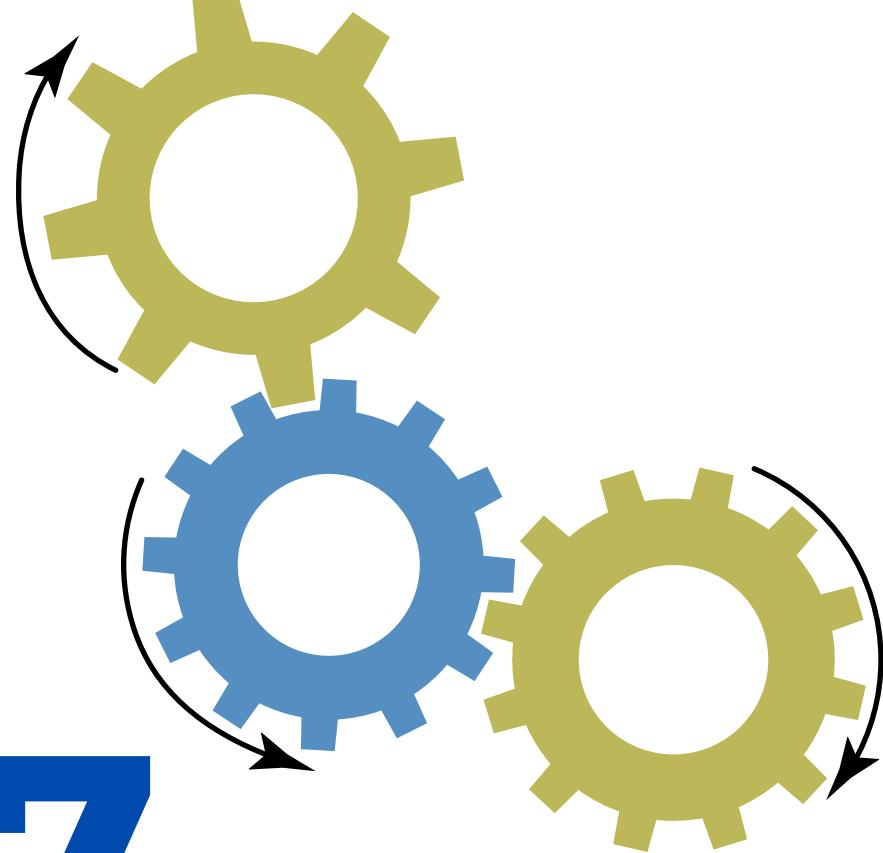


 Berkay, Elnara, Yen

# PROJECT 7

# Machine Learning



# 1. THE PROCESS CONTENT

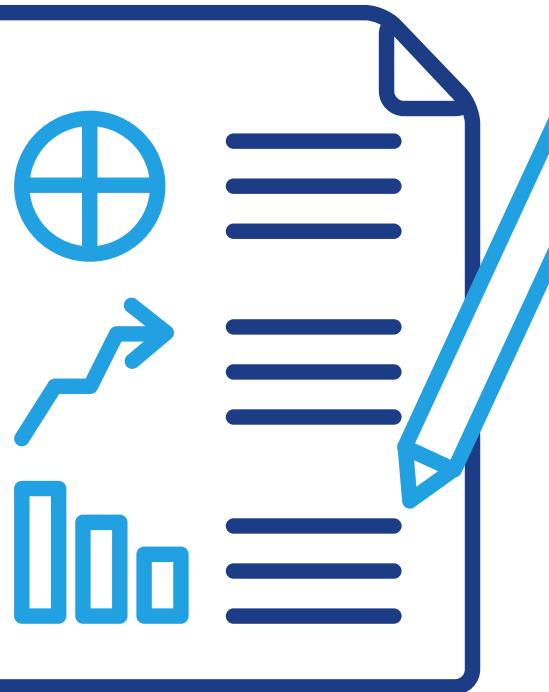
Banking data for machine learning



- 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



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

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

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

```
1 from imblearn.over_sampling import SMOTE
2 smote = SMOTE()
3 X_train, y_train = smote.fit_resample(X_train, y_train)
4 y_train.value_counts()
```

