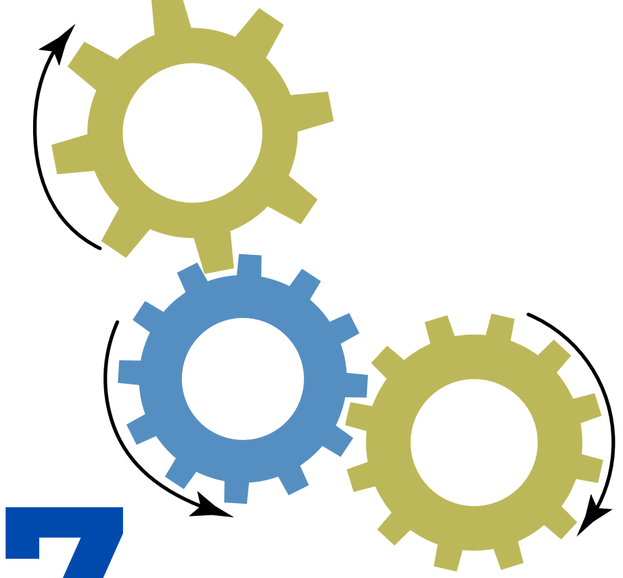
Berkay, Elnara, Yen



PROJECT 7
Machine Learning

1. THE PROCESS CONTENT

Banking data for machine learning

1 Data processing

Shape (6819, 96)
df.dtypes (float, int)
No null and missing values
Low variance 2 columns found
Dropping high correlation columns = 22

2 Pre modeling part

Choice of target - Bankrupt?
SMOTE for oversampling imbalanced column
RandomForestClassifier for best features search
As result - 21 columns for modeling selected

3 Ours models

MultinomialNB
BaggingClassifier
RidgeClassifier
Stacking

2. PRE MODELING PART I

```python
# Find the Column Collinearity
# # Create correlation matrix
corr_matrix = df.corr().abs()
# # Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
# # Find features with correlation greater than 0.85
to_drop = [column for column in upper.columns if any(upper[column] > 0.85)]
print(to_drop)
print(len(to_drop))
# # Drop features
df.drop(to_drop, axis=1, inplace=True)
```

```python
df['Bankrupt?'].value_counts() # distribution of target "B" (binary label)
```

```
0    6599
1     220
Name: Bankrupt?, dtype: int64
```

```python
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X_train, y_train = smote.fit_resample(X_train, y_train)
y_train.value_counts()
```

```
0    5286
1    5286
```

3. PRE MODELING PART II

```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    'n_estimators': [50, 100,500],
    'min_samples_split': [2, 4],
    'min_samples_leaf' : [1, 2],
    'max_features': ['sqrt']
    }
clf = RandomForestClassifier(random_state=RAND_STATE)
```

```python
grid_search = GridSearchCV(clf, param_grid, cv=5,return_train_score=True,n_jobs=-1)
grid_search.fit(X_train,y_train)
best_params = grid_search.best_params_ #To check the best set of parameters returned
best_params
```

```
'max_features': 'sqrt',
'min_samples_leaf': 1,
'min_samples_split': 2,
'n_estimators': 500}
```

```python
from sklearn.model_selection import cross_val_score
clf = RandomForestClassifier(random_state=RAND_STATE, **best_params)
cross_val_scores = cross_val_score(clf, X_test, y_test, cv=5)
print(np.mean(cross_val_scores))
```

```
0.9655462184873949
```

```python
clf.fit(X_train, y_train)
len(X_train.columns)
```

```
71
```

```python
feature_names = X_train.columns
feature_names = list(feature_names)
```

