IMPERIAL

SUPERVISED LEARNING

SURG70098 - SURGICAL DATA SCIENCE AND AI

STUART BOWYER

INTENDED LEARNING OUTCOMES

1. Understand what distinguishes supervised learning
2. Understand when to use regression and classification models
3. Be able to explain and apply basic regression and classification models to 'real-world' data
4. Be able to fairly and meaningfully evaluate the performance of different supervised learning models

SESSION OUTLINE

1. **Introduction to Supervised Machine Learning**
2. **Regression Models**
3. **Classification Models**
4. **Model Reliability and Reproducibility**
5. **Wrap Up**

MIMIC DATASET

The following code will load the datasets used in this lecture notes

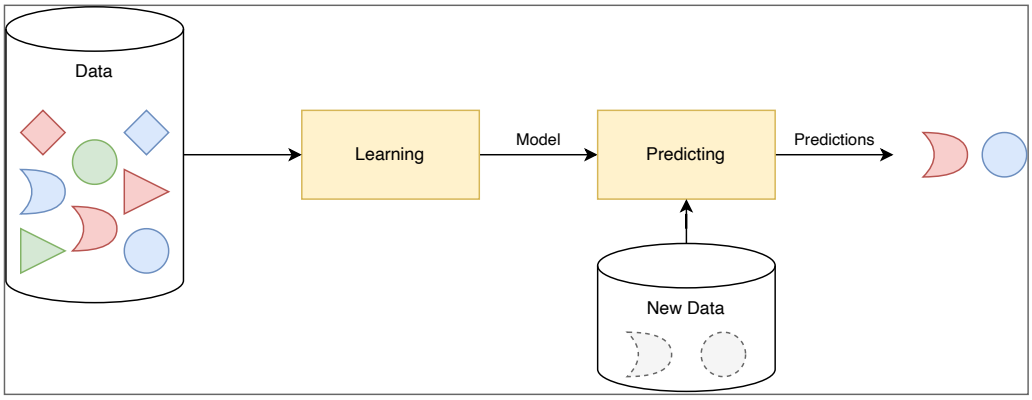
In [ ]:

```
%pip install pandas_gbq

import pandas as pd
import pandas_gbq

project_id = 'mimic-project-439314'  # @param {type:"string"}

df_day1_vitalsign = pandas_gbq.read_gbq("""
  SELECT *
  FROM `physionet-data.mimiciv_derived.first_day_vitalsign`
  LEFT JOIN (
    SELECT
      subject_id,
      stay_id,
      gender,
      race,
      admission_age,
      dod IS NOT NULL AS mortality
    FROM
      `physionet-data.mimiciv_derived.icustay_detail`
  )
  USING(subject_id, stay_id)
  LEFT JOIN (
    SELECT
      stay_id,
      AVG(weight) as weight
    FROM
      `physionet-data.mimiciv_derived.weight_durations`
    GROUP BY
      stay_id
  )
  USING(stay_id)
  LEFT JOIN (
    SELECT
      stay_id,
      CAST(AVG(height) AS FLOAT64) AS height
    FROM
      `physionet-data.mimiciv_derived.height`
    GROUP BY
      stay_id
  )
  USING(stay_id)
  WHERE heart_rate_mean IS NOT NULL
  LIMIT 10000
""", project_id=project_id)
```

INTRODUCTION TO SUPERVISED MACHINE LEARNING

WHAT IS SUPERVISED LEARNING

Creating a model based on a labelled set of data that can be used to predict something about future (unlabelled) data

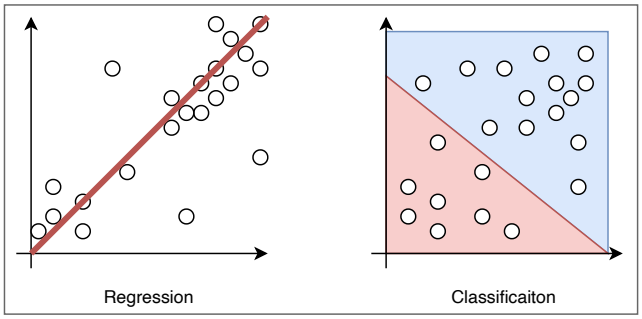


STAGES IN THE SUPERVISED LEARNING PROCESS

1. (Data Acquisition)
2. (Data Preparation)
3. Training
4. Validation/Evaluation
5. Deployment/Use

REGRESSION VS CLASSIFICATION

- **Regression:** Supervised learning that predicts a continuous value output based on one or more features
- **Classification:** Supervised learning that predicts a categorical value output based on one or more features