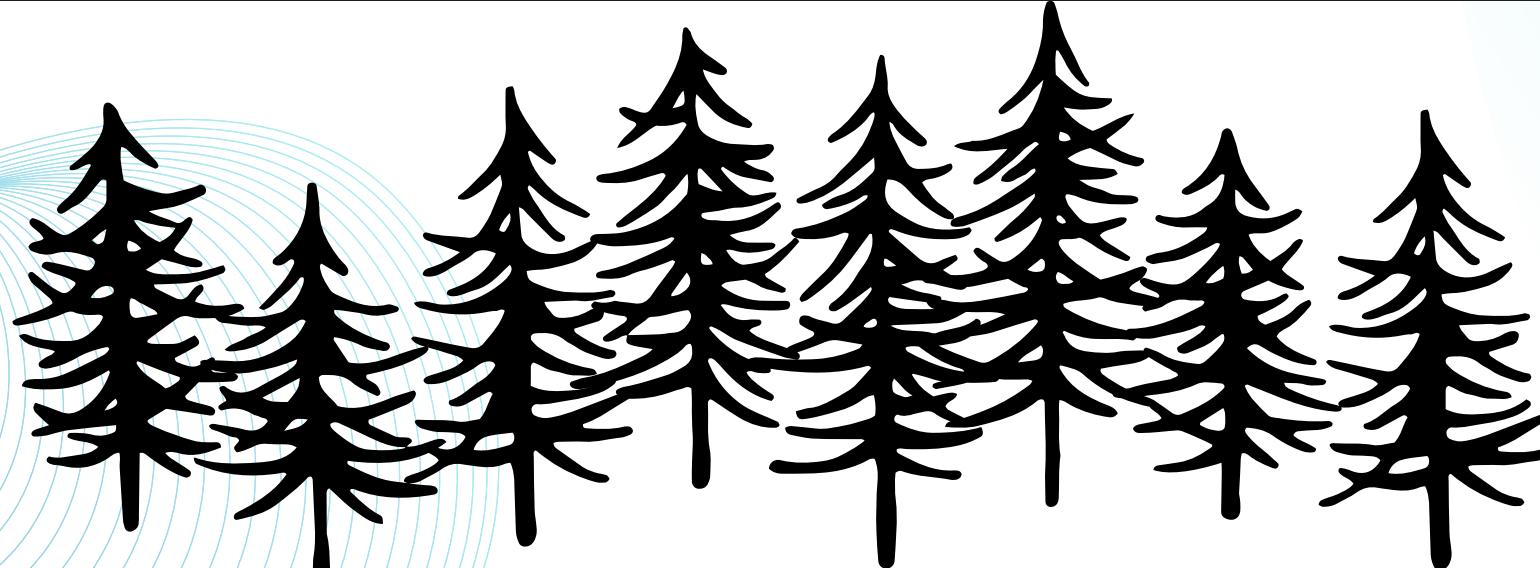
```
0    5286
1    5286
```

# 3. PRE MODELING PART II

```
1 from sklearn.model_selection import GridSearchCV
2 from sklearn.ensemble import RandomForestClassifier
3
4 param_grid = {
5     'n_estimators': [50, 100, 500],
6     'min_samples_split': [2, 4],
7     'min_samples_leaf' : [1, 2],
8     'max_features': ['sqrt']
9 }
10 clf = RandomForestClassifier(random_state=RAND_STATE)

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

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



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

0.9655462184873949

```
1 clf.fit(X_train, y_train)
2 len(X_train.columns)
```

71

```
1 feature_names = X_train.columns
2 feature_names = list(feature_names)
```

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

# 4. MULTINOMIAL NB

```
1 NB_model = MultinomialNB().fit(X_train, y_train)
2 y_pred_test = NB_model.predict(X_test)
3 y_pred_train = NB_model.predict(X_train)
4
5 print('Accuracy:', accuracy_score(y_test, y_pred_test))
6 print('F1 score:', f1_score(y_test, y_pred_test, average="macro"))
7 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
---	------	------	------	------

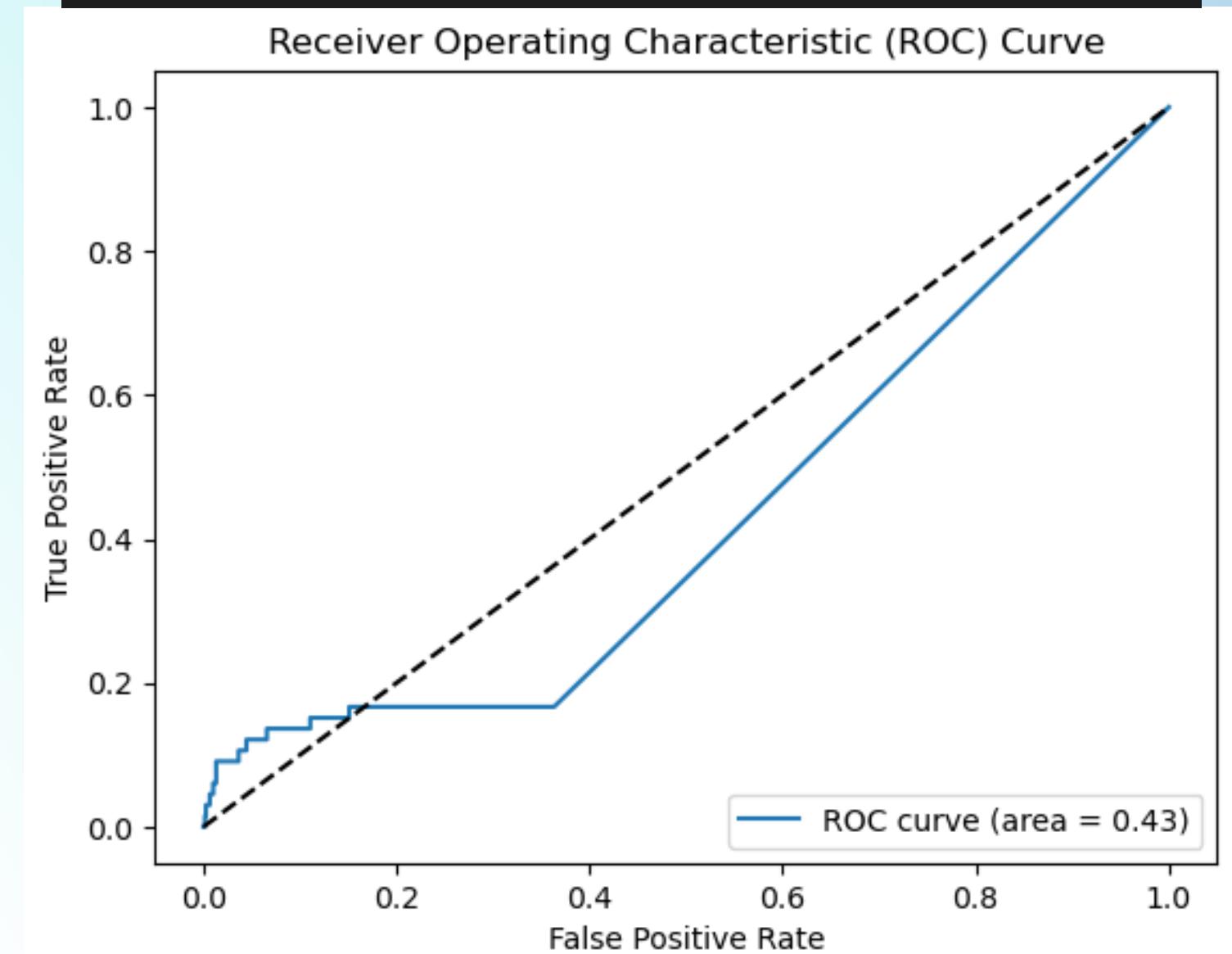
1	0.03	0.17	0.04	66
---	------	------	------	----

