

**School of InfoComm Technology**

**Machine Learning**

Diploma in Data Science (DS)

Diploma in Information Technology (IT)

October 2022 Semester

**INDIVIDUAL ASSIGNMENT 2**

(40% of Machine Learning Module)

**Deadline for Submission:**

**Presentation: 29th Jan 2023 (Sunday), 2359 Hours**

**Report: 11th Feb 2023 (Saturday), 2359 Hours**

|  |  |  |
| --- | --- | --- |
| Student Name | : | Lim Wee Liang Kelven |
| Student Number | : | S10221788K |

**Penalty for late submission:**

10% of the marks will be deducted every day after the deadline.

**NO** submission will be accepted after 18th Feb 2023, 23:59.

Contents

[Introduction 4](#_Toc127033822)

[HR Analytics 4](#_Toc127033823)

[Problem Understanding 4](#_Toc127033824)

[Build the Models 4](#_Toc127033825)

[Logistic Classifier 4](#_Toc127033826)

[DecisionTreeClassifier 5](#_Toc127033827)

[MLPClassifier 5](#_Toc127033828)

[RandomForestClassifier 5](#_Toc127033829)

[BaggingClassifier 5](#_Toc127033830)

[AdaBoostClassifier 6](#_Toc127033831)

[XGBClassifier 6](#_Toc127033832)

[SVC 6](#_Toc127033833)

[VotingClassifier 6](#_Toc127033834)

[Evaluate and Improve the Models 7](#_Toc127033835)

[Logistic Classifier 7](#_Toc127033836)

[DecisionTreeClassifier 8](#_Toc127033837)

[MLPClassifier 9](#_Toc127033838)

[RandomForestClassifier 10](#_Toc127033839)

[BaggingClassifier 11](#_Toc127033840)

[AdaBoostClassifier 12](#_Toc127033841)

[XGBClassifier 13](#_Toc127033842)

[SVC 14](#_Toc127033843)

[VotingClassifier 15](#_Toc127033844)

[Summary 15](#_Toc127033845)

[Airbnb 16](#_Toc127033846)

[Problem Understanding 16](#_Toc127033847)

[Build the Models 16](#_Toc127033848)

[Linear Regressor 16](#_Toc127033849)

[DecisionTreeRegressor 16](#_Toc127033850)

[MLPRegressor 16](#_Toc127033851)

[RandomForestRegressor 16](#_Toc127033852)

[BaggingRegressor 17](#_Toc127033853)

[AdaBoostRegressor 17](#_Toc127033854)

[XGBRegressor 17](#_Toc127033855)

[SVR 17](#_Toc127033856)

[VotingRegressor 18](#_Toc127033857)

[Evaluate and Improve the Models 18](#_Toc127033858)

[Linear Regressor 18](#_Toc127033859)

[DecisionTreeRegressor 18](#_Toc127033860)

[MLPRegressor 20](#_Toc127033861)

[RandomForestRegressor 21](#_Toc127033862)

[BaggingRegressor 22](#_Toc127033863)

[AdaBoostRegressor 23](#_Toc127033864)

[XGBRegressor 24](#_Toc127033865)

[SVR 25](#_Toc127033866)

[VotingRegressor 26](#_Toc127033867)

[Summary 27](#_Toc127033868)

[Conclusion 27](#_Toc127033869)

[HR Analytics 27](#_Toc127033870)

[Airbnb 28](#_Toc127033871)

[Reflection 28](#_Toc127033872)

[Further Improvements 28](#_Toc127033873)

[Skills Learnt 28](#_Toc127033874)

# Introduction

This report contains the documentation of the analysis and findings from two datasets: hr\_data.csv from HR Analytics, and listings.csv from Airbnb. The objectives of this report are:

* Building machine learning models
* Evaluating and improving the models’ performance
* Summarising the findings

This report will explain what models were used, why they were used, how they are improved, and what can be seen from the models.

Some of the approaches and explanations used in HR Analytics are also used in Airbnb. Hence, some explanations will be omitted when going through the Airbnb section to avoid repetition.

