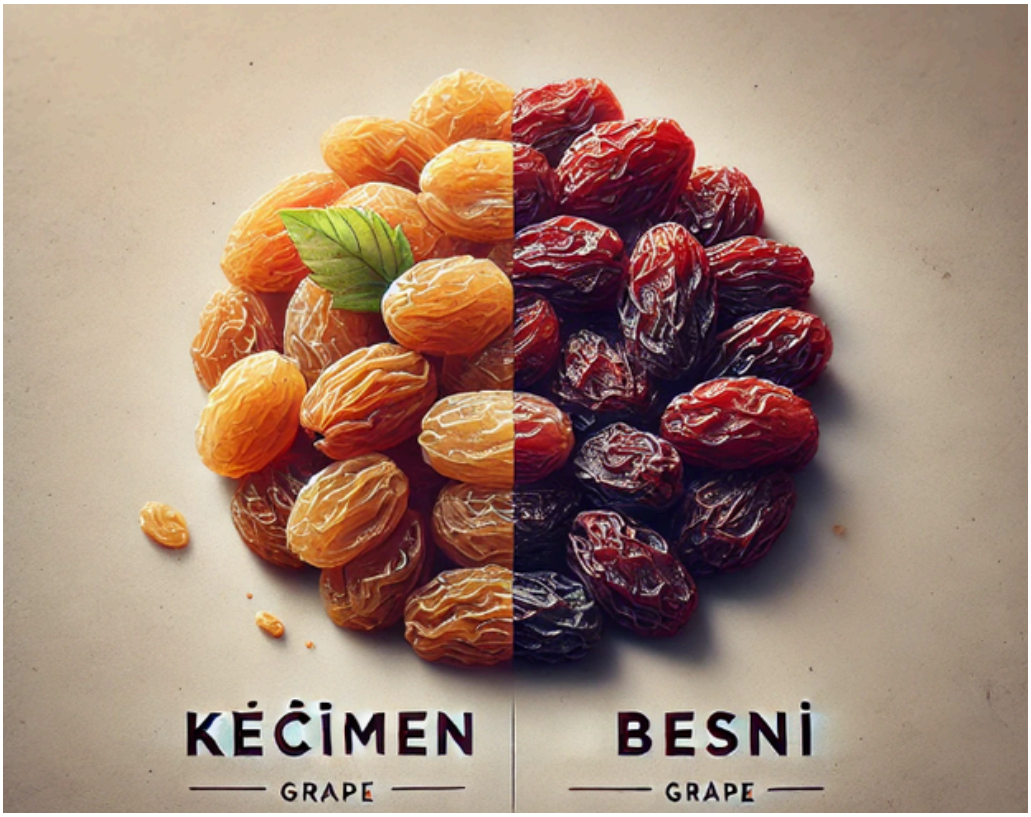
RAISIN CLASS PREDICTION



INTRODUCTION AND OVERVIEW

Project Objective:

The primary goal of this project is to classify two varieties of raisins, Kecimen and Besni, using machine learning techniques. The dataset contains 900 samples, with 450 samples from each variety, and includes several morphological features like Area, MajorAxisLength, MinorAxisLength, Eccentricity, ConvexArea, Extent, and Perimeter. The aim is to build a predictive model that accurately classifies these raisin types based on these features.

DATASET OVERVIEW

Dataset Summary:

- This dataset is designed for the classification of Kecimen and Besni raisin varieties grown in Turkey.
- Data Collection: Images of Kecimen and Besni raisin varieties were obtained using CVS (Computer Vision System). A total of 900 raisin grains were used, with 450 samples from each variety.
- Pre-processing: The images underwent various pre-processing stages, resulting in the extraction of 7 key morphological features.
- Classification Techniques: These features have been classified using three different artificial intelligence techniques.

Feature Descriptions:

1. **Area**: Represents the number of pixels within the boundaries of the raisin.
2. **MajorAxisLength**: Indicates the length of the longest line that can be drawn on the raisin.
3. **MinorAxisLength**: Indicates the length of the shortest line that can be drawn on the raisin.
4. **Eccentricity**: Measures the elliptical shape of the raisin.

5.**ConvexArea:** Represents the number of pixels in the smallest convex shell that can be formed around the raisin.
6.**Extent:** The ratio of the area of the raisin to the area of the bounding box.
7.**Perimeter:** The perimeter length of the raisin.
8.**Class:** The type of raisin, labeled as either Kecimen or Besni.

Data Types:

- The features are primarily numerical (float64 and int64).
- The class label is categorical (Kecimen, Besni).

STEPS UNDERTAKEN

Data Exploration:

The dataset was loaded and basic exploratory data analysis (EDA) was performed, including checking for null values, duplicates, and descriptive statistics.
The distribution of the two classes (Kecimen and Besni) was visualized, confirming that the dataset is balanced, with an equal number of samples for both classes.

Data Visualization:

Various plots were created to visualize the features and their distributions. Box plots and correlation heatmaps were utilized to understand the relationships between features and the separation between classes.

Model Development:

Logistic Regression: The initial model chosen was Logistic Regression, implemented with a pipeline that included scaling of the data using StandardScaler.

Cross-Validation and Model Tuning: Cross-validation was performed to assess the model's performance and avoid overfitting. Additionally, GridSearchCV was used to tune hyperparameters to improve model accuracy.

Model Evaluation:

Various metrics were computed, including accuracy, precision, recall, F1-score, and ROC-AUC. The results showed a well-performing model with an accuracy of 0.88 and an ROC-AUC score of 0.93, indicating strong predictive performance.

Prediction:

The final model was applied to new data for prediction, and the results were presented, including the probability of each prediction.

1 Exploratory Data Analysis (EDA) and Visualization

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.express as px


from plotly.offline import download_plotlyjs,init_notebook_mode,plot,iplot
init_notebook_mode(connected=True)


%matplotlib inline

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss, recall_score, accuracy_score, precision_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay
from sklearn.metrics import RocCurveDisplay, roc_auc_score, auc
from sklearn.metrics import roc_curve, average_precision_score, precision_recall_curve
from sklearn.pipeline import Pipeline

import warnings
warnings.filterwarnings("ignore")
warnings.warn("this will not show")
```

executed in 9.78s, finished 15:30:07 2024-08-15

plotly.express: A high-level, simpler interface for creating visualizations with Plotly.

cufflinks: Allows for direct interaction with Pandas DataFrames to create interactive Plotly visualizations.

download_plotlyjs, init_notebook_mode, plot, iplot: These functions enable the display of interactive graphs in Jupyter Notebook in offline mode.

init_notebook_mode(connected=True): Initializes the mode for displaying interactive Plotly visualizations in Jupyter Notebook. The connected=True argument allows these visualizations to run locally without connecting to the online Plotly server.

cf.go_offline(): Used to run the cufflinks library in offline mode. It enables the creation of interactive graphs in Jupyter Notebook without an internet connection.

%matplotlib inline: Ensures that graphs created with the matplotlib library are displayed directly as cell outputs in Jupyter Notebook.

In [2]:
```
df = pd.read_excel("Raisin_Dataset.xlsx")
```

executed in 462ms, finished 15:30:08 2024-08-15

In [3]:
```
df
```

executed in 22ms, finished 15:30:08 2024-08-15

Out[3]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class
0	87524	442.246011	253.291155	0.819738	90546	0.758651	1184.040	Kecimen
1	75166	406.690687	243.032436	0.801805	78789	0.684130	1121.786	Kecimen
2	90856	442.267048	266.328318	0.798354	93717	0.637613	1208.575	Kecimen
3	45928	286.540559	208.760042	0.684989	47336	0.699599	844.162	Kecimen
4	79408	352.190770	290.827533	0.564011	81463	0.792772	1073.251	Kecimen
...	...	...	...	...	...	...	...	...
895	83248	430.077308	247.838695	0.817263	85839	0.668793	1129.072	Besni
896	87350	440.735698	259.293149	0.808629	90899	0.636476	1214.252	Besni
897	99657	431.706981	298.837323	0.721684	106264	0.741099	1292.828	Besni
898	93523	476.344094	254.176054	0.845739	97653	0.658798	1258.548	Besni
899	85609	512.081774	215.271976	0.907345	89197	0.632020	1272.862	Besni

900 rows × 8 columns

In [4]:
```
df.info()
```

executed in 12ms, finished 15:30:08 2024-08-15

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Area             900 non-null    int64  
 1   MajorAxisLength  900 non-null    float64
 2   MinorAxisLength  900 non-null    float64
 3   Eccentricity     900 non-null    float64
 4   ConvexArea       900 non-null    int64  
 5   Extent           900 non-null    float64
 6   Perimeter        900 non-null    float64
 7   Class            900 non-null    object 
dtypes: float64(5), int64(2), object(1)
```

