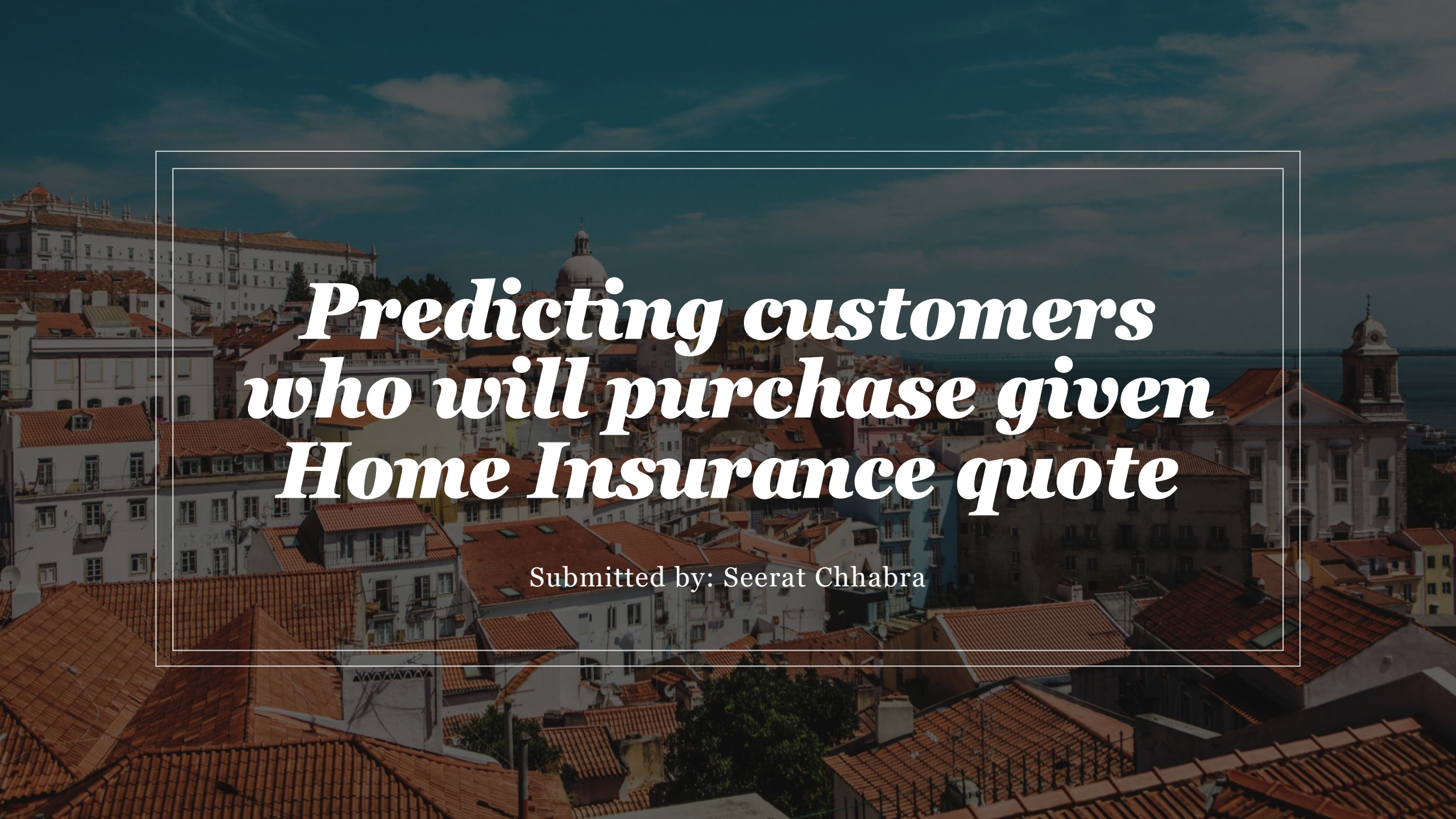


Predicting customers who will purchase given Home Insurance quote

Submitted by: Seerat Chhabra

Data Summary

Training Data

- 65,000 data points
- 596 attributes after 1 hot encoding
- Includes Quote Conversion Flag
  - 1 - Purchased
  - 0 – Did not purchase
- Unbalanced data:

