

Gebze Technical University
Department of Computer Engineering

CSE 557 **Data Mining**

Fall 2023

Lecturer



Dr. Burcu Yilmaz

Prepared by



Bülent Karadeniz



Project Presentation

**Comparison of Machine Learning Models such as
Logistic Regression, SVC, DT, RF
in the Classification of Raisin Grains.**




January / 2024

- 1. Introduction and Objectives:**
- 2. Dataset**
- 3. Machine Learning Models**
- 4. Model Performance Evaluations**
- 5. Results & Model Comparison & Conclusion**



1. Introduction and Objectives

Classification of Raisin Grains Using Machine Vision and Artificial Intelligence Methods

Ilkay CINAR^{*,a} , Murat KOKLU^b , Sakir TASDEMİR^c 

^{a,*} Selcuk University, Faculty Of Technology, Department Of Computer Engineering, 42130, KONYA, TÜRKİYE

^b Selcuk University, Faculty Of Technology, Department Of Computer Engineering, 42130, KONYA, TÜRKİYE

^c Selcuk University, Faculty Of Technology, Department Of Computer Engineering, 42130, KONYA, TÜRKİYE

In this study, machine vision system was developed in order to distinguish between two different variety of raisins (Kecimen and Besni) grown in Turkey. Firstly a total of 900 pieces raisin grains were obtained, from an equal number of both varieties. These images were subjected to various preprocessing steps and 7 morphological feature extraction operations were performed using image processing techniques. In addition, minimum, mean, maximum and standard deviation statistical information was calculated for each feature. The distributions of both raisin varieties on the features were examined and these distributions were shown on the graphs. Later, models were created using LR, MLP, and SVM machine learning techniques and performance measurements were performed. The classification achieved 85.22% with LR, 86.33% with MLP and 86.44% with the highest classification accuracy obtained in the study with SVM. Considering the number of data available, it is possible to say that the study was successful.



1. Introduction and Objectives

- There are many applications of traditional methods for assessing and determining the **quality of foods**.
- However, these can be **time-consuming** and **expensive**. In addition, human-made procedures from traditional methods can be **inconsistent** and more **inefficient**, as well as physical conditions such as **fatigue** and even **people's psychological mood** can affect the outcome of the work.
- My aim is to test the machine learning used in the authors' study on the same data set and to observe different model results by trying different additional methods.

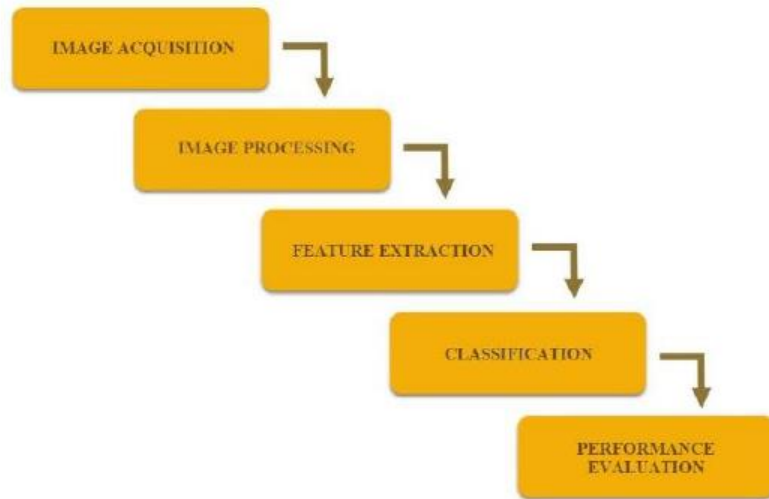


Figure 1. The necessary process steps for classification of raisins (*Kuru üzümün sınıflandırılması için gerekli işlem adımları*)

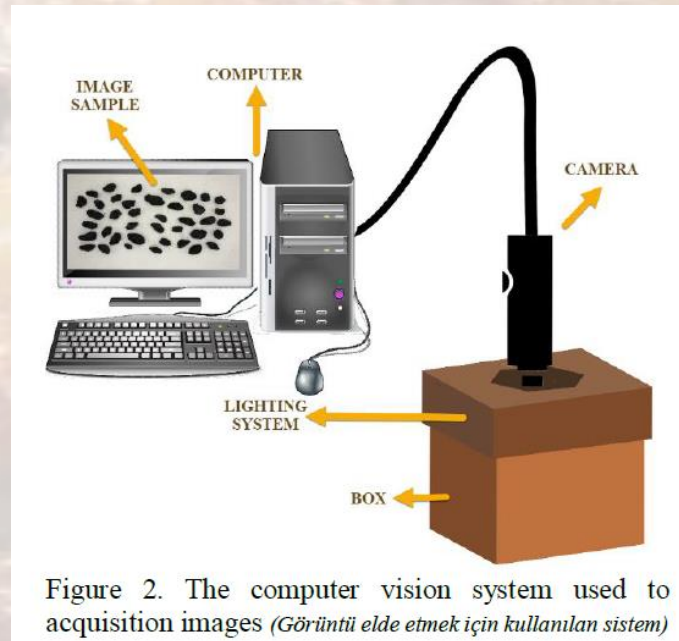


Figure 2. The computer vision system used to acquisition images (*Görüntü elde etmek için kullanılan sistem*)

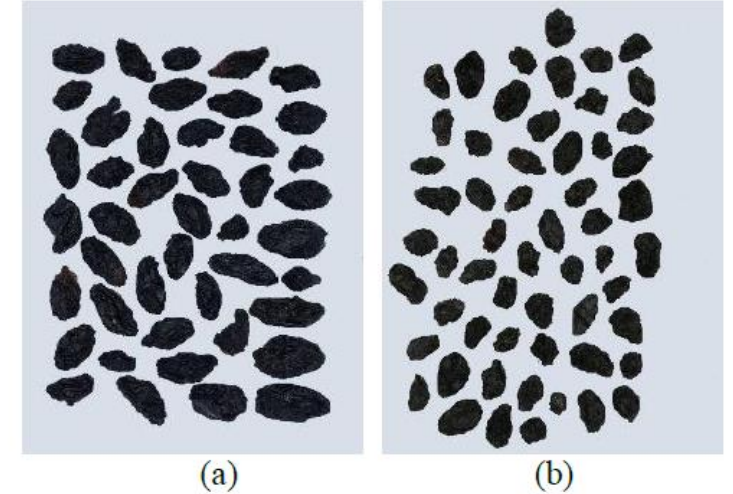


Figure 3. Sample image of raisin varieties used in the study ((a) Besni, (b) Kecimen) (*Çalışmada kullanılan kuru üzüm çeşitlerine ait örnek görüntü ((a) Besni, (b) Keçimen)*)

1. Introduction and Objectives

Table 1. Confusion matrix used in raisin classification
(Kuru üzüm sınıflandırmasında kullanılan karmaşıklık matrisi)

		Predicted	
		Kecimen	Besni
Actual	Kecimen	tp	fp
	Besni	fn	tn

Table 5. The confusion matrix data belonging to the algorithms
(Algoritmalarla ait karmaşıklık matrisi verileri)