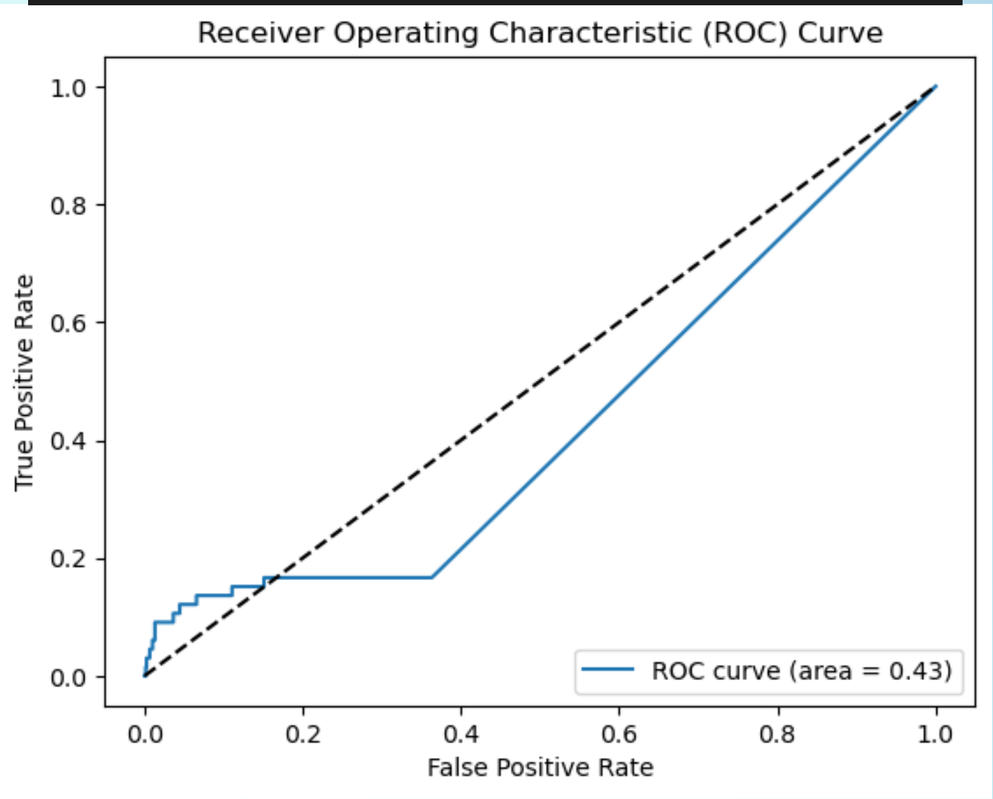
```python
data = pd.DataFrame(list(zip(feature_names, clf.feature_importances_)))
data.columns = ['columns_name', 'score_feature_importance']
data.sort_values(by=['score_feature_importance'], ascending = False)
```

4. MULTINOMIAL NB

```python
NB_model = MultinomialNB().fit(X_train, y_train)
y_pred_test = NB_model.predict(X_test)
y_pred_train = NB_model.predict(X_train)

print('Accuracy:', accuracy_score(y_test, y_pred_test))
print('F1 score:', f1_score(y_test, y_pred_test, average="macro"))
print(classification_report(y_test, y_pred_test)) # The closer to 1
```

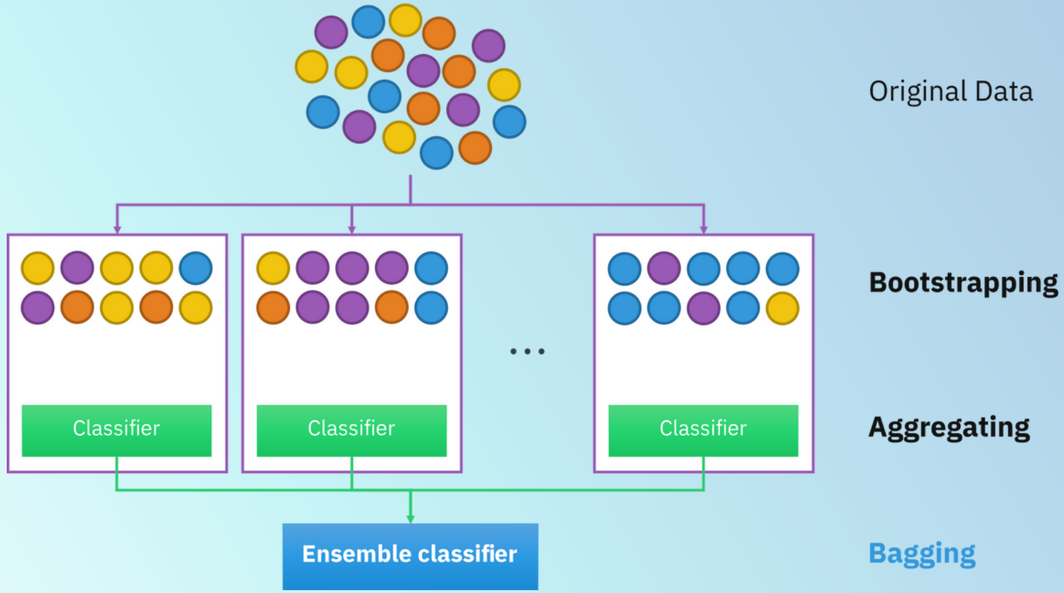
```
Accuracy: 0.7653958944281525
F1 score: 0.4550599829095872
              precision    recall  f1-score   support

           0       0.97      0.79      0.87      1980
           1       0.03      0.17      0.04        66

    accuracy                           0.77      2046
   macro avg       0.50      0.48      0.46      2046
weighted avg       0.94      0.77      0.84      2046
```