Both problems will utilise the classifier and regressor variants of the following learning models:

* Decision Tree
* Multi-layer Perceptron (MLP)
* Random Forest
* Bagging
* Ada Boost
* X Gradient Boost
* Support Vector Machines (SVM)
* Voting

This report aims to create a highly accurate Voting model for both problems. The Voter aggregates the results of all other models to create a more accurate and less bias score. Hence, to create an accurate Voting model, many other models must be created.

For all models, some codes and explanations are similar. Hence, to avoid repetition, those codes and explanations will be omitted.

# HR Analytics

## Problem Understanding

The company, HR Analytics, wants to identify employees who are most likely to get promoted efficiently. Since the target column, ‘is\_promoted’, has boolean values, with 0 representing no promotion and 1 representing a promotion, this becomes a classification problem.

## Build the Models

### Logistic Classifier

Using the train and test data from Assignment 1, a logistic classifier is built using the code shown below.

# Create the model  
lg = LogisticRegression(solver = 'lbfgs', max\_iter = 10000, random\_state = 0)  
  
# Fit the model  
lg.fit(hr\_xtrain, hr\_ytrain)

### DecisionTreeClassifier

A DecisionTreeClassifier is built using the code shown below.

# Create the model  
dtc = DecisionTreeClassifier(max\_depth = 2, random\_state = 0)  
  
# Fit the model  
dtc.fit(hr\_xtrain, hr\_ytrain)

### MLPClassifier

A MLPClassifier is built using the code shown below.

# Create the model  
mlpc = MLPClassifier(hidden\_layer\_sizes = (10,), max\_iter = 2000, random\_state = 0)  
  
# Fit the model  
mlpc.fit(hr\_xtrain, hr\_ytrain)

### RandomForestClassifier

A RandomForestClassifier is built using the code shown below.

# Create the model  
rfc = RandomForestClassifier(n\_estimators = 10, max\_depth = 4, random\_state = 0)  
  
# Fit the model  
rfc.fit(hr\_xtrain, hr\_ytrain)

### BaggingClassifier

A BaggingClassifier is built using the code shown below.

# Create the model  
bgc = BaggingClassifier(n\_estimators = 10, random\_state = 0)  
  
# Fit the model  
bgc.fit(hr\_xtrain, hr\_ytrain)

### AdaBoostClassifier

A AdaBoostClassifier is built using the code shown below.

# Create the model  
adbc = AdaBoostClassifier(DecisionTreeClassifier(max\_depth = 3), n\_estimators = 10, learning\_rate = 0.1, random\_state = 0)  
  
# Fit the model  
adbc.fit(hr\_xtrain, hr\_ytrain)

### XGBClassifier

A XGBClassifier is built using the code shown below.

# Create the model  
xgbc = XGBClassifier(n\_estimators = 20, learning\_rate = 0.1, eval\_metric = 'logloss', random\_state = 0)  
  
# Fit the model  
xgbc.fit(hr\_xtrain, hr\_ytrain)

### SVC

An SVC is built using the code shown below.

# Create the model  
svc = SVC(C = 0.8, kernel = 'rbf', random\_state = 0)  
# test C, kernal  
  
# Fit the model  
svc.fit(hr\_xtrain, hr\_ytrain)

### VotingClassifier

A VotingClassifier is built using the code shown below.

# Create the model  
vc = VotingClassifier(estimators = [('dtc', dtc), ('mlpc', mlpc), ('rfc', rfc),   
 ('bgc', bgc), ('adbc', adbc), ('xgbc', xgbc), ('svc', svc)],   
 voting = 'hard')  
  
# Fit the model  
vc.fit(hr\_xtrain, hr\_ytrain)

## Evaluate and Improve the Models

### Logistic Classifier

After building the model, the accuracy scores for the train and test sets are calculated. Accuracy scores closer to 1 imply an accurate model. The code used to calculate the scores is shown below.

# Accuracy  
train\_acc = lg.score(hr\_xtrain, hr\_ytrain)  
test\_acc = lg.score(hr\_xtest, hr\_ytest)  
  
print(f'Training accuracy: {train\_acc}')  
print(f'Testing accuracy: {test\_acc}')



We can see in the image above that the logistic classifier is highly accurate in the train data but fairly accurate in the test data. To improve the model, GridSearchCV from scikit-learn is used to find the best combination of model parameters. The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {"solver": ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'], max\_iter": [25, 50, 75, 100, 125, 150, 175]}  
  
gs = GridSearchCV(lg, param\_grid = param\_grid, scoring = 'accuracy', cv = 10, n\_jobs = -1)  
  
gs = gs.fit(hr\_xtrain, hr\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



The **.best\_score\_** method is the mean cross-validated score of the best estimator. For example, cv = 10 means the data is split into 10, and each is used to train the model 10 times. Afterwards, the best score of each is aggregated.

