MEX #1 - Geyzson Kristoffer

SN:2023-21036

https://uvle.upd.edu.ph/mod/assign/view.php?id=531880

Problem #1 - Linear Regression

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge, LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split

# reading the data
df = pd.read_excel('ENB2012_data.xlsx')

new_column_names = {
    'X1': 'Relative Compactness',
    'X2': 'Surface Area',
    'X3': 'Wall Area',
    'X4': 'Roof Area',
    'X5': 'Overall Height',
    'X6': 'Orientation',
    'X7': 'Glazing Area',
    'X8': 'Glazing Area Distribution',
    'Y1': 'Heating Load',
    'Y2': 'Cooling Load'
}

# Rename the DataFrame columns
df.rename(columns=new_column_names, inplace=True)
print(df.head())

# print nan values
print(f'\nTotal NaN values: {df.isna().sum().sum()}')

# print null values
print(f'\nTotal null values: {df.isnull().sum().sum()}')

# print data types
print(f'\nData types:\n{df.dtypes}')
```

```
   Relative Compactness  Surface Area  Wall Area  Roof Area  Overall Height  \
0                  0.98         514.5      294.0     110.25             7.0
1                  0.98         514.5      294.0     110.25             7.0
2                  0.98         514.5      294.0     110.25             7.0
3                  0.98         514.5      294.0     110.25             7.0
4                  0.90         563.5      318.5     122.50             7.0

   Orientation  Glazing Area  Glazing Area Distribution  Heating Load  \
0            2           0.0                          0         15.55
1            3           0.0                          0         15.55
2            4           0.0                          0         15.55
3            5           0.0                          0         15.55
4            2           0.0                          0         20.84

   Cooling Load
0         21.33
1         21.33
2         21.33
3         21.33
4         28.28

Total NaN values: 0

Total null values: 0

Data types:
Relative Compactness         float64
Surface Area                 float64
Wall Area                    float64
Roof Area                    float64
Overall Height               float64
Orientation                    int64
Glazing Area                 float64
Glazing Area Distribution      int64
Heating Load                 float64
Cooling Load                 float64
dtype: object
```

Initial training using all of the features

In [ ]:

```python
# splitting the data into X and y
X = df.iloc[:, :8] # X1 to X8
y = df.iloc[:, 8] # heating load only

# splitting the data into train, validation, and test sets
X_train, X2, y_train, y2 = train_test_split(X, y,
                                            test_size=0.4,
                                            random_state=69) # 60/40 split
 # splitting further to get the 60/20/20 split
X_val, X_test, y_val, y_test = train_test_split(X2, y2,
                                                test_size=0.5,
                                                random_state=69)

alphas = np.logspace(-1, 1, 10) # from 10**-1 to 10**1, 10 values
models = [] # list of models
```

```python
# creating models with different alphas and
# making pipelines with standard scaler and ridge regression
for i, alpha in enumerate(alphas):
    models.append(make_pipeline(StandardScaler(),
                                Ridge(alpha=alpha)).fit(X_train, y_train))

# printing the accuracy and alpha of each model
for model in models:
    alpha = model.get_params()['ridge__alpha']
    accuracy_train = model.score(X_train, y_train)
    accuracy_val = model.score(X_val, y_val)
    print(f'Alpha: {alpha : .6f} | Training Accuracy: {accuracy_train : .6f} | Vali

# get the best model based on validation accuracy
best_model = models[np.argmax([model.score(X_val, y_val) for model in models])]
alpha = best_model.get_params()['ridge__alpha']
coef = best_model.get_params()['ridge'].coef_
intercept = best_model.get_params()['ridge'].intercept_
```

```
Alpha:  0.100000 | Training Accuracy:  0.909115 | Validation Accuracy:  0.922755
Alpha:  0.166810 | Training Accuracy:  0.909110 | Validation Accuracy:  0.922726
Alpha:  0.278256 | Training Accuracy:  0.909097 | Validation Accuracy:  0.922681
Alpha:  0.464159 | Training Accuracy:  0.909067 | Validation Accuracy:  0.922613
Alpha:  0.774264 | Training Accuracy:  0.909000 | Validation Accuracy:  0.922520
Alpha:  1.291550 | Training Accuracy:  0.908865 | Validation Accuracy:  0.922411
Alpha:  2.154435 | Training Accuracy:  0.908609 | Validation Accuracy:  0.922316
Alpha:  3.593814 | Training Accuracy:  0.908128 | Validation Accuracy:  0.922263
Alpha:  5.994843 | Training Accuracy:  0.907209 | Validation Accuracy:  0.922222
Alpha:  10.000000 | Training Accuracy:  0.905468 | Validation Accuracy:  0.922028
```

In [ ]:

```python
# print the model with the highest accuracy among all models
print(f'Best Model:')
print(f'\talpha: {alpha : .6f}')
print(f'\ttraining accuracy: {best_model.score(X_train, y_train) : .5f}')
print(f'\tvalidation accuracy: {best_model.score(X_val, y_val) : .5f}')
print(f'\tcoefficients: {coef}')
print(f'\tintercept: {intercept}')
```

```
Best Model:
        alpha:  0.100000
        training accuracy:  0.90912
        validation accuracy:  0.92275
        coefficients: [-6.05743799 -3.03910856  0.80505193 -3.40185598  7.8326222  -
0.1128605
  2.71987256  0.52375461]
        intercept: 22.539934782608697
```

Final Evaluation in Test Data

In [ ]:

```python
print(f'Best Model:')
print(f'\ttest accuracy: {best_model.score(X_test, y_test) : .5f}')
```

```
Best Model:
        test accuracy:  0.92564
```

Item 1-a

Question: What is the best model's coefficients, intercept, and its training, validation, and test accuracy?

Answer:

Evaluation Metrics & Parameters	All Features - Best Model
Coefficients	[-6.05743799, -3.03910856, 0.80505193, -3.40185598, 7.8326222, -0.1128605, 2.71987256, 0.52375461]
Intercept	22.53993
Alpha	0.100000
Training-Accuracy	0.90912
Validation-Accuracy	0.92275
Testing-Accuracy	0.92564

Performing the same procedure but using the top 5 features

In [ ]:
```
sorted_indices = np.argsort(np.abs(coef))[::-1] # sort in descending order

top5_coef = coef[sorted_indices[:5]] # top 5 coefficients
top5_feature_names = df.columns[:8][sorted_indices[:5]].tolist()

print("Top 5 Features:")
print(" names:", ', '.join(top5_feature_names))
print(" coefficients:", top5_coef)
print(" indices:", sorted_indices[:5])
```

```
Top 5 Features:
 names: Overall Height, Relative Compactness, Roof Area, Surface Area, Glazing Area
 coefficients: [ 7.8326222  -6.05743799 -3.40185598 -3.03910856  2.71987256]
 indices: [4 0 3 1 6]
```

Repeating the same process but with the top 5 features

In [ ]:
```
# X will now be the top 5 features
X = df.iloc[:, [4, 0, 3, 1, 6]]
y = df.iloc[:, 8] # still the heating load

# splitting the data into train, validation, and test sets
X_train, X2, y_train, y2 = train_test_split(X, y,
                                            test_size=0.4,
                                            random_state=69) # 60/40 split
 # splitting further to get the 60/20/20 split
```

```python
X_val, X_test, y_val, y_test = train_test_split(X2, y2,
                                                test_size=0.5,
                                                random_state=69)

alphas = np.logspace(-1, 1, 10) # from 10**-1 to 10**1, 10 values
models = [] # list of models

# creating models with different alphas and
# making pipelines with standard scaler and ridge regression
for i, alpha in enumerate(alphas):
    models.append(make_pipeline(StandardScaler(),
                                Ridge(alpha=alpha)).fit(X_train, y_train))

# printing the accuracy and alpha of each model
for model in models:
    alpha = model.get_params()['ridge__alpha']
    accuracy_train = model.score(X_train, y_train)
    accuracy_val = model.score(X_val, y_val)
    print(f'Alpha: {alpha : .6f} | Training Accuracy: {accuracy_train : .6f} | Vali

# get the best model based on validation accuracy
best_model = models[np.argmax([model.score(X_val, y_val) for model in models])]
alpha = best_model.get_params()['ridge__alpha']
coef = best_model.get_params()['ridge'].coef_
intercept = best_model.get_params()['ridge'].intercept_
```

```
Alpha:  0.100000 | Training Accuracy:  0.906421 | Validation Accuracy:  0.926517
Alpha:  0.166810 | Training Accuracy:  0.906417 | Validation Accuracy:  0.926491
Alpha:  0.278256 | Training Accuracy:  0.906409 | Validation Accuracy:  0.926451
Alpha:  0.464159 | Training Accuracy:  0.906389 | Validation Accuracy:  0.926396
Alpha:  0.774264 | Training Accuracy:  0.906348 | Validation Accuracy:  0.926331
Alpha:  1.291550 | Training Accuracy:  0.906269 | Validation Accuracy:  0.926277
Alpha:  2.154435 | Training Accuracy:  0.906124 | Validation Accuracy:  0.926275
Alpha:  3.593814 | Training Accuracy:  0.905857 | Validation Accuracy:  0.926372
Alpha:  5.994843 | Training Accuracy:  0.905330 | Validation Accuracy:  0.926548
Alpha:  10.000000 | Training Accuracy:  0.904202 | Validation Accuracy:  0.926578
```

In [ ]:

```python
# print the model with the highest accuracy among all models
print(f'Best Model:')
print(f'\talpha: {alpha : .6f}')
print(f'\ttraining accuracy: {best_model.score(X_train, y_train) : .5f}')
print(f'\tvalidation accuracy: {best_model.score(X_val, y_val) : .5f}')
print(f'\ttest accuracy: {best_model.score(X_test, y_test) : .5f}')
print(f'\tcoefficients: {coef}')
print(f'\tintercept: {intercept}')
```

```
Best Model:
	alpha:  10.000000
	training accuracy:  0.90420
	validation accuracy:  0.92658
	test accuracy:  0.92500
	coefficients: [ 7.71068437 -3.28294732 -4.67906309  0.91757006  2.76335503]
	intercept: 22.539934782608697
```

In [ ]:

```python
# for comparison, printing the model with the same alpha but with the top 5 feature
alpha = models[0].get_params()['ridge__alpha']
coef = models[0].get_params()['ridge'].coef_
```

```
intercept = models[0].get_params()['ridge'].intercept_

print(f'Same Alpha:')
print(f'\talpha: {alpha : .6f}')
print(f'\ttraining accuracy: {models[0].score(X_train, y_train) : .5f}')
print(f'\tvalidation accuracy: {models[0].score(X_val, y_val) : .5f}')
print(f'\tcoefficients: {coef}')
print(f'\tintercept: {intercept}')
```

```
Same Alpha:
	alpha:  0.100000
	training accuracy:  0.90642
	validation accuracy:  0.92652
	coefficients: [ 7.89264074 -5.97144026 -5.02829022 -1.25881311  2.83435557]
	intercept: 22.539934782608697
```

In [ ]:
```
print(f'Same Alpha:')
print(f'\ttest accuracy: {models[0].score(X_test, y_test) : .5f}')
```

```
Same Alpha:
	test accuracy:  0.92760
```

Item 1-b

Q: If you repeat the procedure above using only the 5 top features, what are the results?

A: Using the same method with only the top 5 features yields the following comparison:

Evaluation Metrics & Parameters	All Features - Best Model	Top 5 Features - Same Alpha	Top 5 Features - Best Model
Coefficients	[-6.05743799, -3.03910856, 0.80505193, -3.40185598, 7.8326222, -0.1128605, 2.71987256, 0.52375461]	[7.89264074 -5.97144026 -5.02829022 -1.25881311 2.83435557]	[7.71068437 -3.28294732 -4.67906309 0.91757006 2.76335503]
Intercept	22.53993	22.53993	22.53993
Alpha	0.100000	0.100000	10.000000
Training Accuracy	0.90912	0.90642	0.90420
Validation Accuracy	0.92275	0.92652	0.92658
Testing Accuracy	0.92564	0.92760	0.92500

Problem #1 - Insights

Q: Based on your results for this Problem, what insights did you gain?

A: Here are some insights

- In the preliminary analysis, the five most significant features, ranked by magnitude in descending order, are *Overall Height, Relative Compactness, Roof Area, Surface Area, and Glazing Area*.
- As anticipated, the coefficients associated with these top 5 features were different.
- No variance was observed in the intercept values.
- Notably, there was a decline in Training accuracy across both the "Top 5 with same alpha" and "Top 5 best model" scenarios.
- Conversely, Validation accuracy exhibited an uptick in both aforementioned scenarios.
- Testing accuracy showed a mixed results: it improved under the "Top 5 with same alpha" but declined in the "Top 5 best model" setup.
- If computational resources permit, it would be advantageous to incorporate all available features.
- Opting for just the top-ranked features doesn't guarantee an enhancement in testing accuracy.
- The regularization parameter, alpha, barely made an impact since the range is from 10**-1 to 10**1. But when I tested it on a wider range, the results were dramatically different.
- The accuracy may increase if we add Polynomial Features of a certain degree to the pipeline. For instance, a separate analysis I made with PolynomialFeatures(degree=3) yield higher accuracy in its best model compared to the models above.
- For cold countries aiming to optimize heating load, focus on compact, lower-height designs with increased roof areas and limited glazed surfaces for enhanced energy efficiency.

Problem #2 - Logistic Regression

In [ ]:
```
# column names taken from the website
column_names = ['ID', 'Diagnosis',
                'radius1', 'texture1', 'perimeter1', 'area1',
                'smoothness1', 'compactness1', 'concavity1',
                'concave_points1', 'symmetry1', 'fractal_dimension1',
                'radius2', 'texture2', 'perimeter2', 'area2',
                'smoothness2', 'compactness2', 'concavity2',
                'concave_points2', 'symmetry2', 'fractal_dimension2',
```