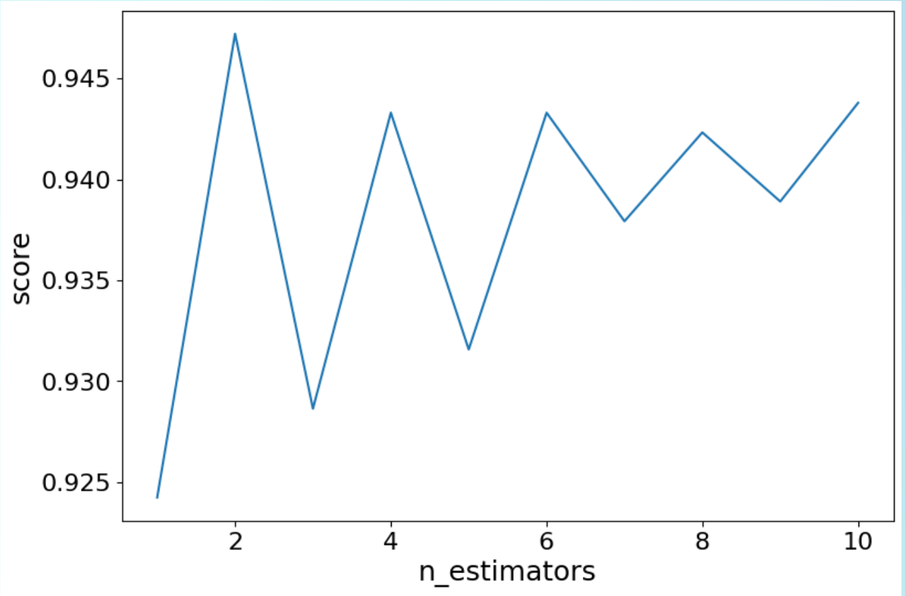
	Error_metric	Train	Test
0	Accuracy	0.468175	0.765396
1	Precision	0.414336	0.025229
2	Recall	0.153929	0.166667



5.BAGGING CLASSIFIER

```python
estimator_range = [1,2,3,4,5,6,7,8,9,10]

models = []
scores = []

for n_estimators in estimator_range:

    # Create bagging classifier
    clf = BaggingClassifier(n_estimators = n_estimators, random_state = 42)

    # Fit the model
    clf.fit(X_train, y_train)

    # Append the model and score to their respective list
    models.append(clf)

    scores.append(accuracy_score(y_true = y_test, y_pred = clf.predict(X_test)))

# Generate the plot of scores against number of estimators
plt.figure(figsize=(9,6))
plt.plot(estimator_range, scores)

# Adjust labels and font (to make visable)
plt.xlabel("n_estimators", fontsize = 18)
plt.ylabel("score", fontsize = 18)
plt.tick_params(labelsize = 16)

# Visualize plot
plt.show()
```





