Introduction to Machine Learning in Python – Workshop 2

Presented by Jiajia Li

31.07.2024

Australian National University

Agenda

Recap

Machine learning approaches:

- Supervised learning
- Unsupervised
- Reinforcement

The machine learning models we are going to talk in this course is supervised learning.

Machine learning models:

- Artificial neural networks (ANNs)
- Decision trees
- Support-vector machines (SVMs)
- Regression analysis
- Bayesian networks
- Gaussian processes
- Genetic algorithms

Recap - Data

Critical patients' physiological data on the first day of admission to the intensive care unit (ICU).

```python
import pandas as pd

cohort = pd.read_csv('./eicu_cohort.csv')
cohort.head()
```

Target

	gender	age	admissionweight	unabridgedhosplos	acutephysiologyscore	apachescore	actualhospitalmortality	heartrate	meanbp	creatinine	temperature
0	Male	45.0	116.0	3.0778	41	46	ALIVE	109.0	154.0	1.01	36.20
1	Male	57.0	NaN	7.6736	26	31	ALIVE	106.0	46.0	-1.00	36.30
2	Female	59.0	66.6	15.0778	56	61	ALIVE	134.0	172.0	1.03	34.80
3	Male	63.0	71.9	1.3201	77	88	EXPIRED	133.0	40.0	4.30	32.60
4	Male	67.0	104.8	1.5257	75	88	EXPIRED	31.0	133.0	0.70	36.44

Recap

Encoding

Transformation of categorical data into a numerical format that can be used by machine learning algorithms.

Categories	Encoded
ALIVED	0
EXPIRED	1

```
# add the encoded value to a new column
cohort['actualhospitalmortality_enc'] = cohort['actualhospitalmortality'].cat.codes
```

Recap

Partitioning

Divide our dataset to “training” and “test” set. We use a split of 70/30.


```
from sklearn.model_selection import train_test_split

x = cohort.drop(['actualhospitalmortality', 'actualhospitalmortality_enc'], axis=1)
y = cohort['actualhospitalmortality_enc']

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.7, random_state=42)
```

Recap – Handle missing data

Check if any data is missing: `cohort.isnull().sum()`

Imputing missing data

To avoid data leaking between our training and test sets, we take the median from the training set only, and use it to impute missing values in both training and test set.

```
# impute missing values from the training set
x_train = x_train.fillna(x_train.median())
x_test = x_test.fillna(x_train.median())
```

Recap - Normalisation

To adjust the scales of features to a common range without distorting differences in the ranges of values. This is crucial for many machine learning algorithms that are sensitive to the scale of data.

Min-Max Normalisation

Scales the data to a fixed range, usually [0, 1] or [-1, 1].