```
                'radius3', 'texture3', 'perimeter3', 'area3',
                'smoothness3', 'compactness3', 'concavity3',
                'concave_points3', 'symmetry3', 'fractal_dimension3']

df = pd.read_csv('wdbc.data', header=None, names=column_names)
df
```

Out[ ]:

	ID	Diagnosis	radius1	texture1	perimeter1	area1	smoothness1	compactne
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13
...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04

569 rows × 32 columns

In [ ]:

```
# print nan values
print(f'\nTotal NaN values: {df.isna().sum().sum()}')

# print null values
print(f'\nTotal null values: {df.isnull().sum().sum()}')

# print data types
print(f'\nData types:\n{df.dtypes}')
```

```
Total NaN values: 0

Total null values: 0

Data types:
ID                          int64
Diagnosis                  object
radius1                   float64
texture1                  float64
perimeter1                float64
area1                     float64
smoothness1               float64
compactness1              float64
concavity1                float64
concave_points1           float64
symmetry1                 float64
fractal_dimension1        float64
radius2                   float64
texture2                  float64
perimeter2                float64
area2                     float64
smoothness2               float64
compactness2              float64
concavity2                float64
concave_points2           float64
symmetry2                 float64
fractal_dimension2        float64
radius3                   float64
texture3                  float64
perimeter3                float64
area3                     float64
smoothness3               float64
compactness3              float64
concavity3                float64
concave_points3           float64
symmetry3                 float64
fractal_dimension3        float64
dtype: object
```

In [ ]:

```
df = df.drop(columns=['ID']) # dropping ID column
df['Diagnosis'] = np.where(df['Diagnosis'] == 'M', 1, 0) # M = 1, B = 0
df
```

Out[ ]:

	Diagnosis	radius1	texture1	perimeter1	area1	smoothness1	compactness1	conca
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1
...	...	...	...	...	...	...	...	
564	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.2
565	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.1
566	1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.0
567	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.3
568	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.0

569 rows × 31 columns

In [ ]:

```
X = df.iloc[:, 1:] # features
y = df.iloc[:, :1].values.ravel() # diagnosis

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=69,
                                                    stratify=y) # 70/30 split

model = make_pipeline(StandardScaler(),
                      LogisticRegression()).fit(X_train, y_train) # model

train_accuracy = model.score(X_train, y_train) # accuracy
test_accuracy = model.score(X_test, y_test) # accuracy

logistic_model = model.named_steps['logisticregression']
coefficients = logistic_model.coef_.flatten()
feature_importance = np.abs(coefficients)
feature_names = X_train.columns
sorted_indices = np.argsort(feature_importance)[::-1]

print(f'Training Accuracy: {train_accuracy : .6f}')
print(f'Testing Accuracy: {test_accuracy : .6f}')

print("Feature Magnitudes in Descending Order:")
for index in sorted_indices:
    print(f"\t{feature_names[index]}: {feature_importance[index] :.5f}")
```