6. BAGGING CLASSIFIER-2

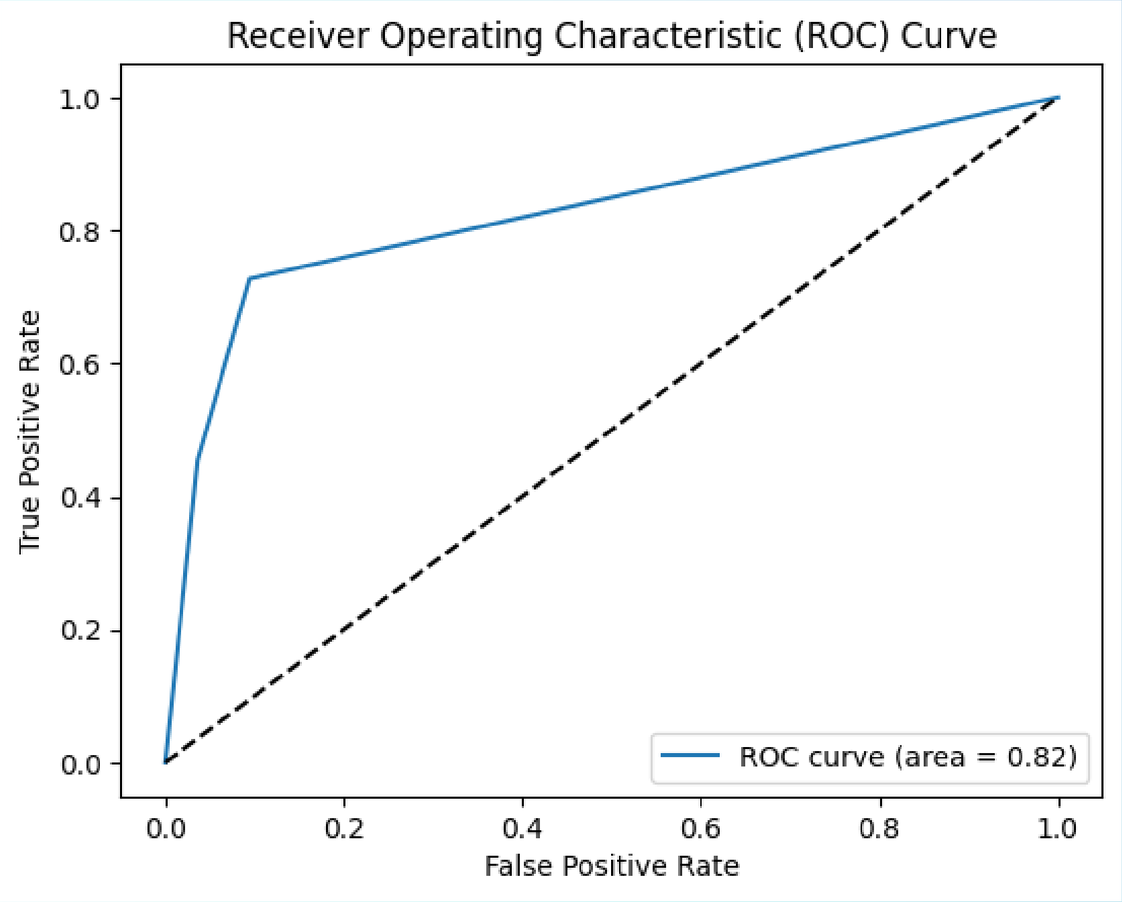
```python
# Create bagging classifier
clf = BaggingClassifier(n_estimators = 2, random_state = 42)

# Fit the model
clf.fit(X_train, y_train)

# Doing the predictions
y_pred = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
```

✓ 0.4s

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score

performance_log = pd.DataFrame({'Error_metric': ['Accuracy','Precision','Recall'],
                                'Train': [accuracy_score(y_train, y_pred),
                                          precision_score(y_train, y_pred),
                                          recall_score(y_train, y_pred)],
                                'Test': [accuracy_score(y_test, y_pred_test),
                                         precision_score(y_test, y_pred_test),
                                         recall_score(y_test, y_pred_test)]})