Normalized Value

Original Value

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Maximum Value of x

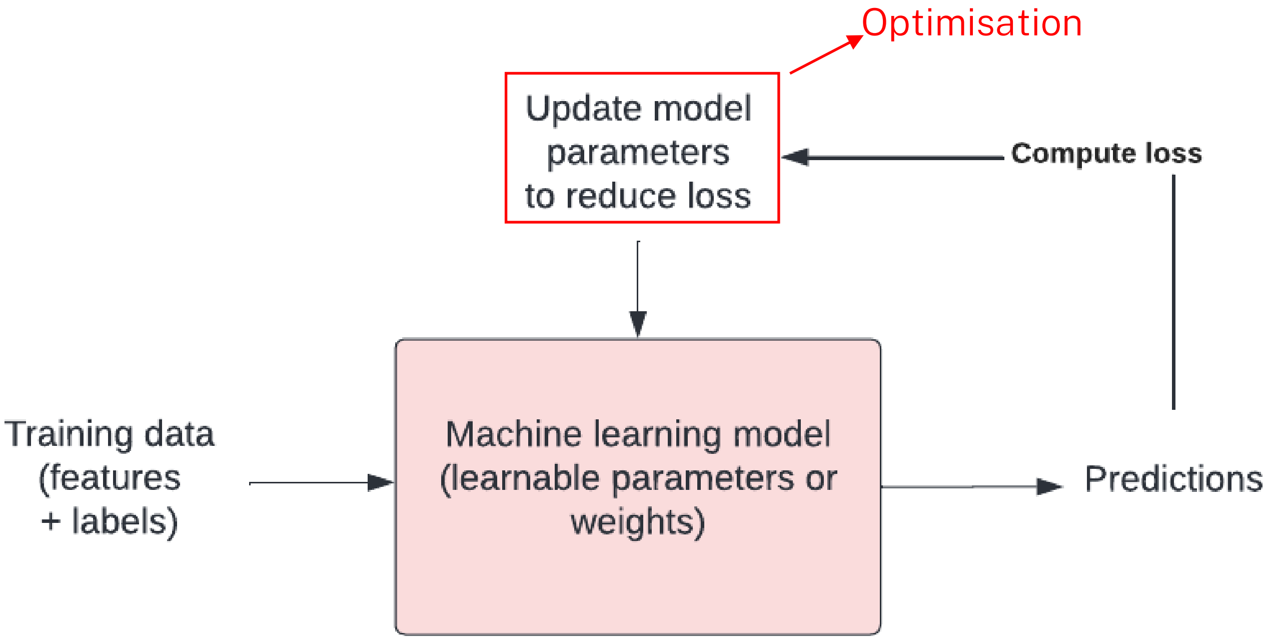
Minimum Value of x

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

Recap – How do machines learn?

Up until this point, our data is ready to be fed into a machine learning model.



Use 2-dimensional linear regression as an example

$$y = mx + c$$

First, we initialise m=0 and c=0.

Then, we can calculate the loss:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - (mx_i + c))^2$$

x	y
1	2
2	3
3	5
4	7
5	11

Loss = 41.6

Optimisation – Gradient Descent

Gradient descent is an iterative optimisation algorithm used to minimise the cost function. The updates for **m** and **c** are computed as follows:

$$m \leftarrow m - \alpha \frac{\partial \mathrm{MSE}}{\partial m} \qquad c \leftarrow c - \alpha \frac{\partial \mathrm{MSE}}{\partial c}$$

Where alpha is the **learning rate**, and partial derivatives are:

$$\frac{\partial \mathrm{MSE}}{\partial m} = -\frac{2}{n} \sum_{i=1}^{n} (y_i - (mx_i + c))x_i \qquad \frac{\partial \mathrm{MSE}}{\partial c} = -\frac{2}{n} \sum_{i=1}^{n} (y_i - (mx_i + c))$$

We set learning rate to **0.01** here.

Optimisation – Gradient Descent

After calculation, our new parameters should be:

- m=0.424
- c=0.112

Then we can use the new parameters to run the next iteration.

Let's have a look of the code implementation in Jupyter notebook.

Use 2-dimensional linear regression as an example

The linear regression model with gradient descent optimisation was implemented as the “**SGDRegressor()**” function in “sklearn”.

Where the “**LinearRegression()**” function from “sklearn” is an **ordinary least squares linear regression** model, with a normal equation to calculate the parameters.

$$m = \frac{n\sum(x_i y_i) - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$$

$$c = \frac{\sum y_i - m\sum x_i}{n}$$

Fit data to LinearRegression()

Here, we use “apachescore” as “x”, and “actualhospitalmortality” as “y”.