In [5]:
```
df.shape
```
executed in 4ms, finished 15:30:08 2024-08-15

Out[5]:
```
(900, 8)
```

In [6]:
```
df.duplicated().sum()
```
executed in 13ms, finished 15:30:08 2024-08-15

Out[6]:
```
0
```

In [7]:
```
df.isnull().sum().any()
```
executed in 5ms, finished 15:30:08 2024-08-15

Out[7]:
```
False
```

In [8]:
```
df.describe().T
```
executed in 22ms, finished 15:30:08 2024-08-15

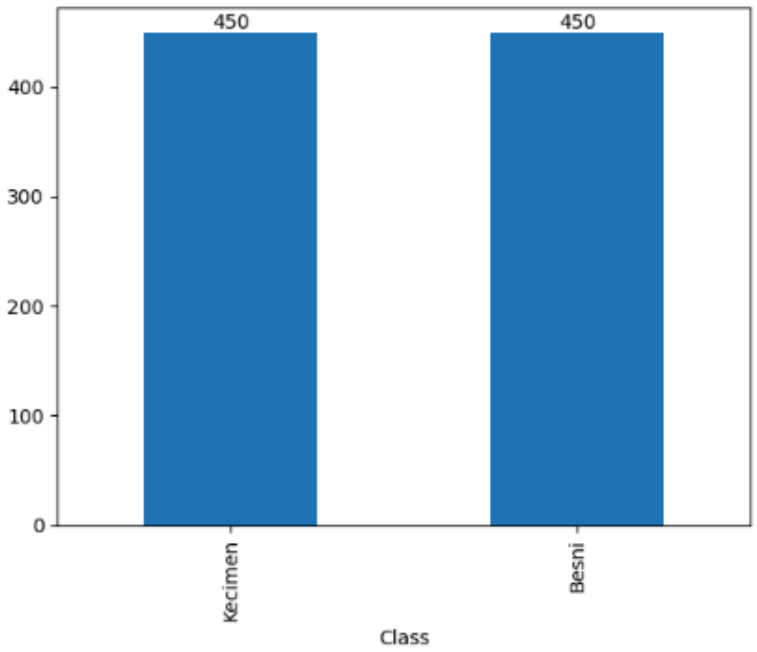
Out[8]:

	count	mean	std	min	25%	50%	75%	max
Area	900.0	87804.127778	39002.111390	25387.000000	59348.000000	78902.000000	105028.250000	235047.000000
MajorAxisLength	900.0	430.929950	116.035121	225.629541	345.442898	407.803951	494.187014	997.291941
MinorAxisLength	900.0	254.488133	49.988902	143.710872	219.111126	247.848409	279.888575	492.275279
Eccentricity	900.0	0.781542	0.090318	0.348730	0.741766	0.798846	0.842571	0.962124
ConvexArea	900.0	91186.090000	40769.290132	26139.000000	61513.250000	81651.000000	108375.750000	278217.000000
Extent	900.0	0.699508	0.053468	0.379856	0.670869	0.707367	0.734991	0.835455
Perimeter	900.0	1165.906636	273.764315	619.074000	966.410750	1119.509000	1308.389750	2697.753000

In [9]:
```
df["Class"].value_counts()
```
executed in 7ms, finished 15:30:08 2024-08-15

Out[9]:
```
Class
Kecimen    450
Besni      450
Name: count, dtype: int64
```

In [10]:
```
ax = df["Class"].value_counts().plot(kind='bar')
ax.bar_label(ax.containers[0]);
```
executed in 147ms, finished 15:30:08 2024-08-15



Balanced Dataset: A balanced dataset is one in which each class contains a similar number of examples, making it suitable for classification problems. For example, Kecimen: 450 - Besni: 450.

Unbalanced/Imbalanced Dataset: An imbalanced dataset occurs when one class has significantly more examples than the others. For instance, Diabetes (1): 300 - Non-Diabetes (0): 5000.

In the case of this dataset, we have an equal number of examples for both classes. Therefore, we can use accuracy to evaluate the model's performance.

In [11]:
```
df["Class"] = df["Class"].map({"Kecimen":0,"Besni":1})
```
executed in 3ms, finished 15:30:08 2024-08-15

In [12]:
```
df
```
executed in 14ms, finished 15:30:08 2024-08-15

Out[12]:

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class
0	87524	442.246011	253.291155	0.819738	90546	0.758651	1184.040	0
1	75166	406.690687	243.032436	0.801805	78789	0.684130	1121.786	0
2	90856	442.267048	266.328318	0.798354	93717	0.637613	1208.575	0
3	45928	286.540559	208.760042	0.684989	47336	0.699599	844.162	0
4	79408	352.190770	290.827533	0.564011	81463	0.792772	1073.251	0
...	...	...	...	...	...	...	...	...
895	83248	430.077308	247.838695	0.817263	85839	0.668793	1129.072	1
896	87350	440.735698	259.293149	0.808629	90899	0.636476	1214.252	1
897	99657	431.706981	298.837323	0.721684	106264	0.741099	1292.828	1
898	93523	476.344094	254.176054	0.845739	97653	0.658798	1258.548	1
899	85609	512.081774	215.271976	0.907345	89197	0.632020	1272.862	1

900 rows × 8 columns

In [13]:
```
# features = df.columns[:-1]
# for i in features:
#     fig = px.box(df, x=i)
#     fig.show()
```
executed in 6ms, finished 15:30:08 2024-08-15

In [14]:
```
df.iloc[:,:-1].iplot(kind="box")

#The iplot function from the plotly library needs to be installed for this code to work.
#The plotly library is used to create interactive and customizable graphs.
#The iplot function is a function in the plotly library that allows you to directly plot a DataFrame as a graph
```
executed in 632ms, finished 15:30:09 2024-08-15

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_3812\863862614.py in ?()
----> 1 df.iloc[:,:-1].iplot(kind="box")
      2
      3 #The iplot function from the plotly library needs to be installed for this code to work.
      4 #The plotly library is used to create interactive and customizable graphs.