```
Training Accuracy:  0.987437
Testing Accuracy:  0.982456
Feature Magnitudes in Descending Order:
	texture3: 1.22297
	radius2: 1.08966
	concavity3: 1.04051
	concavity1: 1.02640
	area2: 1.00383
	compactness2: 0.99125
	symmetry3: 0.91455
	area3: 0.88047
	radius3: 0.85664
	concave_points1: 0.84292
	concave_points3: 0.80083
	perimeter3: 0.68378
	smoothness3: 0.57744
	fractal_dimension2: 0.52852
	texture1: 0.48488
	compactness1: 0.44272
	perimeter2: 0.41542
	symmetry2: 0.39675
	area1: 0.36899
	concave_points2: 0.35139
	smoothness2: 0.32736
	perimeter1: 0.29978
	radius1: 0.29569
	texture2: 0.23883
	compactness3: 0.19358
	concavity2: 0.18930
	fractal_dimension1: 0.11576
	fractal_dimension3: 0.08977
	smoothness1: 0.02459
	symmetry1: 0.01009
```

Item 2-a

Q: After fitting the data, what is the model's training and testing accuracy? Which features are most important?

A: The model's training and testing accuracy, together with 5 of the most important features:

Evaluation Metric	Model
Training Accuracy	0.987437
Testing Accuracy	0.982456

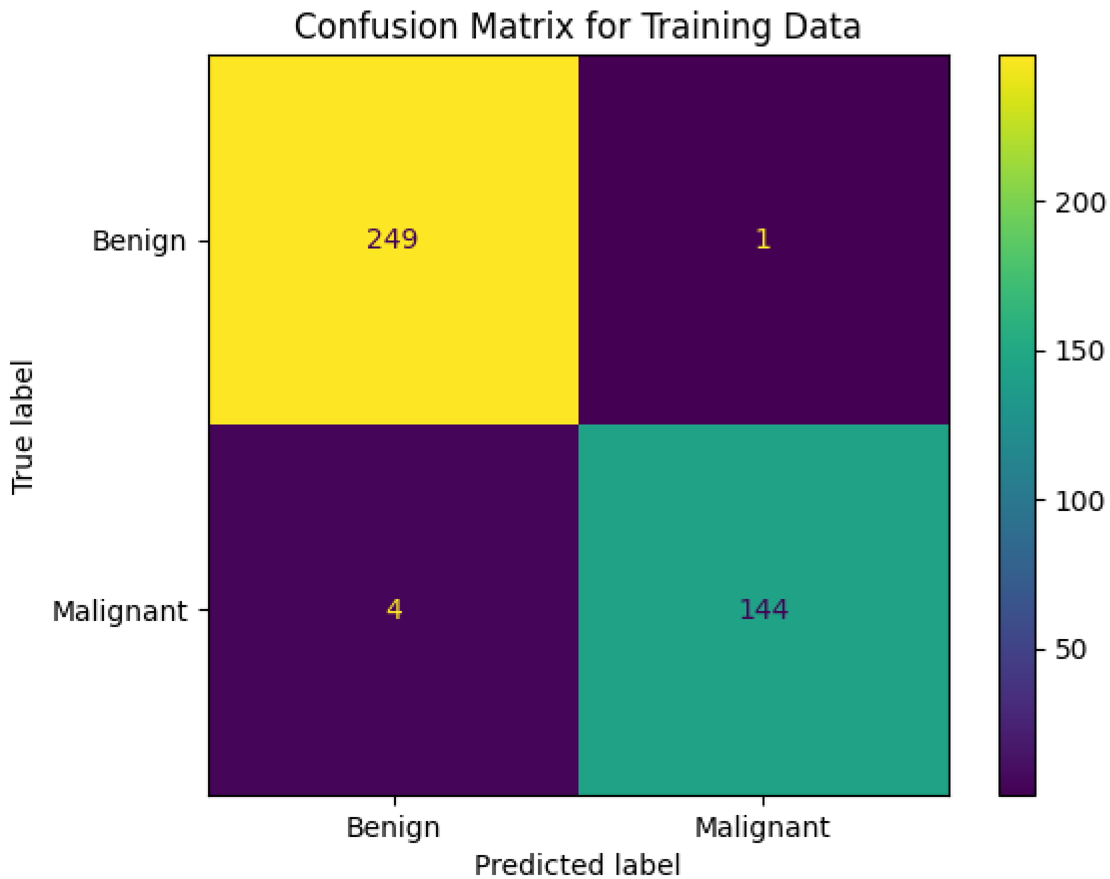
Rank	Feature	Magnitude
#1	texture3	1.22297
#2	radius2	1.08966

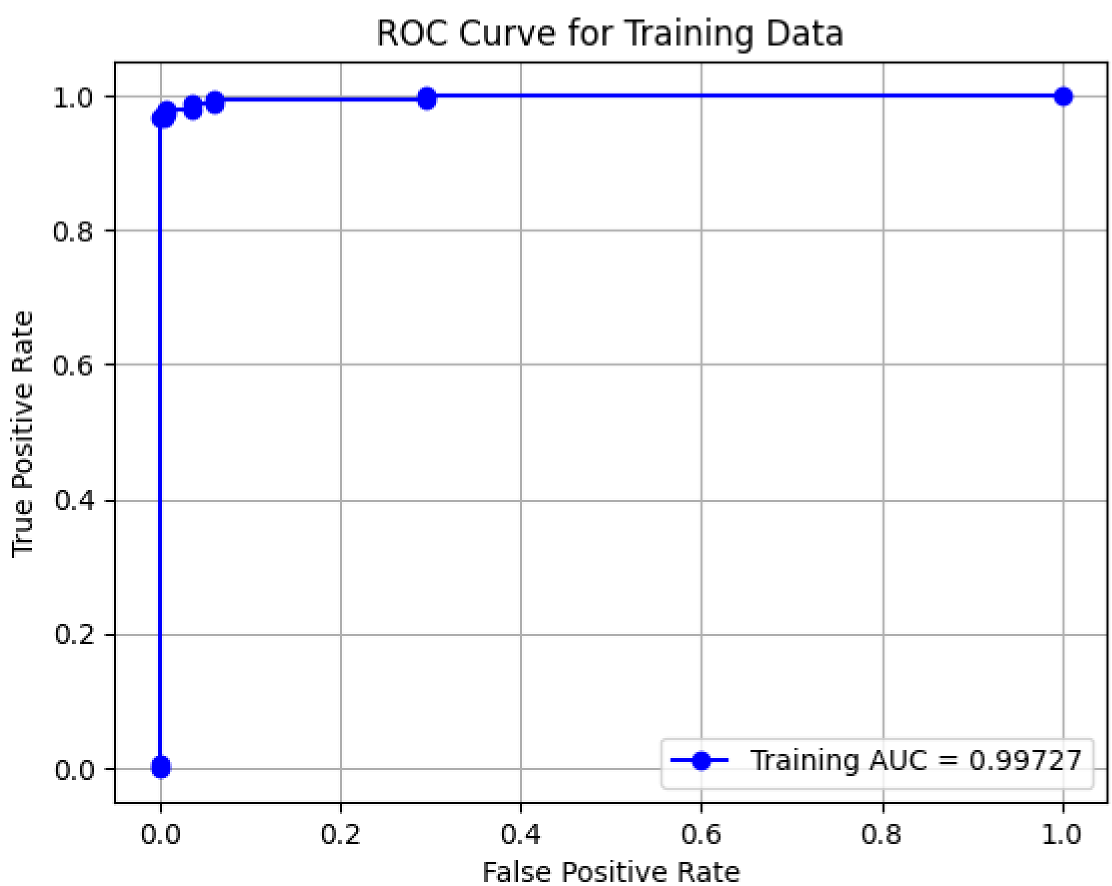
Rank	Feature	Magnitude
#3	concavity3	1.04051
#4	concavity1	1.02640
#5	area2	1.00383

Training Set

In [ ]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn import metrics