The steps to fit a model:

- Data partition
- Normalisation
- Fit the model
- Predict values

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression

x_apache = cohort_enc['apachescore'].values.reshape(-1, 1)
y = cohort['actualhospitalmortality_enc']

x_apache_train, x_apache_test, y_train, y_test = train_test_split(x_apache, y, train_size=.7, random_state=42)

scaler = MinMaxScaler()
scaler.fit(x_apache_train)
x_apache_train = scaler.transform(x_apache_train)
x_apache_test = scaler.transform(x_apache_test)

model = LinearRegression()
model = model.fit(x_apache_train, y_train)
y_pred = model.predict(x_apache_test)
```

Plot the fitted line

After fitting the model and making predictions. Let’s plot the fitted line to see the relationship between our prediction and the true values.

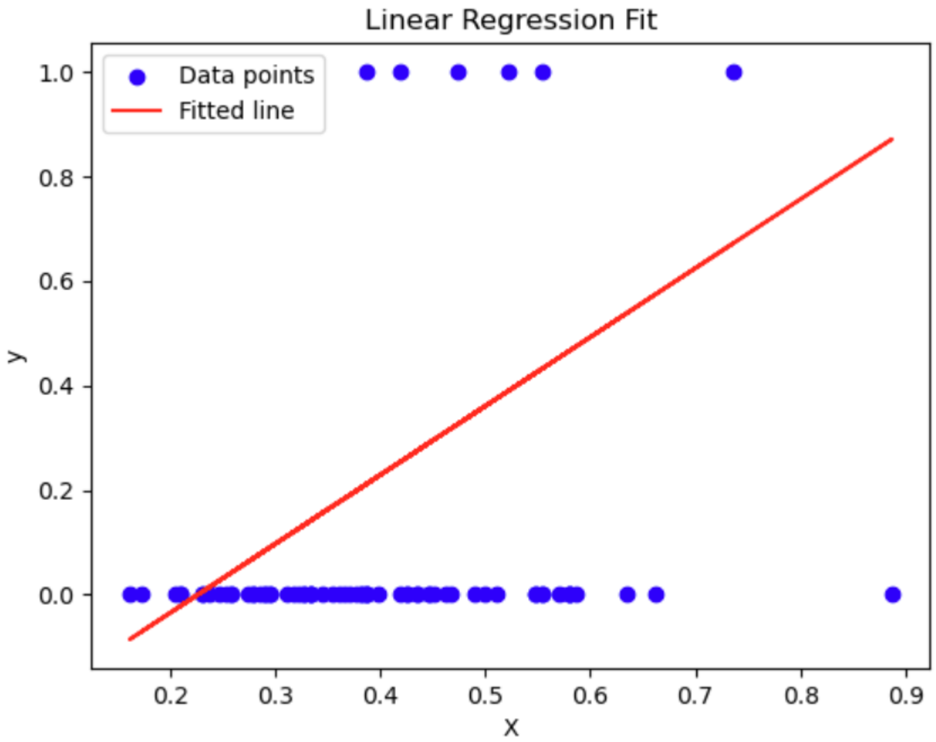
```
import matplotlib.pyplot as plt

plt.scatter(x_apache_test, y_test, color='blue', label='Data points')
plt.plot(x_apache_test, y_pred, color='red', label='Fitted line')

plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Fit')
plt.legend()