Algorithms LR, MLP, SVM		Predicted	
		Kecimen	Besni
Actual	Kecimen	391	59
		400	50
		404	46
	Besni	74	376
		73	377
		76	374

Table 2. Calculation formulas for performance success criteria (Performans başarı ölçütleri için hesaplanma formülleri)

No	Performance Measure	Formula
1	Accuracy	$\frac{tp + tn}{tp + fp + tn + fn} \times 100$
2	Sensitivity	$\frac{tp}{tp + fn} \times 100$
3	Specificity	$\frac{tn}{tn + fp} \times 100$
4	Precision	$\frac{tp}{tp + fp} \times 100$
5	F1-Score	$\frac{2tp}{2tp + fp + fn} \times 100$
6	Negative Predictive Value	$\frac{tn}{tn + fn} \times 100$
7	False Positive Rate	$\frac{fp}{tn + fp} \times 100$
8	False Discovery Rate	$\frac{fp}{tp + fp} \times 100$
9	False Negative Rate	$\frac{fn}{tp + fn} \times 100$

Table 6. Classification performance measurement results (Sınıflandırma performans ölçüm sonuçları)

Performance Measure	LR	MLP	SVM
Accuracy	85.22	86.33	86.44
Sensitivity	84.09	84.57	84.17
Specificity	86.44	88.29	89.05
Precision	86.89	88.89	89.78
F1-Score	85.46	86.67	86.88
Negative Predictive Value	83.56	83.78	83.11
False Positive Rate	13.56	11.71	10.95
False Discovery Rate	13.11	11.11	10.22
False Negative Rate	15.91	15.43	15.83

Table 4. Statistical data on the properties obtained (Elde edilen özelliklere ait istatistik verileri)

No	Feature	Minimum	Mean	Maximum	Standard Deviation
1	Area	25387	87804.128	235047	39002.111
2	Perimeter	619.074	1165.907	2697.753	273.764
3	MajorAxisLength	225.63	430.93	997.292	116.035
4	MinorAxisLength	143.711	254.488	492.275	49.989
5	Eccentricity	0.349	0.782	0.962	0.09
6	ConvexArea	26139	91186.09	278217	40769.29
7	Extent	0.38	0.7	0.835	0.053

2. Dataset

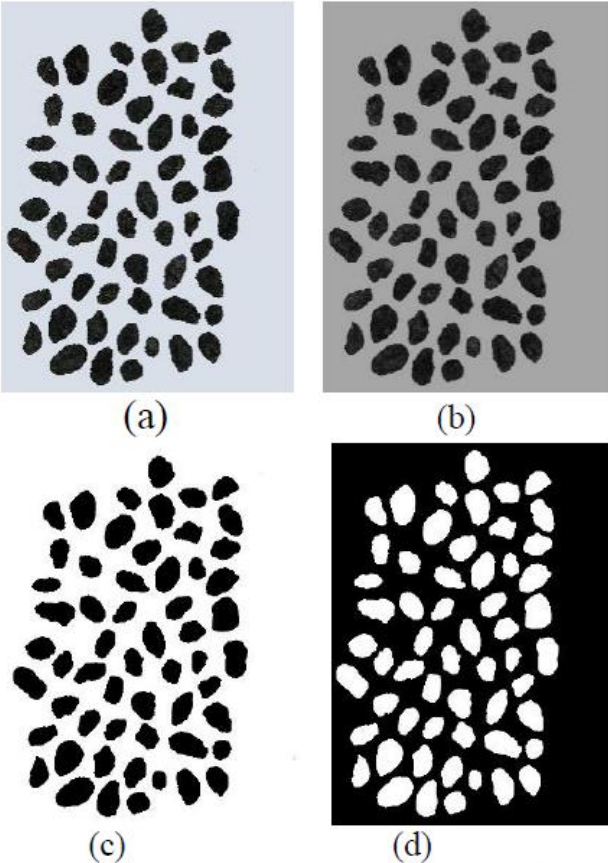


Figure 4. The preprocessing steps performed on images ((a) Real image, (b) Grayscale image, (c) Binary image, (d) Incomplement image) (Görüntüler üzerinde gerçekleştirilen ön işlem aşamaları ((a) Gerçek görüntü, (b) Gri tonlamalı görüntü, (c) İkili görüntü, (d) Terslenmiş görüntü))

A total of **900 images** of raisins were obtained from **both types of raisins** used in the study, including **450 pieces**.

Features

1. **Area**: Gives the number of pixels within the boundaries of the raisin grain.
2. **Perimeter**: It measures the environment by calculating the distance between the boundaries of the raisin grain and the pixels around it.
3. **MajorAxisLength**: Gives the length of the main axis, which is the longest line that can be drawn on the raisin grain.
4. **MinorAxisLength**: Gives the length of the small axis, which is the shortest line that can be drawn on the raisin grain.
5. **Eccentricity**: It gives a measure of the eccentricity of the ellipse, which has the same moments as raisins.
6. **ConvexArea**: Gives the number of pixels of the smallest convex shell of the region formed by the raisin grain.
7. **Extent**: Gives the ratio of the region formed by the raisin grain to the total pixels in the bounding box.

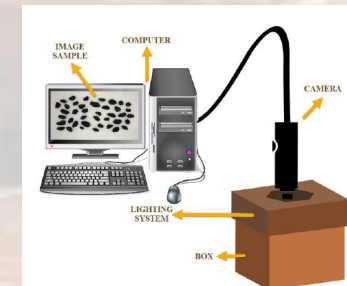
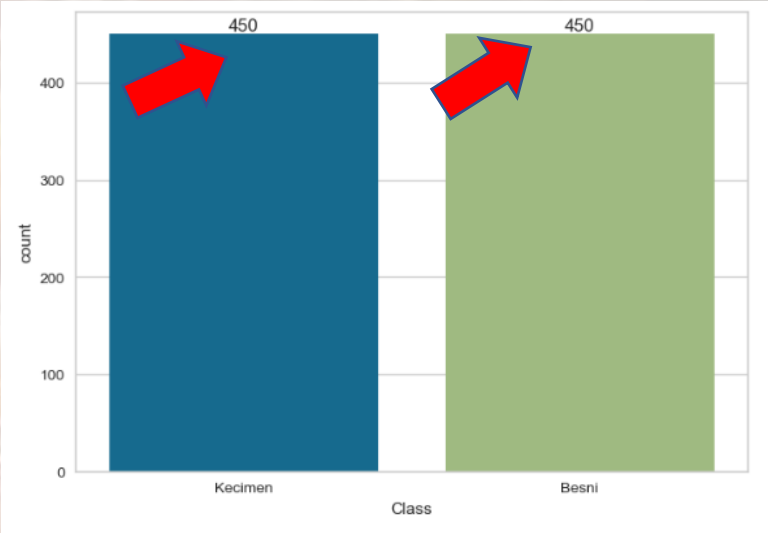