0	81.35%
1	18.86%

Test Data

- 173,836 data points
- 595 attributes after 1 hot encoding
- No Target column
  - Predict whether customer will purchase (1) or not (0)

Data Pre-Processing

- **Removed 34 attributes with 0 standard deviation**

Data -[65000,562]

Data -[65000,561]

- **Removed Quote Number column**

Train-Test split

80%

Training Data

- 52000 data points
- 561 attributes
- Unbalanced data:

0	42183
1	9817

20%

Test Data

- 13000 data points
- 560 attributes


MY DATASET HAS
IMBALANCED CLASSES
SMOTE

SMOTE

```
[ ]  # Resample the minority class.
     sm = SMOTE(sampling_strategy='minority', random_state=7)

     # Fit the model to generate the data.
     oversampled_trainX, oversampled_trainY = sm.fit_sample(X_Train, Y_Train)
     print('Resampled dataset shape %s' % Counter(oversampled_trainY))
```

Before SMOTE

0	42183
1	9817

Oversampling using SMOTE

After SMOTE

0	42183
1	**42183**

DECISION TREE

DecisionTreeClassifier()

1 Default model

```
#Make default decision tree using training dataset
clf = DecisionTreeClassifier()
clf.fit(X_Train, Y_Train)
#Prediction on test data created from split
pred=pd.DataFrame(clf.predict(X_Test_1),columns=["Prediction"])
print("-----------------------------Decision Tree - Test Data created from split-----------------------------------\n")
print("Accuracy Score on test data:",accuracy_score(Y_Test_1,pred["Prediction"]))
print("Confusion Matrix on test data\n", confusion_matrix(Y_Test_1,pred["Prediction"]))
print("Classification report\n", classification_report(Y_Test_1,pred["Prediction"]))
```

Precision of 1	Accuracy
0.68	0.88

2 Hyperparameter Tuning

```
print("RandomizedSearchCV-Decision tree")
parameters={'max_depth': range(10,100,10), 'min_samples_leaf' : range(2,25,5), 'criterion':['gini','entropy']}
clf_random = RandomizedSearchCV(clf,parameters,n_iter=20, cv=5, scoring ="precision")
clf_random.fit(X_Train, Y_Train)
rand_parm=clf_random.best_params_
print(rand_parm)
```

```
RandomizedSearchCV-Decision tree
{'min_samples_leaf': 22, 'max_depth': 20, 'criterion': 'gini'}
```

Optimize Precision score!
Want to correctly predict more customers who will purchase at quoted price!

3 Tuned model

```
#Using the parameters obtained from HyperParameterTuning in the DecisionTreeClassifier
clf_r = DecisionTreeClassifier(**rand_parm)
clf_r.fit(X_Train,Y_Train)
# Random model prediction on test data
pred_r=pd.DataFrame(clf_r.predict(X_Test_1),columns=["Prediction"])
print("Accuracy Score on test data using Decision Tree - Random Search:",accuracy_score(Y_Test_1,pred_r["Prediction"]))
print("Balanced Accuracy Score on test data using Decision Tree - Random Search:",balanced_accuracy_score(Y_Test_1,pred_r["Prediction"]
print("Confusion Matrix on test data using Decision Tree - Random Search\n", confusion_matrix(Y_Test_1,pred_r["Prediction"]))
```

0.78	0.90

Using tuned model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_DT.csv	a few seconds ago	0 seconds	4 seconds	0.82131

Complete

Jump to your position on the leaderboard

Kaggle score | **82.13%**

RANDOM FOREST

RandomForestClassifier()

1 Default model

```
#Random forest
rfc = RandomForestClassifier()
rfc.fit(X_Train, Y_Train)
# prediction on test data
pred_rf=pd.DataFrame(rfc.predict(X_Test_1),columns=["Prediction"])
print("----------------------------Random Forest: Test Data----------------------------------------------\n")
print("Accuracy Score on test data using Random Forest:",accuracy_score(Y_Test_1,pred_rf["Prediction"]))
print("Balanced Accuracy Score on test data using Random Forest:",balanced_accuracy_score(Y_Test_1,pred_rf["Prediction"]))
print("Confusion Matrix on test data using Random Forest\n", confusion_matrix(Y_Test_1,pred_rf["Prediction"]))
print("Classification report on test data using Random Forest\n", classification_report(Y_Test_1,pred_rf["Prediction"]))
```