It is also shown that the best parameters for max\_iter is 25 and solver is lbfgs. Using these 2 results, a new model is built.

# Create the model  
lg = LogisticRegression(solver = 'lbfgs',   
 max\_iter = 25,   
 random\_state = 0)  
  
# Fit the model  
lg.fit(hr\_xtrain, hr\_ytrain)

This time, the scores look like this.



For this logistic classifier, there were no changes to the training and testing accuracy.

### DecisionTreeClassifier

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lg’ with the model’s name.



To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {"criterion" : ["gini", "entropy"],   
 'splitter': ['best', 'random'],  
 "min\_samples\_leaf" : [1, 2, 3, 4, 5],   
 "min\_samples\_split" : [2, 4, 6, 8, 10],   
 'max\_depth' : [None, 1, 2, 3, 4, 5]  
 }  
  
gs = GridSearchCV(dtc, param\_grid = param\_grid, scoring = 'accuracy', cv = 10, n\_jobs = -1)  
gs = gs.fit(hr\_xtrain, hr\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
dtc = DecisionTreeClassifier(criterion = 'entropy',   
 max\_depth = None,   
 min\_samples\_leaf = 5,   
 min\_samples\_split = 2,  
 splitter = 'random',  
 random\_state = 0)  
  
# Fit the model  
dtc.fit(hr\_xtrain, hr\_ytrain)



In this DecisionTreeClassifier, the training accuracy improved by 0.15 while the test accuracy remained unchanged.

### MLPClassifier

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lg’ with the model’s name.



To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {"activation": ['logistic', 'relu', 'identity','relu'],  
 'hidden\_layer\_sizes': [(50,),(100,),(150,)],  
 'learning\_rate':['constant','invscaling','adaptive'],  
 'max\_iter': [100,200,300],  
 'solver': ['sgd', 'adam']  
 }  
  
gs = GridSearchCV(mlpc, param\_grid = param\_grid, scoring = 'accuracy', cv = 5, n\_jobs = -1)  
gs = gs.fit(hr\_xtrain, hr\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
mlpc = MLPClassifier(activation = 'relu',   
 hidden\_layer\_sizes = (150,),   
 learning\_rate = 'constant',  
 max\_iter = 300,   
 solver = 'adam',   
 random\_state = 0)  
  
# Fit the model  
mlpc.fit(hr\_xtrain, hr\_ytrain)



In this MLPClassifier, the training accuracy improved by 0.02 while the test accuracy remained unchanged.

### RandomForestClassifier

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lg’ with the model’s name.



To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {"criterion" : ["gini", "entropy"],   
 "max\_depth": [1,2,3,4,5],   
 'min\_samples\_split':[2,3,4,5],  
 "min\_samples\_leaf" : [1,2,3,4,5],   
 "n\_estimators": [25,50,100,125,150]  
 }  
  
gs = GridSearchCV(estimator = rfc, param\_grid = param\_grid, scoring = 'accuracy', cv = 5, n\_jobs = -1)  
gs = gs.fit(hr\_xtrain, hr\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
rfc = RandomForestClassifier(criterion = 'gini',   
 n\_estimators = 150,   
 max\_depth = 5,   
 min\_samples\_leaf = 4,   
 min\_samples\_split = 2,  
 random\_state = 0)  
  
# Fit the model  
rfc.fit(hr\_xtrain, hr\_ytrain)



In this RandomForestClassifier, the training accuracy worsened by 0.01 while the test accuracy remained unchanged.

### BaggingClassifier

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lg’ with the model’s name.



An important feature to take note of is the training accuracy is 0.98. This implies that the model may be overfitted.