Figure 2. The computer vision system used to

2. Dataset

	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



#	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)

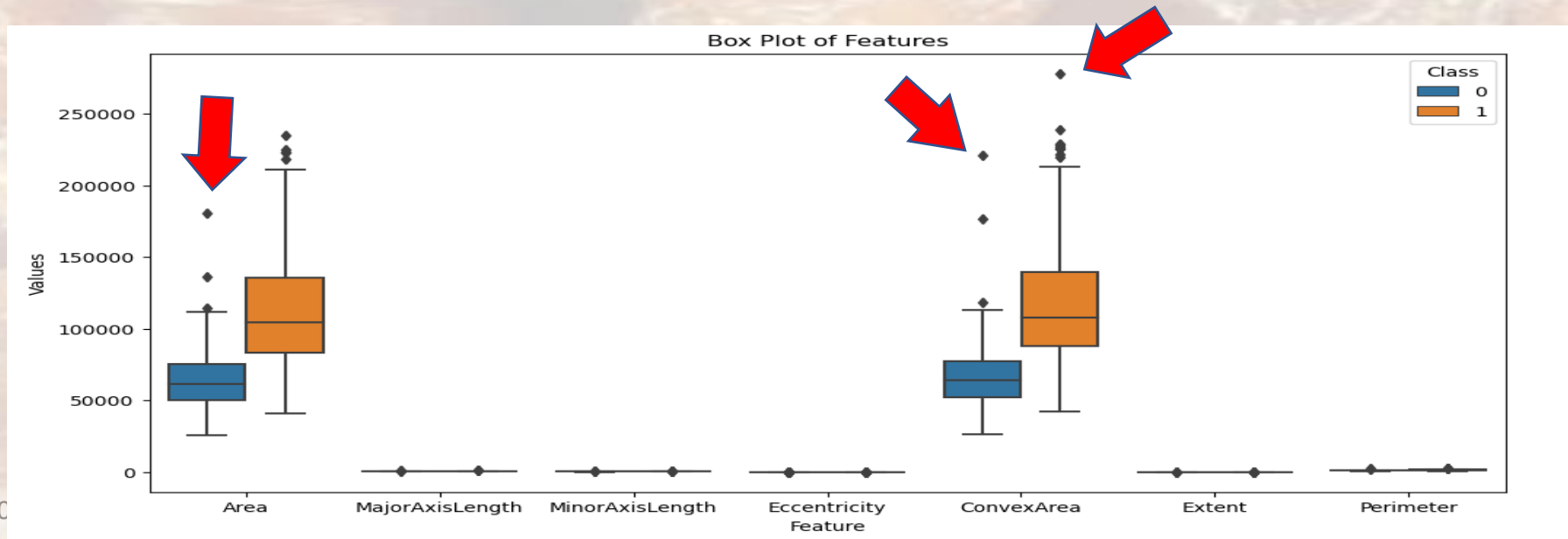
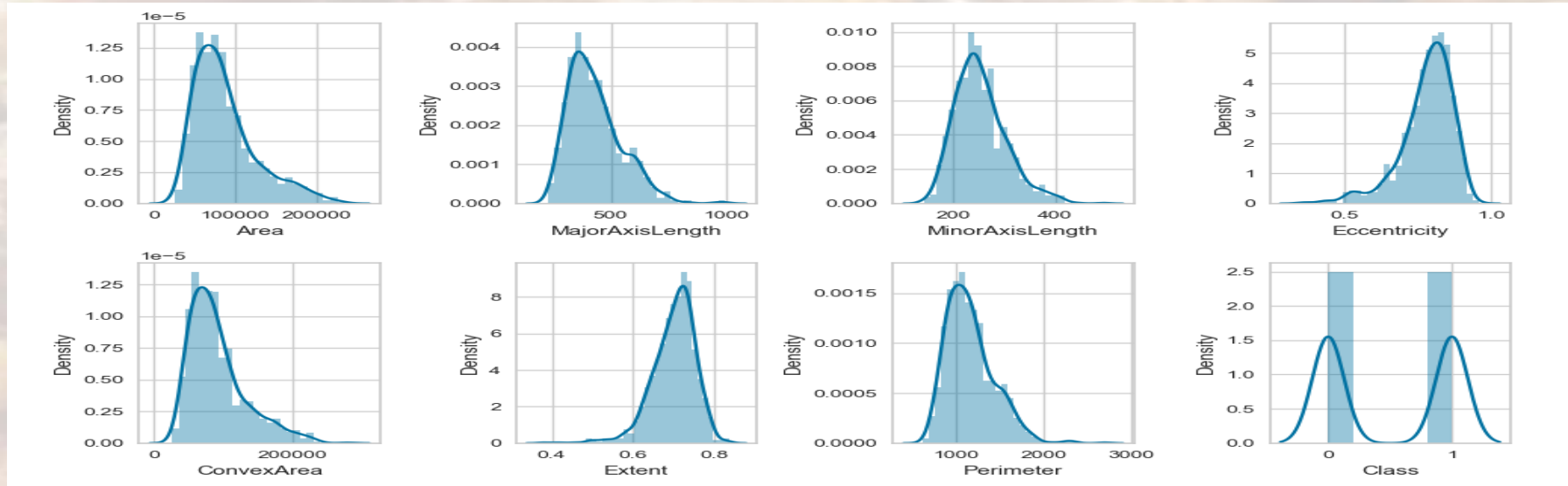
```
# Target değişkenine nümeric dönüşüm gerçekleştirelim
df["Class"] = df["Class"].map({"Kecimen": 0, "Besni": 1})
```

```
df.sample(5)
```