accuracy				
----------	--	--	--	--

macro avg	0.50	0.48	0.46	2046
-----------	------	------	------	------

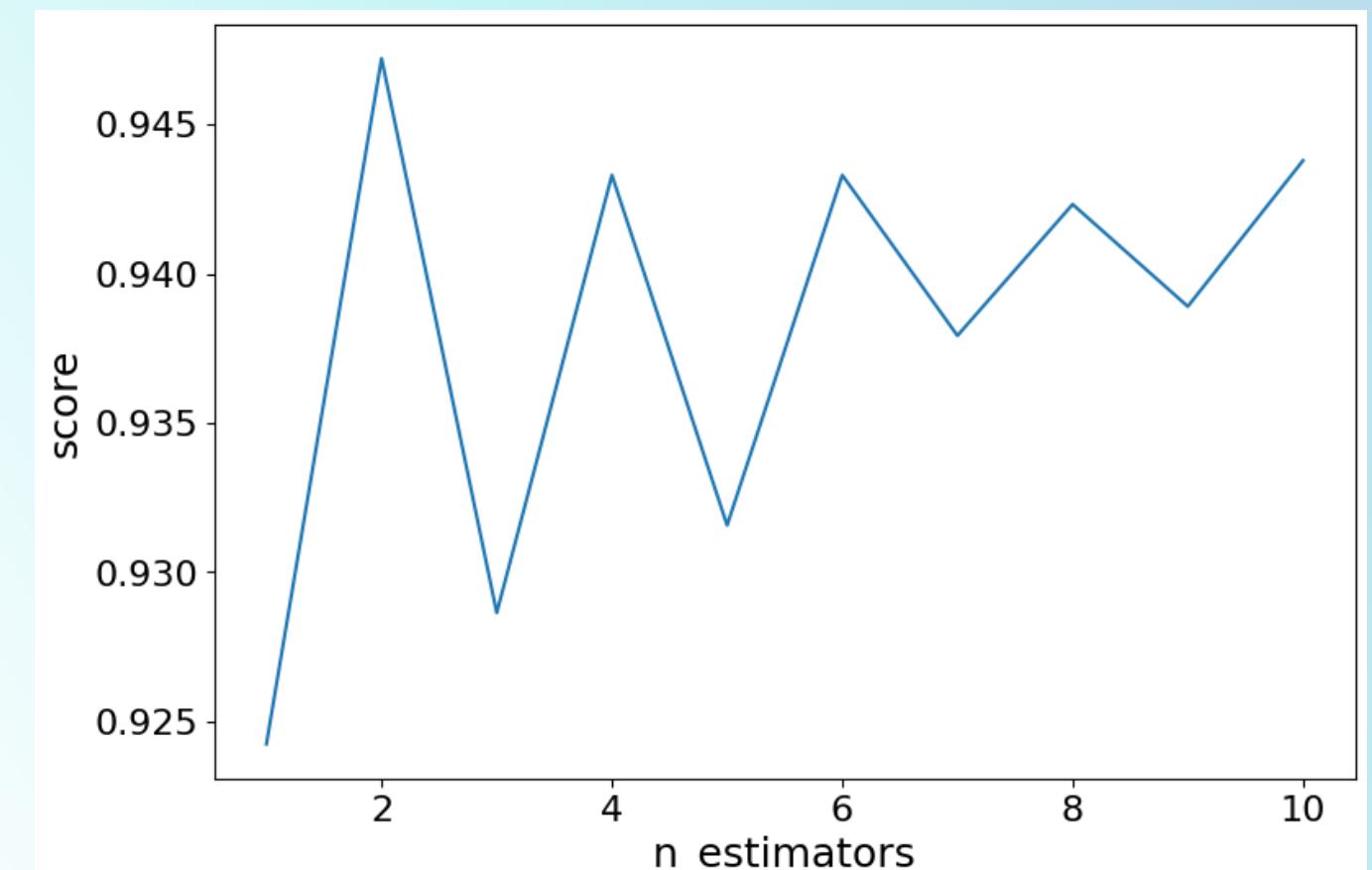
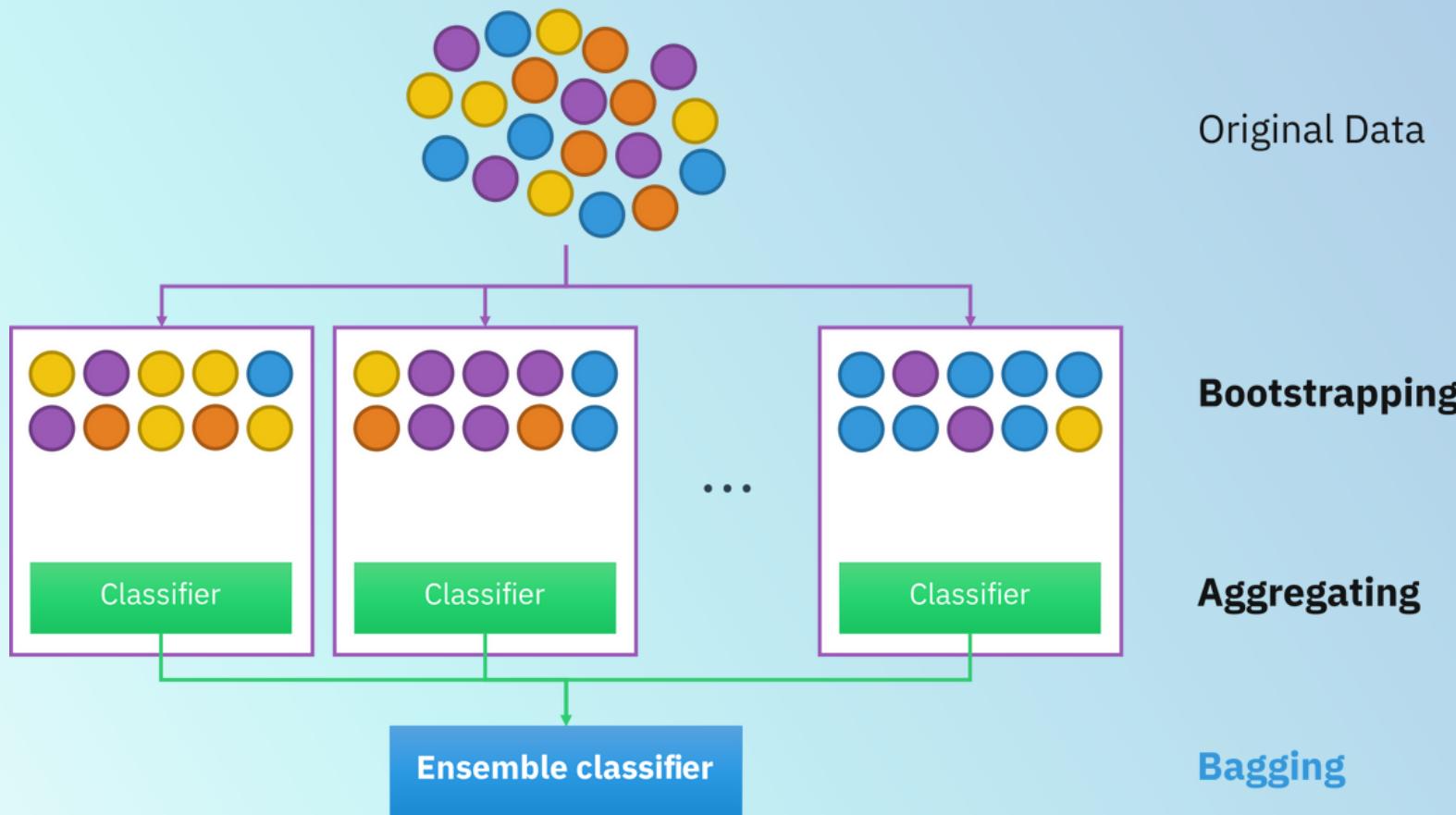
weighted avg	0.94	0.77	0.84	2046
--------------	------	------	------	------

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

```
1 estimator_range = [1,2,3,4,5,6,7,8,9,10]
2
3 models = []
4 scores = []
5
6 for n_estimators in estimator_range:
7
8     # Create bagging classifier
9     clf = BaggingClassifier(n_estimators = n_estimators, random_state = 42)
10
11    # Fit the model
12    clf.fit(X_train, y_train)
13
14    # Append the model and score to their respective list
15    models.append(clf)
16
17    scores.append(accuracy_score(y_true = y_test, y_pred = clf.predict(X_test)))
18
19 # Generate the plot of scores against number of estimators
20 plt.figure(figsize=(9,6))
21 plt.plot(estimator_range, scores)
22
23 # Adjust labels and font (to make visable)
24 plt.xlabel("n_estimators", fontsize = 18)
25 plt.ylabel("score", fontsize = 18)
26 plt.tick_params(labelsize = 16)
27
28 # Visualize plot
29 plt.show()
```

