Logistic Regression

Logistic regression is a fundamental classification technique. It belongs to the group of linear classifiers and is somewhat similar to polynomial and linear regression. Logistic regression is fast and relatively uncomplicated, and it’s convenient for you to interpret the results. Although it’s essentially a method for binary classification, it can also be applied to multiclass problems.

You’ll need an understanding of the sigmoid function and the natural logarithm function to understand what logistic regression is and how it works.

This image shows the sigmoid function (or S-shaped curve) of some variable x:

The sigmoid function has values very close to either 0 or 1 across most of its domain. This fact makes it suitable for application in classification methods.

Single-Variate Logistic Regression

Single-variate logistic regression is the most straightforward case of logistic regression. There is only one independent variable (or feature), which is $\mathbf{x} = x$. This figure illustrates single-variate logistic regression:

Here, you have a given set of input-output (or x-y) pairs, represented by green circles. These are your observations. Remember that y can only be 0 or 1. For example, the leftmost green circle has the input $x = 0$ and the actual output $y = 0$. The rightmost observation has $x = 9$ and $y = 1$.

Logistic regression finds the weights b_0 and b_1 that correspond to the maximum log-likelihood function (LLF). These weights define the logit $f(x) = b_0 + b_1x$, which is the dashed black line. They also define the predicted probability $p(x) = 1/(1 + exp(-f(x)))$, shown here as the full black line. In this case, the threshold $p(x) = 0.5$ and $f(x) = 0$ corresponds to the value of x slightly higher than 3. This value is the limit between the inputs with the predicted outputs of 0 and 1.

Logistic Regression in Python

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

In [3]:
```
np.arange(10).reshape(-1, 1)
```

Out[3]:
```
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
```

```
       [8],
       [9]])
```

In [3]:

```
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

In [4]:

```
model = LogisticRegression(random_state = 42)
```

In [5]:

```
model.fit(x, y)
```

Out[5]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=42, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=0, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

In [5]:

```
print("Classes: ", model.classes_)
print("Intercept: ",model.intercept_)
print("Coef: ",model.coef_)
```

```
Classes:  [0 1]
Intercept:  [-4.12617727]
Coef:  [[1.18109091]]
```

$\hat{y} = f(x) = 1.18109091 * x_1 - 4.12617727$

In [6]:

```
print("Probability estimates:","\n",model.predict_proba(x))
```

```
Probability estimates:
 [[0.98411203 0.01588797]
 [0.95003074 0.04996926]
 [0.85370936 0.14629064]
 [0.64173546 0.35826454]
 [0.35475873 0.64524127]
 [0.1443924  0.8556076 ]
 [0.04924876 0.95075124]
 [0.01565079 0.98434921]
 [0.00485659 0.99514341]
 [0.00149573 0.99850427]]
```

In [7]:

```
model.predict(x)
print("Actual (class) predictions:","\n",model.predict(x))
```

```
Actual (class) predictions:
 [0 0 0 0 1 1 1 1 1 1]
```

In [7]:

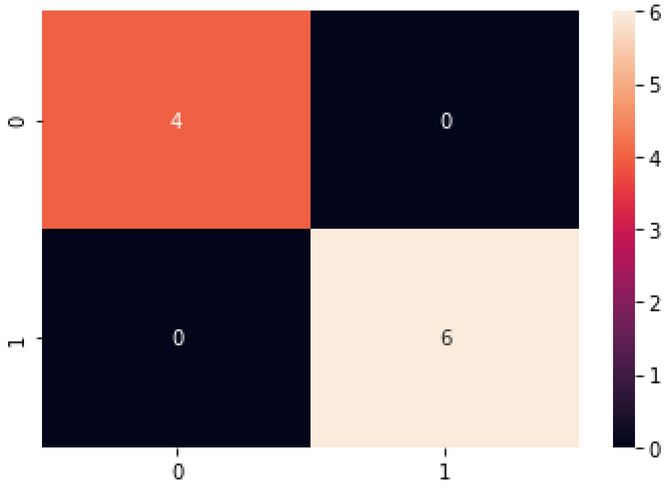
```
#Confusion Matrix
y_pred = model.predict(x)
confusion_matrix(y, y_pred)
```

Out[7]:

```
array([[4, 0],
       [0, 6]], dtype=int64)
```

In [8]:
```
import seaborn as sns

cm = confusion_matrix(y, model.predict(x))
sns.heatmap(cm, annot=True)
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2a36bdd0240>
```



In [9]:
```
y_pred = model.predict(x)
print(classification_report(y,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         6

    accuracy                           1.00        10
   macro avg       1.00      1.00      1.00        10
weighted avg       1.00      1.00      1.00        10
```

In [10]:
```
model.score(x,y)
```

Out[10]:
```
1.0
```

In [11]:
```
#Changing Hyperparameters
model  = LogisticRegression(solver='liblinear', C=0.5, random_state=0)
model.fit(x, y)
```

Out[11]:
```
LogisticRegression(C=0.5, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [12]:
```
print("Intercept of the function:","\n",model.intercept_)
```

```
Intercept of the function:
 [-0.61167085]
```

In [13]:
```
print("Weights of the function:","\n",model.coef_)
```

```
Weights of the function:
 [[0.41299976]]
```

In [16]:
```python
print("Probability estimates:","\n",model.predict_proba(x))
```

```
Probability estimates:
 [[0.64832185 0.35167815]
 [0.54950505 0.45049495]
 [0.44662201 0.55337799]
 [0.34811656 0.65188344]
 [0.26108668 0.73891332]
 [0.18948992 0.81051008]
 [0.13396721 0.86603279]
 [0.09284959 0.90715041]
 [0.06342763 0.93657237]
 [0.04288806 0.95711194]]
```

In [17]:
```python
print("Actual (class) predictions:","\n",model.predict(x))
```

```
Actual (class) predictions:
 [0 0 1 1 1 1 1 1 1 1]
```

In [14]:
```python
print("Accuracy of the model:",model.score(x, y))
```

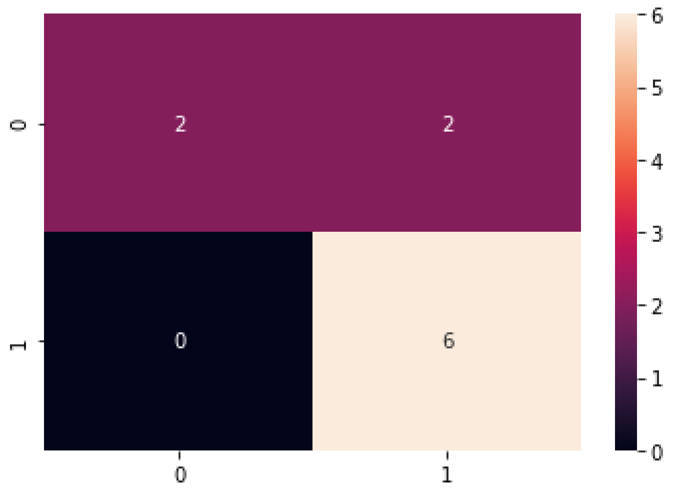
```
Accuracy of the model: 0.8
```

In [19]:
```python
#Confusion Matrix
confusion_matrix(y, model.predict(x))
```

Out[19]:
```
array([[2, 2],
       [0, 6]], dtype=int64)
```

In [15]:
```python
# Visualize confusion matrix with heatmap
sns.heatmap(confusion_matrix(y, model.predict(x)), annot=True)
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x2a36c16c390>



In [21]:
```python
print(classification_report(y, model.predict(x)))
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         4
           1       0.75      1.00      0.86         6
```

```
    accuracy                           0.80        10
   macro avg       0.88      0.75      0.76        10
weighted avg       0.85      0.80      0.78        10
```

Real Life Example

In [2]:

```
from sklearn.datasets import load_breast_cancer
import pandas as pd
X , y = load_breast_cancer(return_X_y=True)
df = pd.DataFrame(X,columns = load_breast_cancer().feature_names)
df.head()
```

Out[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	din
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 × 30 columns

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
```

```
 23  worst area              569 non-null    float64
 24  worst smoothness        569 non-null    float64
 25  worst compactness       569 non-null    float64
 26  worst concavity         569 non-null    float64
 27  worst concave points    569 non-null    float64
 28  worst symmetry          569 non-null    float64
 29  worst fractal dimension 569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

In [4]:

```
df.describe()
```

Out[4]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	co
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.0
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.0
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.0
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.0
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.0
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.0
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.0
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.2

8 rows × 30 columns

In [5]:

```
df.isna().sum()
```

Out[5]:

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
```

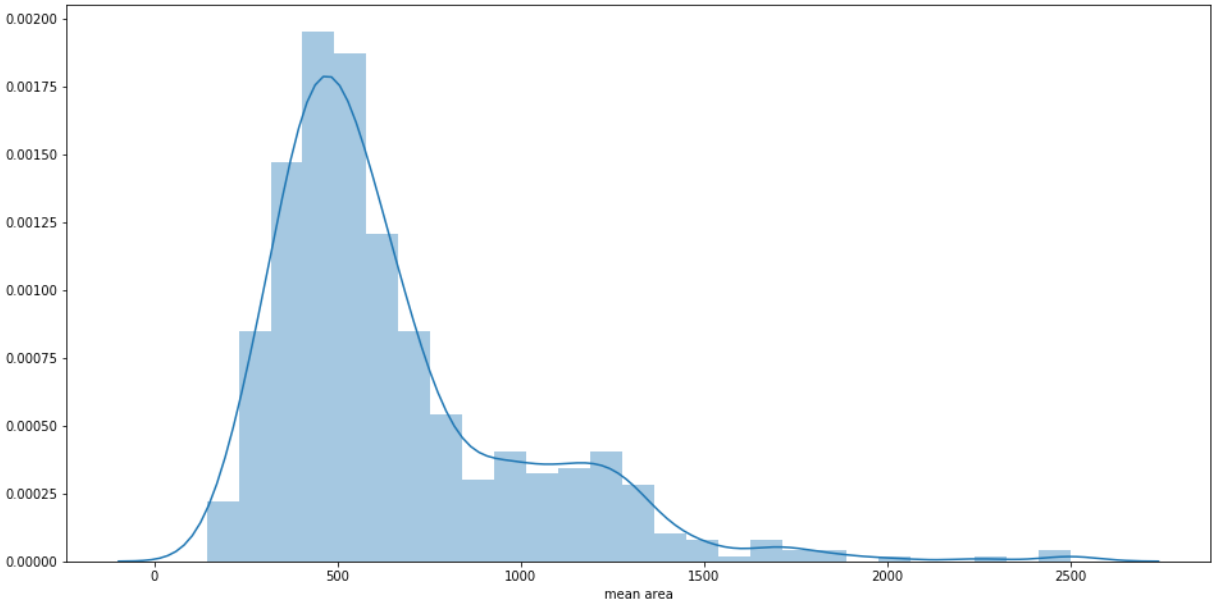
```
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
dtype: int64
```

In [6]:

```
import seaborn as sns
plt.figure(figsize=(16, 8))
sns.distplot(df["mean area"])
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26036d0aeb8>
```



In [8]:

```
# Outlier Detection
from scipy import stats
import numpy as np
z = np.abs(stats.zscore(df))
z
```

Out[8]:

```
array([[1.09706398, 2.07333501, 1.26993369, ..., 2.29607613, 2.75062224,
        1.93701461],
       [1.82982061, 0.35363241, 1.68595471, ..., 1.0870843 , 0.24388967,
        0.28118999],
       [1.57988811, 0.45618695, 1.56650313, ..., 1.95500035, 1.152255  ,
        0.20139121],
       ...,
       [0.70228425, 2.0455738 , 0.67267578, ..., 0.41406869, 1.10454895,
        0.31840916],
       [1.83834103, 2.33645719, 1.98252415, ..., 2.28998549, 1.91908301,
        2.21963528],
       [1.80840125, 1.22179204, 1.81438851, ..., 1.74506282, 0.04813821,
        0.75120669]])
```

In [9]:

```
outliers = list(set(np.where(z > 3)[0]))

len(outliers)
```

Out[9]:

```
74
```

In [11]:

```
new_df = df.drop(outliers,axis = 0).reset_index(drop = False)
display(new_df)
```

```
y_new = y[list(new_df["index"])]
len(y_new)
```

	index	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	symn
0	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0
1	2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0
2	4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0
3	5	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	0
4	6	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	0
...	...	...	...	...	...	...	...	...	...	
490	560	14.05	27.15	91.38	600.4	0.09929	0.11260	0.04462	0.04304	0
491	563	20.92	25.09	143.00	1347.0	0.10990	0.22360	0.31740	0.14740	0
492	564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0
493	565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0
494	566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0