~\anaconda\Lib\site-packages\pandas\core\generic.py in ?(self, name)
   5985             and name not in self._accessors
   5986             and self._info_axis._can_hold_identifiers_and_holds_name(name)
   5987         ):
   5988             return self[name]
-> 5989         return object.__getattribute__(self, name)

AttributeError: 'DataFrame' object has no attribute 'iplot'
```

In [ ]:
```
fig = px.box(df, color="Class", color_discrete_map={"Kecimen":'#FF0000',"Besni":'#00FF00'})
fig.show()

#Here, by examining the data on a class basis, we can better distinguish outlier values.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
df.iplot(kind="bar")

# 0: Keçimen, 1: Besni
# Eccentricity, Extent (not a distinguishing feature)

# The first 450 observations in our data are labeled as 0 = Keçimen
# The last 450 observations are labeled as 1 = Besni
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
fig = px.bar(df,x=df.index,y="Area",color="Class",color_discrete_map={"Kecimen":'#FF0000',"Besni":'#00FF00'})
fig.show();
```
executed in 0ms, finished 15:30:09 2024-08-15

-In logistic regression, the importance of multicollinearity for accurately estimating coefficients is less significant compared to linear regression.

-In logistic regression, since the dependent variable is categorical, a linear model is not used.

Here, the interpretation of coefficients is done based on odds.

-Logistic regression predicts the probability of the dependent variable.

-In this case, high correlation among independent variables does not affect the performance of logistic regression.

-However, severe multicollinearity can reduce the accuracy and reliability of logistic regression, potentially leading to misleading results.

-Additionally, because L1 and L2 regularization techniques are applied in the background in logistic regression, this issue is mitigated.

In [ ]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df.select_dtypes(include='number').corr(),vmin=-1,vmax=1, annot=True, fmt='.3f', cmap='coolwarm');
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
corr_matrix = df.corr()
fig = px.imshow(corr_matrix)
fig.show();

#an alternative heatmap
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
sns.pairplot(df, hue = "Class");

# I can't see a clear separation between the classes.
# However, let's not forget that we're only looking at it in two dimensions.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
fig = px.scatter_3d(df, x='Perimeter', y='Area', z='Extent', color='Class')
fig.show()

# When we look in three dimensions, we can see that the classes actually separate well from each other,
# but we also notice that there are complex points where our model might make mistakes.
```
executed in 0ms, finished 15:30:09 2024-08-15

2 Logistic Regression Model

For example, an e-commerce site wants to predict whether a customer will purchase a product.

Independent variables may include factors such as the customer's age, gender, past purchase history, and click history. The combination of these independent variables creates a model that determines the likelihood of a customer purchasing a product.

Logistic regression is a statistical method used to model the relationship between a dependent variable (in this case, whether a customer will purchase a product or not) and independent variables.

Logistic regression creates a linear model based on the combination of independent variables. This model uses a logit transformation to provide results as a probability between 0 and 1, determining the likelihood of a person purchasing a product. This probability is classified by setting a threshold value (e.g., 0.5); that is, if it is greater than 0.5, the likelihood of a customer purchasing the product is high; otherwise, it is low.

In conclusion, logistic regression is a statistical technique used to model the relationship between a dependent variable (e.g., whether a customer will purchase a product or not) and independent variables (e.g., the customer's age, gender, and past purchase history).

Fundamental Concepts:

Logistic regression is a statistical analysis method used to model the relationship between a dependent variable, often associated with binary outcomes such as yes/no, and one or more independent variables. For example, it can be used to determine whether a patient has a disease or to predict a student's probability of passing an exam.

Probability and Odds Ratio: Logistic regression predicts the probability (P) of an event and the odds ratio associated with that probability. The odds ratio is the ratio of the probability of the event occurring to the probability of it not occurring. For example, the probability of a student passing an exam is compared to the probability of not passing.

Sigmoid Function: Logistic regression uses the sigmoid (S-shaped) function, which limits the results between 0 and 1. This is ideal for probability predictions because we can interpret probabilities directly.

Mathematical Model: The logistic regression model is generally expressed as follows:

$$\log(P / (1 - P)) = \beta_0 + \beta_1 * X_1 + \beta_2 * X_2 + ... + \beta_n * X_n$$

Here ,P represents the probability of the event, and β_0, β_1, ..., β_n are the coefficients, while X_1, X_2, ..., X_n are the independent variables.

Interpretation of the Model: The coefficient (beta) for each independent variable indicates its effect on the log-odds of the event. A positive coefficient suggests that an increase in the variable raises the probability of the event, while a negative coefficient suggests a decrease.

Application Examples: Logistic regression is utilized in various fields, including medicine (disease diagnosis), economics (bankruptcy prediction), and marketing (probability of product purchase).

Model Evaluation: Various metrics are used to assess model performance. These include confusion matrix, ROC curve, and AUC value.

Important Points:

- Logistic regression works with a categorical dependent variable rather than a continuous one, unlike linear regression.
- The model offers a probability-based approach and is particularly suitable for modeling binary outcomes.
- The assumptions and applicability of the model may vary depending on the data analyzed and the research question.

The Presence of the Term "Regression" in Logistic Regression

1- Linear Regression Based: Logistic regression is essentially an extended form of a linear regression model. While linear regression is used to predict a continuous dependent variable, logistic regression adapts this approach for a categorical dependent variable (typically binary).

2- Log-Odds Transformation@: Logistic regression establishes a linear relationship by modeling the probability of the dependent variable through the logit function (log-odds). Thus, it follows the fundamental principles of linear regression but adapts the results for a binary classification problem.

3- Historical Development: The term's origin dates back to the early days of statistics and regression analysis. At that time, the term "regression" was generally used to describe drawing the line (or curve) that best fits the data. Logistic regression was developed as a specific type of regression where the dependent variable is expressed as a probability.

4- Nature of the Model: Although logistic regression is used as a classification algorithm, it fundamentally predicts a continuous probability value (between 0 and 1). Thus, it generates a continuous output that represents a categorical outcome.

5- Statistical Approach: In statistics, the term "regression" defines methods for modeling the effect of independent variables on a dependent variable. In this context, logistic regression is used to model the effect of independent variables on a categorical outcome.

In summary, the name "logistic regression" includes the term "regression" due to both its historical origins and mathematical structure. This reflects that the method follows the fundamental principles of regression analysis while being used for classification purposes.

https://medium.com/academy-team/lojistik-regresyon-g%C3%BCc%C3%BCn%C3%BC-nereden-al%C4%B1yor-1f7cd17dbb05

2.1 Train | Test Split and Scaling

In [ ]:
```
df.info()
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
X = df.drop(["Class"], axis=1)
y = df["Class"]
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    stratify=y,
                                                    random_state=10)

# stratify=y no need to use for the balance data
```
executed in 0ms, finished 15:30:09 2024-08-15

stratify: Ensures that class ratios are preserved when splitting the dataset into training and testing sets.

stratify=y: If the classes in your dataset are imbalanced (for example, if one class has significantly more samples than another), making a random train-test split can result in some classes being underrepresented or not represented at all in the test set. The stratify parameter guarantees that the class distributions in the training and test sets are the same as those in the original dataset, even in cases of such imbalances.

The stratify parameter is used by providing an array that contains the class labels (usually the target variable - y -). This ensures that class ratios are preserved during the separation of the training and test sets.

The default value of the stratify parameter is - None -. This means that the training and test sets will be randomly split without preserving class ratios. If you want to maintain class ratios, you should specify this parameter with the target variable y.

2.2 Logistic Regression with Pipeline

Logistic Regression is a classification method.

- It allows data to be separated according to specific classification criteria.
- It enables the separation of data into binary classes or multi-class categories.

Therefore, it is categorized as a method that solves classification problems.

The sigmoid function is a type of activation function that limits input values between 0 and 1.

The default threshold value is 0.5 but can be changed.