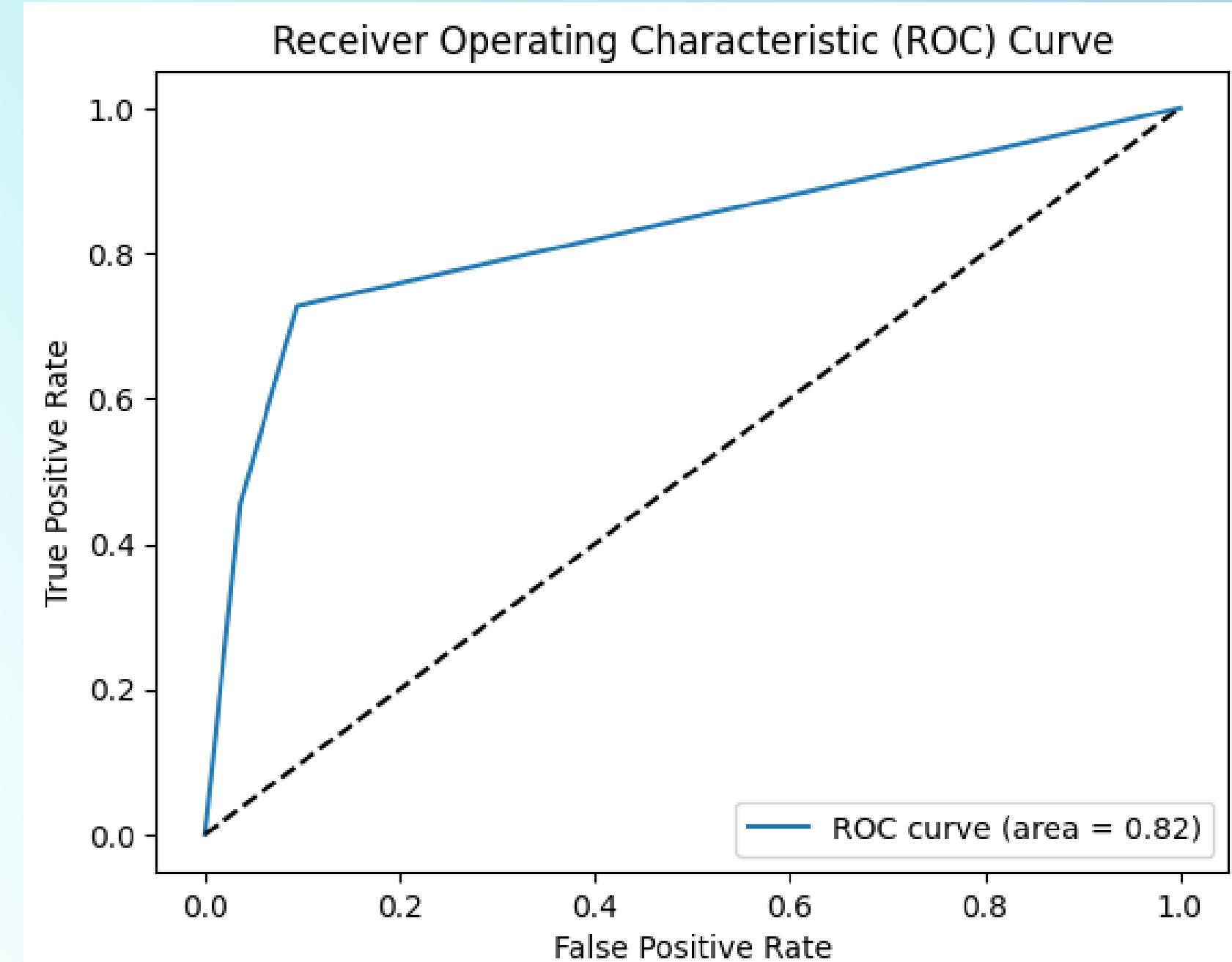


# 6. BAGGING CLASSIFIER-2

```
1 # Create bagging classifier
2 clf = BaggingClassifier(n_estimators = 2, random_state = 42)
3
4 # Fit the model
5 clf.fit(X_train, y_train)
6
7 # Doing the predictions
8 y_pred = clf.predict(X_train)
9 y_pred_test = clf.predict(X_test)
✓ 0.4s
```

```
1 from sklearn.metrics import accuracy_score, precision_score, recall_score
2
3 performance_log = pd.DataFrame({'Error_metric': ['Accuracy', 'Precision', 'Recall'],
4                                 'Train': [accuracy_score(y_train, y_pred),
5                                           precision_score(y_train, y_pred),
6                                           recall_score(y_train, y_pred)],
7                                 'Test': [accuracy_score(y_test, y_pred_test),
8                                           precision_score(y_test, y_pred_test),
9                                           recall_score(y_test, y_pred_test)]})
10
11 display(performance_log)
✓ 0.0s
```

