Logistics Regression

Prof.Dr. Bahadır AKTUĞ
Machine Learning with Python

**Compiled from sources given in the references.*

Logistics Regression

- Logistic regression is a statistical method for predicting binary classes.
- In many problems, the outcome or target variable is dichotomous in nature.
- However, Logistics regression was generalized so as to apply to multi-class classification problems.
- It is a special case of linear regression where the target variable is categorical in nature.
- It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

Logistics Regression

$$\text{logit}(p) = \log(\frac{p}{1-p})$$

p = probability

$\frac{p}{1-p}$ = corresponding odds

- Linear Regression Equation

$$y = \beta 0 + \beta 1X1 + \beta 2X2 + \ldots\ldots + \beta nXn$$

- Sigmoid Function

$$p = 1/1 + e^{-y}$$

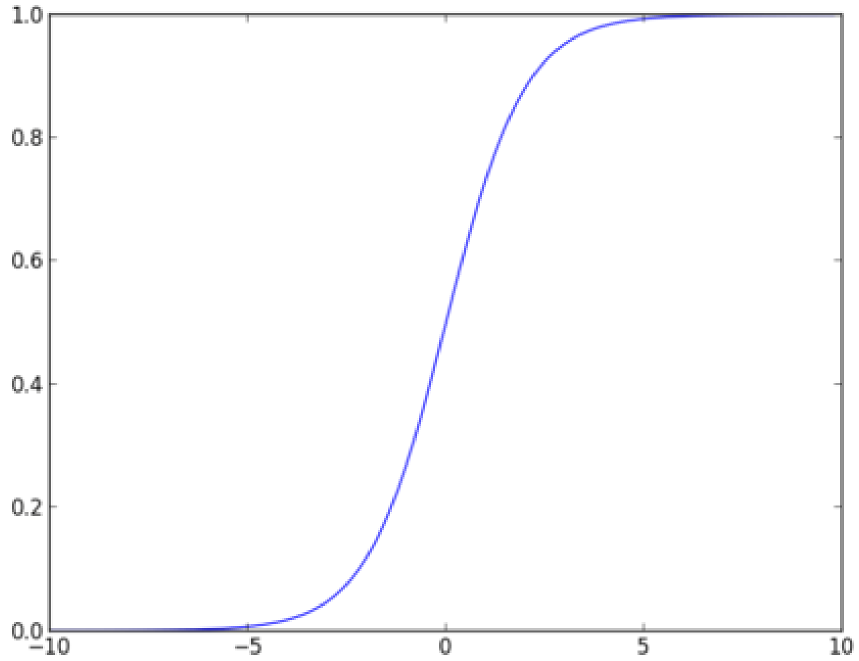
- Apply Sigmoid function on linear regression

$$p = 1/1 + e^{-(\beta 0 + \beta 1X1 + \beta 2X2 \ldots\ldots \beta nXn)}$$

Logistics Regression

- Sigmoid Function

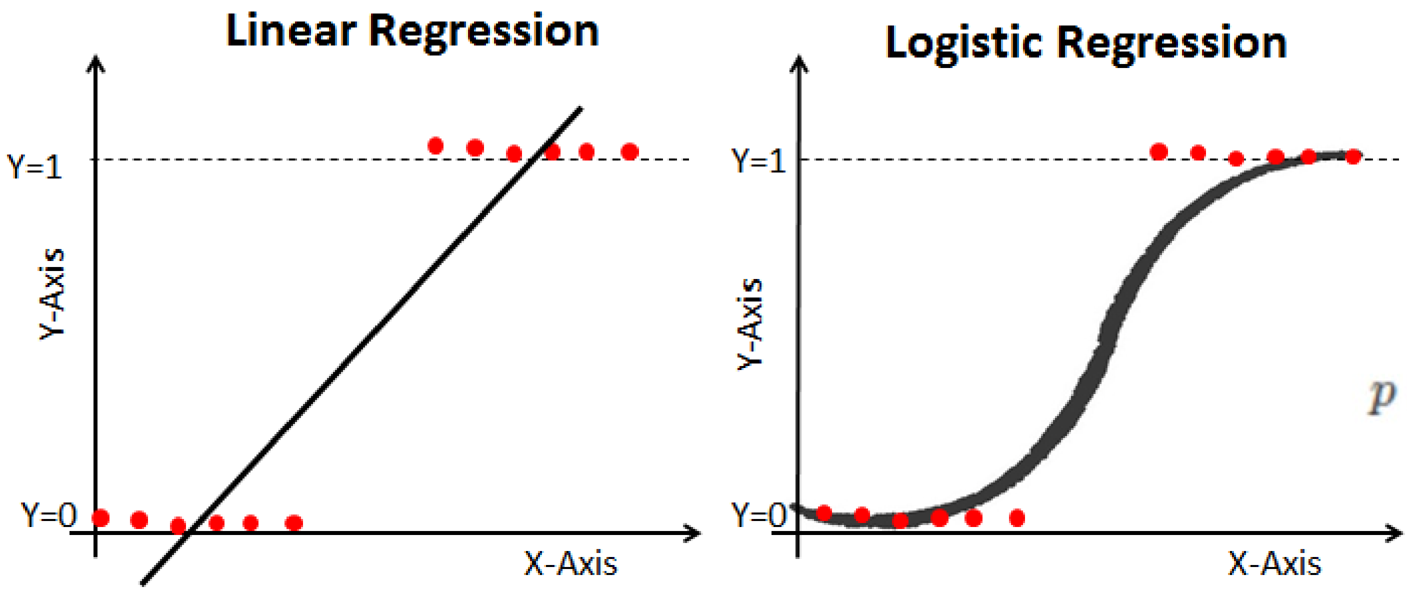
$$f(x) = \frac{1}{1 + e^{-(x)}}$$



Logistics Regression

- Properties of Logistic Regression:
  - The dependent variable in logistic regression follows Bernoulli Distribution.
  - Estimation is done through maximum likelihood.
- Linear regression gives you a continuous output, but logistic regression provides a constant output.
- An example of the continuous output is house price and stock price. Example's of the discrete output is predicting whether a patient has cancer or not, predicting whether the customer will churn.
- Linear regression is estimated using Ordinary Least Squares (OLS) while logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach.