To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {'max\_samples': [1,2,3,4,5],  
 'max\_features': [1,2,3,4,5],  
 "n\_estimators": [5, 10, 15, 20, 25]  
 }  
  
gs = GridSearchCV(estimator = bgc, param\_grid = param\_grid, scoring = 'accuracy', cv = 5, n\_jobs = -1)  
gs = gs.fit(hr\_xtrain, hr\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
bgc = BaggingClassifier(n\_estimators = 25,   
 max\_features = 3,   
 max\_samples = 5,   
 random\_state = 0)  
  
# Fit the model  
bgc.fit(hr\_xtrain, hr\_ytrain)



In this BaggingClassifier, the training accuracy worsened by 0.35 while the test accuracy remained unchanged.

### AdaBoostClassifier

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lg’ with the model’s name.



To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {'learning\_rate': [0.1, 0.5, 1, 1.5, 2],  
 "n\_estimators": [25,50,75,100,125],  
 'algorithm': ["SAMME", "SAMME.R"]  
 }  
  
gs = GridSearchCV(estimator = adbc, param\_grid = param\_grid, scoring = 'accuracy', cv = 5, n\_jobs = -1)  
gs = gs.fit(hr\_xtrain, hr\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
adbc = AdaBoostClassifier(dtc,   
 n\_estimators = 125,   
 learning\_rate = 0.1,   
 algorithm = 'SAMME.R',   
 random\_state = 0)  
  
# Fit the model  
adbc.fit(hr\_xtrain, hr\_ytrain)



In this AdaBoostClassifier, the training accuracy improved by 0.2 while the test accuracy remained unchanged. An important feature to take note of is the training accuracy is 0.99. This high accuracy could be attributed to the AdaBoostClassifier being built using the improved DecisionTreeClassifier

### XGBClassifier

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lg’ with the model’s name.



To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {'learning\_rate': [0.1, 0.3, 0.5, 1],  
 'max\_depth':[2,4,6,8,10],  
 'eval\_metric': ['logloss', 'rmse', 'mae']  
 }  
  
gs = GridSearchCV(estimator = xgbc, param\_grid = param\_grid, scoring = 'accuracy', cv = 5, n\_jobs = -1)  
gs = gs.fit(hr\_xtrain, hr\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
xgbc = XGBClassifier(learning\_rate = 0.5,   
 eval\_metric = 'logloss',   
 max\_depth = 4,  
 random\_state = 0)  
  
# Fit the model  
xgbc.fit(hr\_xtrain, hr\_ytrain)



In this XGBClassifier, the training accuracy improved by 0.03 while the test accuracy remained unchanged.

### SVC

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lg’ with the model’s name.



To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {'C': [0.5, 0.75, 1, 1.25, 1.5],  
 "kernel": ['linear', 'poly', 'rbf', 'sigmoid'],  
 'gamma': ['scale', 'auto']  
 }  
  
gs = GridSearchCV(estimator = svc, param\_grid = param\_grid, scoring = 'accuracy', cv = 5, n\_jobs = -1)  
gs = gs.fit(hr\_xtrain, hr\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
svc = SVC(C = 1.5,   
 kernel = 'rbf',   
 gamma = 'scale',   
 random\_state = 0)  
  
# Fit the model  
svc.fit(hr\_xtrain, hr\_ytrain)



In this SVC, the training accuracy improved by 0.06 while the test accuracy remained unchanged.

### VotingClassifier

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lg’ with the model’s name.



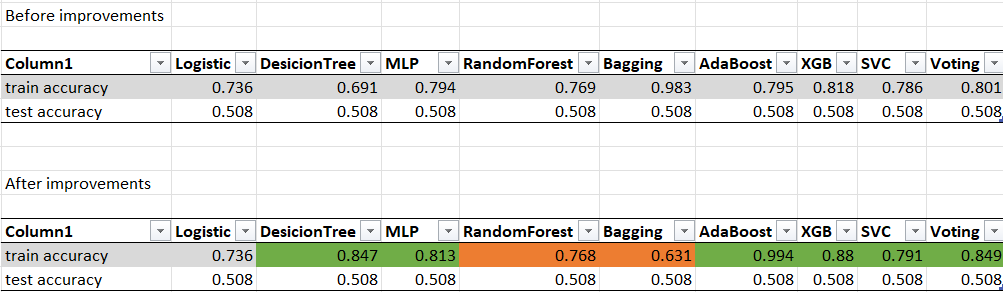
To improve the VotingClassifier, a new one is built using the improved models.