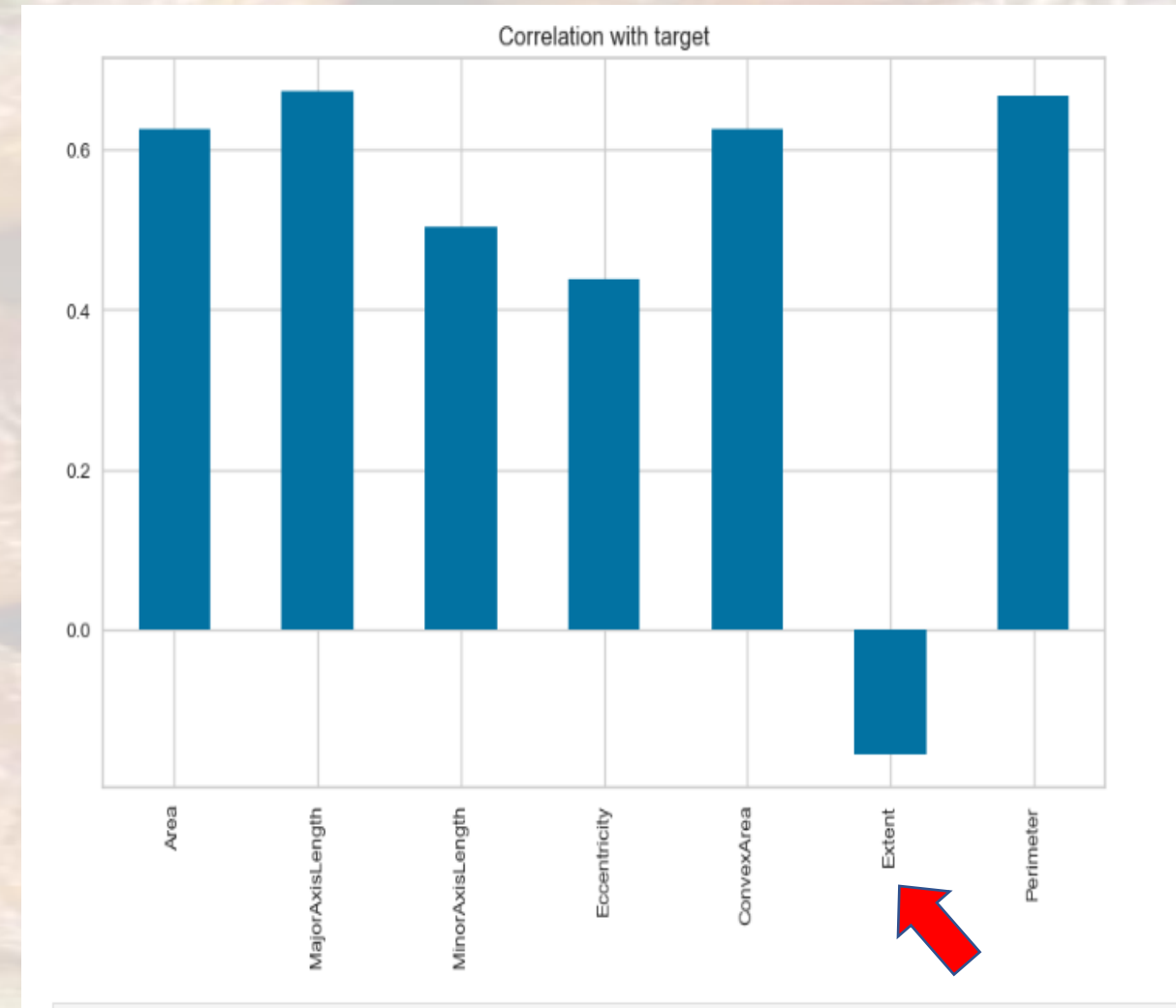
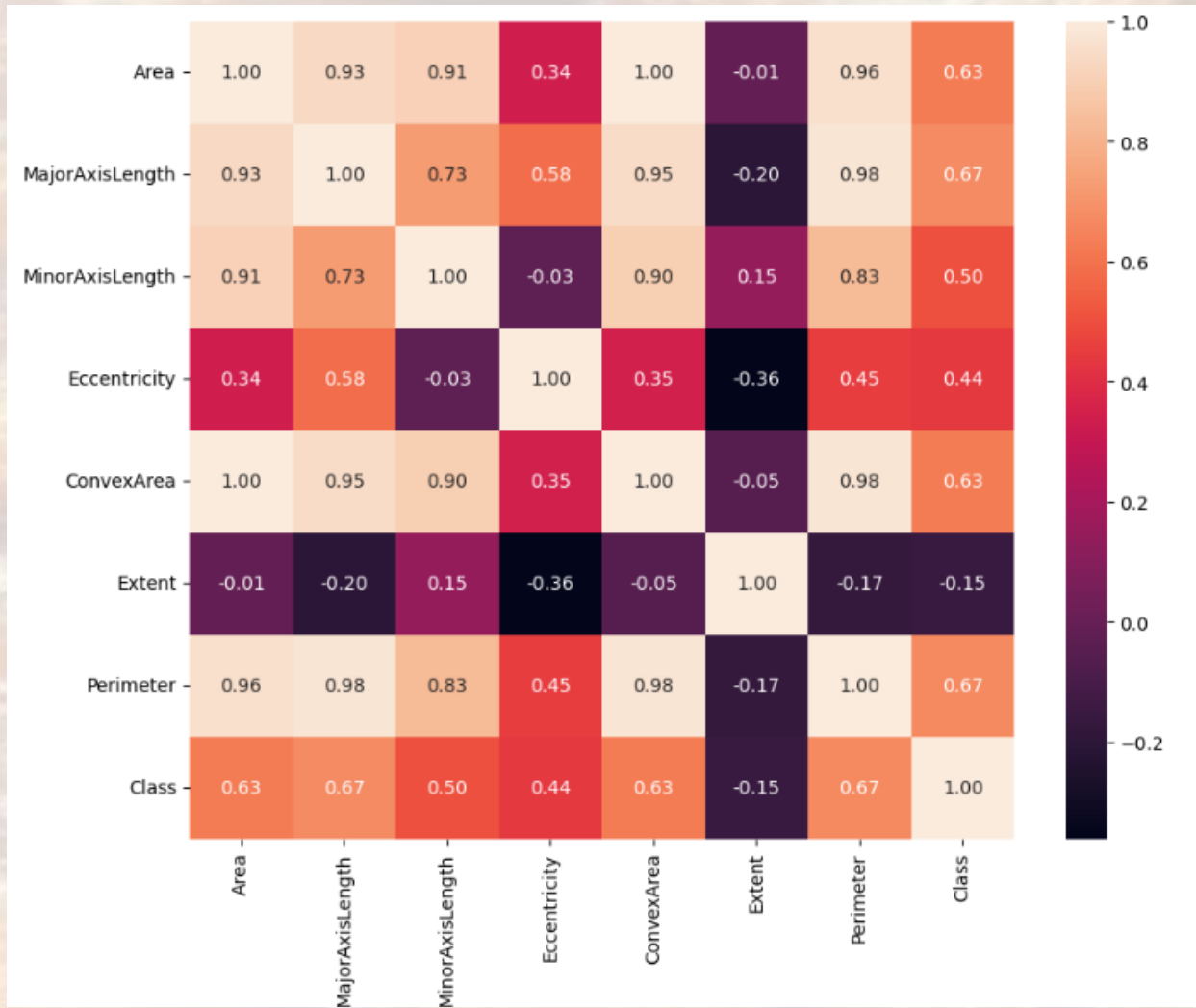
	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class
751	182122	620.487220	376.488606	0.794883	187560	0.721830	1695.230	1
103	55306	353.154609	204.126836	0.816030	57396	0.743990	973.259	0
245	56096	313.386310	232.282440	0.671282	58701	0.711770	929.115	0
276	77043	388.692727	257.546290	0.748977	80094	0.726930	1112.212	0
132	48894	318.098911	197.495503	0.783920	50468	0.663816	875.174	0