Precision of 1	Accuracy
0.81	0.89

2 Hyperparameter Tuning

```
#Hyperparameter tuning done for random forest classifier
#RANDOM SEARCH-----------------------------------------

print("RandomizedSearchCV-Decision tree")
parameters={'max_features': range(10,50,5) ,'n_estimators':[25,30,40,50], 'criterion':['gin
rfc_random = RandomizedSearchCV(rfc,parameters,n_iter=20,cv=5, scoring ='precision')
rfc_random.fit(X_Train, Y_Train)
rand_parm_rfc=rfc_random.best_params_
print(rand_parm_rfc)
```

```
RandomizedSearchCV-Decision tree
{'n_estimators': 50, 'max_features': 45, 'criterion': 'entropy'}
```

Optimize Precision score!
Want to correctly predict more customers who will purchase at quoted price!

3 Tuned model

```
#Using the parameters obtained from HyperParameterTuning in the randomforestclassifier
rfc_r = RandomForestClassifier(**rand_parm_rfc)
rfc_r.fit(X_Train,Y_Train)
rfc_r_predict = rfc_r.predict(X_Test_1)
# Random model prediction on test data
pred_r_rfc=pd.DataFrame(rfc_r.predict(X_Test_1),columns=["Prediction"])
print("Accuracy Score on test data using Random Forest - Random Search:",accuracy_score(Y_Test_1,pred_r_rfc["Prediction"]))
print("Balanced Accuracy Score on test data using Random Forest - Random Search:",balanced_accuracy_score(Y_Test_1,pred_r_rfc["Prediction"]))
print("Confusion Matrix on test data using Random Forest - Random Search\n", confusion_matrix(Y_Test_1,pred_r_rfc["Prediction"]))
print("Classification report on test data using Random Forest - Random Search\n", classification_report(Y_Test_1,pred_r_rfc["Prediction"]))
```

0.88	0.92

Using tuned model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_RF.csv	a few seconds ago	0 seconds	1 seconds	0.81189

Complete

Jump to your position on the leaderboard ▾

Kaggle score	81.18%

KNearest Neighbor (KNN)

KNeighborsClassifier()

1 Default model

```
#KNN classifier
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_Train, Y_Train)
#Prediction on test data created from split
pred_knn=pd.DataFrame(knn_clf.predict(X_Test_1),columns=["Prediction"])
print("----------------------------KNN - Test Data created from split----------------------------------\n")
print("Accuracy Score on training data using KNN:",accuracy_score(Y_Test_1,pred_knn["Prediction"]))
print("Confusion Matrix on training data using KNN\n", confusion_matrix(Y_Test_1,pred_knn["Prediction"]))
print("Classification report\n", classification_report(Y_Test_1,pred_knn["Prediction"]))
```

Precision of 1	Accuracy
0.23	0.52

2 Hyperparameter Tuning

```
#Random Hyperparameter tuning for KNN
print("RandomizedSearchCV- KNN")
parameters={'algorithm': ['auto', 'kd_tree'], 'n_neighbors' : range(5,10,1), 'p': range(2,5,1)}
knn_random = RandomizedSearchCV(knn_clf,parameters,n_iter=10, cv=3, scoring ="precision")
knn_random.fit(X_Train, Y_Train)
knn_rand_parm=knn_random.best_params_
print(knn_rand_parm)
```

```
RandomizedSearchCV- KNN
{'p': 4, 'n_neighbors': 6, 'algorithm': 'auto'}
```

Optimize Precision score!
Want to correctly predict more customers who will purchase at quoted price!