# Create the model  
vc = VotingClassifier(estimators = [('dtc', dtc), ('mlpc', mlpc), ('rfc', rfc), ('bgc', bgc), ('adbc', adbc), ('xgbc', xgbc), ('svc', svc)], voting = 'hard')  
  
# Fit the model  
vc.fit(hr\_xtrain, hr\_ytrain)



In this final VotingClassifier, the training accuracy improved by 0.04 while the test accuracy remained unchanged.

## Summary



The image above shows a table with all the train and test accuracy scores of all the models before and after their improvements. It is observed that the RandomForestClassifier and BaggingClassifier models worsened after using GridSearchCV while the others improved.

An important feature to take note of is the test accuracy scores of all the models before and after improvements remain unchanged. This could because the chosen features are bad in predicting promotion status. A possible improvement is to add weights the training data. Thus, some features could be manually adjusted to affect promotion more than others.

Ultimately, through using GridSearchCV, most models had better scores which in turn created a better VotingClassifier.

# Airbnb

## Problem Understanding

The company, Airbnb, wants to predict the rental prices of listed properties. Since the target column, ‘price, has continuous values, this becomes a regression problem.

## Build the Models

### Linear Regressor

Using the train and test data from Assignment 1, a linear regressor is built using the code shown below.

# Create the model  
lm = LinearRegression()  
  
# Fit the model  
lm.fit(airbnb\_xtrain, airbnb\_ytrain)

### DecisionTreeRegressor

A DecisionTreeRegressor is built using the code shown below.

# Create the model  
dtr = DecisionTreeRegressor(max\_depth = 2,   
 random\_state = 0)  
  
# Fit the model  
dtr.fit(airbnb\_xtrain, airbnb\_ytrain)

### MLPRegressor

A MLPRegressor isbuilt using the code shown below.

# Create the model  
mlpr = MLPRegressor(hidden\_layer\_sizes = (10,),   
 max\_iter = 2000,   
 random\_state = 0)  
  
# Fit the model  
mlpr.fit(airbnb\_xtrain, airbnb\_ytrain)

### RandomForestRegressor

A RandomForestRegressor is built using the code shown below.

# Create the model  
rfr = RandomForestRegressor(n\_estimators = 10,   
 max\_depth = 4,   
 random\_state = 0)  
  
# Fit the model  
rfr.fit(airbnb\_xtrain, airbnb\_ytrain)

### BaggingRegressor

A BaggingRegressor is built using the code shown below.

# Create the model  
bgr = BaggingRegressor(n\_estimators = 10,   
 random\_state = 0)  
  
# Fit the model  
bgr.fit(airbnb\_xtrain, airbnb\_ytrain)

### AdaBoostRegressor

A AdaBoostRegressor is built using the code shown below.

# Create the model  
adbr = AdaBoostRegressor(dtr,   
 n\_estimators = 10,   
 learning\_rate = 0.1,   
 random\_state = 0)  
  
# Fit the model  
adbr.fit(airbnb\_xtrain, airbnb\_ytrain)

### XGBRegressor

A XGBRegressor is built using the code shown below.

# Create the model  
xgbr = XGBRegressor(n\_estimators = 20,   
 learning\_rate = 0.1,   
 eval\_metric = 'logloss',   
 random\_state = 0)  
  
# Fit the model  
xgbr.fit(airbnb\_xtrain, airbnb\_ytrain)

### SVR

A SVR is built using the code shown below.

# Create the model  
svr = SVR(C = 0.8, kernel = 'rbf')  
  
# Fit the model  
svr.fit(airbnb\_xtrain, airbnb\_ytrain)

### VotingRegressor

A VotingRegressor is built using the code shown below.

# Create the model  
vr = VotingRegressor(estimators = [('dtr', dtr), ('mlpr', mlpr), ('rfr', rfr), ('bgr', bgr), ('adbr', adbr), ('xgbr', xgbr), ('svr', svr)])  
  
# Fit the model  
vr.fit(airbnb\_xtrain, airbnb\_ytrain)

## Evaluate and Improve the Models

### Linear Regressor

After building the model, the accuracy and MSE scores for the train and test sets are calculated. Accuracy scores closer to 1 and MSE scores closer to 0 imply an accurate model. The code used to calculate the scores is shown below.