plt.show()
```

Plot the fitted line



From the plot we can see that, linear regression is not a good solution for our problem. Because the predicted values can go below 0 or above 1.

For classification tasks, **Logistic Regression** may be a better fit.

Logistic Regression

Logistic regression is a statistical method used for **binary classification** problems.

Logistic Function (Sigmoid Function):

The logistic function maps any real-valued number into a value between 0 and 1. The function is defined as:

y_pred we get from logistic regression

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

z = y_pred from linear regression

Logistic Regression

If we convert our y_pred using logistic function, we will have our y_pred values mapped between [0, 1].

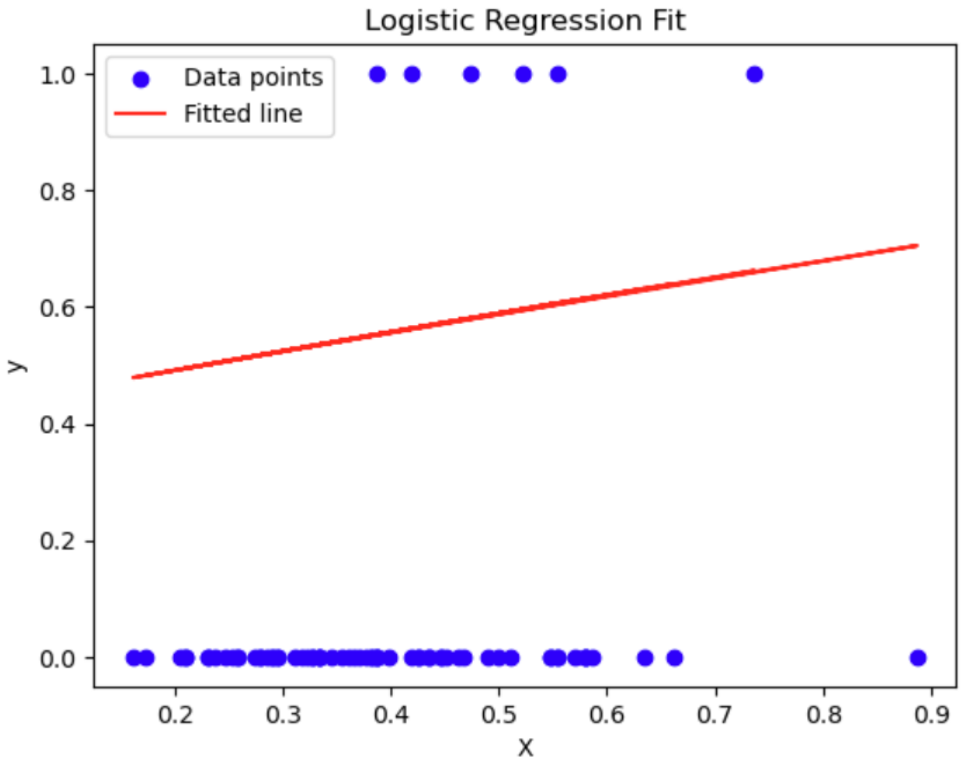
Code implementation:

```
import numpy as mp

y_pred_logistic = 1 / (1 + np.exp(-y_pred))
```

If we plot y_pred_logistic against x_apache_test, we will get:

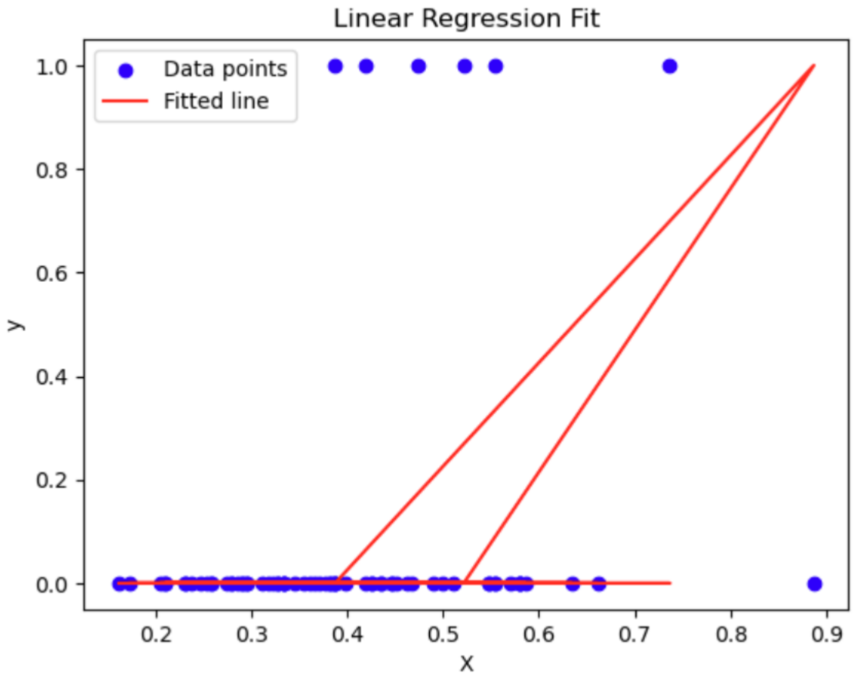
Logistic Regression



y now shows the probability that a given x belongs to class 1, which is “EXPIRED”.

Logistic Regression

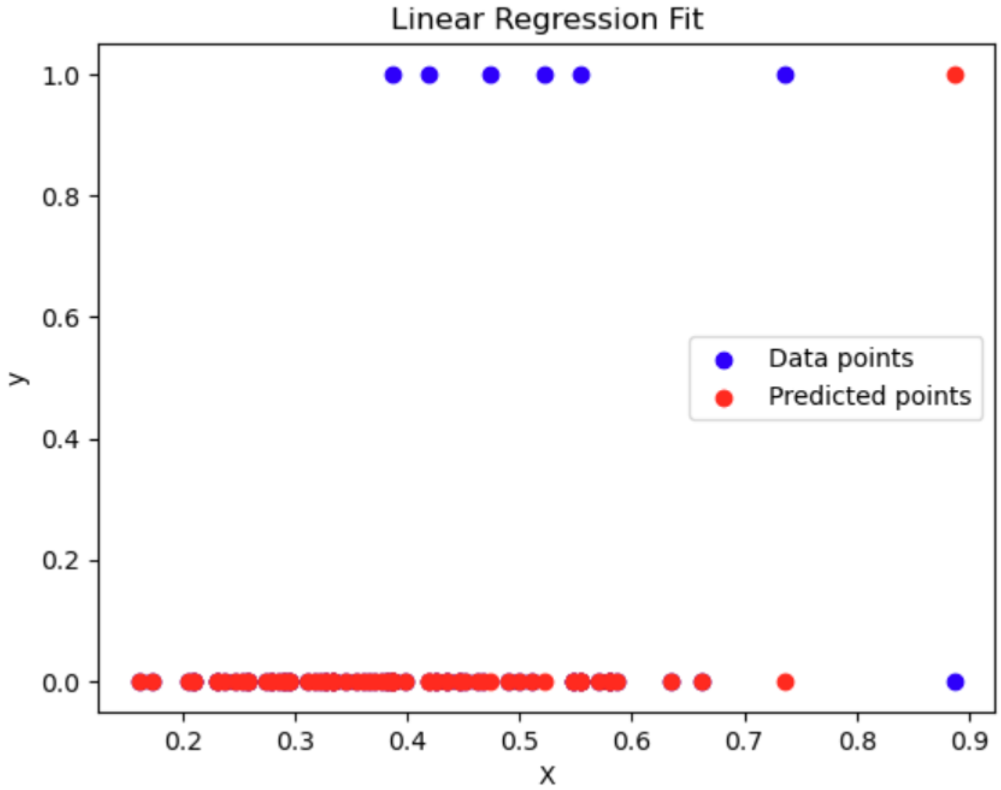
Now, let's try to fit our data to the "LogisticRegression()" function from scikit-learn, and plot the fitted line.



Why is the plot look like this?

Let's try plotting them in scatter plot.

Logistic Regression



What can you see from this graph?

If the red and blue dots are on top of each other, it means our prediction is correct.

But can we evaluate our model using other methods? Rather than plotting them every time.

Evaluation

Evaluating a machine learning model involves several steps and metrics, depending on the type of model and the problem being solved (e.g., regression, classification, clustering).

What is the type of our problem?

A classification problem!

There are several metrics for evaluating a classification model:

- Confusion Matrix, Accuracy, Precision, Recall, F1-Score, and ROC-AUC

Evaluation – Confusion Matrix

A table used to describe the performance of a classification model by showing the actual vs. predicted classifications.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Create a confusion matrix in python

We are using the logistic regression model we have fitted with “apachescore”.

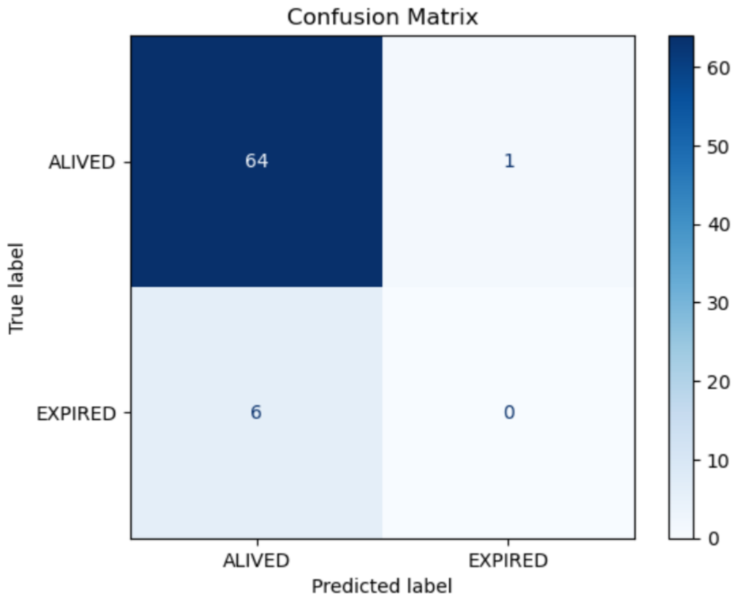
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred_logistic2, labels=[0, 1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["ALIVED", "EXPIRED"])

disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```

Create a confusion matrix in python

We are using the logistic regression model we have fitted with “apachescore”.



Evaluation - Accuracy

The ratio of correctly predicted instances to the total instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- TP: True Positives
- TN: True Negatives
- FP: False Positives
- FN: False Negatives

Accuracy the higher the better!

Evaluation - Precision

The ratio of correctly predicted positive observations to the total predicted positives.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- TP: True Positives
- TN: True Negatives
- FP: False Positives
- FN: False Negatives

High precision indicates a low false positive rate.

Evaluation – Recall/Sensitivity/True positive rate

The ratio of correctly predicted positive observations to all positive observations in the actual class.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- TP: True Positives
- TN: True Negatives
- FP: False Positives
- FN: False Negatives

High recall indicates a low false negative rate.

Evaluation – F1 Score

The harmonic mean of precision and recall, providing a balance between the two.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

An F1 score close to 1 suggests that the model performs well in identifying positive instances with few false positives and false negatives.

Calculate accuracy, precision, recall, specificity, and F1 score in python

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy = accuracy_score(y_test, y_pred_logistic2)
precision = precision_score(y_test, y_pred_logistic2)
recall = recall_score(y_test, y_pred_logistic2)
f1 = f1_score(y_test, y_pred_logistic2)
```