495 rows × 31 columns

Out[11]: 495

In [12]:

```
#Scaling
X_new = new_df.drop('index', axis = 1)

from sklearn.preprocessing import StandardScaler, MinMaxScaler
X_scaled = MinMaxScaler().fit_transform(X_new)
X_scaled
```

Out[12]:

```
array([[0.83424397, 0.38362684, 0.82273105, ..., 0.68863384, 0.3717064 ,
        0.42980015],
       [0.78021978, 0.54926226, 0.79595605, ..., 0.89966679, 0.64240903,
        0.41158614],
       [0.81705445, 0.22037125, 0.84304312, ..., 0.60162903, 0.25062735,
        0.27498103],
       ...,
       [0.89502118, 0.60352213, 0.90674915, ..., 0.82043688, 0.15526976,
        0.20376929],
       [0.80723187, 0.88243693, 0.80703536, ..., 0.60273973, 0.31587202,
        0.14330888],
       [0.59052121, 0.87434555, 0.59560521, ..., 0.52499074, 0.20483061,
        0.29294207]])
```

In [14]:

```
cv['test_score']
```

Out[14]:

```
array([0.96551724, 0.93043478, 0.9826087 ])
```

In [15]:

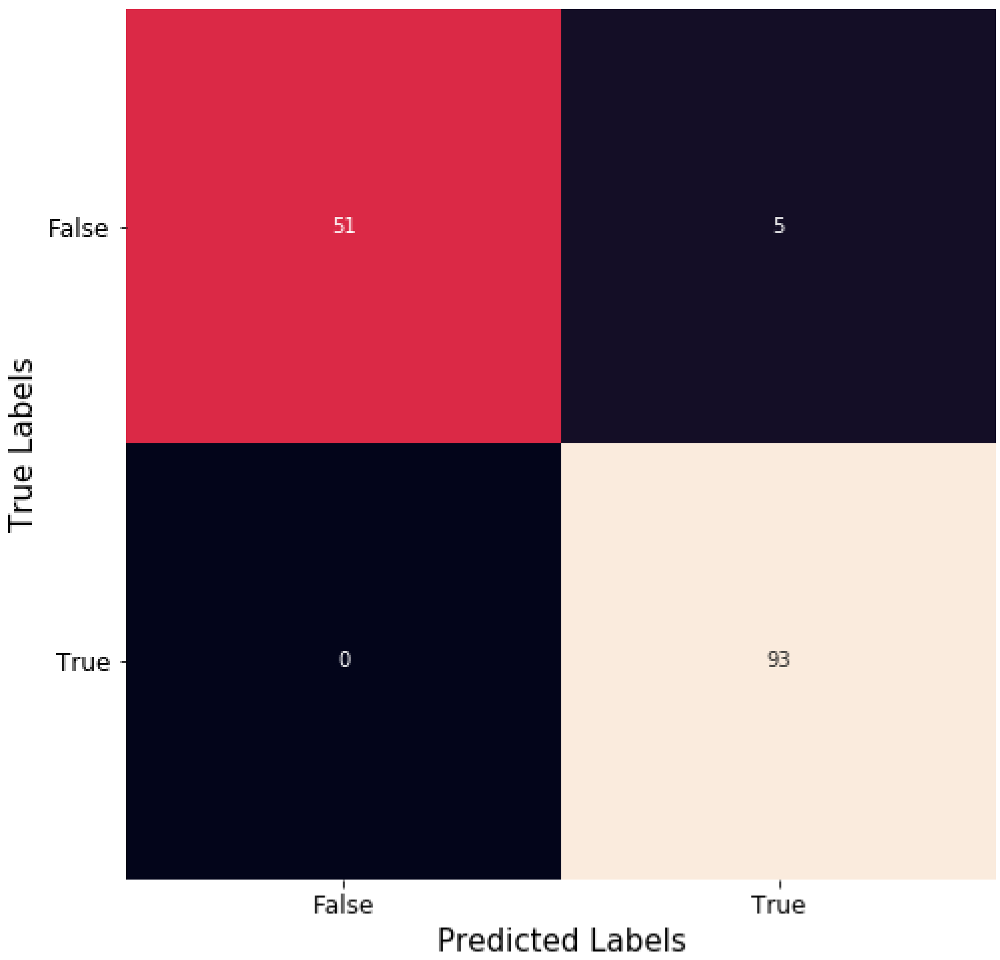
```
from sklearn.model_selection import train_test_split, cross_validate

#Scaling and outlier removed
X_train, X_test, y_train, y_test = train_test_split(X_scaled,y_new, test_size=0.3, r
```

```
models = LogisticRegression(random_state=42, n_jobs=-1)
cv = cross_validate(models,X_train,y_train, cv = 3, n_jobs=-1, return_estimator=True

print("Mean training accuracy: {}".format(np.mean(cv['test_score'])))
print("Test accuracy: {}".format(cv["estimator"][0].score(X_test,y_test)))
```

```
Mean training accuracy: 0.9595202398800601
Test accuracy: 0.9664429530201343
```