Logistics Regression



$$p = 1/1 + e^{-y}$$

$$y = \beta0 + \beta1X1 + \beta2X2 + \ldots\ldots + \beta nXn$$

$$p = 1/1 + e^{-(\beta0+\beta1X1+\beta2X2.....\beta nXn)}$$

Types of Logistic Regression

- **Binary Logistic Regression**: The target variable has only two possible outcomes such as Spam or Not Spam, Cancer or No Cancer.
- **Multinomial Logistic Regression**: The target variable has three or more nominal categories such as predicting the type of Wine.
- **Ordinal Logistic Regression**: the target variable has three or more ordinal categories such as restaurant or product rating from 1 to 5.

Multinomial Logistics Regression

- Multinomial logistic regression is a classification method that generalizes logistic regression to multiclass problems, i.e. with more than two possible discrete outcomes.
- It is a model that is used to predict the probabilities of the different possible outcomes of a categorically distributed dependent variable, given a set of independent variables (which may be real-valued, binary-valued, categorical-valued, etc.).
  - For this purpose the logit function is generalized to multiple classes with the sum of all probabilities is 1.

$$\Pr(Y_i = 1) = \frac{e^{\beta_1 \cdot \mathbf{X}_i}}{1 + \sum_{k=1}^{K-1} e^{\beta_k \cdot \mathbf{X}_i}}$$

$$\Pr(Y_i = 2) = \frac{e^{\beta_2 \cdot \mathbf{X}_i}}{1 + \sum_{k=1}^{K-1} e^{\beta_k \cdot \mathbf{X}_i}}$$

$$\cdots\cdots$$

$$\Pr(Y_i = K-1) = \frac{e^{\beta_{K-1} \cdot \mathbf{X}_i}}{1 + \sum_{k=1}^{K-1} e^{\beta_k \cdot \mathbf{X}_i}}$$

Properties of Logistic Regression

- Estimation is done through maximum likelihood,
- No R Square, Model fitness is calculated through Concordance, KS-Statistics.
- The dependent variable in logistic regression follows Bernoulli Distribution.
- Logistic Regression can be used for various classification problems such as spam detection.
- Logistic Regression is one of the most simple and commonly used Machine Learning algorithms for two-class classification.
- Easy to implement and can be used as the baseline for any binary classification problem.
- Its basic fundamental concepts are also constructive in deep learning.
- Logistic regression describes and estimates the relationship between one dependent binary variable and independent variables.

sklearn.linear_model.LogisticRegression

- class sklearn.linear_model.LogisticRegression(
  - penalty='l2', *,
  - dual=False,
  - tol=0.0001,
  - C=1.0,
  - fit_intercept=True,
  - intercept_scaling=1,
  - class_weight=None,
  - random_state=None,
  - solver='lbfgs’,
  - max_iter=100,
  - multi_class='auto’,
  - verbose=0,
  - warm_start=False,
  - n_jobs=None,
  - l1_ratio=None)

sklearn.linear_model.LogisticRegression

penalty{'l1', 'l2', 'elasticnet', 'none'}, default='l2'

Used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties. 'elasticnet' is only supported by the 'saga' solver. If 'none' (not supported by the liblinear solver), no regularization is applied.

New in version 0.19: l1 penalty with SAGA solver (allowing 'multinomial' + L1)

dual*bool, default=False*

Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n_samples > n_features.

tol*float, default=1e-4*

Tolerance for stopping criteria.

C*float, default=1.0*

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

fit_intercept*bool, default=True*

Specifies if a constant (a.k.a. bias or intercept) should be added to the decision function.

intercept_scaling*float, default=1*

Useful only when the solver 'liblinear' is used and self.fit_intercept is set to True. In this case, x becomes [x, self.intercept_scaling], i.e. a "synthetic" feature with constant value equal to intercept_scaling is appended to the instance vector. The intercept becomes intercept_scaling * synthetic_feature_weight.

Note! the synthetic feature weight is subject to l1/l2 regularization as all other features. To lessen the effect of regularization on synthetic feature weight (and therefore on the intercept) intercept_scaling has to be increased.

sklearn.linear_model.LogisticRegression

class_weight*dict or ‘balanced’, default=None*
Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.
The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)).

Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.
New in version 0.17: class_weight=’balanced’
random_state*int, RandomState instance, default=None*
Used when solver == ‘sag’, ‘saga’ or ‘liblinear’ to shuffle the data. See Glossary for details.
solver*{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=’lbfgs’*
Algorithm to use in the optimization problem.

- For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ and ‘saga’ are faster for large ones.
- For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss; ‘liblinear’ is limited to one-versus-rest schemes.
- ‘newton-cg’, ‘lbfgs’, ‘sag’ and ‘saga’ handle L2 or no penalty
- ‘liblinear’ and ‘saga’ also handle L1 penalty
- ‘saga’ also supports ‘elasticnet’ penalty
- ‘liblinear’ does not support setting penalty='none'

Note that ‘sag’ and ‘saga’ fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing.
New in version 0.17: Stochastic Average Gradient descent solver.
New in version 0.19: SAGA solver.
Changed in version 0.22: The default solver changed from ‘liblinear’ to ‘lbfgs’ in 0.22.

sklearn.linear_model.LogisticRegression

max_iter*int, default=100*
Maximum number of iterations taken for the solvers to converge.
multi_class*{'auto', 'ovr', 'multinomial'}, default='auto'*
If the option chosen is 'ovr', then a binary problem is fit for each label. For 'multinomial' the loss minimised is the multinomial loss fit across the entire probability distribution, *even when the data is binary*. 'multinomial' is unavailable when solver='liblinear'. 'auto' selects 'ovr' if the data is binary, or if solver='liblinear', and otherwise selects 'multinomial'.
New in version 0.18: Stochastic Average Gradient descent solver for 'multinomial' case.
Changed in version 0.22: Default changed from 'ovr' to 'auto' in 0.22.
verbose*int, default=0*
For the liblinear and lbfgs solvers set verbose to any positive number for verbosity.
warm_start*bool, default=False*
When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution. Useless for liblinear solver. See the Glossary.
New in version 0.17: warm_start to support *lbfgs*, *newton-cg*, *sag*, *saga* solvers.
n_jobs*int, default=None*
Number of CPU cores used when parallelizing over classes if multi_class='ovr'". This parameter is ignored when the solver is set to 'liblinear' regardless of whether 'multi_class' is specified or not. None means 1 unless in a **joblib.parallel_backend** context. -1 means using all processors. See Glossary for more details.
l1_ratio*float, default=None*
The Elastic-Net mixing parameter, with 0 <= l1_ratio <= 1. Only used if penalty='elasticnet'. Setting l1_ratio=0 is equivalent to using penalty='l2', while setting l1_ratio=1 is equivalent to using penalty='l1'. For 0 < l1_ratio <1, the penalty is a combination of L1 and L2.