3 Tuned model

```
#Using the parameters obtained from HyperParameterTuning in the KNN
knn_clf_r = KNeighborsClassifier(**knn_rand_parm)
# random model prediction on test data
pred_knn=pd.DataFrame(knn_clf_r.predict(X_Test_1),columns=["Prediction"])
print("Accuracy Score on test data using SVM - random Search:",accuracy_score(Y_Test_1,pred_knn["Prediction"]))
print("Balanced Accuracy Score on test data using SVM - random Search:",balanced_accuracy_score(Y_Test_1,pred_knn["Prediction"]))
print("Confusion Matrix on test data using SVM - random Search\n", confusion_matrix(Y_Test_1,pred_knn["Prediction"]))
print("Classification report on test data using SVM - random Search\n", classification_report(Y_Test_1,pred_knn["Prediction"]))
```

0.24	0.56

Using tuned model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_KNN.csv	a few seconds ago	0 seconds	1 seconds	0.58227

Complete

Jump to your position on the leaderboard

Kaggle score | **58.27%**

Multi-Layer Perceptron (MLP)

MLPClassifier()

1 Default model

```
# neural networks - mlp
nn_clf = MLPClassifier()
nn_clf.fit(X_Train, Y_Train)
#Prediction on test data created from split
pred=pd.DataFrame(nn_clf.predict(X_Test_1),columns=["Prediction"])
print("-----------------------------Neural Networks - Test Data created from split-----------------------------------\n")
print("Accuracy Score on training data using Neural networks:",accuracy_score(Y_Test_1,pred["Prediction"]))
print("Confusion Matrix on training data using Neural networks\n", confusion_matrix(Y_Test_1,pred["Prediction"]))
print("Classification report\n", classification_report(Y_Test_1,pred["Prediction"]))
```

Precision of 1	Accuracy
0.92	0.89

2 Hyperparameter Tuning

```
[ ] #Random Hyperparameter tuning for neural networks model
    print("RandomizedSearchCV-Multi Layer Perceptron")
    parameters={'activation': ['logistic','relu'], 'hidden_layer_sizes' : range(100,130,10), 'solver' : ['sgd', 'adam']}
    nn_random = RandomizedSearchCV(nn_clf,parameters,n_iter=10, cv=8, scoring ="precision")
    nn_random.fit(X_Train, Y_Train)
    nn_rand_parm=nn_random.best_params_
    print(nn_rand_parm)
```

```
RandomizedSearchCV-Multi Layer Perceptron
{'solver': 'adam', 'hidden_layer_sizes': 100, 'activation': 'relu'}
```

Optimize Precision score!
Want to correctly predict more customers who will purchase at quoted price!

3 Tuned model

```
#Using the parameters obtained from HyperParameterTuning in the MLP
nn_clf_r = MLPClassifier(**nn_rand_parm)
# random model prediction on test data
pred_nn=pd.DataFrame(nn_clf_r.predict(X_Test_1),columns=["Prediction"])
print("Accuracy Score on test data using MLP - random Search:",accuracy_score(Y_Test_1,pred_nn["Prediction"]))
print("Balanced Accuracy Score on test data using MLP - random Search:",balanced_accuracy_score(Y_Test_1,pred_nn["Prediction"]))
print("Confusion Matrix on test data using MLP - random Search\n", confusion_matrix(Y_Test_1,pred_nn["Prediction"]))
print("Classification report on test data using MLP - random Search\n", classification_report(Y_Test_1,pred_nn["Prediction"]))
```

0.51	0.82

Using default model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_mlp.csv	a few seconds ago	0 seconds	1 seconds	0.71622

Complete

Jump to your position on the leaderboard ▾

Kaggle score | 71.62%

Using tuned model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_mlp_1.csv	a few seconds ago	0 seconds	1 seconds	0.83394

Complete

Jump to your position on the leaderboard

Kaggle score | 83.39%

Support Vector Machine (SVM)

SVC()

1 Default model

```
from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler(feature_range=(-1,1)).fit(X_Train)
X_train = scaling.transform(X_Train)
X_test_1 = scaling.transform(X_Test_1)
X_test_A = scaling.transform(X_Test_A)
```

```
#svm classifier
svm_clf = SVC(cache_size=7000)
svm_clf.fit(X_train, Y_Train)
```

Precision of 1	Accuracy
0.99	0.91

Normalize data to use for SVC model

2 Hyperparameter Tuning

```
#Random Hyperparameter tuning for svm
print("RandomizedSearchCV- SVM")
parameters={'kernel': ['poly', 'rbf', 'sigmoid'], 'cache_size' : range(7000,7200,100)}
svm_random = RandomizedSearchCV(svm_clf,parameters,n_iter=10, cv=3, scoring ="precision")
svm_random.fit(X_train, Y_Train)
svm_rand_parm=svm_random.best_params_
print(svm_rand_parm)
```

```
RandomizedSearchCV- SVM
{'kernel': 'poly', 'cache_size': 7000}
```