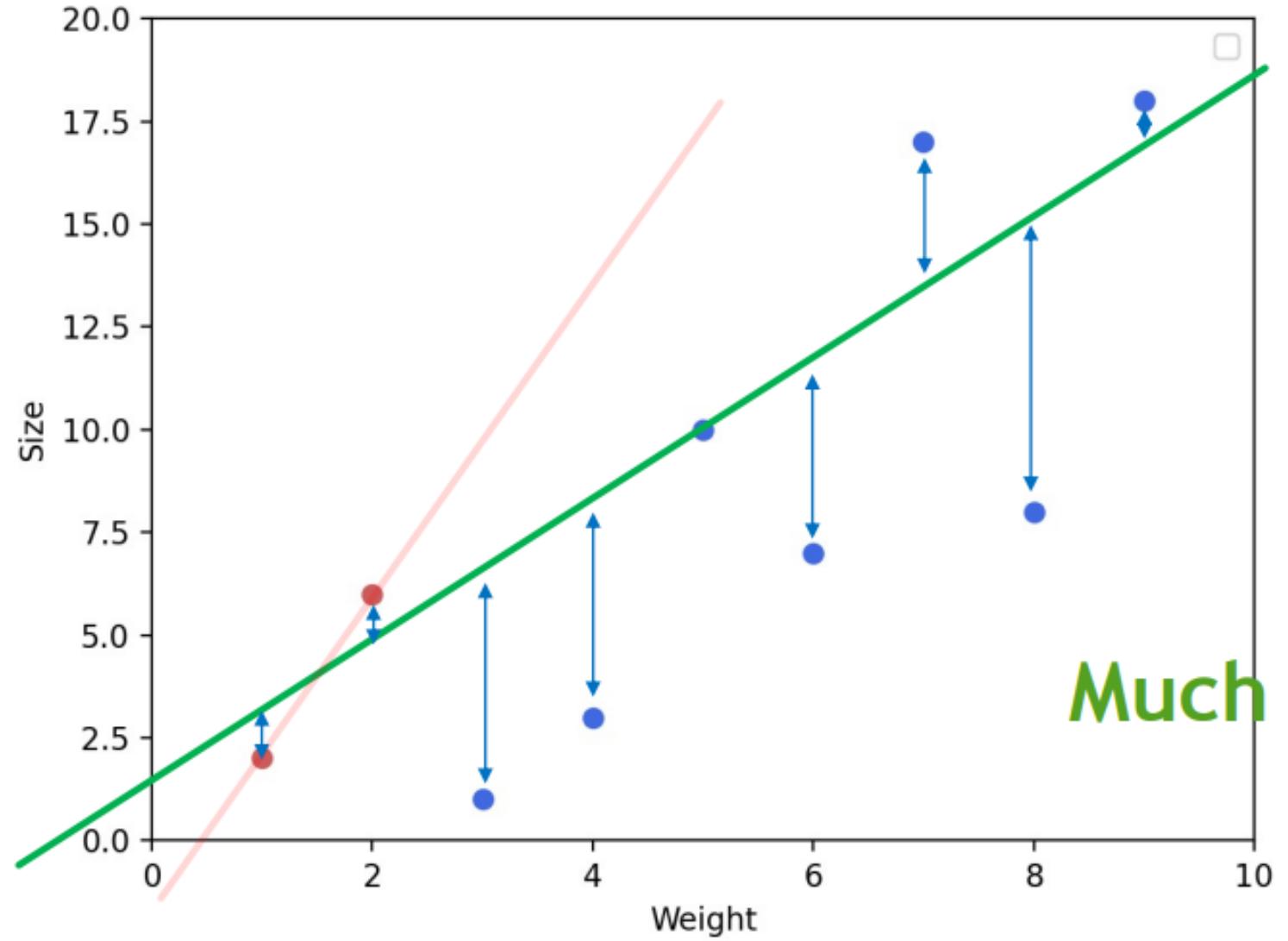
	Error_metric	Train	Test
0	Accuracy	0.982355	0.947214
1	Precision	0.995772	0.294118
2	Recall	0.968824	0.454545