# Accuracy  
train\_acc = lm.score(airbnb\_xtrain, airbnb\_ytrain)  
test\_acc = lm.score(airbnb\_xtest, airbnb\_ytest)  
  
print(f'Training accuracy: {train\_acc}')  
print(f'Testing accuracy: {test\_acc}')  
print()  
  
# MSE  
train\_mse = mean\_squared\_error(lm.predict(airbnb\_xtrain), airbnb\_ytrain)  
test\_mse = mean\_squared\_error(lm.predict(airbnb\_xtest), airbnb\_ytest)  
  
print(f'the training mean squared error is: {train\_mse}')  
print(f'the testing mean squared error is: {test\_mse}')

Text

Description automatically generated

GridSearchCV cannot be used on a Linear Regressor. Thus, it cannot be improved.

### DecisionTreeRegressor

After building the model, the accuracy and MSE scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lm’ with the model’s name.

Text

Description automatically generated

We can see in the image above that the DecisionTreeRegressor is inaccurate for both the train and test data. To improve the model, GridSearchCV from scikit-learn is used to find the best combination of model parameters. The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {'criterion': ['friedman\_mse', 'poisson'],  
 "splitter": ['best', 'random'],  
 'max\_depth': [2, 3, 4, 5],  
 'min\_samples\_split': [2,3,4],  
 'min\_samples\_leaf': [1,2,3]  
 }  
  
gs = GridSearchCV(estimator = dtr, param\_grid = param\_grid, scoring = 'r2', cv = 5, n\_jobs = -1, error\_score = 'raise')  
gs = gs.fit(airbnb\_xtrain, airbnb\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



For regressors, **scoring = ‘accuracy’** is changed to **scoring = ‘r2’**.

From here, a new model is built using the best parameters shown.

# Create the model

dtr = DecisionTreeRegressor(criterion = 'poisson',   
 splitter = 'random',   
 max\_depth = 5,   
 min\_samples\_leaf = 1,  
 min\_samples\_split = 3,  
 random\_state = 0)  
  
# Fit the model  
dtr.fit(airbnb\_xtrain, airbnb\_ytrain)

This time, the scores look like this.

Text

Description automatically generated

For this DecisionTreeRegressor, the training accuracy improved by 0.01 and test accuracy improved by 0.03. The train MSE improved by about 1000 and test MSE improved by about 5000.

### MLPRegressor

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lm’ with the model’s name.

Text

Description automatically generated

To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {"activation": ['logistic', 'relu', 'identity','tanh'],  
 'hidden\_layer\_sizes': [(90,), (100,), (110,)],  
 'max\_iter': [100, 200, 300],  
 'solver': ['sgd', 'adam'],  
 'learning\_rate':['constant', 'invscaling', 'adaptive']  
 }  
  
gs = GridSearchCV(mlpr, param\_grid = param\_grid, scoring = 'r2', cv = 5, n\_jobs = -1)  
gs = gs.fit(airbnb\_xtrain, airbnb\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
mlpr = MLPRegressor(activation = 'relu',   
 hidden\_layer\_sizes = (90,),   
 learning\_rate = 'adaptive',  
 max\_iter = 100,   
 solver = 'sgd',   
 random\_state = 0)  
  
# Fit the model  
mlpr.fit(airbnb\_xtrain, airbnb\_ytrain)

Text

Description automatically generated

For this MLPRegressor, the training accuracy improved by 0.11 and test accuracy worsened by a lot. The train MSE improved by about 9000 and test MSE worsened by a lot.

### RandomForestRegressor

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lm’ with the model’s name.

Text

Description automatically generated

To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {'criterion': ['friedman\_mse', 'absolute\_error', 'poisson'],  
 'n\_estimators': [90, 100, 200, 300],  
 "max\_depth": [1, 2, 3, 4, 5],   
 "min\_samples\_leaf" : [1, 2, 3, 4, 5]  
 }  
  
gs = GridSearchCV(rfr, param\_grid = param\_grid, scoring = 'r2', cv = 5, n\_jobs = -1)  
gs = gs.fit(airbnb\_xtrain, airbnb\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
rfr = RandomForestRegressor(criterion = 'friedman\_mse',   
 max\_depth = 5,  
 min\_samples\_leaf = 4,   
 n\_estimators = 90,   
 random\_state = 0)  
  
# Fit the model  
rfr.fit(airbnb\_xtrain, airbnb\_ytrain)

Text

Description automatically generated

For this RandomForestRegressor, the training accuracy worsened by 0.06 and test accuracy improved by 1.5. The train MSE worsened by about 8000 and test MSE improved by about 200,000.