In [16]:

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
pred = cv["estimator"][0].predict(X_test)

cm = confusion_matrix(y_test, pred)
plt.figure(figsize=(12, 8))
ax =sns.heatmap(cm, square=True, annot=True, cbar=False)
ax.xaxis.set_ticklabels(["False","True"], fontsize = 12)
ax.yaxis.set_ticklabels(["False","True"], fontsize = 12, rotation=0)
ax.set_xlabel('Predicted Labels',fontsize = 15)
ax.set_ylabel('True Labels',fontsize = 15)
plt.show()
```



In [17]:

```
# Without any preprocess
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state

models = LogisticRegression(random_state=42,n_jobs=-1)
cv = cross_validate(models,X_train,y_train,cv = 3, n_jobs=-1, return_estimator=True)

print("Mean training accuracy: {}".format(np.mean(cv['test_score'])))
print("Test accuracy: {}".format(cv["estimator"][0].score(X_test,y_test)))
```

```
Mean training accuracy: 0.944691273638642
Test accuracy: 0.9707602339181286
```

Evaluation Metrics

F1-score = $F_1 = 2 * \frac{precision * recall}{precision + recall}$

Find all evaluation metrics in sklearn library by clicking here.

In [36]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, cv["estimator"][0].predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.98      0.94      0.96        63
           1       0.96      0.99      0.98       108

    accuracy                           0.97       171
   macro avg       0.97      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```

In [37]:
```
from sklearn.metrics import f1_score, accuracy_score, recall_score, precision_score

final_model = cv["estimator"][0]

y_pred = final_model.predict(X_test)

print("Accuracy:",accuracy_score(y_test,y_pred))
print("Precision:",precision_score(y_test,y_pred))
print("Recall:",recall_score(y_test,y_pred))
print("F1 Score:",f1_score(y_test,y_pred))
```