display(performance_log)
```

✓ 0.0s

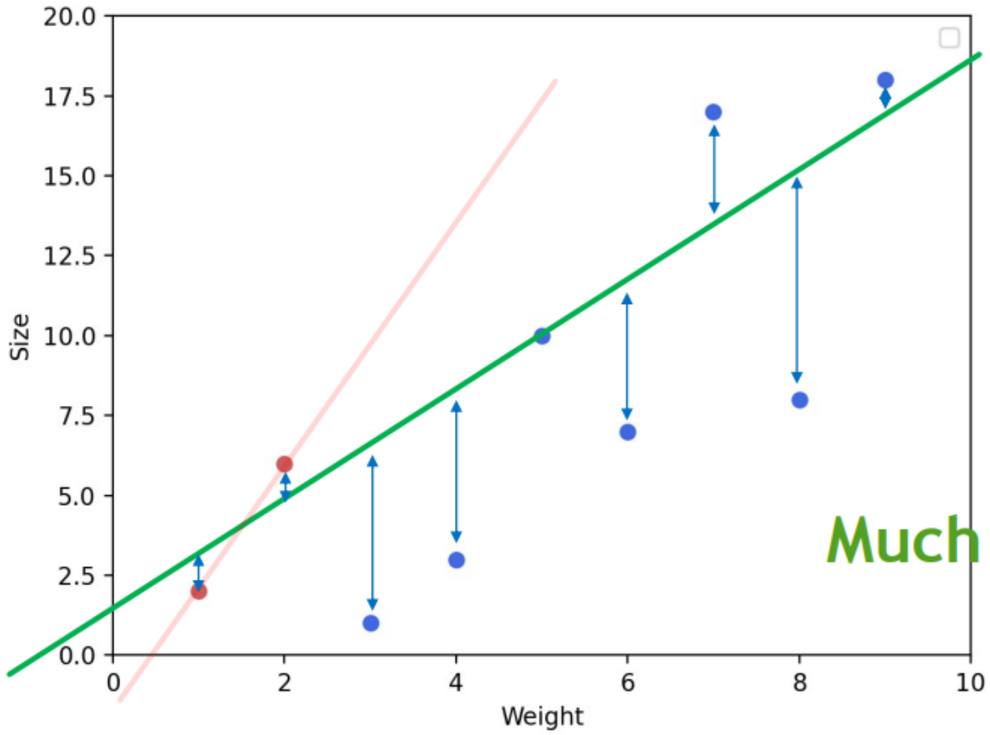
	Error_metric	Train	Test
0	Accuracy	0.982355	0.947214
1	Precision	0.995772	0.294118
2	Recall	0.968824	0.454545



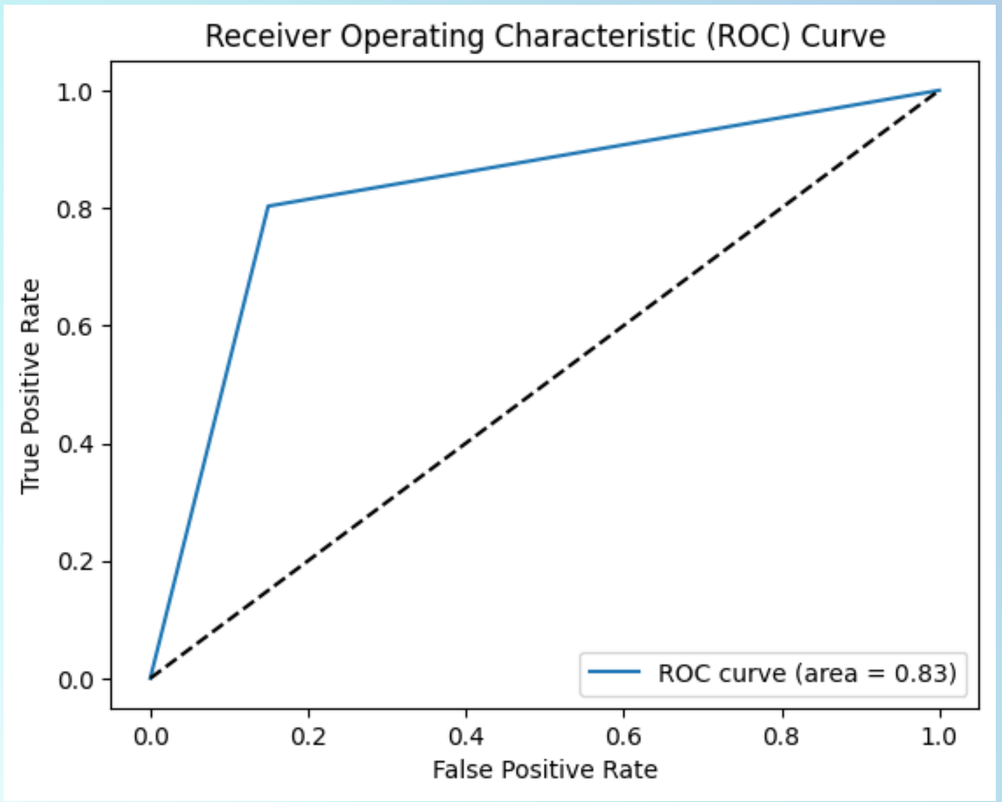
7. RIDGE CLASSIFIER

This classifier first converts the target values into {-1, 1} and then treats the problem as a regression task (multi-output regression in the multiclass case).