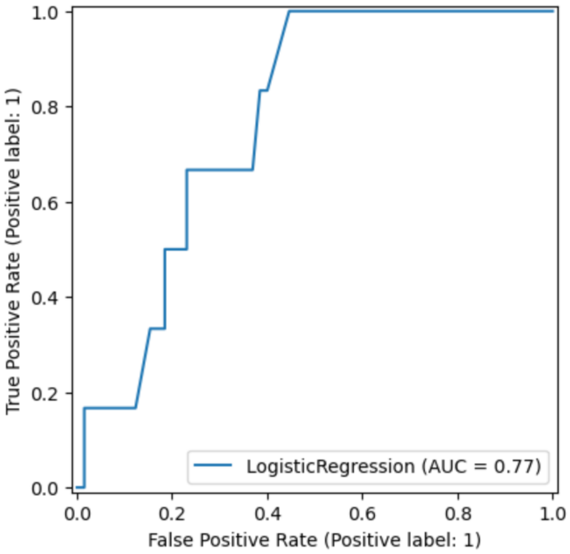
```
Accuracy: 0.9014
Precision: 0.0000
Recall: 0.0000
F1 Score: 0.0000
```

Why there are zeros?

Evaluation – ROC-AUC

Receiver Operating Characteristic – Area Under Curve

The **ROC curve** is a graphical representation of the **true positive rate** versus **false positive rate** at various threshold settings.

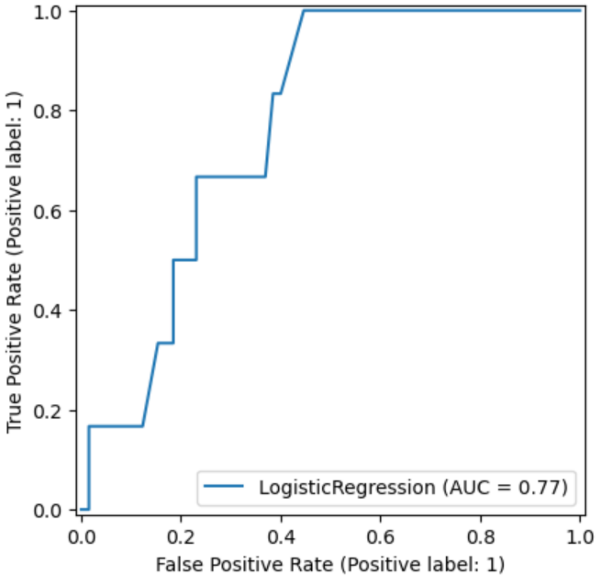


Threshold:

the cutoff point for assigning class labels based on the predicted probabilities.

Evaluation – ROC-AUC

AUC (Area Under the Curve) measures the area under the ROC curve. It provides a single value that summarises the performance of the model across all classification thresholds.



- **AUC = 1**: perfect model.
- **AUC = 0.5**: model performs no better than random guessing.
- **AUC < 0.5**: the model is worse than random guessing. This might indicate that the model is predicting the classes in reverse.

```
metrics.RocCurveDisplay.from_estimator(model, x_apache_test, y_test)
```

Multiple Linear Regression

Before, we only used “apachescore” to predict the mortality of our patients. Now, let’s try use more features.

The “LinearRegression()” function in scikit-learn can take multiple features in. The code implementation for a multiple linear regression is:

Multiple Linear Regression

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression

x = cohort_enc.drop('actualhospitalmortality_enc', axis=1)
y = cohort_enc['actualhospitalmortality_enc']

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=.7, random_state=42)

x_train = x_train.fillna(x_train.median())
x_test = x_test.fillna(x_train.median())

scaler = MinMaxScaler()
scaler.fit(x_train)
x_apache_train = scaler.transform(x_train)
x_apache_test = scaler.transform(x_test)

model = LinearRegression()
model = model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

Multiple Logistic Regression

To fit multiple features to a logistic regression in python is also easy.

It is similar to what we did for multiple linear regression.

You just need to change the function to “LogisticRegression()”.

Exercise

- Fit all features to the logistic regression model.

- Evaluate the fitted model by creating the confusion matrix.
- Evaluate the fitted model by calculating accuracy, precision, recall, and F1 score.
- Evaluate the fitted model by plotting ROC-AUC.

- Download our new data "Melbourne Housing Snapshot" and fit it into a linear regression model and make predictions for price.

References

- The Carpentries Incubator – Introduction to Machine Learning in Python
- OpenAI – ChatGPT
- scikit-learn – API Reference
- scikit-learn – User Guide
- Ankita Banerji – Gradient Descent in Linear Regression
- Kaggle – Melbourne Housing Snapshot

THANK YOU

Contact Us

Biological Data Science Institute
ANU College of Science

RN Robertson Building
46 Sullivans Creek Rd
Acton ACT 2601

jiajia.li1@anu.edu.au

Australian National University