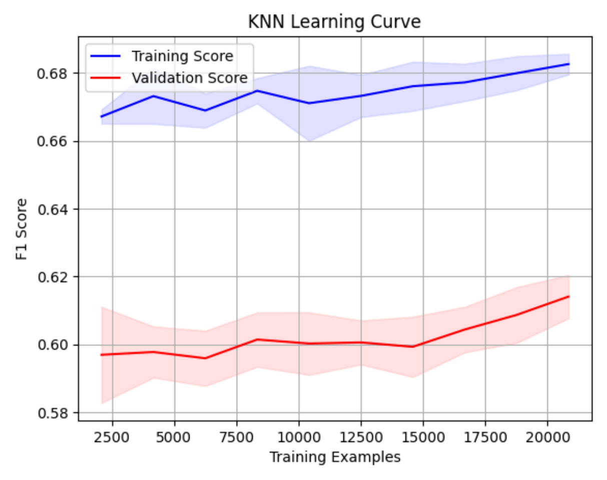
In both cases, the kernel polynomial suggested by CV did not yield the best scores on the testing set. Based on the testing set scores, the income dataset prefers a simpler model and bank dataset prefers a more complex one, with increasing testing set scores even up to 5 degrees.

K-Nearest Neighbor (KNN)

The library used for the SVM algorithm is scikit-learn's `KNeighborsClassifier`. KNN models are sensitive to feature scale so inputs are fit through a standard scaler. We experimented with manhattan (p=1) and euclidean distance (p=2). We are interested in finding the K value which is able to classify the dataset without overfitting.

Hyper parameters	{'n_neighbors': 9, 'p': 1}	{'n_neighbors': 11, 'p': 2}
Train F1 score	0.6843293954134816	0.6324302788844621
Test F1 score	0.6252170892671066	0.5832668050455053

Table 5.1: Income and Bank Churn data KNN hyper-params & scores



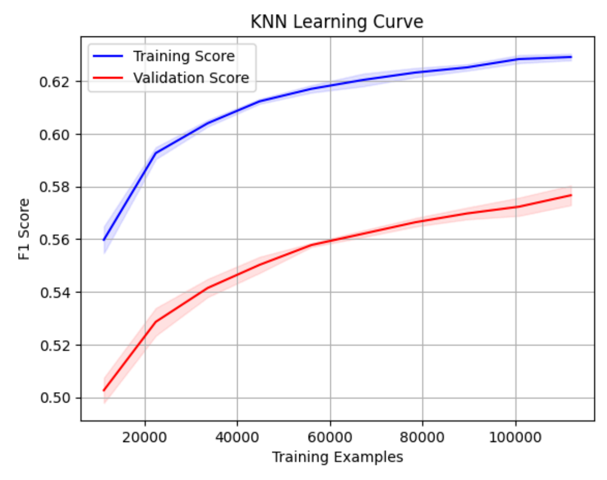
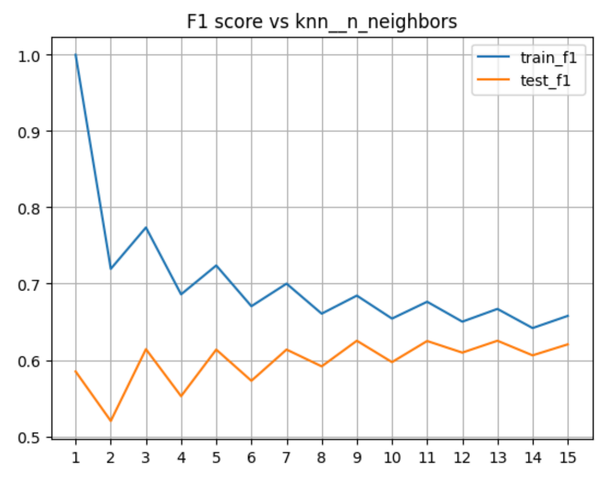


Table 5.2: Income and Bank Churn data KNN learning curve (training sample)

In both instances, the validation scores are still increasing suggests it needs more training data.