	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

2. Dataset

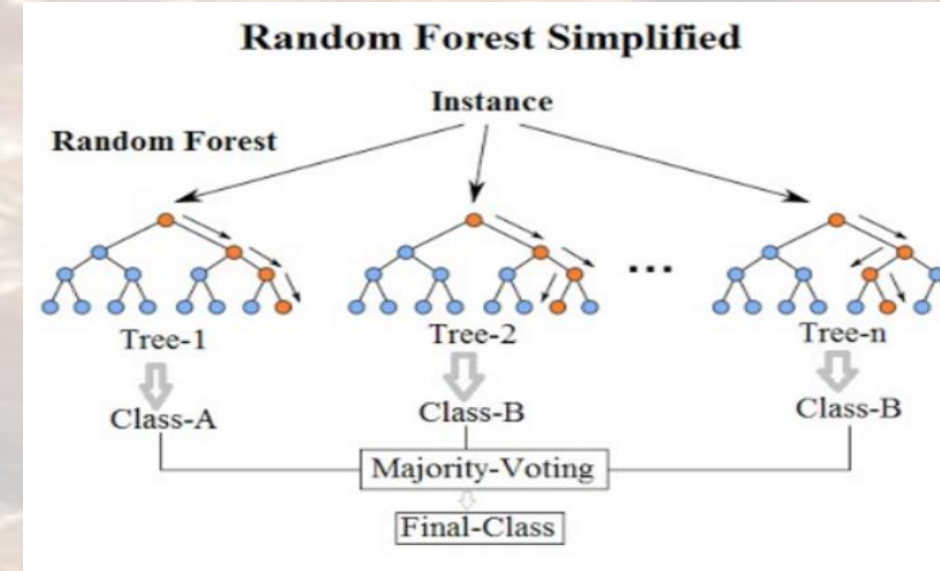
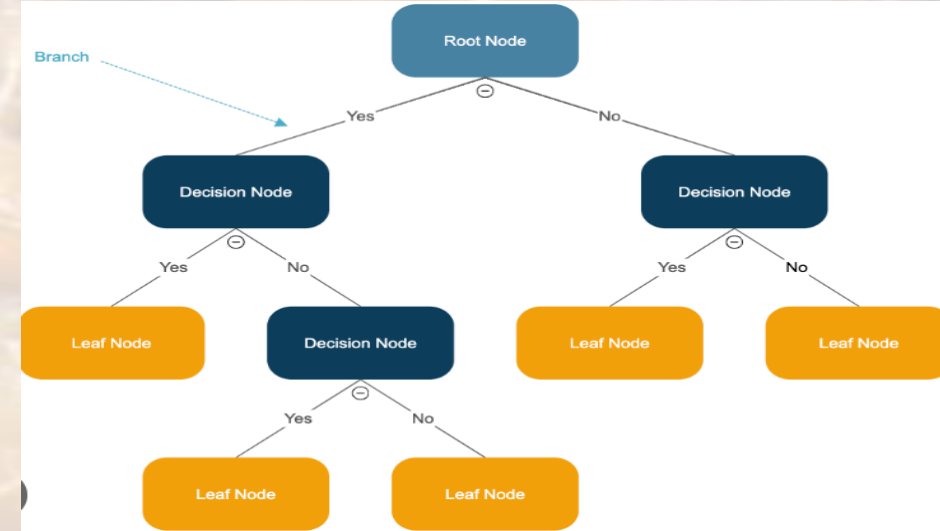
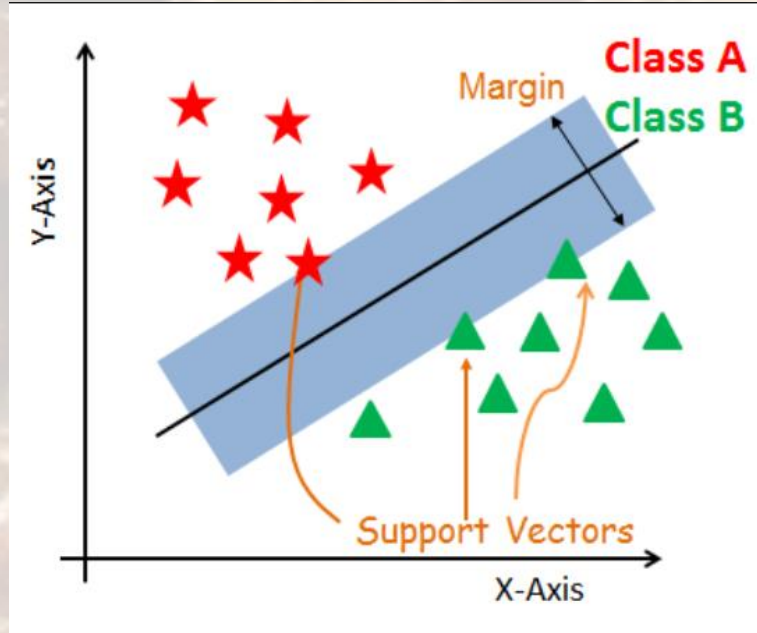
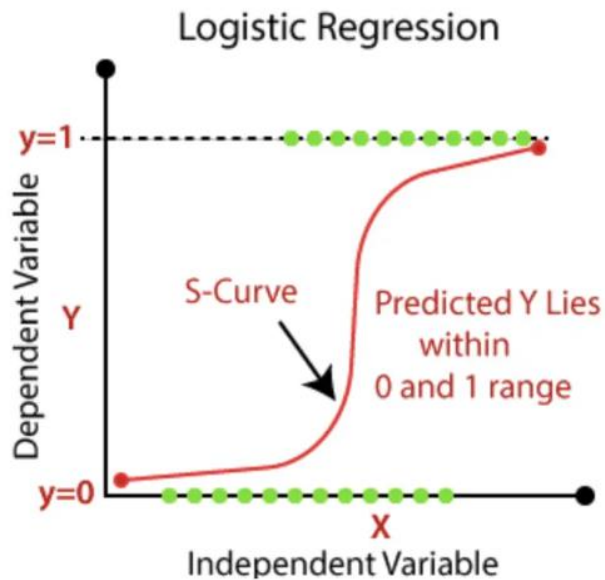


2. Dataset



3. Machine Learning Models

- ✓ Logistic regression
- ✓ Support Vector Machine
- ✓ Decision Tree
- ✓ Random Forest



4. Model & Performance Evaluations

Logistic Regression

```
] log_model = LogisticRegression()  
log_model.fit(X_train_scaled, y_train)
```

```
eval_metric(log_model, X_train_scaled, y_train, X_test_scaled, y_test)
```

```
Test_Set  
[[79 15]  
 [11 75]]  
  
      precision    recall  f1-score   support  
  
 0       0.88      0.84      0.86         94  
 1       0.83      0.87      0.85         86  
  
 accuracy          0.86         180  
 macro avg       0.86      0.86      0.86         180  
weighted avg       0.86      0.86      0.86         180
```

```
Train_Set  
[[317 39]  
 [ 56 308]]  
  
      precision    recall  f1-score   support  
  
 0       0.85      0.89      0.87        356  
 1       0.89      0.85      0.87        364  
  
 accuracy          0.87        720  
 macro avg       0.87      0.87      0.87        720  
weighted avg       0.87      0.87      0.87        720
```

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class	pred_proba	pred
70	95347	451.526154	280.226153	0.784111	99256	0.674956	1255.24	0	0.777842	1
827	61861	345.943650	235.430468	0.732706	67390	0.702280	1063.621	1	0.225457	0
231	52693	283.504239	242.113954	0.520265	54860	0.737749	895.74	0	0.020714	0
588	112808	542.504780	267.201878	0.870293	116961	0.743155	1390.400	1	0.959072	1
39	49882	287.264327	222.185873	0.633852	50880	0.766378	843.764	0	0.022046	0

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

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

```
test_accuracy    0.868056  
test_precision    0.889884  
test_recall      0.846396  
test_f1          0.866050  
dtype: float64
```