import matplotlib.pyplot as plt

ypred_train = model.predict(X_train)
cfm = confusion_matrix(y_train, ypred_train)
cm_display = ConfusionMatrixDisplay(confusion_matrix = cfm,
                                    display_labels = ["Benign", "Malignant"])
cm_display.plot()
plt.title('Confusion Matrix for Training Data')
plt.show()

# For Training Data
y_train_prob = model.predict_proba(X_train)[:, 1]
fpr_train, tpr_train, _ = metrics.roc_curve(y_train, y_train_prob)
auc_train = metrics.roc_auc_score(y_train, y_train_prob)
plt.figure()
plt.plot(fpr_train, tpr_train, 'b-o', label=f'Training AUC = {auc_train:.5f}')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Training Data')
plt.legend()
plt.grid()
plt.show()
print(f"Training AUC: {auc_train:.5f}")

print(f'Train Accuracy: {train_accuracy : .6f}')

# Calculate the False Positive Rate (FPR)
true_negatives, false_positives = cfm[0]
false_negatives, true_positives = cfm[1]

recall = true_positives / (true_positives + false_negatives)
precision = true_positives / (true_positives + false_positives)
f1_score = 2 * (recall * precision) / (recall + precision)
false_alarm_rate = false_positives / (false_positives + true_negatives)

print(f'Recall: {recall : .6f}')
print(f'Precision: {precision : .6f}')
print(f'F1 Score: {f1_score : .6f}')
print(f'False Alarm Rate: {false_alarm_rate : .6f}')
```