EXAMPLE USES OF SUPERVISED LEARNING

Regression	Classification
Predict surgical procedure duration (Linear Regression) - **https://doi.org/10.3389/fmed.2017.00085**	Predict risk of postoperative pul complications (Logistic Regressi **https://doi.org/10.1016/S258**
Predict FEV1 and FVC from chest x-ray (Deep NN) - **https://doi.org/10.1016/S2589-7500(24)00113-4**	Identify (and rule out) patients w myocardial infarction (Logistic R **https://doi.org/10.1016/S258**
Predicting length of stay after appendectomy surgery (Linear Regression) - **https://doi.org/10.1186/s12911-022-01884-9**	Post surgery mortality predictio **https://doi.org/10.3390/bioe**

REGRESSION MODELS

- Supervised learning that predicts a continuous value output based on one or more features
- Learns the relationship between feature values and an output value

LINEAR REGRESSION

Linear regression is a long standing statistical method for regression modelling It has the following equation simplified equation univariate model (i.e. the equation for a straight line)

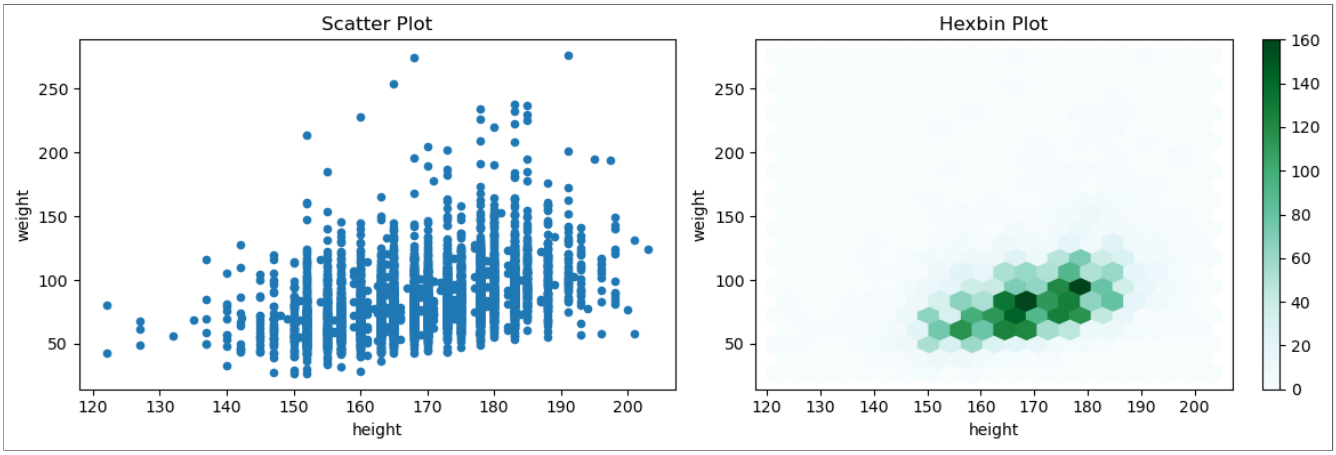
$$y = mx + c$$

where:

- y is the predicted output
- x is the feature input
- m is the slope of the relationship
- c is the relationship's intercept

WORKED EXAMPLE

- Assume we want to predict a patient's weight, given their height
- Here are scatter and density plots for the available data from MIMIC-IV



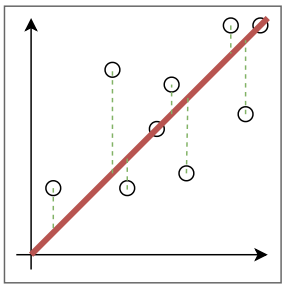
- Our linear regression equation becomes:

$$weight = m \times height + c$$

- **How do we calculate the gradient, m, and the intercept, c ?**

SOLVING FOR THE MODEL PARAMETERS

- While solving this equation for pair of points is trivial (GCSE maths)
- The challenge here is that for typical (imperfect) data, there is no single line that goes through all points
- The problem becomes an optimisation problem ...
  - i.e. find a line that minimises the prediction error
  - Specifically, minimises the sum of the square of all errors



ORDINARY LEAST SQUARES

The Python function we use to model, relies on a method called 'Ordinary Least Squares'
The cost function it tries to minimise is:

$$S(b) = \sum_{i=1}^{n} (y_i - x_i^T b)^2$$

where:

- x_i is the feature values of the i-th observation
- y_i is the i-th output values
- b is a candidate for the linear model parameters

Which can be solved using linear algebra:

$$S(b) = (X^T X)^{-1} X^T y$$

PYTHON IMPLEMENTATION IN `scikit-learn`

In [40]:

```
# Import the linear regression function from sklearn
from sklearn.linear_model import LinearRegression

# Prepare our data
#  - first remove any nan values as LR cannot work with them (if not already cleaned)
data = df_day1_vitalsign.dropna(subset=['height', 'weight'])
X = data[['height']]     # Note: Double brackets to maintain X as a DataFrame
Y = data['weight']       # Single bracket for a Series

# Create the model
model = LinearRegression()

# Train (fit) the model
model.fit(X,Y)

# Print model coefficients
print(f'Slope:     {model.coef_[0]}')
print(f'Intercept: {model.intercept_}')
```