4. Model & Performance Evaluations

Logistic Regression

```
] log_model = LogisticRegression()  
  
log_model.fit(X_train_scaled, y_train)
```

```
model = LogisticRegression()  
  
penalty = ["l1", "l2"]  
C = np.logspace(-1, 5, 20)  
class_weight= ["balanced", None]  
solver = ["liblinear"]  
  
param_grid = {"penalty": penalty,  
              "C": [1],  
              "class_weight": class_weight,  
              "solver": solver}  
  
grid_model = GridSearchCV(estimator = model,  
                           param_grid = param_grid,  
                           cv = 10,  
                           scoring = 'accuracy',  
                           n_jobs = -1)  
  
grid_model.fit(X_train_scaled, y_train)
```

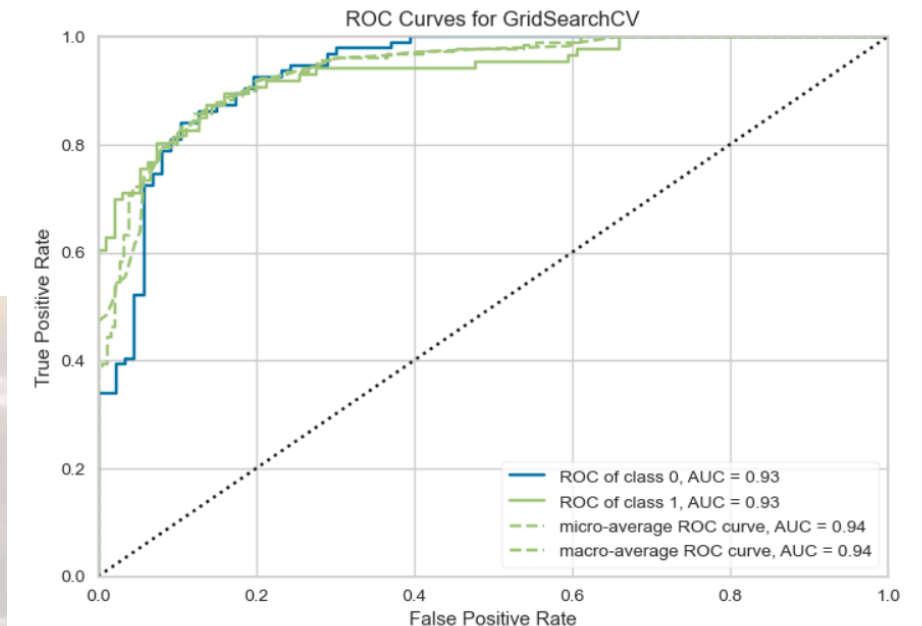
```
eval_metric(grid_model, X_train_scaled, y_train, X_test_scaled, y_test)
```

```
Test_Set  
[[79 15]  
 [11 75]]
```

	precision	recall	f1-score	support
0	0.88	0.84	0.86	94
1	0.83	0.87	0.85	86
accuracy			0.86	180
macro avg	0.86	0.86	0.86	180
weighted avg	0.86	0.86	0.86	180

```
Train_Set  
[[317 39]  
 [ 55 309]]
```

	precision	recall	f1-score	support
0	0.85	0.89	0.87	356
1	0.89	0.85	0.87	364
accuracy			0.87	720
macro avg	0.87	0.87	0.87	720
weighted avg	0.87	0.87	0.87	720



SVM Model ¶

Train | Test Split and Scaling

```
: X = df.drop(["Class"], axis = 1)
  y = df["Class"]

X_train, X_test, y_train_svm, y_test_svm = train_test_split(X, y, test_size = 0.2, random_state = 42)

scaler = StandardScaler()

X_train_scaled_svm = scaler.fit_transform(X_train)

X_test_scaled_svm = scaler.transform(X_test)

# (SVM) modelini oluşturma
SVM_model = SVC(kernel='linear', random_state=42)
SVM_model.fit(X_train_scaled_svm, y_train_svm)
```

4. Model Performance Evaluations

SVM Model

```
eval_metric(SVM_model, X_train_scaled_svm, y_train_svm, X_test_scaled_svm, y_test_svm)
```

Test_Set

[[79 15]

[11 75]]

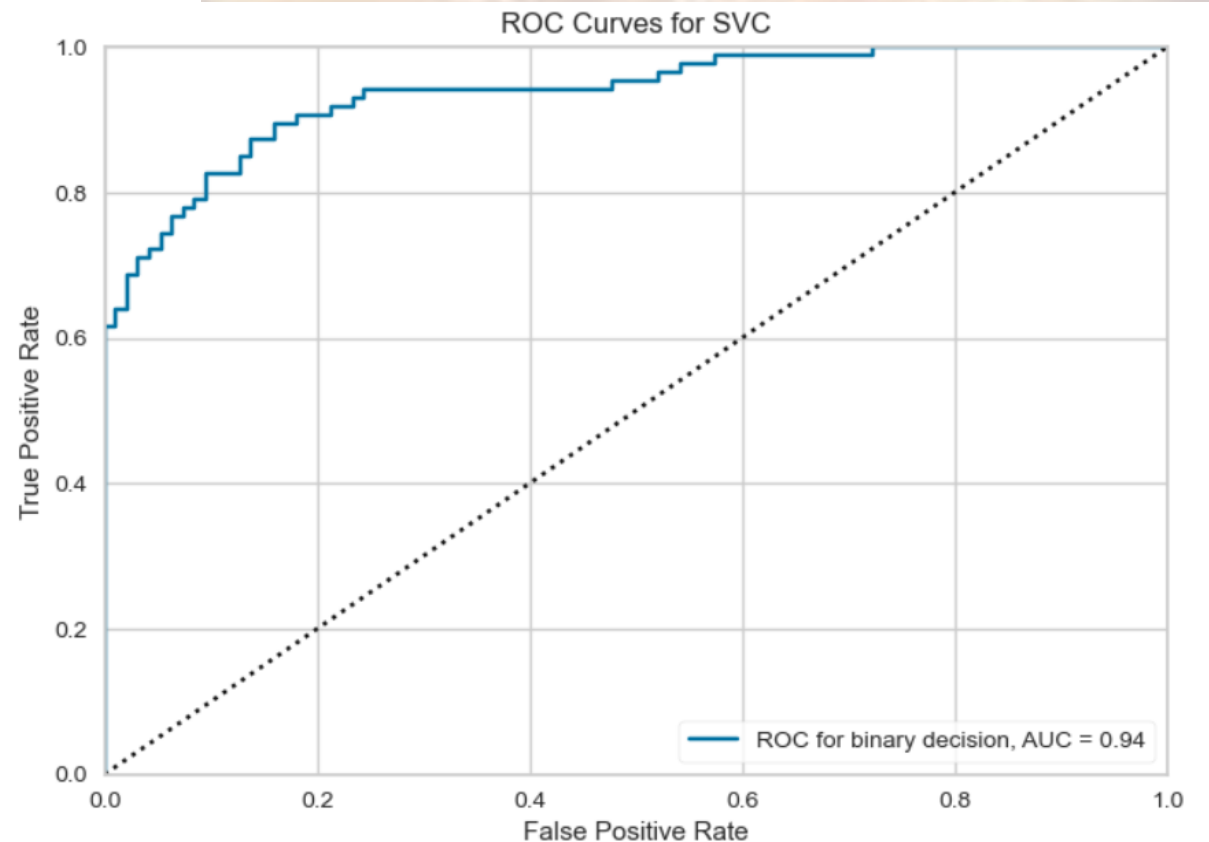
	precision	recall	f1-score	support
0	0.88	0.84	0.86	94
1	0.83	0.87	0.85	86
accuracy			0.86	180
macro avg	0.86	0.86	0.86	180
weighted avg	0.86	0.86	0.86	180