In logistic regression, we cannot interpret coefficients directly as in linear regression; we interpret them through odds ratios.

Logistic regression transforms our classical linear regression equation (b2X2 + b1X1 + b0) into probabilities.

Where is it used?

Solving binary classification problems (e.g., spam/ham emails) Solving multi-class classification problems (e.g., priority/normal emails) Classifying medical data (e.g., disease/healthy) Classifying financial data (e.g., profitable/unprofitable stocks) Classifying advertising data (e.g., relevant/irrelevant ads) Determining the Threshold Value

The threshold value is a cutoff used to classify a model's predicted probabilities into binary outcomes. The optimal threshold value is determined to maximize classification performance.

ROC Curve and AUC

The ROC curve is a graph that shows the sensitivity (recall) and specificity values of the model at different threshold values. AUC (Area Under the Curve) represents the area under the ROC curve and is a measure of classification performance. The optimal threshold value corresponds to the point with the highest AUC on the ROC curve.

Youden's J Statistic

Youden's J statistic is used to find the threshold value corresponding to the highest AUC on the ROC curve. Its formula is: J = Sensitivity + Specificity - 1. This statistic determines the optimal threshold value by considering sensitivity and specificity values.

F1 Score

The F1 score represents the harmonic mean of precision and recall values. The F1 score is a measure of classification performance, and the optimal threshold value is determined as the one that yields the highest F1 score.

By selecting one of these methods, you can determine the optimal threshold value. The choice of method should depend on the characteristics of your dataset and your classification goals.

Pipeline is a concept where the steps in a data processing workflow are executed in a connected manner.

A pipeline generally consists of several steps, where each step processes the output of the previous step and provides input to the next step.

Within a pipeline, transformation algorithms that perform fit and transform operations are specified first, followed by machine learning algorithms that perform fit and predict operations.

In [ ]:
```
scaler = StandardScaler()
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
log_model = LogisticRegression()
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
log_pipe = Pipeline([("scaler",scaler),("log_model",log_model)])
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
log_pipe.fit(X_train, y_train)
```
executed in 0ms, finished 15:30:09 2024-08-15

log_pipe.fit(X,y) burda sırasıyla ne işlem yapar?

- pipe_model içerisindeki ilk yapılacak işlem StandardScaler() olduğundan;
- X'e standard scale uygulanır.
- pipe_model içerisindeki ikinci yapılacak işlem LogisticRegression() olduğundan;
- Scallenmiş X ile birlikte y Logistic modele verilerek eğitim tamamlanır.

In [ ]:
```
# operations = [("scaler", StandardScaler()), ("logistic", LogisticRegression())]
# pipe_model = Pipeline(steps=operations)
# pipe_model.fit(X_train, y_train)
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
y_pred = log_pipe.predict(X_test)
y_pred

# The predict function predicts which classes the test data belongs to.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
y_pred_proba = log_pipe.predict_proba(X_test)
y_pred_proba

# The predict_proba function shows the probability of each observation in the test data belonging to each class.
# The first value represents the probability of the observation being in class 0,
# while the second value represents the probability of it being in class 1.
# The observation is assigned to the class with the higher probability.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
# # A pipeline with MinMaxScaler
# pipeline_minmax = Pipeline([
#     ('scaler', MinMaxScaler()),
#     ('log_model', log_model)
# ])

# #  A pipeline with StandardScaler
# pipeline_standard = Pipeline([
#     ('scaler', StandardScaler()),
#     ('log_model', log_model)
# ])
```
executed in 0ms, finished 15:30:09 2024-08-15

2.3 X_test + y_yest + y_pred + y_pred_proba

In [ ]:
```
test_data = pd.concat([X_test, y_test], axis=1)
test_data

# In this code, X_test and y_test are combined along the columns (axis=1) to form the test_data DataFrame.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
test_data["pred_proba"] = y_pred_proba[:,1]

test_data

# we add a pred_proba column/feature to the test_data DataFrame that shows the probability of belonging to class 1.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
test_data["pred"] = y_pred
test_data

# When the model performs the prediction process, it assigns probabilities of 0.5 and above to class 1, and probabilities belo
# The predictions made by the model are then added as a pred feature to the test_data DataFrame.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
test_data.sample(10)

# df'den rastgele 10 tane gözlem seçiyoruz
```
executed in 0ms, finished 15:30:09 2024-08-15

2.4 Model Performance

accuracy: The proportion of true results (both true positives and true negatives) among the total number of cases examined.

recall (sensitivity): The rate of capturing positives (the rate of capturing 1s)

recall: How many of the actual 1s in real life did my model predict as 1

specificity: The rate of capturing negatives, the opposite of recall; the rate of capturing 0s

precision: Out of the ones I predicted as positive, how many were correct

precision: How many of the predictions labeled as 1 by my model were actually 1 in real life

precision: The predictive power of my model

F1: Measures the balance between precision and recall

$$
\begin{aligned}
precision &= \frac{TP}{TP+FP} \\
recall &= \frac{TP}{TP+FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP+TN}{TP+FN+TN+FP} \\
specificity &= \frac{TN}{TN+FP}
\end{aligned}
$$

		Predicted		
		Yes	No	
Actual	Yes	2 (True +ve)	1 (False -ve)	2/(2+1)=⅔ Recall (Sensitivity)
	No	2 (False +ve)	3 (True -ve)	3/(3+2)=⅗ (Specificity)
		2/(2+2)=50% (Precision)		Accuracy= (2+3)/(2+1+2+3)= 5/8

Accuracy: Indicates how many of the model's predictions are correct. It is calculated by dividing the sum of True Positives (TP) and True Negatives (TN) by the total number of predictions (including TP, TN, False Positives (FP), and False Negatives (FN)). Formula:

```
Accuracy = (TP + TN) / (TP + TN + FP + FN)
```

Healthcare Sector: In a disease screening test, accuracy represents the rate at which the presence or absence of a disease is correctly identified. Accuracy reflects the overall performance of the test and indicates how reliable the test is.

Weather Forecasting: In meteorology, the accuracy of a weather forecasting model shows how accurately it predicts weather conditions, such as rain or sunshine, for a given day. Accurate predictions help people plan their daily activities more effectively.

Recall (Sensitivity or True Positive Rate): Indicates how well the model captures the positive class. It expresses how many of the actual positive examples were correctly predicted as positive by our model. Formula:

```
Recall = TP / (TP + FN)
```

Cancer-Heart Disease Diagnosis: In disease diagnosis, the recall rate indicates how well the test correctly identifies patients who truly have cancer or heart disease as positive. High recall reduces the risk of missing patients with cancer or heart disease, which is critical for early intervention and can increase the chances of survival. **Specificity (True Negative Rate):** Indicates how well the model captures the negative class and is the inverse of recall. It measures how many of the actual negative examples were correctly predicted as negative. Formula:

```
Specificity = TN / (TN + FP)
```

Spam Filtering: In an email service, the specificity rate shows how accurately non-spam emails are correctly classified as safe. High specificity reduces the risk of important emails being mistakenly marked as spam.

Security Systems: High specificity is necessary to reduce false alarms in a security system. A false positive classification by a security camera or motion sensor can lead to false alarms and unnecessary security interventions.

Precision: Indicates how many of the examples predicted as positive are actually positive. It reflects how "precise" the model is, meaning how many of the things it labeled as positive are truly positive. Formula:

```
Precision = TP / (TP + FP)
```

Credit Scoring: In a credit risk model used by a bank, the precision rate indicates how many of the applicants predicted to be high-risk actually experience issues with loan repayments. High precision helps the bank reduce the risk of credit losses.