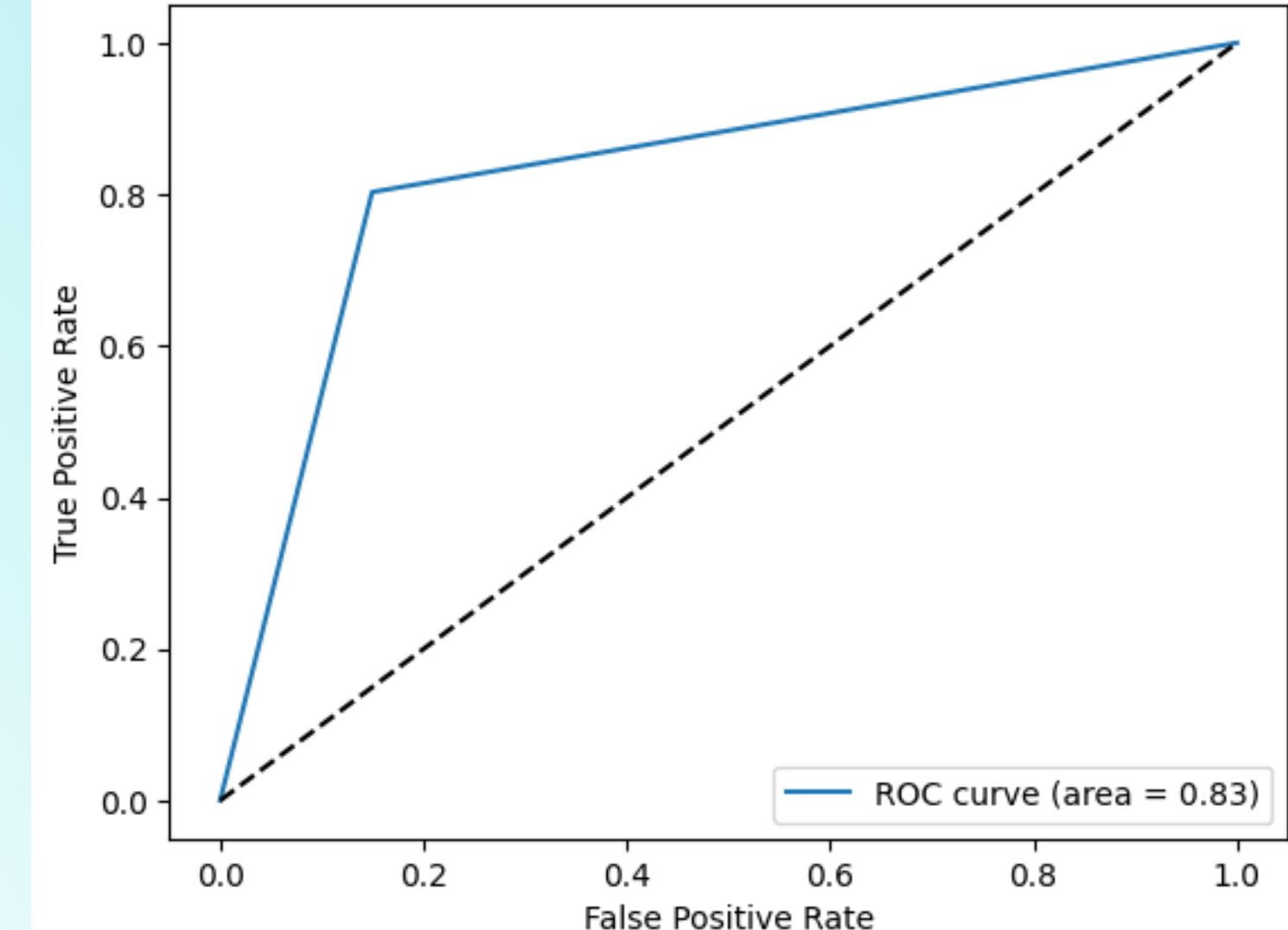
# 7. RIDGE CLASSIFIER

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).

```
1 from sklearn.linear_model import RidgeClassifier
2
3 rdgclassifier = RidgeClassifier()
4 rdgclassifier.fit(X_train, y_train)
5
6 # Doing the predictions
7 y_pred = rdgclassifier.predict(X_train)
8 y_pred_test = rdgclassifier.predict(X_test)
```



Receiver Operating Characteristic (ROC) Curve



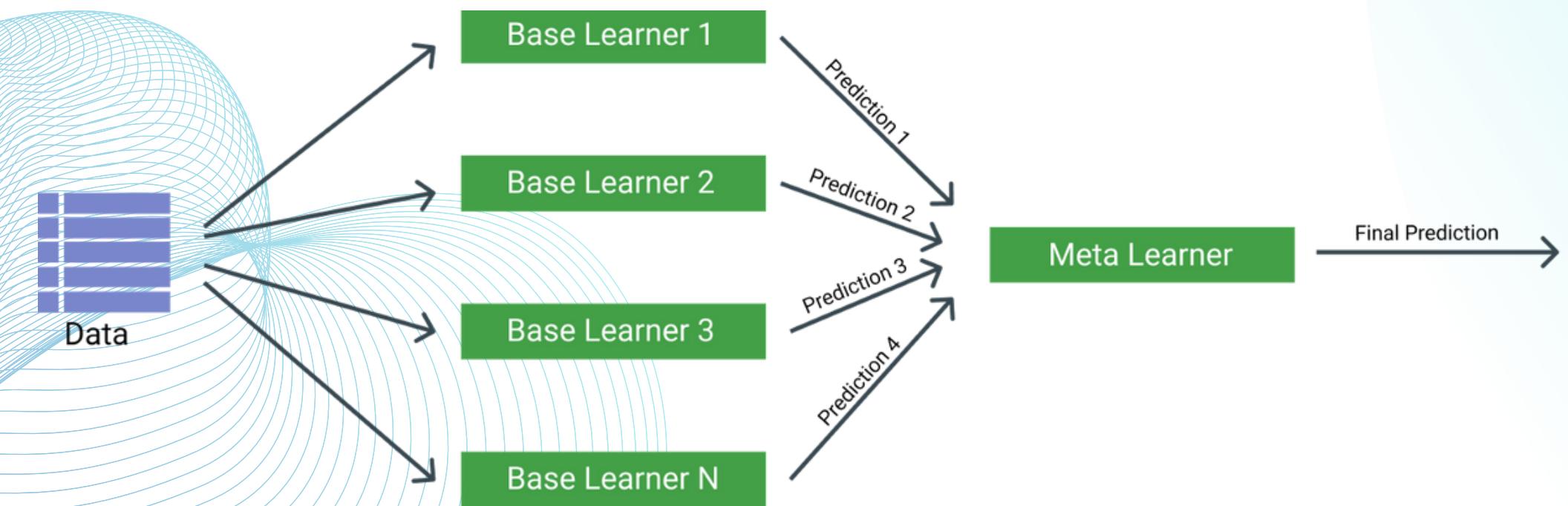
Error_metric	Train	Test
0 Accuracy	0.896081	0.848974
1 Precision	0.868331	0.151862
2 Recall	0.933752	0.803030