```python
from sklearn.linear_model import RidgeClassifier

rdgclassifier = RidgeClassifier()
rdgclassifier.fit(X_train, y_train)

# Doing the predictions
y_pred = rdgclassifier.predict(X_train)
y_pred_test = rdgclassifier.predict(X_test)
```



Much better!



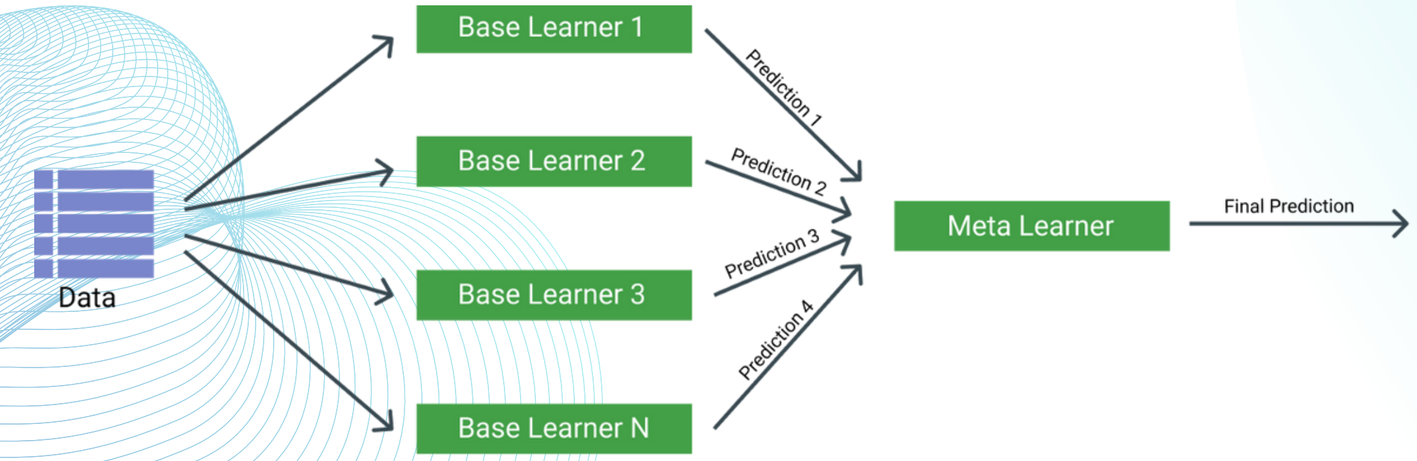
	Error_metric	Train	Test
0	Accuracy	0.896081	0.848974
1	Precision	0.868331	0.151862
2	Recall	0.933752	0.803030

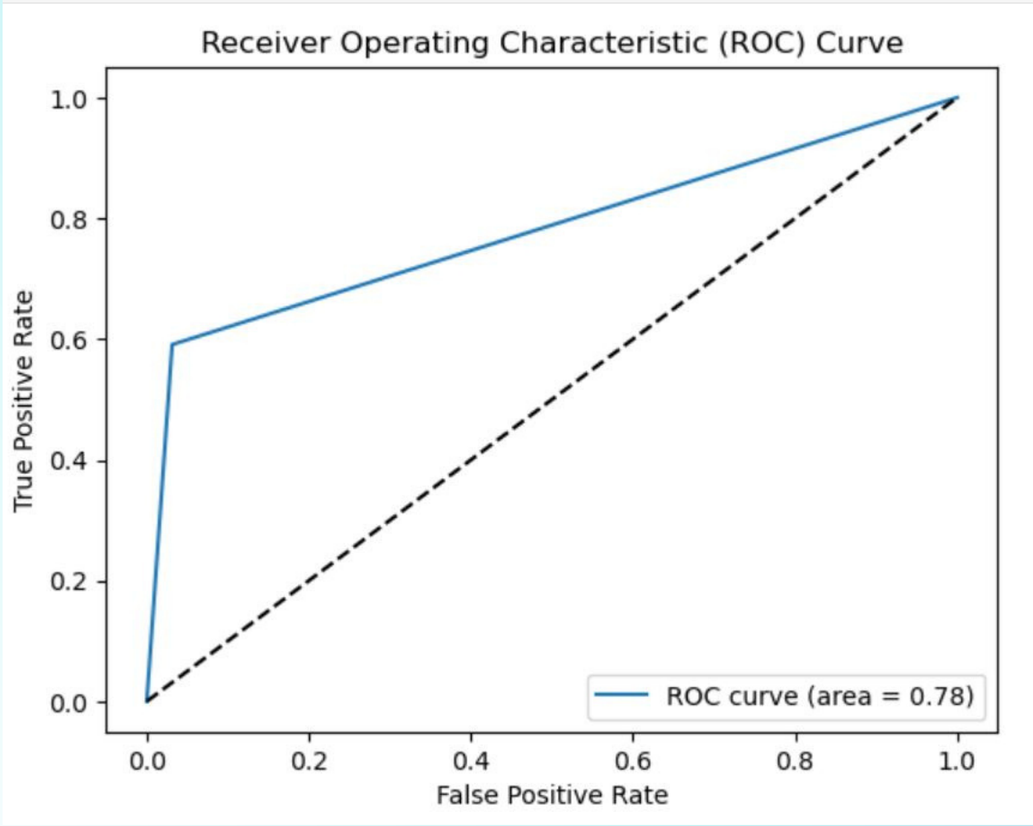
8. STACKING