1. Example: Classifying IRIS Dataset

- We can use Logistics Regression over IRIS dataset.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

dataLoaded = load_iris()
print("Data Shape" , dataLoaded.data.shape)
print("Label Shape", dataLoaded.target.shape)
```

```
Data Shape (150, 4)
Label Shape (150,)
```

1. Example: Classifying IRIS Dataset

- Except for the command for loading dataset, the same code can be used:
  - from sklearn.datasets import load_iris
- The data consists of 150 individual observations of 4 features:

```
>>> iris.head()
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
>>>
```

1. Example: Classifying IRIS Dataset

```
x_train, x_test, y_train, y_test = train_test_split(dataLoaded.data,
                   dataLoaded.target, test_size=0.25, random_state=0)
logisticRegr = LogisticRegression(max_iter=100)
logisticRegr.fit(x_train, y_train)
predictions = logisticRegr.predict(x_test)
score = logisticRegr.score(x_test, y_test)
print(score)
```

```
>>> predictions
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2])
```

```
>>> y_test
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1])
```

```
>>> predictions == y_test
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False])
```

1. Example: Classifying IRIS Dataset

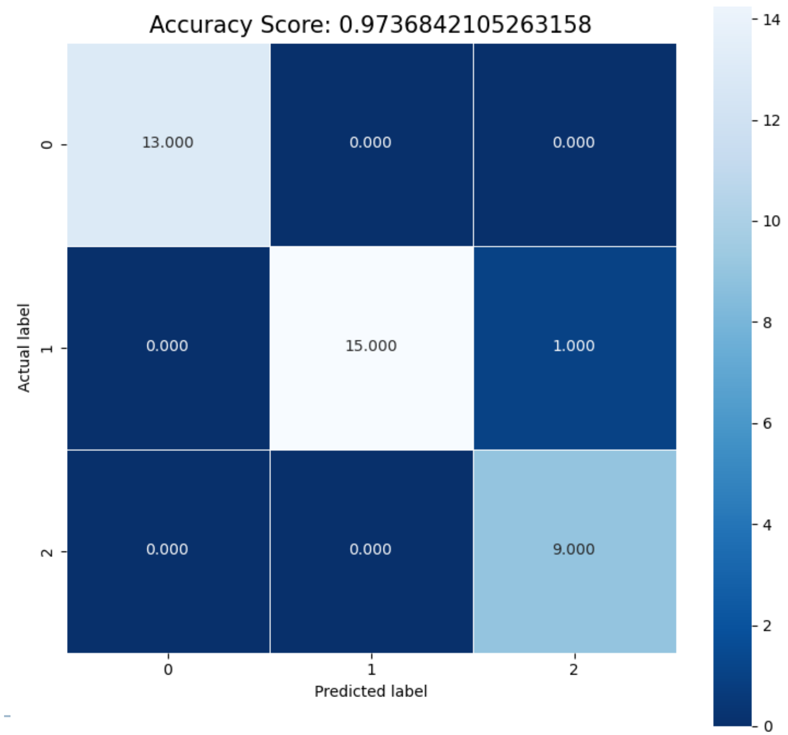
```
x_train, x_test, y_train, y_test = train_test_split(dataLoaded.data,
                   dataLoaded.target, test_size=0.25, random_state=0)
logisticRegr = LogisticRegression(max_iter=100)
logisticRegr.fit(x_train, y_train)
predictions = logisticRegr.predict(x_test)
score = logisticRegr.score(x_test, y_test)
print(score)
```

```
cm = metrics.confusion_matrix(y_test, predictions)
print(cm)
```

```
0.9736842105263158
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```

1. Example: Classifying IRIS Dataset

```
# Seaborn
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15)
plt.show()
```



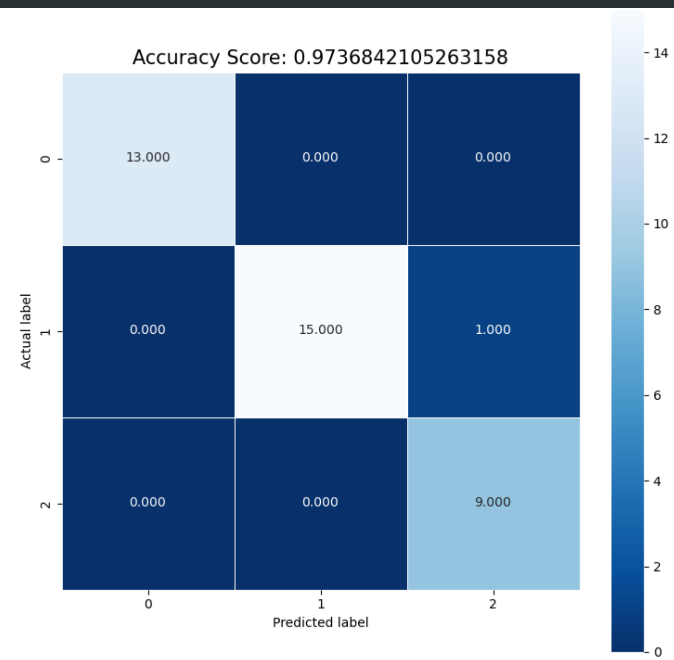
1. Example: Classifying IRIS Dataset

- Let’s see if scaling the input data helps.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test  = sc.transform(x_test)
```

- Exactly the same results!