```
Accuracy: 0.9707602339181286
Precision: 0.963963963963964
Recall: 0.9907407407407407
F1 Score: 0.9771689497716894
```

When to use Accuracy Metric

When there are roughly equal number of samples belonging to each class.

When to use Precision

Precision is a good measure to determine, when the costs of False Positive is high. For instance, email spam detection. In email spam detection, a false positive means that an email that is non-spam (actual negative) has been identified as spam (predicted spam). The email user might lose important emails if the precision is not high for the spam detection model.

When to use Recall

For instance, in fraud detection or sick patient detection. If a fraudulent transaction (Actual Positive) is predicted as non-fraudulent (Predicted Negative), the consequence can be very bad for the bank.

When to use F1 Score

F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

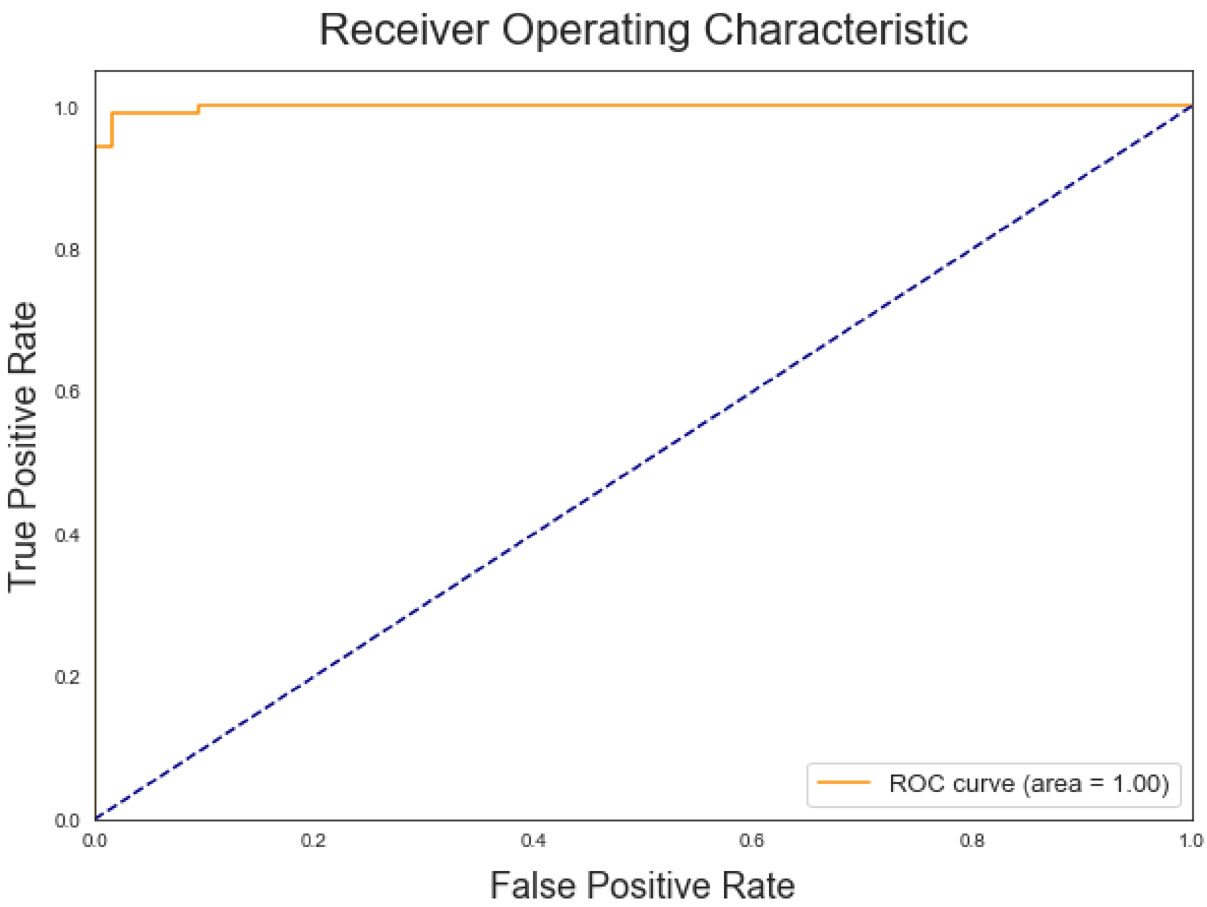
ROC (Receiver Operating Characteristic) & AUC (Area Under the Curve)

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s.

True Positive Rate (TPR) = $TP\,/\,(TP + FN)$
False Positive Rate (FPR) = $FP\,/\,(FP + FN)$

In [38]:

```
from sklearn.metrics import roc_curve, auc

y_pred_prop = final_model.predict_proba(X_test)[:,1]

fpr_log, tpr_log, _ = roc_curve(y_test, y_pred_prop)
roc_auc_log = auc(fpr_log, tpr_log)

sns.set_style("white")
plt.figure(figsize=(10, 7))
plt.plot(fpr_log, tpr_log, color='darkorange',
         label='ROC curve (area = %0.2f)' % roc_auc_log)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate',fontsize=18,labelpad =10)
plt.ylabel('True Positive Rate',fontsize=18)
plt.title('Receiver Operating Characteristic',fontsize=22).set_position([.5, 1.02])
plt.legend(loc="lower right",fontsize=13)
plt.show()
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0