Confusion Matrix for Training Data
Benign
Malignant
True label
249
1
4
144
200
150
100
50
Benign
Malignant
Predicted label


ROC Curve for Training Data
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
Training AUC = 0.99727
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

```
Training AUC: 0.99727
Train Accuracy:  0.987437
Recall:  0.972973
Precision:  0.993103
F1 Score:  0.982935
False Alarm Rate:  0.004000
```

Item 2-b-training

Confusion Matrix, ROC Curve, AUC are above

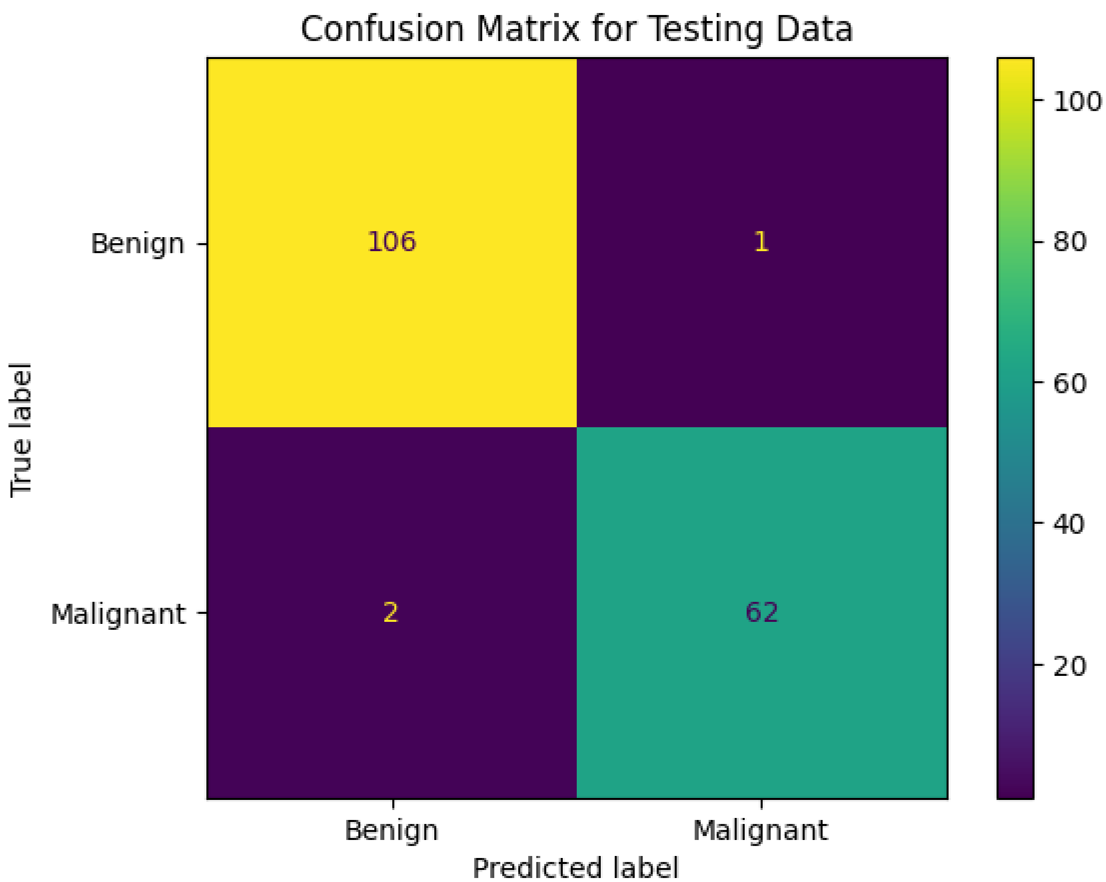
Evaluation Metric	Score
Training AUC	0.99727
Train Accuracy	0.987437
Recall	0.972973
Precision	0.993103
F1 Score	0.982935
False Alarm Rate	0.004000