```
0.9736842105263158
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```



1. Example: Classifying IRIS Dataset

y_test	y_pred	setosa(%)	versicolor(%)	virginica(%)
virginica	virginica	0.00	0.03	0.97
versicolor	versicolor	0.01	0.95	0.04
setosa	setosa	1.00	0.00	0.00
virginica	virginica	0.00	0.08	0.92
setosa	setosa	0.98	0.02	0.00
virginica	virginica	0.00	0.01	0.99
setosa	setosa	0.98	0.02	0.00
versicolor	versicolor	0.01	0.71	0.28
versicolor	versicolor	0.00	0.73	0.27
versicolor	versicolor	0.02	0.89	0.08
virginica	virginica	0.00	0.44	0.56
versicolor	versicolor	0.02	0.76	0.22
versicolor	versicolor	0.01	0.85	0.13
versicolor	versicolor	0.00	0.69	0.30
versicolor	versicolor	0.01	0.75	0.24
setosa	setosa	0.99	0.01	0.00
versicolor	versicolor	0.02	0.72	0.26
versicolor	versicolor	0.03	0.86	0.11
setosa	setosa	0.94	0.06	0.00
setosa	setosa	0.99	0.01	0.00
virginica	virginica	0.00	0.17	0.83
versicolor	versicolor	0.04	0.71	0.25
setosa	setosa	0.98	0.02	0.00
setosa	setosa	0.96	0.04	0.00

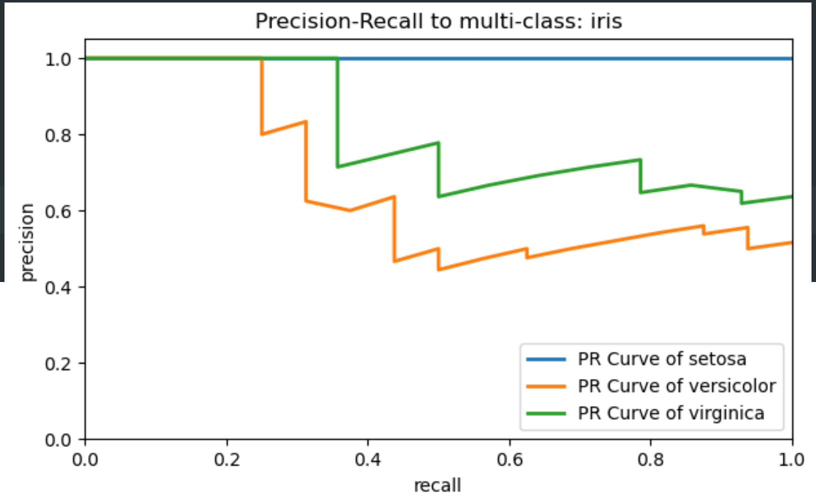
Internally, Logistics Regression produces a probabilty for each class.

This probability is then transformed into classification based on a threshold.

For the binary case, this threshold is 0.5

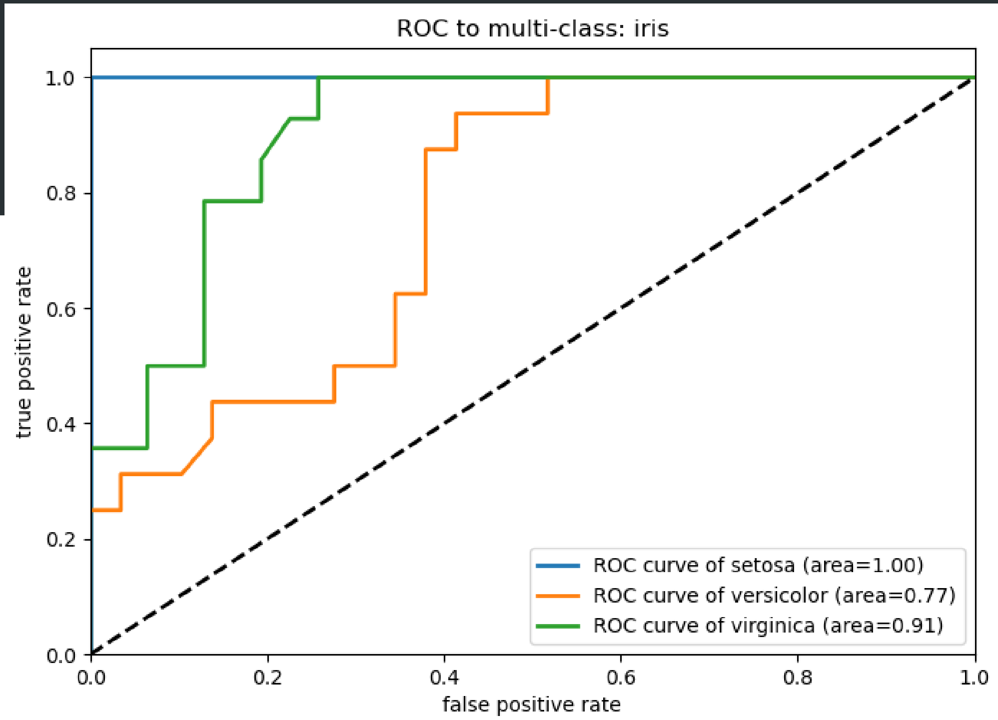
1. Example: Classifying IRIS Dataset

```python
# precision recall curve
from sklearn.metrics import precision_recall_curve,roc_curve
from sklearn.preprocessing import label_binarize

n_classes = len(set(dataLoaded.target))
y_test = label_binarize(y_test, classes=[*range(n_classes)])
precision = dict()
recall = dict()
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test[:, i],
                                                        y_probs[:, i])
    plt.plot(recall[i], precision[i], lw=2, label=dataLoaded.target_names[i])

plt.xlabel("recall")
plt.ylabel("precision")
plt.legend(loc="best")
plt.title("precision vs. recall curve")
plt.show()
```