Customer Targeting Campaigns: In a marketing campaign, high precision is necessary to reach the correct customer segment. The more accurately promotions and advertisements are targeted to customers, the higher the conversion rates, leading to a more efficient use of the marketing budget.

Precision and recall can often have an inverse relationship, meaning it's important to remember that increasing one can lead to a decrease in the other. Therefore, it is often necessary to balance these two metrics.

F1 Score: A metric based on the harmonic mean that takes both precision and recall into account.

```
F1 Skoru = 2 * (Precision * Recall) / (Precision + Recall)
```

Fraud Detection: In the finance sector, the F1 score is very important in fraud detection systems. These systems typically contain very low rates of positive examples (actual fraud cases), so it is crucial to minimize both false positives (false alarms) and false negatives (missed fraud cases). A high F1 score indicates that the fraud detection system is both precise and sensitive, meaning it has the ability to accurately identify real fraud incidents without missing them. This metric tends to favor models where both precision and recall are high, which is particularly important in areas that can lead to critical outcomes. The F1 score is very useful for evaluating the overall performance of a model, especially in situations where the costs of false negatives and false positives are high.

In [ ]:
```
accuracy_score(y_test,y_pred)

# This metric shows how many of the actual values I predicted correctly in my data.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
log_pipe.score(X_test, y_test)

# When we provide X_test and y_test to the score function, it returns the accuracy score.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
precision_score(y_test, y_pred, pos_label=1)

#By default, it returns the precision score for class 1.
# It shows how many of the predictions made for predicting class 1 are accurate.
# A precision of 0.1 means that the model was correct in 10% of its predictions.
# To obtain the score for class 0, it shoul
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
precision_score(y_test, y_pred, pos_label=0)

# By default, it returns the precision score for class 1. It indicates how many of the predictions made for class 1 are accurc
# A precision of 0.1 means that the model was correct in 10% of its predictions.
# To obtain the score for class 0, it should be adjusted with pos_label=0.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
recall_score(y_test, y_pred,  pos_label=1)

# By default, it returns the recall score for class 1. It indicates how much of class 1 the model was able to correctly identi
# A recall of 0.1 means that the model was only able to detect 10% of class 1.
# To obtain the score for class 0, it should be adjusted with pos_label=0.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
recall_score(y_test, y_pred,  pos_label=0)
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
f1_score(y_test,y_pred, pos_label=1)

# Returns the harmonic mean of the precision and recall scores.
# There is always a trade-off between precision and recall scores.
# If precision increases/decreases, recall decreases/increases.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
f1_score(y_test,y_pred, pos_label=0)
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
confusion_matrix(y_test,y_pred)

# The confusion matrix is a simplified representation.
# It can cause confusion since the true label and predicted label axes are not displayed..
```
executed in 0ms, finished 15:30:09 2024-08-15

		Predicted	
		Negative **(N)** -	Positive **(P)** +
Actual	Negative -	True Negative **(TN)**	**False Positive (FP) Type I Error**
	Positive +	**False Negative (FN) Type II Error**	True Positive **(TP)**

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_estimator(log_pipe, X_test, y_test, cmap="inferno")

# Since the true label and predicted label axes are displayed, it is easier to interpret the confusion matrix.
# True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) can be derived from the confusion m
# You can visualize the confusion matrix with the following command:
```
executed in 0ms, finished 15:30:09 2024-08-15

plasma-inferno-magma-cividis-Greys-Blues-PuBu-YlGnBu-hot

https://matplotlib.org/stable/users/explain/colors/colormaps.html

In [ ]:
```
def eval_metric(model, X_train, y_train, X_test, y_test):

    """ to get the metrics for the model """

    y_train_pred = model.predict(X_train)
    y_pred = model.predict(X_test)

    print("Test_Set")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    print()
    print("Train_Set")
    print(confusion_matrix(y_train, y_train_pred))
    print(classification_report(y_train, y_train_pred))

# To observe the scores for both the train and test sets, we define our eval_metric function.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
eval_metric(log_pipe, X_train, y_train, X_test, y_test)

# If we talk about the label 1:
# A precision of 0.89 means that 89% of my predictions are accurate.
# A recall of 0.84 means that I correctly identified 84% of the actual class 1 instances.
# The F1 score provides the harmonic mean of precision and recall, so it should be interpreted based on your target metric (re
# Interpreting it in isolation can lead to misinterpretation.
# To determine if there is overfitting in the data, we compare the scores of the target class in the test and train sets.
# If the training set scores are significantly better than the test set scores, indicating a large gap, it suggests that overf
# However, we do not make definitive overfitting or underfitting decisions solely based on the train and test set performance.
# A definitive decision on overfitting or underfitting will be made after cross-validation.
```
executed in 0ms, finished 15:30:09 2024-08-15

2.5 Cross Validate

In [ ]:
```
from sklearn.metrics import get_scorer_names

scorers = get_scorer_names()
print(scorers)

# from sklearn.metrics import SCORERS
# SCORERS.keys()

# The list of metrics we will use for cross-validation and GridSearchCV for classification models.

# We will take the 'accuracy', 'precision', 'recall', and 'f1' metrics from this list.

# The metric names must be written in the following format for use in cross-validation and GridSearchCV; otherwise, you will r
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
model = Pipeline([("scaler",scaler),("log_model",log_model)])

scores = cross_validate(model, X_train, y_train, scoring = ['precision','recall','f1','accuracy'], cv = 10, return_train_score
df_scores = pd.DataFrame(scores, index = range(1, 11))
df_scores

# To make a definitive decision on overfitting/underfitting through cross-validation, we set return_train_score=True
# in order to compare the scores of both the train and validation sets in each iteration.

# The scores returned below are always for class 1. To obtain the CV scores for metrics related to class 0,
# we need to perform additional steps.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
df_scores.mean()[2:]

# The train and validation scores for label 1 are compared at this stage to make a definitive decision on overfitting/underfit
# Based on the scores below, we can say that there is no overfitting.
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
eval_metric(log_pipe, X_train, y_train, X_test, y_test)

# The scores obtained from here are the definitive and final scores of the model.
# We determine whether these scores are truly consistent by comparing them with the validation set scores obtained from cross-
# If the scores are close to each other, we can say that the test data scores are consistent.
```
executed in 0ms, finished 15:30:09 2024-08-15

2.6 GridSearchCV

In [ ]:
```
log_pipe.get_params()

# If we are using a pipeline model within GridSearchCV,
# we can view the list of hyperparameters for the ML algorithm defined in the pipeline using get_params().

# The hyperparameters we will write into the param_grid parameter of GridSearchCV must be formatted as shown below.

# Note that after the name of the ML model we provided in the pipeline (e.g., "log_model"),
# there are two underscores (__) followed by the actual name of the ML model's hyperparameter.
```
executed in 0ms, finished 15:30:09 2024-08-15

Logistic Regression Hyperparametreleri : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

In [ ]:
```
from sklearn.model_selection import GridSearchCV
```
executed in 0ms, finished 15:30:09 2024-08-15

penalty: Specifies which regularization parameter will be applied in the background. Default is l2; l1 represents Lasso, while l2 represents Ridge.

C: Represents the strength of the regularization, similar to alpha in ridge and lasso. However, as the value increases, regularization decreases, acting contrary to alpha. Thus, if overfitting is present, the value of C should be decreased. Default: 1.

class_weight: In unbalanced datasets, we can use `class_weight="balanced"` to balance the classes. When the data is balanced, the model tries to increase the precision score of the underrepresented class, thereby increasing its recall score. Class weights can be set to balanced, or None for no class weights.

In [ ]:
```
# pipeline for logistic regression

model = Pipeline([("scaler", scaler), ("log_model", log_model)])

# l1: Lasso, l2: Ridge
penalty = ["l1", "l2"]

# To obtain 20 values of C between -1 and 5,
# as C increases, regularization decreases, acting contrary to alpha.
# The reason for using np.logspace for C is that the optimal value of regularization
# is typically not within a specific range.
# Therefore, it makes sense to generate values on a logarithmic scale
# to test different values across a wide range and find the one that yields the best result..