3 Tuned model

```
#Using the parameters obtained from HyperParameterTuning in the SVM
svm_clf_r = SVC(**svm_rand_parm)
#grid model
svm_clf_r.fit(X_train,Y_Train)
```

0.99	0.91

Using tuned model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_SVC.csv	a few seconds ago	0 seconds	4 seconds	0.75657

Complete

Jump to your position on the leaderboard ▾

Kaggle score	75.65%

Linear Support Vector Machine (SVM)

LinearSVC()

1 Default model

```
#svm classifier linear svc
from sklearn.svm import LinearSVC
svm_clf = LinearSVC()
svm_clf.fit(X_Train, Y_Train)
#Prediction on test data created from split
pred_svm=pd.DataFrame(svm_clf.predict(X_Test_1),columns=["Prediction"])
print("----------------------------SVC - Test Data created from split-----------------------------------\n")
print("Accuracy Score on test data:",accuracy_score(Y_Test_1,pred_svm["Prediction"]))
print("Confusion Matrix on test data\n", confusion_matrix(Y_Test_1,pred_svm["Prediction"]))
print("Classification report\n", classification_report(Y_Test_1,pred_svm["Prediction"]))
```

Precision of 1	Accuracy
0.50	0.81

2 Hyperparameter Tuning

```
#Random Hyperparameter tuning for svm
print("RandomizedSearchCV- SVC")
parameters={'loss': ['hinge', 'squared_hinge'], 'C' : range(1,10,2)}
svm_random = RandomizedSearchCV(svm_clf,parameters,n_iter=10, cv=3, scoring
svm_random.fit(X_Train, Y_Train)
svm_rand_parm=svm_random.best_params_
print(svm_rand_parm)
```

```
RandomizedSearchCV- SVC
{'loss': 'squared_hinge', 'C': 7}
```

Optimize Precision score!
Want to correctly predict more customers who will purchase at quoted price!

3 Tuned model

```
#Using the parameters obtained from HyperParameterTuning in the SVM
svm_clf_r = LinearSVC(**svm_rand_parm)
#grid model
svm_clf_r.fit(X_Train,Y_Train)
# random model prediction on test data
pred_svm=pd.DataFrame(svm_clf_r.predict(X_Test_1),columns=["Prediction"])
print("Accuracy Score on test data using SVM - random Search:",accuracy_score(Y_Test_1,pred_svm["Prediction"]))
print("Balanced Accuracy Score on test data using SVM - random Search:",balanced_accuracy_score(Y_Test_1,pred_svm["Prediction"]))
print("Confusion Matrix on test data using SVM - random Search\n", confusion_matrix(Y_Test_1,pred_svm["Prediction"]))
print("Classification report on test data using SVM - random Search\n", classification_report(Y_Test_1,pred_svm["Prediction"]))
```

0.22	0.57

Using default model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_SVC_Linear_def.csv	a few seconds ago	0 seconds	1 seconds	0.69314

Complete

Jump to your position on the leaderboard

Kaggle score	**69.31%**

Stacking

Stacking()

Stacking Code

```
models = [ MLPClassifier(), RandomForestClassifier(**rand_parm_rfc),
           DecisionTreeClassifier(**rand_parm) ]

S_Train, S_Test = stacking(models,
                           X_Train, Y_Train, X_Test_A,
                           regression=False,

                           mode='oof_pred_bag',

                           needs_proba=False,

                           save_dir=None,

                           metric=accuracy_score,

                           n_folds=3,

                           stratified=True,

                           shuffle=True,

                           random_state=0,

                           verbose=2)
```