### BaggingRegressor

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lm’ with the model’s name.

Text

Description automatically generated

To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {'max\_samples': [1, 2, 3, 4, 5],  
 'max\_features': [1, 2, 3, 4, 5],  
 "n\_estimators": [5, 10, 15, 20, 25]  
 }  
  
gs = GridSearchCV(estimator = bgr, param\_grid = param\_grid, scoring = 'r2', cv = 5, n\_jobs = -1)  
gs = gs.fit(airbnb\_xtrain, airbnb\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
bgr = BaggingRegressor(max\_features = 4,   
 max\_samples = 4,   
 n\_estimators = 5,   
 random\_state = 0)  
  
# Fit the model  
bgr.fit(airbnb\_xtrain, airbnb\_ytrain)

Text

Description automatically generated

For this BaggingRegressor, the training accuracy worsened by 0.74 and test accuracy improved by 1.91. The train MSE worsened by about 75,000 and test MSE reduced by more than half.

### AdaBoostRegressor

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lm’ with the model’s name.

Text

Description automatically generated

To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {'learning\_rate': [0.1, 0.5, 1, 1.5, 2],  
 "n\_estimators": [25, 50, 75, 100, 125],  
 'loss': ["linear", "square", 'exponential']  
 }  
  
gs = GridSearchCV(estimator = adbr, param\_grid = param\_grid, scoring = 'r2', cv = 5, n\_jobs = -1)  
gs = gs.fit(airbnb\_xtrain, airbnb\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
adbr = AdaBoostRegressor(dtr, learning\_rate = 0.1,   
 loss = 'exponential',   
 n\_estimators = 25,   
 random\_state = 0)  
  
# Fit the model  
adbr.fit(airbnb\_xtrain, airbnb\_ytrain)

Text

Description automatically generated

For this AdaBoostRegressor, the training accuracy improved by 0.21 and test accuracy improved by 1.15. The train MSE improved by about 21,000 and test MSE improved by about 400.

### XGBRegressor

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lm’ with the model’s name.

Text

Description automatically generated

To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {'learning\_rate': [0.1, 0.3, 0.5, 1, 1.5],  
 'max\_depth':[1,4,6,8,10],  
 'sampling\_method':['uniform','subsample','gradient\_based'],  
 "n\_estimators": [50, 100, 200, 300, 400],  
 'eval\_metric': ["logloss", "mae"]  
 }  
  
gs = GridSearchCV(estimator = xgbr, param\_grid = param\_grid, scoring = 'r2', cv = 5, n\_jobs = -1)  
gs = gs.fit(airbnb\_xtrain, airbnb\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
xgbr = XGBRegressor(n\_estimators = 50,   
 learning\_rate = 0.1,   
 eval\_metric = 'logloss',   
 max\_depth = 1,  
 sampling\_method = 'uniform',  
 random\_state = 0)  
  
# Fit the model  
xgbr.fit(airbnb\_xtrain, airbnb\_ytrain)

Text

Description automatically generated

For this XGBRegressor, the training accuracy worsened by 0.44 and test accuracy also worsened by 0.01. The train MSE doubled to 100,000 and test MSE also worsened by about 2000.

### SVR

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lm’ with the model’s name.

Text

Description automatically generated

To improve the model, GridSearchCV The parameters used are found on the scikit-learn website. The code used to create the GridSearchCV model is shown below.

# Create a GridSearch model to find the best parameters  
  
param\_grid = {'C': [0.5, 0.75, 1, 1.25, 1.5],  
 "kernel": ['linear', 'poly', 'rbf', 'sigmoid'],  
 'gamma': ['scale', 'auto'],  
 'degree': [1,2,3,4,5]  
 }  
  
gs = GridSearchCV(estimator = svr, param\_grid = param\_grid, scoring = 'r2', cv = 5, n\_jobs = -1)  
gs = gs.fit(airbnb\_xtrain, airbnb\_ytrain)  
print(gs.best\_score\_)  
print(gs.best\_params\_)



Using the best combination of parameters, a new model is built.