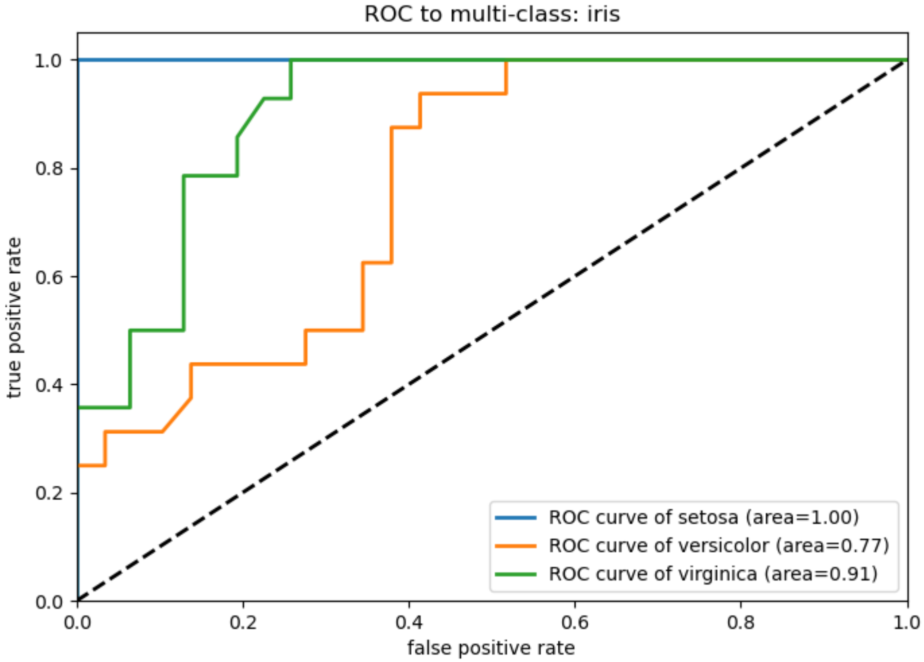
1. Example: Classifying IRIS Dataset

```
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i],
                                  y_probs[:, i])
    plt.plot(fpr[i], tpr[i], lw=2, label=dataLoaded.target_names[i])

plt.xlabel("false positive rate")
plt.ylabel("true positive rate")
plt.legend(loc="best")
plt.title("ROC curve")
plt.show()
```



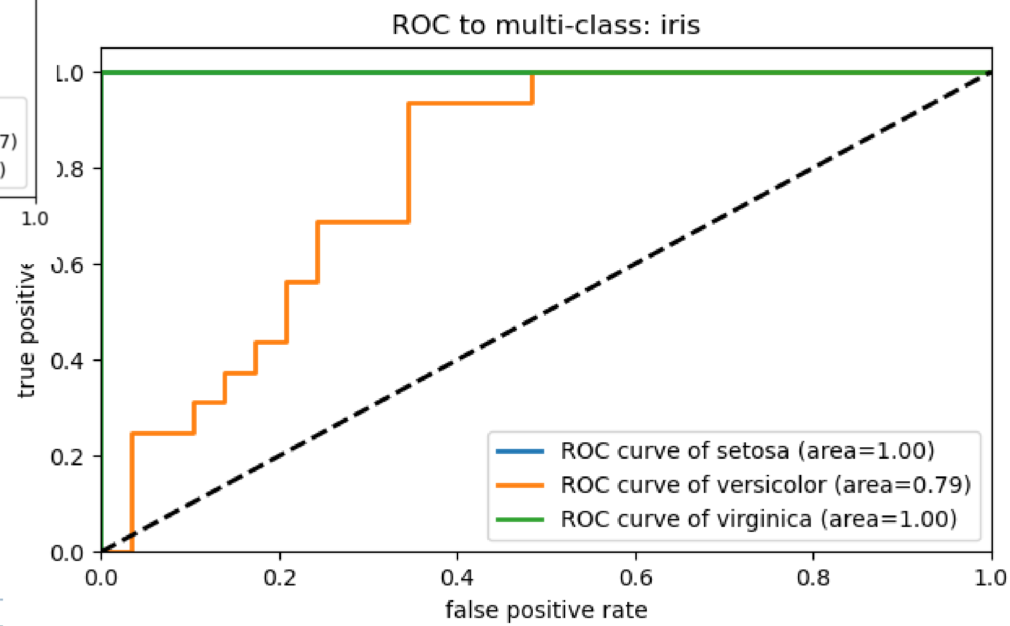
1. Example: Classifying IRIS Dataset

Only first two features are used



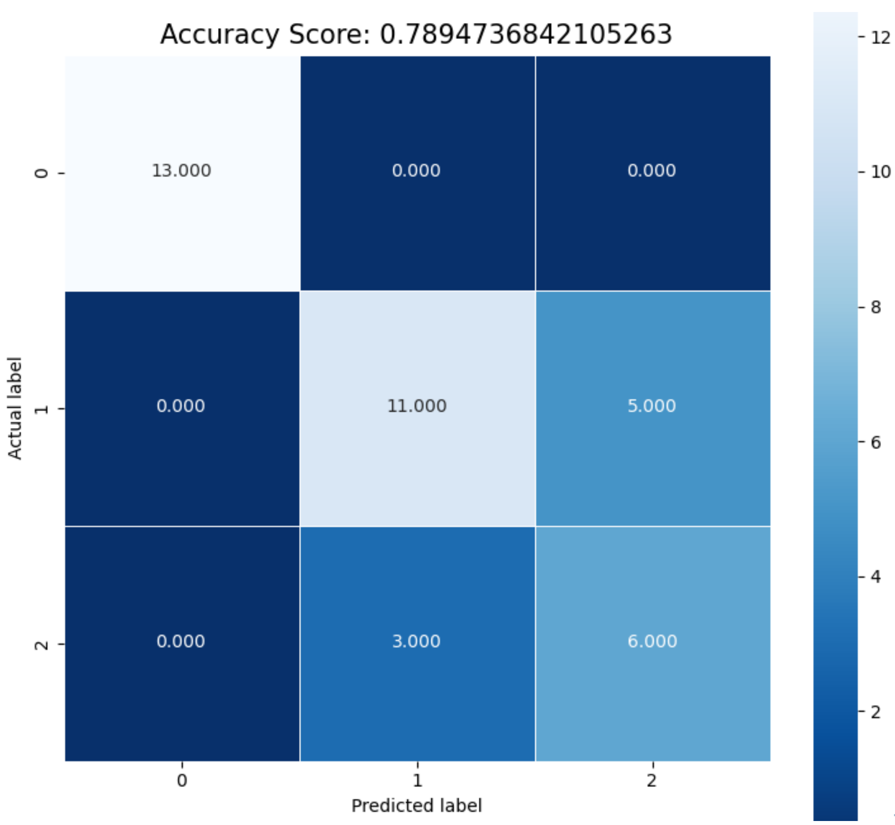

```
x = dataLoaded.data[:,:2]
y = dataLoaded.target
```