3 Base models:

- MLP
- Random Forest
- Decision Tree

Output

```
task:           [classification]
n_classes:      [2]
metric:         [accuracy_score]
mode:           [oof_pred_bag]
n_models:       [3]

model  0:       [MLPClassifier]
    fold  0:    [0.86348766]
    fold  1:    [0.74663964]
    fold  2:    [0.63896593]
    ----
    MEAN:       [0.74969775] + [0.09168611]
    FULL:       [0.74969775]

model  1:       [RandomForestClassifier]
    fold  0:    [0.94843894]
    fold  1:    [0.94662542]
    fold  2:    [0.94509637]
    ----
    MEAN:       [0.94672024] + [0.00136625]
    FULL:       [0.94672024]

model  2:       [DecisionTreeClassifier]
    fold  0:    [0.93311287]
    fold  1:    [0.93396629]
    fold  2:    [0.93321954]
    ----
    MEAN:       [0.93343290] + [0.00037967]
    FULL:       [0.93343290]
```

3-fold cross validation for all 3 base models

1 Meta model – Random Forest

```
#STACKING - CONTRUCT A Random forest model==============================
model_st = RandomForestClassifier()

model_st = model_st.fit(S_Train, Y_Train)

pred_st=pd.DataFrame(model_st.predict(S_Test),columns=["Prediction"])

QuoteNumber = testData['QuoteNumber']
submission_st = pd.DataFrame({"QuoteNumber": QuoteNumber, "QuoteConversion_Flag":pred_st["Prediction"]})
submission_st.to_csv("/gdrive/My Drive/CIS 508 Python/Assignment_3/Submission_ST.csv", index = None)
```

Kaggle score

0.81

2 Hyperparameter Tuning

```
#Hyperparameter tuning done for stacking random forest classifier
#RANDOM SEARCH-----------------------------------------

print("RandomizedSearchCV-Stacking")
parameters={'max_features': range(1,3,1) ,'n_estimators':[25,30,40,50], 'criterion':['gini
st_random = RandomizedSearchCV(model_st,parameters,n_iter=20,cv=5, scoring ='precision')
st_random.fit(S_Train, Y_Train)
rand_parm_st=st_random.best_params_
print(rand_parm_st)
```

```
RandomizedSearchCV-Stacking
{'n_estimators': 25, 'max_features': 1, 'criterion': 'gini'}
```

Optimize Precision score!
Want to correctly predict more customers who will purchase at quoted price!

3 Tuned model

```
#STACKING - CONTRUCT A Random forest model==============================
model_st = RandomForestClassifier(**rand_parm_st)

model_st = model_st.fit(S_Train, Y_Train)

pred_st=pd.DataFrame(model_st.predict(S_Test),columns=["Prediction"])

QuoteNumber = testData['QuoteNumber']
submission_st = pd.DataFrame({"QuoteNumber": QuoteNumber, "QuoteConversion_Flag":pred_st["Prediction"]})
submission_st.to_csv("/gdrive/My Drive/CIS 508 Python/Assignment_3/Submission_ST_tuned.csv", index = None)
```

0.81

Using default Random model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_ST.csv	32 minutes ago	219 seconds	4 seconds	0.81311

Complete

Jump to your position on the leaderboard

Kaggle score	81.31%

Using tuned Random model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_ST_tuned.csv	a few seconds ago	0 seconds	1 seconds	0.81311

Complete

Jump to your position on the leaderboard ▾

Kaggle score | 81.31%

Kaggle score

1 Meta model – MLP

```
#STACKING - CONTRUCT A Random forest model==============================
model_st = RandomForestClassifier()

model_st = model_st.fit(S_Train, Y_Train)

pred_st=pd.DataFrame(model_st.predict(S_Test),columns=["Prediction"])

QuoteNumber = testData['QuoteNumber']
submission_st = pd.DataFrame({"QuoteNumber": QuoteNumber, "QuoteConversion_Flag":pred_st["Prediction"]})
submission_st.to_csv("/gdrive/My Drive/CIS 508 Python/Assignment_3/Submission_ST.csv", index = None)
```

0.81

2 Meta model -Decision Tree

```
#STACKING - CONTRUCT A MLP model==============================
model_st_mlp = MLPClassifier()

model_st_mlp = model_st_mlp.fit(S_Train, Y_Train)

pred_st_mlp=pd.DataFrame(model_st_mlp.predict(S_Test),columns=["Prediction"])

QuoteNumber = testData['QuoteNumber']
submission_st_mlp = pd.DataFrame({"QuoteNumber": QuoteNumber, "QuoteConversion_Flag":pred_st_mlp["Prediction"]})
submission_st_mlp.to_csv("/gdrive/My Drive/CIS 508 Python/Assignment_3/Submission_ST_MLP.csv", index = None)
```