# Create the model  
svr = SVR(C = 1.5,   
 degree = 1,  
 gamma = 'scale',   
 kernel = 'linear')  
  
# Fit the model  
svr.fit(airbnb\_xtrain, airbnb\_ytrain)

Text

Description automatically generated

For this SVR, the training accuracy improved by 0.01 and test accuracy worsened by 264. The train MSE improved by about 500 and test MSE worsened by a lot.

### VotingRegressor

After building the model, the accuracy scores for the train and test sets are calculated. The code used to calculate the scores are similar, the only changes needed are to replace all instances of ‘lm’ with the model’s name.

Text

Description automatically generated

To improve the VotingRegressor, a new one is built using the improved models.

# Create the model  
vr = VotingRegressor(estimators = [('dtr', dtr), ('mlpr', mlpr), ('rfr', rfr),   
 ('bgr', bgr), ('adbr', adbr), ('xgbr', xgbr), ('svr', svr)])  
  
# Fit the model  
vr.fit(airbnb\_xtrain, airbnb\_ytrain)

Text

Description automatically generated

In this final VotingRegressor, Both the training accuracy and test accuracy worsened. Both the train MSE and test MSE also worsened.

## Summary

Table

Description automatically generated with medium confidence

The image above shows a table with all the train and test R2 scores of all the models before and after their improvements. It is observed that there is a mixture of improvements and losses. Some models had improved train and test accuracy scores, some improved in one but worsened in another, and some had worser scores in both.

Graphical user interface

Description automatically generated with low confidence

The image above shows a table with all the train and test MSEs of all the models before and after their improvements. Like the R2 scores, the models had a mixture of improvements and losses.

Ultimately, through using GridSearchCV, most models had worser scores which in turn created a worser VotingRegressor.

# Conclusion

## HR Analytics

In the HR Analytics dataset, there was an overall improvement in all the models after using GridSearchCV. The purpose in creating many models was to create an accurate and unbiased VotingClassifier. However, it cannot be concluded if this was successful for there was an issue regarding the test accuracy scores. Across all the models, the test accuracy remained the same at 0.508. The test accuracy scores were expected to be different across all the models. Hence, unless the issue with the test scores is resolved, the final VotingClassifier cannot be concluded as accurate.

## Airbnb

In the Airbnb dataset, there was an overall worsening in all the models after using GridSearchCV. Like HR Analytics, the purpose in creating many models was to create an accurate and unbiased VotingRegressor. Since the VotingRegressor is dependent on other models, the worser scores from them created a worser VotingRegressor than the one before GridSearchCV. Thus, the final VotingRegressor is inaccurate.

# Reflection

## Further Improvements

Changes were also made in the data preparation portion in Assignment 1 notebook. These changes include removing outliers using IQR range instead of a YeoJohnsonTransformer, and not removing any columns. However, these changes had minimal effect on the accuracy scores.

If more time was allotted, more experimentation could have been carried out to discover why the test scores for HR Analytics remained unchanged across all models.

For Airbnb, many changes were made to the data preparation portion in Assignment 1 notebook. These changes include sampling the column ‘neighbourhood\_group’, removing outliers using IQR range instead of a YeoJohnsonTransformer, and not removing any columns. However, these changes had minimal effect on the R2 and MSE scores.

If more time was allotted, more research could have been performed to discover why the MSE scores were different to other student’s scores. References could have been made to their data preparation methods which could have improved the VotingRegressor model.

## Skills Learnt

Throughout this assignment, I have grown more knowledgeable about the different types of learning models and how to implement them. I have also grown accustomed to using GridSearchCV to adjust and fine-tune different hyperparameters. This skill will be highly useful in the workforce. I am also able to evaluate model performance, infer their meanings, and document the findings.

A skill that I could have learnt better is data preparation. Other students had different, and sometimes better scores than my models. This is mainly due to their different way of preparing data. By preparing the data in a more appropriate fashion, the scores for both HR Analytics and Airbnb would turn out more reasonable. For example, in Airbnb, the maximum value in the ‘price’ column is 10,000, but the MSE is more than 10,000 and can be 6-digits long.