# 8. STACKING

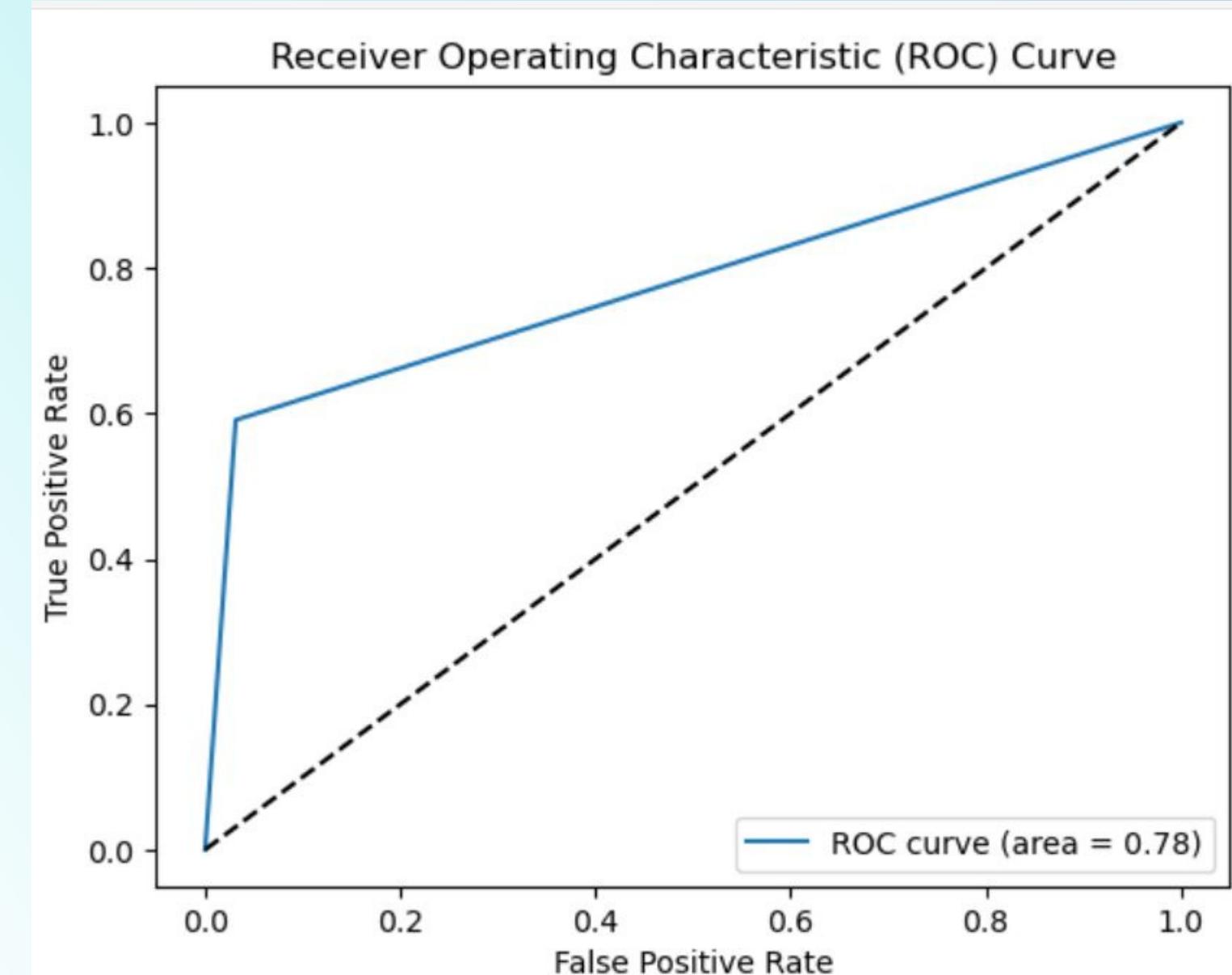
# Fit the model

```
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

	<b>BaggingClassifier</b>	<b>Train</b>	<b>Test</b>
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

	<b>RidgeClassifier</b>	<b>Train</b>	<b>Test</b>
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

	<b>MultinomialNB</b>	<b>Train</b>	<b>Test</b>
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

	<b>Stacking</b>	<b>Train</b>	<b>Test</b>
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

```
1 from sklearn.metrics import roc_auc_score
2 # Evaluate the models on the train and test sets
3 from sklearn.metrics import roc_auc_score
4 models = {'BaggingClassifier': bagg_model, 'RidgeClassifier': rc_model, "MultinomialNB": NB_model, "Stacking": stack_model}
5 for name, mod in models.items():
6     y_train_pred = mod.predict(x_train)
7     y_test_pred = mod.predict(x_test)
8     performance_log = pd.DataFrame({name: ['Accuracy','Precision','Recall','ROC AUC SCORE'],
9                                     'Train': [accuracy_score(y_train, y_train_pred),
10                                              precision_score(y_train, y_train_pred),
11                                              recall_score(y_train, y_train_pred),
12                                              roc_auc_score(y_train, y_train_pred)],
13                                     'Test': [accuracy_score(y_test, y_test_pred),
14                                              precision_score(y_test, y_test_pred),
15                                              recall_score(y_test, y_test_pred),
16                                              roc_auc_score(y_train, y_train_pred)]})
17     display(performance_log)
```

# THANK YOU!