0.81

3 Meta model –Gradient Boosting

```
#STACKING - CONTRUCT A Decision model==============================
model_st_GB = GradientBoostingClassifier()

model_st_GB = model_st_GB.fit(S_Train, Y_Train)

pred_st_GB=pd.DataFrame(model_st_GB.predict(S_Test),columns=["Prediction"])

QuoteNumber = testData['QuoteNumber']
submission_st_GB = pd.DataFrame({"QuoteNumber": QuoteNumber, "QuoteConversion_Flag":pred_st_GB["Prediction"]})
submission_st_GB.to_csv("/gdrive/My Drive/CIS 508 Python/Assignment_3/Submission_ST_GB.csv", index = None)
```

0.81

Using MLP model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_ST_MLP.csv	a few seconds ago	1 seconds	1 seconds	0.81311

Complete

Jump to your position on the leaderboard ▾

Kaggle score	81.31%

Using Decision Tree model

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
Submission_ST_DT.csv	a few seconds ago	0 seconds	1 seconds	0.81311

Complete

Jump to your position on the leaderboard ▾

Kaggle score | **81.31%**

Using Gradient Boosting model

Your most recent submission

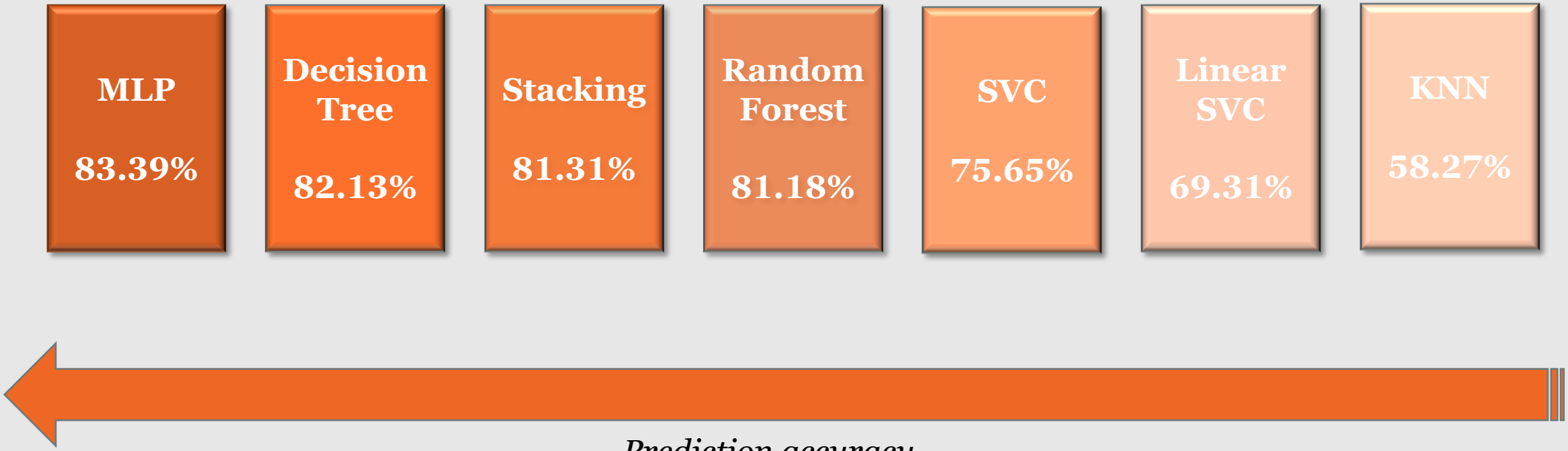
Name	Submitted	Wait time	Execution time	Score
Submission_ST_GB.csv	a few seconds ago	4 seconds	1 seconds	0.81311

Complete

Jump to your position on the leaderboard

Kaggle score | **81.31%**

Comparison of models – Kaggle score



Learnings from assignment

Unbalanced data can be made used using SMOTE

In case of large data, split data into train and test data

Multi-Layer Perceptron Classification algorithm

K nearest Neighbor algorithm

Support Vector Classifier algorithm

Stacking algorithm