```
# Fit the model
estimators = [('rf', RandomForestClassifier(n_estimators=10, random_state=42)),('knn', KNeighborsClassifier(n_neighbors=1))
model_yen = StackingClassifier(estimators)
model_yen.fit(X_train, y_train)
y_pred_train = model_yen.predict(X_train)
```

Stacking classifier : is a meta-ensemble learning technique that involves training several base classifiers and combining their predictions using another classifier.

	Stacking	Train	Test
0	Accuracy	0.958649	0.987781
1	Precision	0.989374	0.780822
2	Recall	0.927257	0.863636
3	ROC AUC SCORE	0.958649	0.958649





9. RESULTS

	BaggingClassifier	Train	Test
0	Accuracy	0.895324	0.979961
1	Precision	0.988758	0.676056
2	Recall	0.799740	0.727273
3	ROC AUC SCORE	0.895324	0.895324

	RidgeClassifier	Train	Test
0	Accuracy	0.891102	0.846530
1	Precision	0.861807	0.141618
2	Recall	0.931587	0.742424
3	ROC AUC SCORE	0.891102	0.891102

	MultinomialNB	Train	Test
0	Accuracy	0.467201	0.768328
1	Precision	0.411867	0.025581
2	Recall	0.153280	0.166667
3	ROC AUC SCORE	0.467201	0.467201

	Stacking	Train	Test
0	Accuracy	0.958649	0.987781
1	Precision	0.989374	0.780822
2	Recall	0.927257	0.863636
3	ROC AUC SCORE	0.958649	0.958649

```python
from sklearn.metrics import roc_auc_score
# Evaluate the models on the train and test sets
from sklearn.metrics import roc_auc_score
models = {'BaggingClassifier': bagg_model, 'RidgeClassifier': rc_model, "MultinomialNB": NB_model, "Stacking": stack_model}
for name, mod in models.items():
    y_train_pred = mod.predict(X_train)
    y_test_pred = mod.predict(X_test)
    performance_log = pd.DataFrame({name: ['Accuracy','Precision','Recall','ROC AUC SCORE'],
                                    'Train': [accuracy_score(y_train, y_train_pred),
                                              precision_score(y_train, y_train_pred),
                                              recall_score(y_train, y_train_pred),
                                              roc_auc_score(y_train, y_train_pred)],
                                    'Test': [accuracy_score(y_test, y_test_pred),
                                             precision_score(y_test, y_test_pred),
                                             recall_score(y_test, y_test_pred),
                                             roc_auc_score(y_train, y_train_pred)]})
    display(performance_log)
```

THANK YOU!