C = np.logspace(-1, 5, 20)

# balanced: Class weights are balanced, meaning that the algorithm will adjust the weights inversely proportional to class fre
# None: No class weights are applied, meaning that all classes are treated equally during training.

class_weight = ["balanced", None]

# In unbalanced datasets, we can use `class_weight="balanced"` to balance the classes.
# When the data is balanced, the model attempts to increase the precision score of the underrepresented class,
# thereby improving its recall score.


# The four values for the `solver` parameter are:
solver = ["lbfgs", "liblinear", "sag", "saga"]


param_grid = {
    "log_model__penalty": penalty,
    "log_model__C": [C, 1],  # hata alan alanlar "log_model__C" : C, yapsınlar
    "log_model__class_weight": class_weight,
    "log_model__solver": solver
}

# to select nest model:
grid_model = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=10,
    scoring='accuracy',  # Default olarak 1 sınıfının scorunu max. eder
    n_jobs=-1)  # bütün işlemciler burda çalışsın diye

# Since GridSearchCV also performs Cross Validation, we specify `return_train_score=True` to obtain the scores for both the va
# (The `scoring` parameter will return only the specified metric score.)
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
grid_model.fit(X_train,y_train)
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
grid_model.best_params_
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
grid_model.best_index_
```
executed in 0ms, finished 15:30:09 2024-08-15

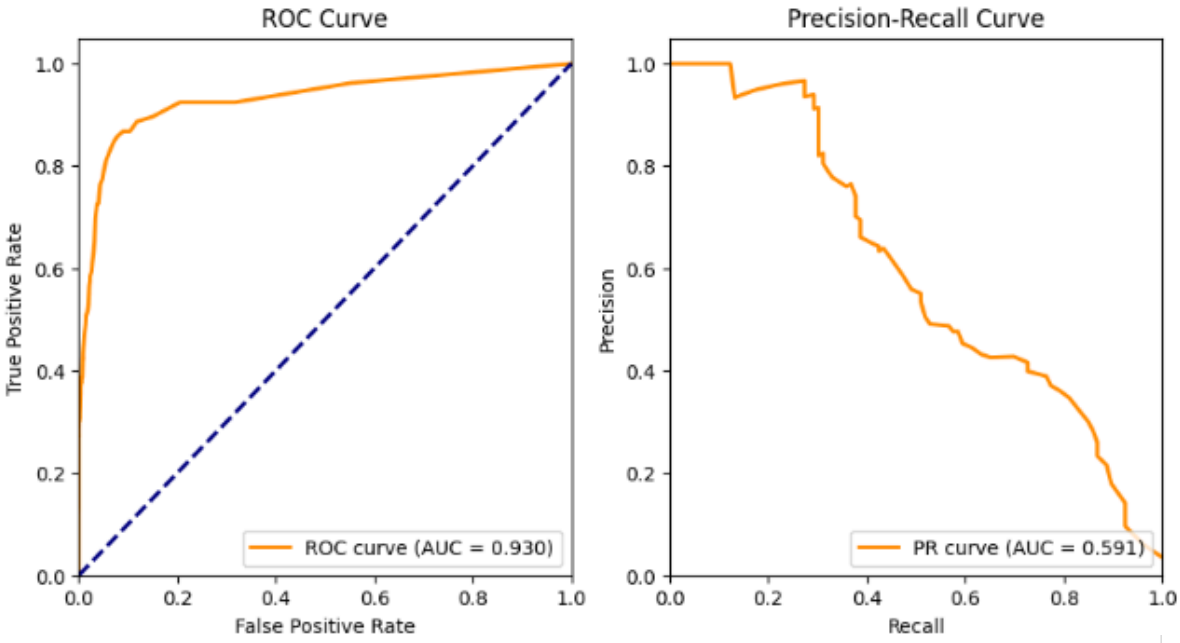
In [ ]:
```
grid_model.best_score_
```
executed in 0ms, finished 15:30:09 2024-08-15

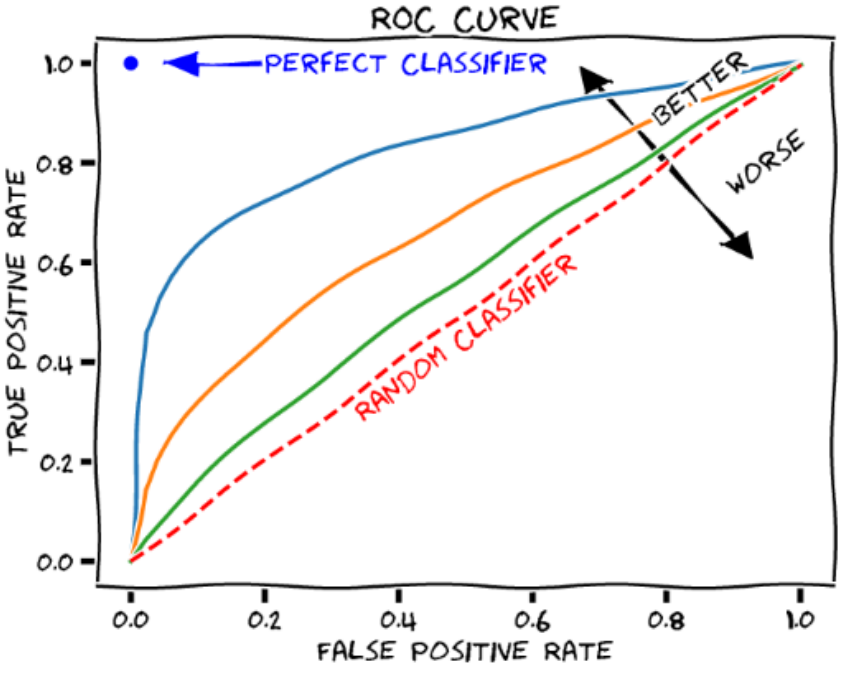
In [ ]:
```
eval_metric(grid_model, X_train, y_train, X_test, y_test)