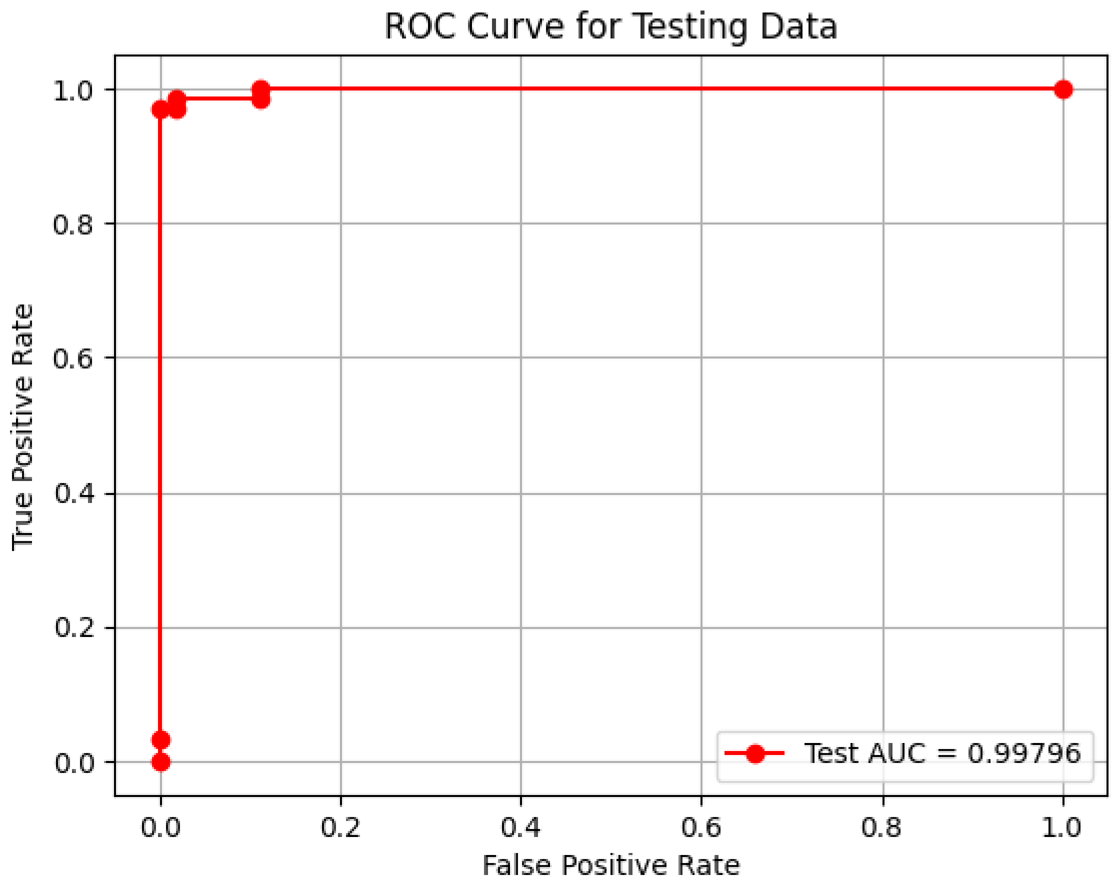
Testing Set

In [ ]:

```
ypred_test = model.predict(X_test)
cfm2 = confusion_matrix(y_test, ypred_test)
cm_display2 = ConfusionMatrixDisplay(confusion_matrix = cfm2,
                                     display_labels = ["Benign", "Malignant"])
cm_display2.plot()
plt.title('Confusion Matrix for Testing Data')
plt.show()

# For Testing Data
y_test_prob = model.predict_proba(X_test)[:, 1]
fpr_test, tpr_test, _ = metrics.roc_curve(y_test, y_test_prob)
auc_test = metrics.roc_auc_score(y_test, y_test_prob)
plt.figure()
plt.plot(fpr_test, tpr_test, 'r-o', label=f'Test AUC = {auc_test:.5f}')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Testing Data')
plt.legend()
plt.grid()
plt.show()
print(f"Test AUC: {auc_test:.5f}")

print(f'Test Accuracy: {test_accuracy : .6f}')
# Calculate the False Positive Rate (FPR)
true_negatives, false_positives = cfm2[0]
false_negatives, true_positives = cfm2[1]
```

```python
recall = true_positives / (true_positives + false_negatives)
precision = true_positives / (true_positives + false_positives)
f1_score = 2 * (recall * precision) / (recall + precision)
false_alarm_rate = false_positives / (false_positives + true_negatives)

print(f'Recall: {recall : .6f}')
print(f'Precision: {precision : .6f}')
print(f'F1 Score: {f1_score : .6f}')
print(f'False Alarm Rate: {false_alarm_rate : .6f}')
```






```
Test AUC: 0.99796
Test Accuracy:  0.982456
Recall:  0.968750
Precision:  0.984127
F1 Score:  0.976378
False Alarm Rate:  0.009346
```

Item 2-b-testing

Confusion Matrix, ROC Curve, AUC are above

Evaluation Metric	Score
Testing AUC	0.99796
Testing Accuracy	0.982456
Recall	0.968750
Precision	0.984127
F1 Score	0.976378
False Alarm Rate	0.009346

Problem #2 - Insights

Q: Based on your results for this Problem, what insights did you gain?

A: Here are some insights

- Interestingly, the testing data's AUC marginally surpasses that of the training data.
- As expected, training data accuracy outperforms testing data accuracy.
- Metrics such as Recall, Precision, F1 Score, and False Alarm rate exhibit better values in the training data compared to the testing data.
- Since the accuracy of the testing data is relatively high, the model may generize well with unseen data.
- The model demonstrates a typical performance pattern, where training accuracy is higher than testing accuracy, suggesting no overfitting issues.
- The model effectively balances sensitivity and specificity, minimizing both false negatives and false positives, which is vital in medical diagnostics.
- To accurately predict tumor malignancy, medical practitioners should prioritize evaluating texture, radius, concavity, and tumor area, as these features have the highest predictive power.

=======END OF SOLUTIONS=======

Course Code: AI 221

Course Name: Classical Machine Learning

Submitted By: Geyzson Kristoffer S. Homena

Submitted To: Karl Ezra S. Pilario, PhD

1st Semester 2023-2024