All four features are used

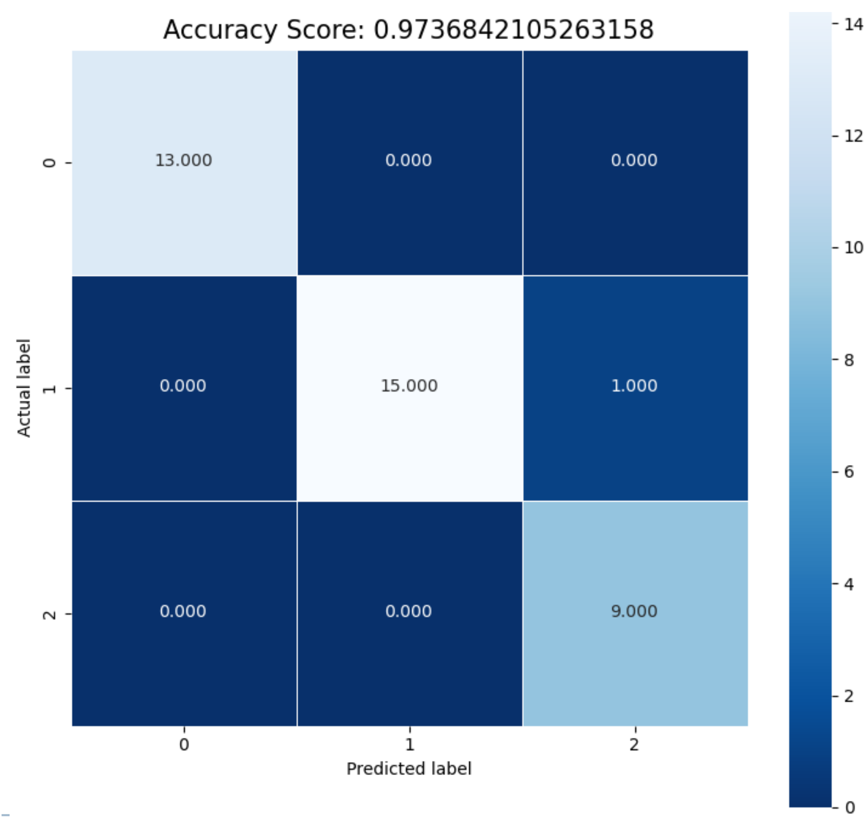


1. Example: Classifying IRIS Dataset

Only first two features are used



All four features are used



2. Example: Recognizing Handwritten Digits

```python
from sklearn.datasets import load_digits
digits = load_digits()
print("Image Data Shape" , digits.data.shape)
print("Label Data Shape", digits.target.shape)
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:5],
                                           digits.target[0:5])):
        plt.subplot(1, 5, index + 1)
        plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
        plt.title('Training: %i\n' % label, fontsize = 20)
plt.show()
```

Example: Recognizing Handwritten Digits

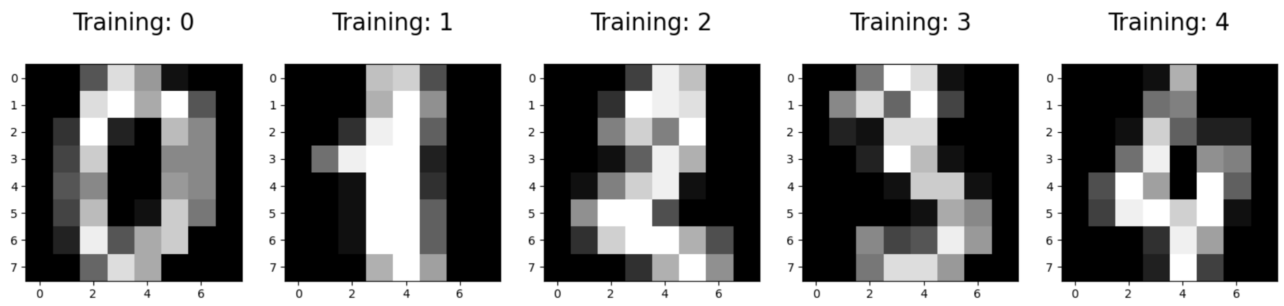
```
print("Image Data Shape" , digits.data.shape)
print("Label Data Shape", digits.target.shape)
```

The commands above will produce the following output:

Image Data Shape (1797, 64)

Label Data Shape (1797,)

The plot command will plot the first 5 data and associated label:



Splitting Data into Training and Test Sets (Digits Dataset)

Split the data into training and test sets:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.25, random_state=0)
from sklearn.linear_model import LogisticRegression
```

Make an instance of the Model (Logistics Regression)

```
logisticRegr = LogisticRegression()
logisticRegr.fit(x_train, y_train)
```

Predict for whole test dataset:

```
predictions = logisticRegr.predict(x_test)
```

Measure Model Performance

- The accuracy (fraction of correct predictions): correct predictions / total number of data points:

```
score = logisticRegr.score(x_test, y_test)
print(score)
```

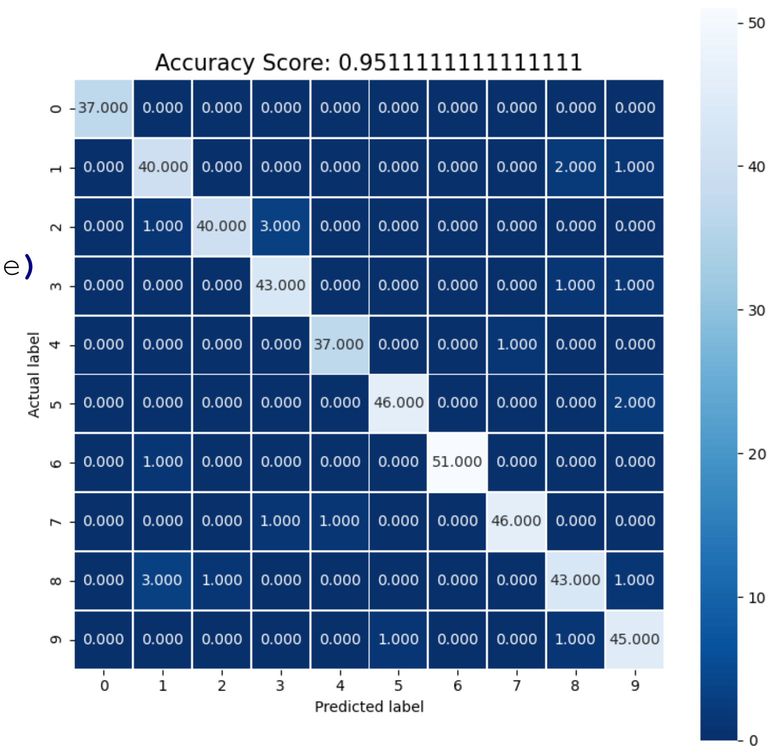
- The computed accuracy is ~0.95 (%95)
- The confusion matrix

```
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 40  0  0  0  0  0  0  2  1]
 [ 0  1 40  3  0  0  0  0  0  0]
 [ 0  0  0 43  0  0  0  0  1  1]
 [ 0  0  0  0 37  0  0  1  0  0]
 [ 0  0  0  0  0 46  0  0  0  2]
 [ 0  1  0  0  0  0 51  0  0  0]
 [ 0  0  0  1  1  0  0 46  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  0  0  1  0  0  1 45]]
```

Measure Model Performance

- A better plot for confusion matrix can be produced either by «seaborn» or «matplotlib» module:

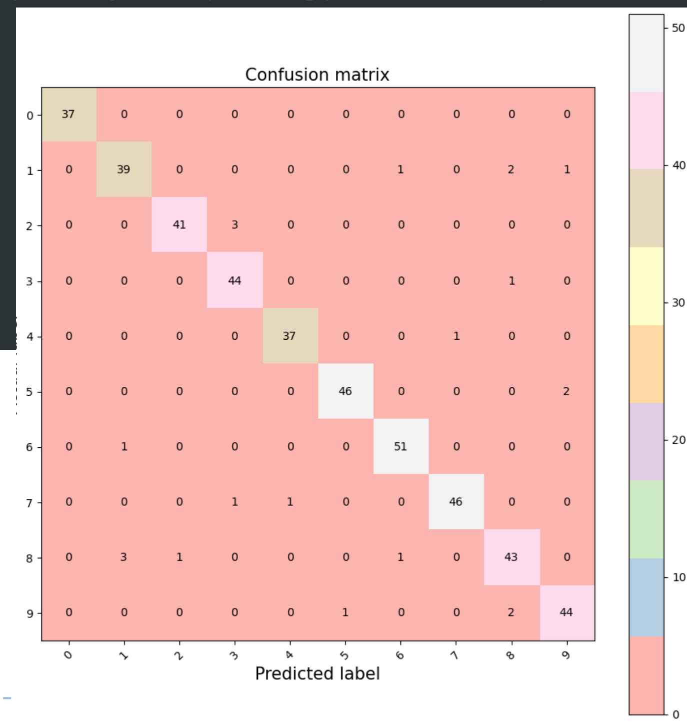
```
import seaborn as sns
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f",
linewidths=.5, square = True, cmap = 'Blues_r’)
plt.ylabel('Actual label’)
plt.xlabel('Predicted label’)
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15)
plt.show()
```



Measure Model Performance

- Matplotlib can also be used:

```
plt.figure(figsize=(9,9))
plt.imshow(cm, interpolation='nearest', cmap='Pastel1')
plt.title('Confusion matrix', size = 15)
plt.colorbar()
tick_marks = np.arange(10)
plt.xticks(tick_marks, ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"], rotation=45, size = 10
plt.yticks(tick_marks, ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"], size = 10)
plt.tight_layout()
plt.ylabel('Actual label', size = 15)
plt.xlabel('Predicted label', size = 15)
width, height = cm.shape
for x in range(width):
    for y in range(height):
        plt.annotate(str(cm[x][y]), xy=(y, x),
        horizontalalignment='center',
        verticalalignment='center')
```



Measure Model Performance

