Practical Activity 4.4.1: Logistic Regression

Logistic regression is a linear model for classification. Logistic regression is also known as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. More technical details can be found at https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

In this practical, we will use the Breast Cancer Wisconsin (Diagnostic) Dataset. The dataset is downloaded from the UCI Machine Learning repository https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic) (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)).

The dataset consists of 569 samples. Each sample has 30 features. The features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

The task is to classify each sample into two classes Benign (1) or Malignant (0).

In [84]:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

In [85]:

```
# Load the dataset
data = pd.read_csv('wdbc.csv', header=0)
```

In [86]:

```python
# inspect the data, drop rows with missing data
data = data.dropna()
print(data.shape)
data.head()
```

```
(569, 31)
```

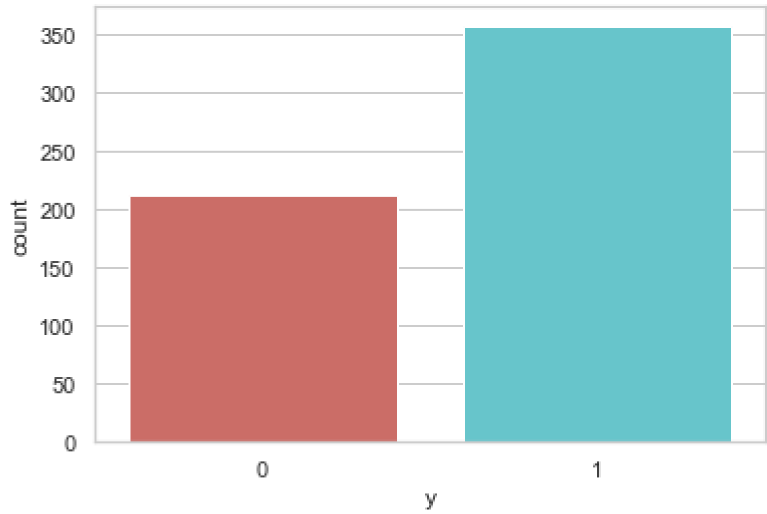
Out[86]:

	Feature1	Feature2	Feature3	Feature4	Feature5	Feature6	Feature7	Feature8	Feature9	Fe
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 31 columns

In [87]:

```python
#See label distribution
sns.countplot(x='y',data=data, palette='hls')
plt.show()
```



In [88]:

```
count_Benign = len(data[data['y']==1])
count_Malignant= len(data[data['y']==0])
Total = count_Benign + count_Malignant
pct_of_Ben = count_Benign/Total
print("Percentage of benign samples in the dataset:", pct_of_Ben*100)
pct_of_Malignant = count_Malignant/Total
print("Percentage of malignant samples in the dataset: ", pct_of_Malignant*100)

```

```
Percentage of benign samples in the dataset: 62.741652021089635
Percentage of malignant samples in the dataset:  37.258347978910365
```

We can see that the classes are slightly imbalanced. The number of samples available for benign class are significantly higher than the malignant class.

In [75]:

```
X = data.iloc[:, data.columns != 'y']
y = data.iloc[:, data.columns == 'y']
```

In [89]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                     random_state=0)
columns = X_train.columns
```

Feature scaling or Feature normalization

Feature normalization is a common pre-processing requirement for many machine learning models. In this example, we use the min-max normalization.

In [90]:

```
#Feature scaling or feature normalization
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
x_train_scaled = scaler.fit_transform(X_train)
```

In [91]:

```
X_train = pd.DataFrame(x_train_scaled, columns=X_train.columns)
x_test_scaled = scaler.fit_transform(X_test)
X_test = pd.DataFrame(x_test_scaled, columns=X_test.columns)
```

In [92]:

```
X_train = X_train.iloc[:,:]
X_test = X_test.iloc[:,:]
```

Feature selection

Different kinds of feature selection algorithms can be used select only the most important features. In this example we use the SlectKBest feature selection method.

In [93]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif

skb = SelectKBest(score_func=mutual_info_classif, k=10)
sel_skb = skb.fit(X_train, y_train.values.ravel())
selected_features_ind = sel_skb.get_support()
#print('scores: ', sel_skb.scores_)
#print(sel_skb_index)
```

In [94]:

```
X_train_new = X_train.iloc[:, selected_features_ind]
X_test_new = X_test.iloc[:, selected_features_ind]

X_train_new.shape
```

Out[94]:

```
(398, 10)
```

Train and test a Logistic Regression model

In [51]:

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

In [96]:

```
#Train a Logistic regression model
logreg = LogisticRegression()
logreg.fit(X_train_new, y_train.values.ravel())
```

Out[96]:

```
LogisticRegression()
```

In [97]:

```
#Test the model
y_pred = logreg.predict(X_test_new)
```

Results and Analysis

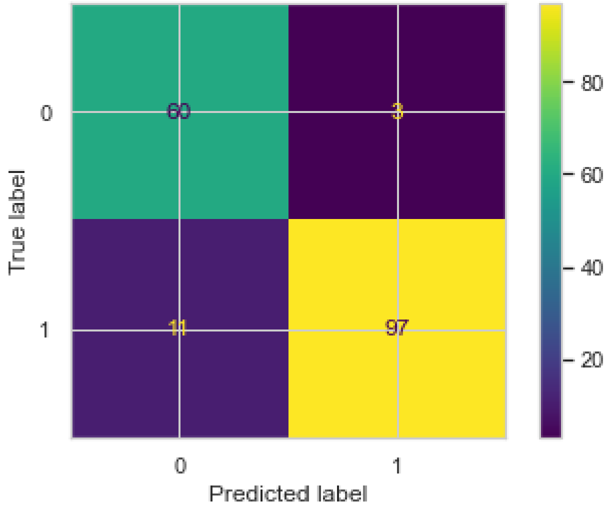
In [99]:

```
#Classification accuracy
print('Accuracy of the logistic regression classifier on test set: {:.2f}'
      .format(logreg.score(X_test_new, y_test)))
```

```
Accuracy of the logistic regression classifier on test set: 0.92
```

In [100]:

```
# Confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import plot_confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
#print(confusion_matrix)
disp = ConfusionMatrixDisplay(confusion_matrix)
disp.plot()
plt.show()
```



In [103]:

```
#Performance measures
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

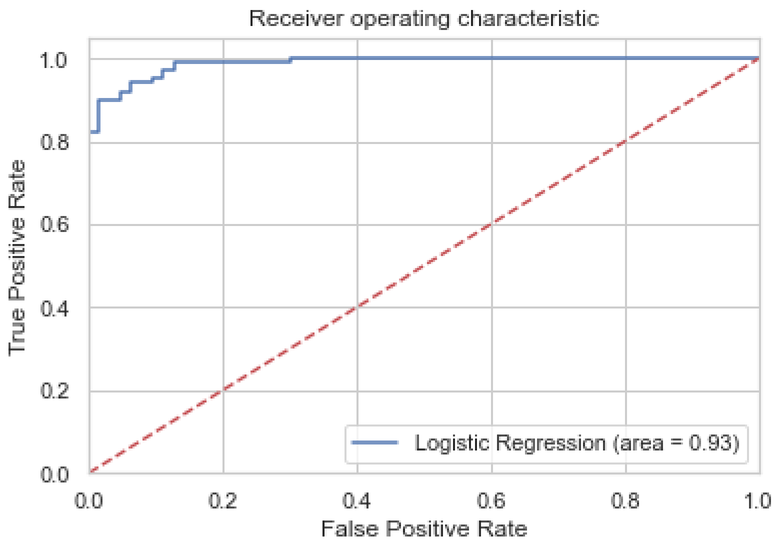
```
              precision    recall  f1-score   support

           0       0.85      0.95      0.90        63
           1       0.97      0.90      0.93       108

    accuracy                           0.92       171
   macro avg       0.91      0.93      0.91       171
weighted avg       0.92      0.92      0.92       171
```

In [101]:

```
#ROC Curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test_new))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test_new)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



Model tuning

You can perform the following experiments for experimentally tuning your model to achieve the best possible accuracy.

1. Use different types of feature scaling and analyze the classification accuracy of the model. [https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)](https://scikit-learn.org/stable/modules/preprocessing.html)
2. Vary parameter k (number of features) in the SelectKBest and see it's effect on the classification accuracy of the model. Can you experimentally find a good parameter k that gives the maximum accuracy?

3. Use a different kind of feature selection method such as Recursive Feature Elimination and see it's effect on the accuracy. [https://scikit-learn.org/stable/modules/feature_selection.html (https://scikit-learn.org/stable/modules/feature_selection.html)](https://scikit-learn.org/stable/modules/feature_selection.html)
4. Use different parameters for the LogisticRegression() model and analyze the results. [https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

In [ ]:

```

```