Train_Set

[[317 39]

[55 309]]

	precision	recall	f1-score	support
0	0.85	0.89	0.87	356
1	0.89	0.85	0.87	364
accuracy			0.87	720
macro avg	0.87	0.87	0.87	720
weighted avg	0.87	0.87	0.87	720



4. Model & Performance Evaluations

DecisionTreeClassifier Model

Train | Test Split and Scaling

```
0]: X = df.drop(["Class"], axis = 1)
    y = df["Class"]

X_train, X_test, y_train_dt, y_test_dt = train_test_split(X, y, test_size = 0.2, random_state = 42)

scaler = StandardScaler()

X_train_scaled_dt = scaler.fit_transform(X_train)

X_test_scaled_dt = scaler.transform(X_test)

# modelini oluşturma
DT_model = DecisionTreeClassifier(random_state=42)
DT_model.fit(X_train_scaled_dt, y_train_dt)
```

4. Model & Performance Evaluations

DecisionTreeClassifier

```
eval_metric(DT_model, X_train_scaled_dt, y_train_dt, X_test_scaled_dt, y_test_dt)
```

Test_Set

[[80 14]

[11 75]]

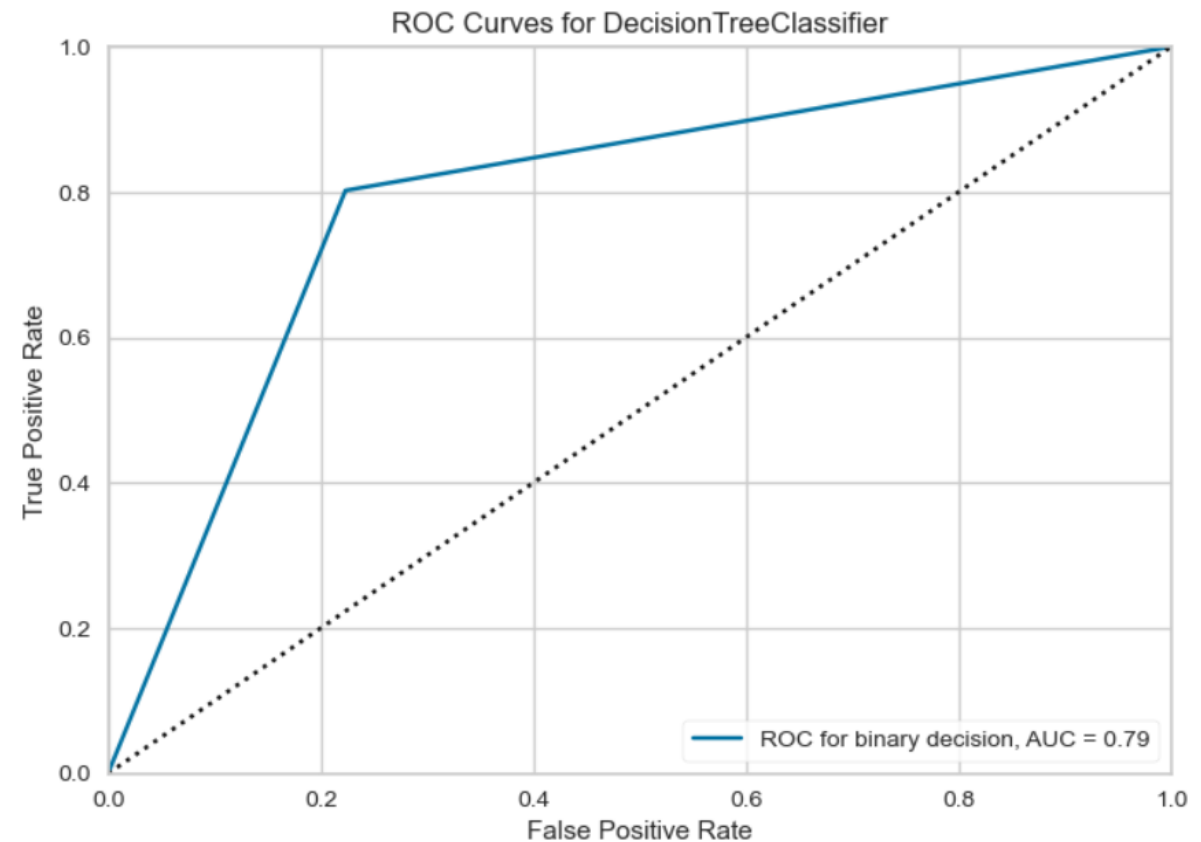
	precision	recall	f1-score	support
0	0.88	0.85	0.86	94
1	0.84	0.87	0.86	86
accuracy			0.86	180
macro avg	0.86	0.86	0.86	180
weighted avg	0.86	0.86	0.86	180

Train_Set

[[326 30]

[59 305]]

	precision	recall	f1-score	support
0	0.85	0.92	0.88	356
1	0.91	0.84	0.87	364
accuracy			0.88	720
macro avg	0.88	0.88	0.88	720
weighted avg	0.88	0.88	0.88	720



Random Forest Model

Train | Test Split and Scaling

```
] X = df.drop(["Class"], axis = 1)
y = df["Class"]

X_train, X_test, y_train_rf, y_test_rf = train_test_split(X, y, test_size = 0.2, random_state = 42)

scaler = StandardScaler()

X_train_scaled_rf = scaler.fit_transform(X_train)

X_test_scaled_rf = scaler.transform(X_test)

# Random Forest modelini oluşturma
RF_model = RandomForestClassifier(n_estimators=10, random_state=42)
RF_model.fit(X_train_scaled_rf, y_train_rf)
```

4. Model & Performance Evaluations

Random Forest

```
eval_metric(RF_model, X_train_scaled_rf, y_train_rf, X_test_scaled_rf, y_test_rf)
```