```
>>> predictions
array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5, 2, 8, 6, 6, 6, 6, 1, 0, 5, 8, 8, 7,
       8, 4, 7, 5, 4, 9, 2, 9, 4, 7, 6, 8, 9, 4, 3, 1, 0, 1, 8, 6, 7, 7,
       1, 0, 7, 6, 2, 1, 9, 6, 7, 9, 0, 0, 9, 1, 6, 3, 0, 2, 3, 4, 1, 9,
       2, 6, 9, 1, 8, 3, 5, 1, 2, 8, 2, 2, 9, 7, 2, 3, 6, 0, 5, 3, 7, 5,
       1, 2, 9, 9, 3, 1, 4, 7, 4, 8, 5, 8, 5, 5, 2, 5, 9, 0, 7, 1, 4, 7,
       3, 4, 8, 9, 7, 9, 8, 2, 1, 5, 2, 5, 8, 4, 1, 7, 0, 6, 1, 5, 5, 9,
       9, 5, 9, 9, 5, 7, 5, 6, 2, 8, 6, 9, 6, 1, 5, 1, 5, 9, 9, 1, 5, 3,
       6, 1, 8, 9, 8, 7, 6, 7, 6, 5, 6, 0, 8, 8, 9, 9, 6, 1, 0, 4, 1, 6,
       3, 8, 6, 7, 4, 9, 6, 3, 0, 3, 3, 3, 0, 7, 7, 5, 7, 8, 0, 7, 1, 9,
       6, 4, 5, 0, 1, 4, 6, 4, 3, 3, 0, 9, 5, 9, 2, 8, 4, 2, 1, 6, 8, 9,
       2, 4, 9, 3, 7, 6, 2, 3, 3, 1, 6, 9, 3, 6, 3, 3, 2, 0, 7, 6, 1, 1,
       9, 7, 2, 7, 8, 5, 5, 7, 5, 3, 3, 7, 2, 7, 5, 5, 7, 0, 9, 1, 6, 5,
       9, 7, 4, 3, 8, 0, 3, 6, 4, 6, 3, 2, 6, 8, 8, 8, 4, 6, 7, 5, 2, 4,
       5, 3, 2, 4, 6, 9, 4, 5, 4, 3, 4, 6, 2, 9, 0, 1, 7, 2, 0, 9, 6, 0,
       4, 2, 0, 7, 9, 8, 5, 7, 8, 2, 8, 4, 3, 7, 2, 6, 9, 9, 5, 1, 0, 8,
       2, 8, 9, 5, 6, 2, 2, 7, 2, 1, 5, 1, 6, 4, 5, 0, 9, 4, 1, 1, 7, 0,
       8, 9, 0, 5, 4, 3, 8, 8, 6, 5, 3, 4, 4, 4, 8, 8, 7, 0, 9, 6, 3, 5,
       2, 3, 0, 8, 8, 3, 1, 3, 3, 0, 0, 4, 6, 0, 7, 7, 6, 2, 0, 4, 4, 2,
       3, 7, 1, 9, 8, 6, 8, 5, 6, 2, 2, 3, 1, 7, 7, 8, 0, 3, 3, 1, 1, 5,
       5, 9, 1, 3, 7, 0, 0, 3, 0, 4, 5, 8, 9, 3, 4, 3, 1, 8, 9, 8, 3, 6,
       3, 1, 6, 2, 1, 7, 5, 5, 1, 9])
>>> y_test
array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5, 2, 8, 6, 6, 6, 6, 1, 0, 5, 8, 8, 7,
       8, 4, 7, 5, 4, 9, 2, 9, 4, 7, 6, 8, 9, 4, 3, 1, 0, 1, 8, 6, 7, 7,
       1, 0, 7, 6, 2, 1, 9, 6, 7, 9, 0, 0, 5, 1, 6, 3, 0, 2, 3, 4, 1, 9,
       2, 6, 9, 1, 8, 3, 5, 1, 2, 8, 2, 2, 9, 7, 2, 3, 6, 0, 5, 3, 7, 5,
       1, 2, 9, 9, 3, 1, 7, 7, 4, 8, 5, 8, 5, 5, 2, 5, 9, 0, 7, 1, 4, 7,
       3, 4, 8, 9, 7, 9, 8, 2, 6, 5, 2, 5, 8, 4, 8, 7, 0, 6, 1, 5, 9, 9,
       9, 5, 9, 9, 5, 7, 5, 6, 2, 8, 6, 9, 6, 1, 5, 1, 5, 9, 9, 1, 5, 3,
       6, 1, 8, 9, 8, 7, 6, 7, 6, 5, 6, 0, 8, 8, 9, 8, 6, 1, 0, 4, 1, 6,
       3, 8, 6, 7, 4, 5, 6, 3, 0, 3, 3, 3, 0, 7, 7, 5, 7, 8, 0, 7, 8, 9,
       6, 4, 5, 0, 1, 4, 6, 4, 3, 3, 0, 9, 5, 9, 2, 1, 4, 2, 1, 6, 8, 9,
       2, 4, 9, 3, 7, 6, 2, 3, 3, 1, 6, 9, 3, 6, 3, 2, 2, 0, 7, 6, 1, 1,
       9, 7, 2, 7, 8, 5, 5, 7, 5, 2, 3, 7, 2, 7, 5, 5, 7, 0, 9, 1, 6, 5,
       9, 7, 4, 3, 8, 0, 3, 6, 4, 6, 3, 2, 6, 8, 8, 8, 4, 6, 7, 5, 2, 4,
       5, 3, 2, 4, 6, 9, 4, 5, 4, 3, 4, 6, 2, 9, 0, 1, 7, 2, 0, 9, 6, 0,
       4, 2, 0, 7, 9, 8, 5, 4, 8, 2, 8, 4, 3, 7, 2, 6, 9, 1, 5, 1, 0, 8,
       2, 1, 9, 5, 6, 8, 2, 7, 2, 1, 5, 1, 6, 4, 5, 0, 9, 4, 1, 1, 7, 0,
       8, 9, 0, 5, 4, 3, 8, 8, 6, 5, 3, 4, 4, 4, 8, 8, 7, 0, 9, 6, 3, 5,
       2, 3, 0, 8, 3, 3, 1, 3, 3, 0, 0, 4, 6, 0, 7, 7, 6, 2, 0, 4, 4, 2,
       3, 7, 8, 9, 8, 6, 8, 5, 6, 2, 2, 3, 1, 7, 7, 8, 0, 3, 3, 2, 1, 5,
       5, 9, 1, 3, 7, 0, 0, 7, 0, 4, 5, 9, 3, 3, 4, 3, 1, 8, 9, 8, 3, 6,
       2, 1, 6, 2, 1, 7, 5, 5, 1, 9])
```

```
>>> predictions == y_test
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True, False,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True, False,  True,
        True,  True,  True,  True, False,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True, False,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True, False,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
       False,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True, False,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
       False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True, False, False,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
        True,  True,  True,  True,  True,  True,  True,  True,  True])
>>>
```

References

1 https://scikit-learn.org/
2 https://towardsdatascience.com/
3 McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython 2nd Edition.
4 Albon, C. (2018). Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning
5 Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 1st Edition
6 Müller, A. C., Guido, S. (2016). Introduction to Machine Learning with Python: A Guide for Data Scientists
7 Burkov, A. (2019). The Hundred-Page Machine Learning Book.
8 Burkov, A. (2020). Machine Learning Engineering.
9 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). Data Structures and Algorithms in Python, Wiley.
10 https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python
11 https://docs.python.org/3/tutorial/
12 http://www.python-course.eu
13 https://developers.google.com/edu/python/
14 http://learnpythonthehardway.org/book/