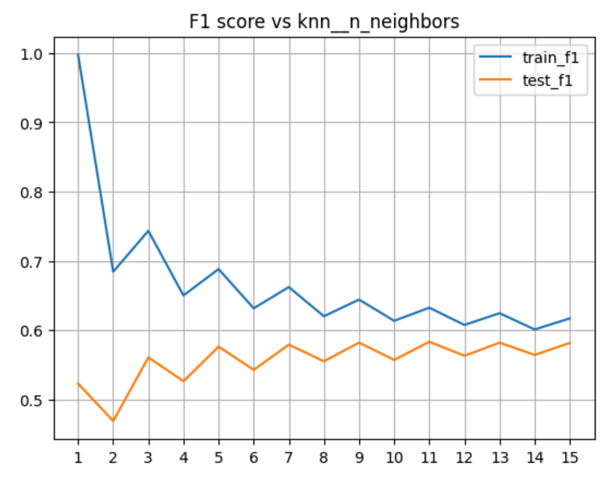


Table 5.3: Income and Bank Churn data KNN number of neighbors

As expected, a higher K value helps with overfitting and both datasets have the highest score when K equals to 9 and 11 respectively. Both scores follow a zig-zag fashion for odd and even K. An odd K tends to have a better score because there's a clear majority while even K can sometimes create a more ambiguous boundary when there's ties.

MODEL COMPARISON

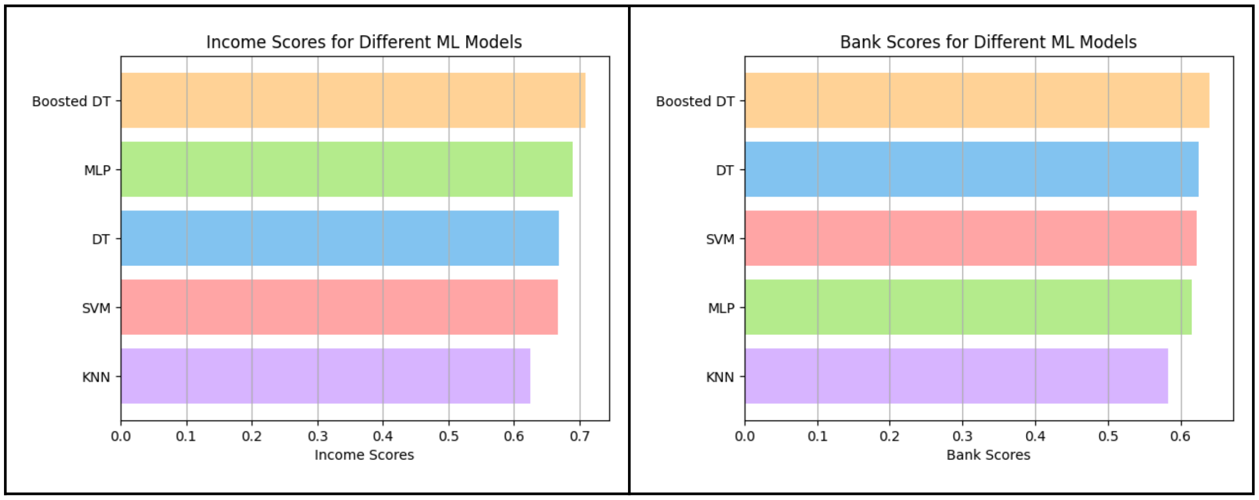


Table 6.1: Testing set F1 scores

Boosted DT outperforms all the other models for several reasons; it is an ensemble learning algorithm which makes it robust to overfitting. Boosted ensemble algorithm also 'learns' from its mistakes from previous iterations.

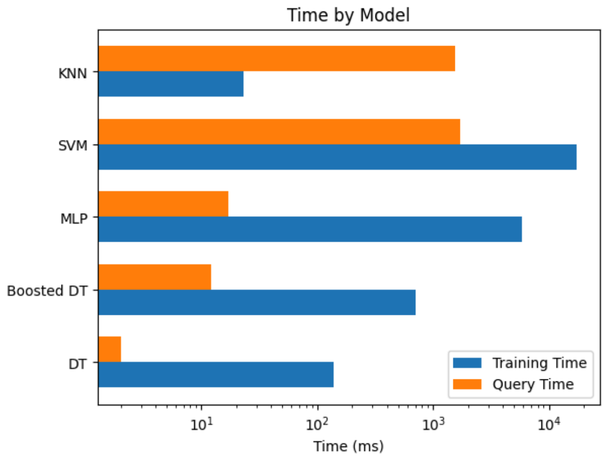


Table 6.2: Training and querying time for income dataset (log scale time)

All of the models except for KNN spent more time training than querying. KNN is an instance based learner so the main bulk of computation is only done during every query.

Bank churn data training and querying time omitted due to large dataset but the general trend should look the same as *table 6.2*.