Test_Set

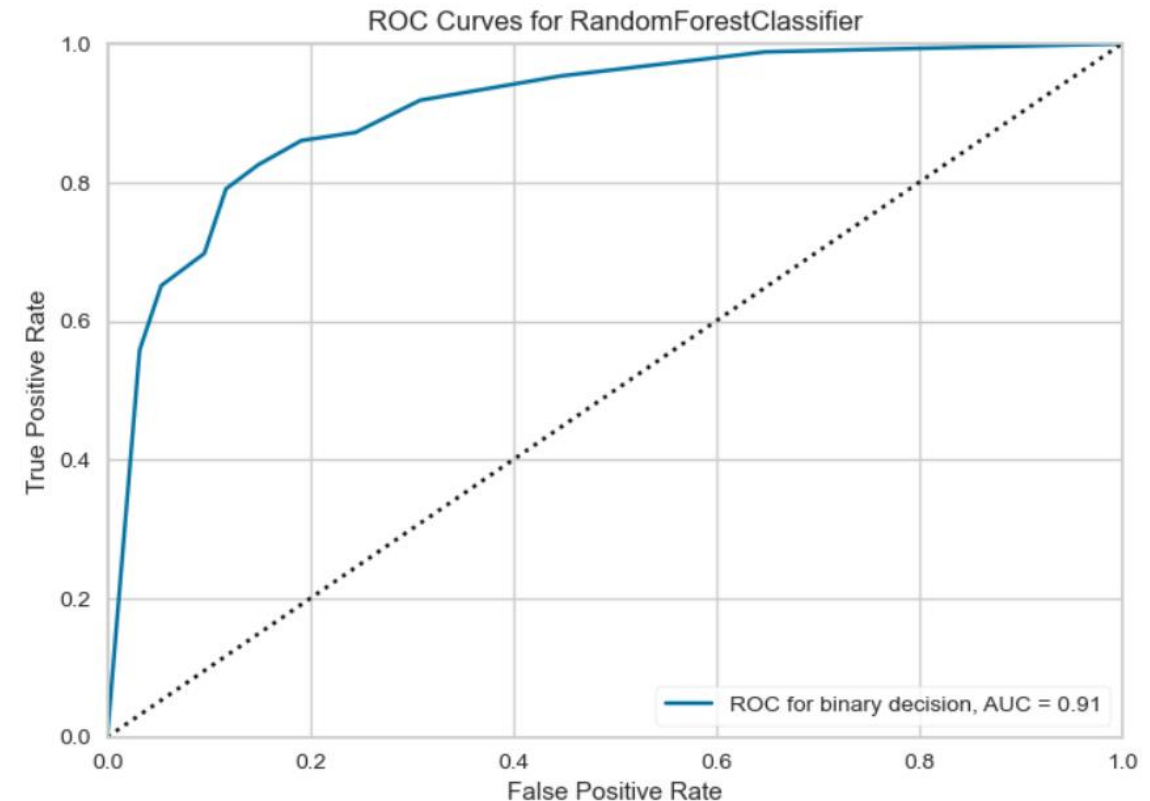
```
[[73 21]
 [17 69]]
```

	precision	recall	f1-score	support
0	0.81	0.78	0.79	94
1	0.77	0.80	0.78	86
accuracy			0.79	180
macro avg	0.79	0.79	0.79	180
weighted avg	0.79	0.79	0.79	180

Train_Set

```
[[356 0]
 [ 0 364]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	356
1	1.00	1.00	1.00	364
accuracy			1.00	720
macro avg	1.00	1.00	1.00	720
weighted avg	1.00	1.00	1.00	720



5. Results & Model Comparison & Conclusion

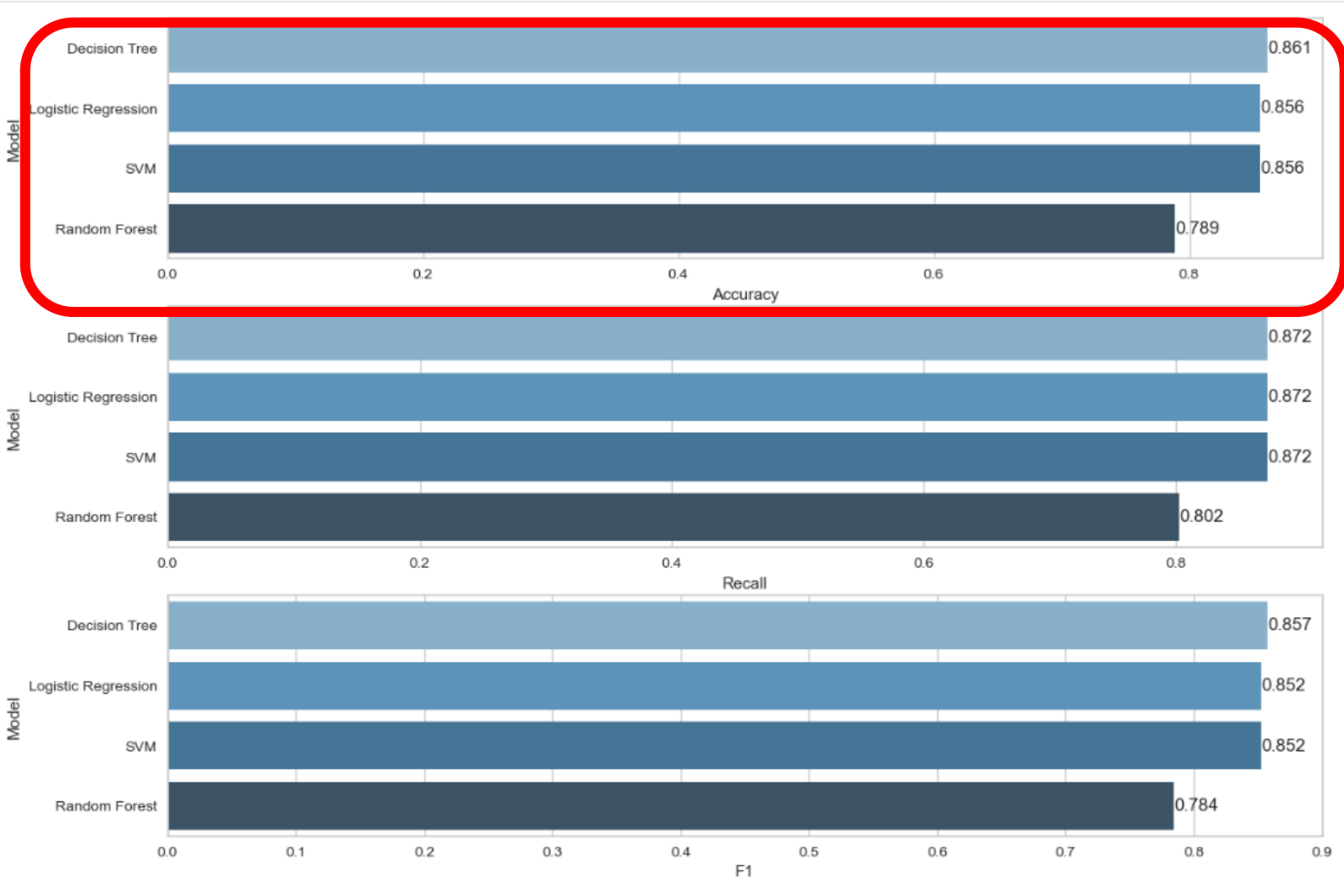


Table 6. Classification performance measurement results (*Sınıflandırma performans ölçüm sonuçları*)

Performance Measure	LR	MLP	SVM
Accuracy	85.22	86.33	86.44
Sensitivity	84.09	84.57	84.17
Specificity	86.44	88.29	89.05
Precision	86.89	88.89	89.78
F1-Score	85.46	86.67	86.88
Negative Predictive Value	83.56	83.78	83.11
False Positive Rate	13.56	11.71	10.95
False Discovery Rate	13.11	11.11	10.22
False Negative Rate	15.91	15.43	15.83



Thanks