# The accuracy of the test set increased from 0.87 to 0.88, with 21 incorrect predictions.
# Compared to the previous test set, there is an improvement of 1 point, and the number of errors decreased from 23 to 21.
```
executed in 0ms, finished 15:30:09 2024-08-15

2.7 Precision-Recall Curve & ROC (Receiver Operating Curve) and AUC (Area Under Curve)

$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP} \\
specificity &= \frac{TN}{TN + FP}
\end{aligned}
$$

		Predicted: Yes	Predicted: No	
Actual	Yes	2 (True +ve)	1 (False -ve)	2/(2+1)=⅔ **Recall (Sensitivity)**
	No	2 (False +ve)	3 (True -ve)	3/(3+2)=⅗ **(Specificity)**
		2/(2+2)=50% **(Precision)**		**Accuracy=** (2+3)/(2+1+2+3)= 5/8





1. **Average Precision (AP):**
    - AP is ideal for unbalanced classification problems, such as medical diagnoses or anomaly detection, where one class may be much rarer than another.
    - AP measures the ratio of true positives that the classification model correctly identifies, focusing on minimizing the number of false positives.
    - AP uses the number of correctly classified positives (True Positives, TP) and the number of false positives (False Positives, FP). It is calculated from the Precision-Recall curve. Precision expresses the ratio of true positives to the total positive predictions, while Recall (sensitivity) represents the ratio of true positives to the total number of actual positive samples.
    - For example, in a cancer detection model, AP measures how many actual cancer cases were correctly identified.
2. **Receiver Operating Characteristic Area Under the Curve (ROC AUC):**
    - ROC AUC is more suitable for balanced datasets, meaning it is used when there are no significant discrepancies in class balance.
    - It measures how well the model separates both positive and negative classes, considering the effects of false positives and false negatives.
    - ROC AUC evaluates different precision and recall values by changing the classification threshold and plots these values. The ROC curve shows the relationship between True Positive Rate (TPR) and False Positive Rate (FPR) based on these threshold values.
    - ROC AUC measures the discriminative power of the classification model between classes. The closer the value is to 1, the better the model's performance.
    - For example, in a spam filter model, ROC AUC measures the ability to correctly distinguish between spam and non-spam emails.

The choice of which metric to use depends on the context of the problem and your objectives.

For instance, AP is preferred when it is crucial to accurately detect rare cases, while ROC AUC is used when a general assessment of the separation between classes is needed.

Typically, both metrics are used together, with high values desired for each.

However, since each problem is unique, it is important to carefully evaluate which metric should be prioritized.

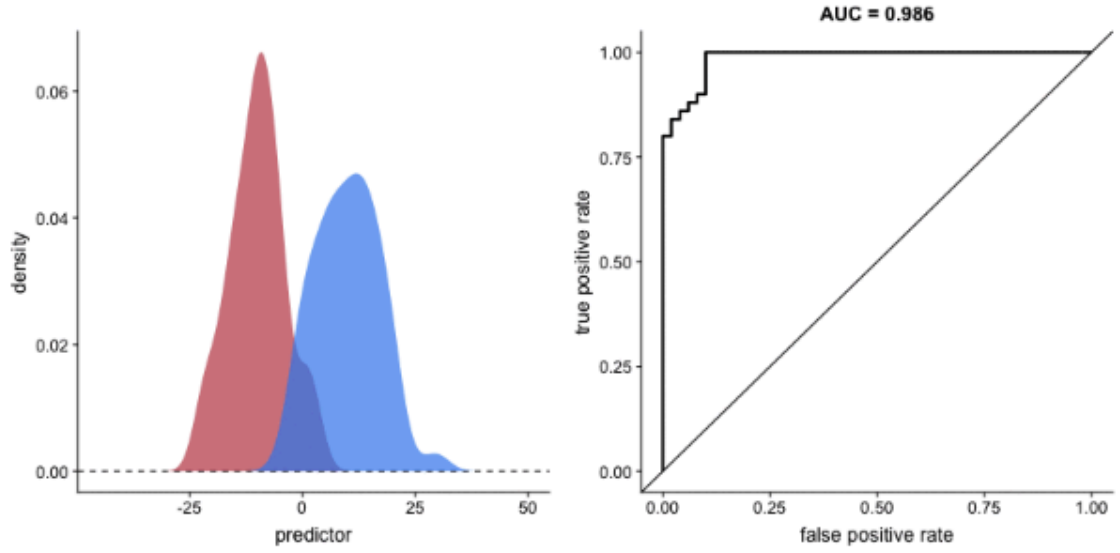
Purpose of the ROC Curve:

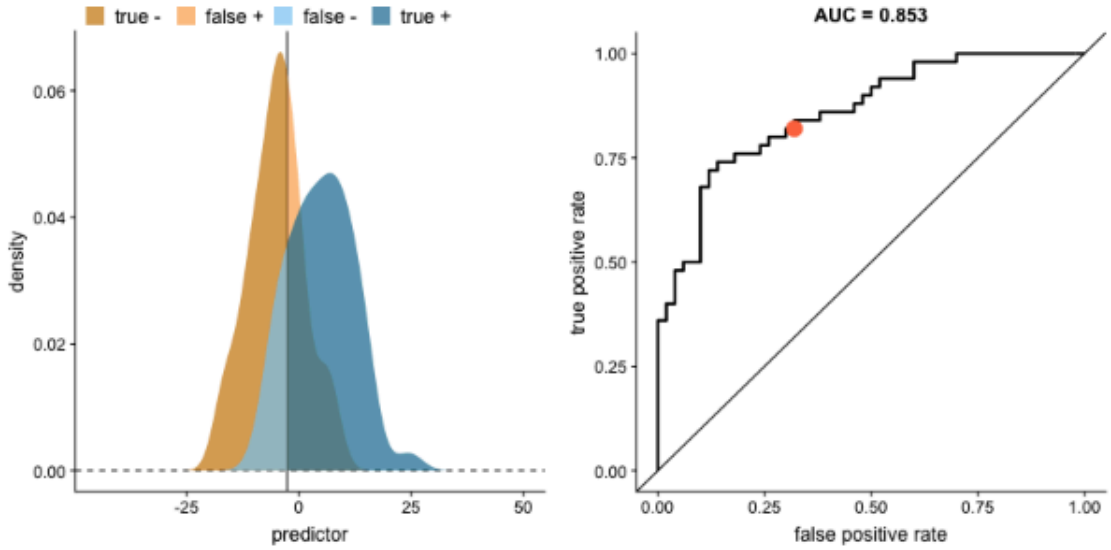
1. To analyze the strength/predictive power of a classifier.
2. To determine the optimal threshold.
3. To compare two models (using the area under the curve).

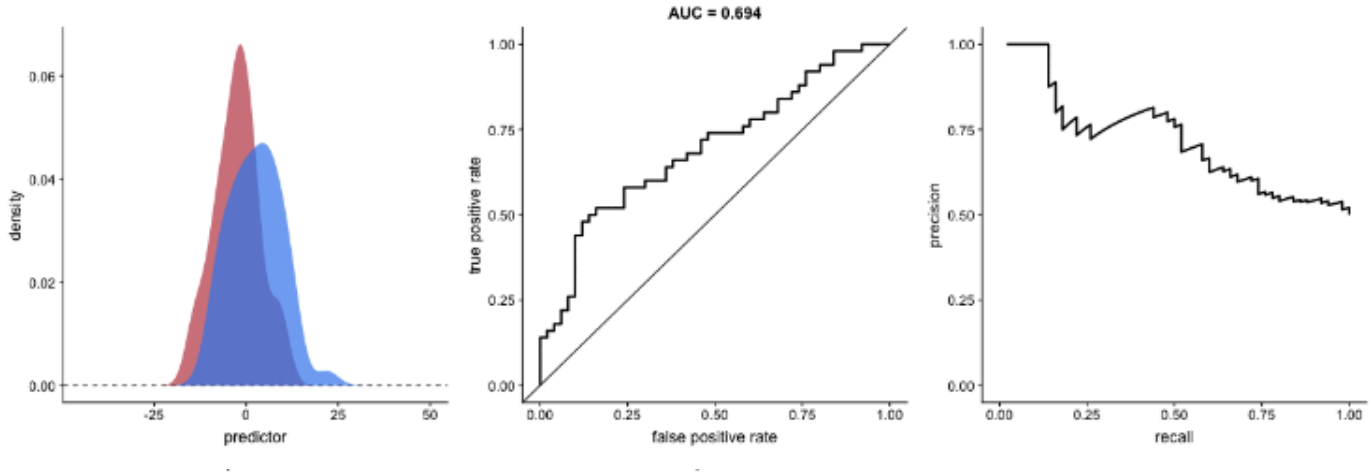
ROC Curve (Receiver Operating Characteristic)

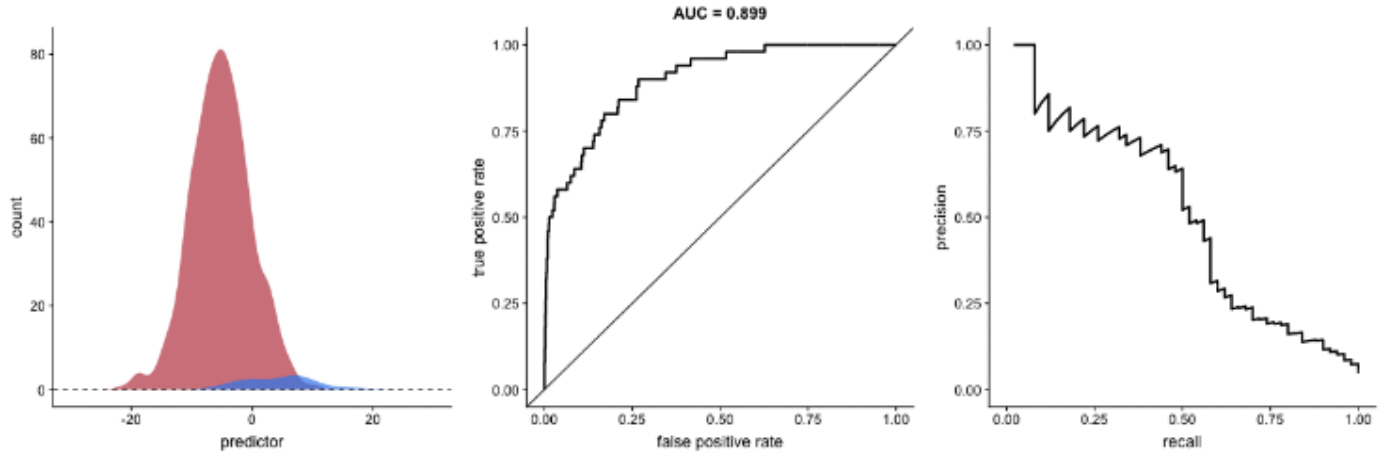
- Its origins date back to World War II.
- It was used as a tool to evaluate radar signal processing techniques during World War II. This curve was developed to measure the ability of radar operators to detect enemy aircraft.
- After the attack on Pearl Harbor, the U.S. military undertook an intense effort to improve radar technology. As part of this effort, the ROC curve was used to assess the accuracy and reliability of radar signals. The primary aim was to minimize the false alarm rate while accurately detecting enemy aircraft.
- The ROC curve illustrates the relationship between the True Positive Rate (TPR) and the False Positive Rate (FPR) for all possible threshold values of a classifier. This makes it an excellent tool for evaluating classifier performance, as it shows performance across all possible thresholds, not just a single threshold.
- The goal is to analyze the predictive power of the classifier in ensuring that as many true positives as possible are detected while minimizing false positives.

ROC AUC Curve









In [ ]:

```
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_estimator(grid_model, X_test, y_test)

# How well does my model distinguish between two types of grapes?

# Since my data is balanced, we are looking at the AUC score. An AUC of 0.93 means that my model is, on average, 93% successfu

# This is the state after performing grid search.

# To visualize the ROC curve, use the following command:
# plot_roc_curve(grid_model, X_test, y_test);
```

executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
y_pred_proba = grid_model.predict_proba(X_test)[:, 1]
auc = roc_auc_score(y_test, y_pred_proba)

RocCurveDisplay.from_estimator(grid_model, X_test, y_test)

plt.text(0.6, 0.2, f'AUC = {auc:.4f}', fontsize=12)
plt.show()
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
RocCurveDisplay.from_estimator(log_pipe, X_test, y_test);

# The state before performing grid search.

# plot_roc_curve(log_pipe, X_test, y_test);
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
y_pred_proba = log_pipe.predict_proba(X_test)[:, 1]
auc = roc_auc_score(y_test, y_pred_proba)

RocCurveDisplay.from_estimator(log_pipe, X_test, y_test)

plt.text(0.8, 0.2, f'AUC = {auc:.4f}', fontsize=12)
plt.show()
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
from sklearn.metrics import precision_recall_curve

# Calculate the Precision-Recall curve using the 'log_pipe' classification model
precision, recall, thresholds = precision_recall_curve(y_test, log_pipe.predict_proba(X_test)[:, 1])

print("Precision:", precision)
print("Recall:", recall)
print("Thresholds:", thresholds)
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
from sklearn.metrics import PrecisionRecallDisplay
PrecisionRecallDisplay.from_estimator(log_pipe, X_test, y_test)

# In balanced datasets, both the AUC score and the Average Precision score may yield similar values. However,
# let's make it a habit to use AUC for balanced datasets and Average Precision scores for unbalanced datasets.

# To plot the Precision-Recall curve, use the following command:
# plot_precision_recall_curve(log_pipe, X_test, y_test);
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
y_pred_proba = log_pipe.predict_proba(X_test)[:, 1]
average_precision = average_precision_score(y_test, y_pred_proba)

PrecisionRecallDisplay.from_estimator(log_pipe, X_test, y_test)

plt.text(0.8, 0.4, f'AP = {average_precision:.4f}', fontsize=12, va='top', ha='left')
plt.show()
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
PrecisionRecallDisplay.from_estimator(grid_model, X_test, y_test);

# plot_precision_recall_curve(grid_model, X_test, y_test);
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
y_pred_proba = grid_model.predict_proba(X_test)[:, 1]
average_precision = average_precision_score(y_test, y_pred_proba)

PrecisionRecallDisplay.from_estimator(grid_model, X_test, y_test)

plt.text(0.8, 0.4, f'AP = {average_precision:.4f}', fontsize=12, va='top', ha='left')
plt.show()
```
executed in 0ms, finished 15:30:09 2024-08-15

3 Prediction

In [ ]:
```
new_data = pd.read_excel("Raisin_Dataset_new.xlsx")
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
final_scaler = StandardScaler()
final_model = LogisticRegression()
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
final_pipe = Pipeline([("scaler", final_scaler), ("log_model", final_model)])
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
final_pipe.fit(X, y)
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
predictions = final_pipe.predict(new_data)
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
positive_class_proba = final_pipe.predict_proba(new_data)[:, 1]

# calculation of probability
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
results = pd.DataFrame({
    'Prediction': predictions,
    'Positive Class Probability': positive_class_proba
})

results

# show prediction result
```
executed in 0ms, finished 15:30:09 2024-08-15

In [ ]:
```
# final_pipe = Pipeline([("scaler", final_scaler), ("log_model", final_model)])
# final_pipe.fit(X, y)
# predictions = final_pipe.predict(new_data)
# positive_class_proba = final_pipe.predict_proba(new_data)[:, 1]
# results = pd.DataFrame({
#     'Prediction': predictions,
#     'Positive Class Probability': positive_class_proba
# })

# results
```
executed in 0ms, finished 15:30:09 2024-08-15

CONCLUSION

After applying logistic regression with a well-tuned pipeline, the model achieved an accuracy of 0.88 and an ROC-AUC score of 0.93. This indicates that the model is highly effective at distinguishing between the two raisin varieties, with only 21 incorrect predictions out of 900 samples. The model's performance suggests it is a reliable tool for classifying Kecimen and Besni raisins based on their morphological features.