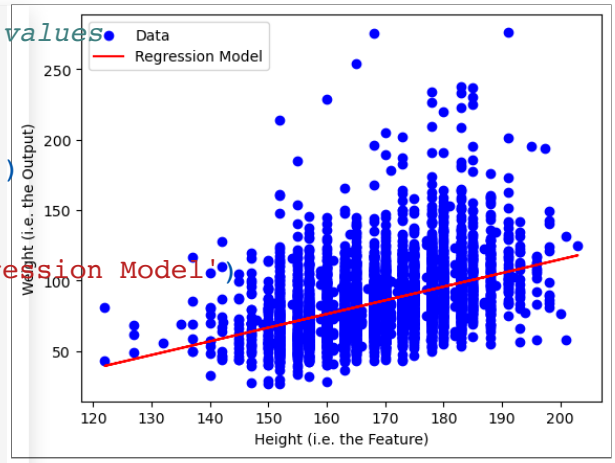
```
Slope:     0.9683731292903554
Intercept: -78.72472270813638
```

PREDICTING AND PLOTTING THE MODEL

- The trained model can be used to predict (new) values with the `.predict()` method

In [47]:

```
# Make predictions on the entire range of X values
Y_pred = model.predict(X)

# Plot the scatter plot of the data
plt.scatter(X, Y, color='blue', label='Data')

# Plot the regression line
plt.plot(X, Y_pred, color='red', label='Regression Model')

# Add labels and a legend
plt.xlabel('Height (i.e. the Feature)')
plt.ylabel('Weight (i.e. the Output)')
plt.legend()

# Show the plot
plt.show()
```



ASSUMPTIONS

- The model assumes the feature values and output value are **linearly** related
- The residual error (ϵ) is normally distributed
- Features are not multicolinear (i.e. highly correlated)

MULTIVARIATE LINEAR REGRESSION

- **Note** that we are have only considered univariate linear regression (i.e. model with a single feature)
- In practice, most of the time you will have several features and perform multivariate linear regression
- You will explore this in the tutorial exercises

MODEL EVALUATION IN REGRESSION

How do you know if your regression model is a good predictor?
How might you quantify the model's performance?

MEAN SQUARED ERROR AND ROOT MEAN SQUARED ERROR

- The 'mean squared error' (MSE) is a common metric for how 'good' a regression predictor is

$$mse = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- It is common to square root the MSE to get a metric with units that match the output of the model
- This is called the 'root mean squared error (RMSE)

$$rmse = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

PYTHON IMPLEMENTATION IN `SCIKIT-LEARN`

In [51]:

```
from sklearn.metrics import mean_squared_error, root_mean_squared_error

mse = mean_squared_error(Y, Y_pred)
print(f'Mean Squared Error:      {mse} kg2')

rmse = root_mean_squared_error(Y, Y_pred)
print(f'Root Mean Squared Error: {rmse} kg')
```

```
Mean Squared Error:      511.8104482250559 kg2
Root Mean Squared Error: 22.623228068183725 kg
```

R-SQUARED AND RESIDUAL ANALYSIS

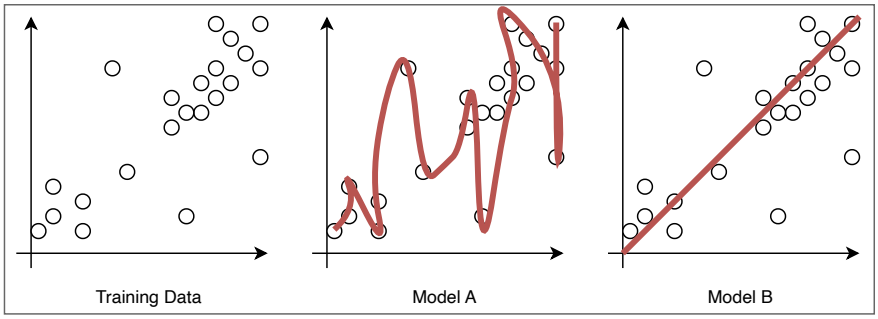
- R-squared is a metric the describes how much of the the variability in an output is 'described' by the model
  - i.e. a perfect model would predict all of the variability in an output, and would have an R-squared of 1
- Residual analysis is used to explore the residual error in a model and validate that it satisfies the models assumption
- There are items in the reading list for you to explore these concepts

MODEL VALIDATION

Can you see any possible issue with using metrics to validate the quality of a model's 'fit' in the way we did it?
Hint - what data are we using to train the model, and what data are we using to compute the metrics?

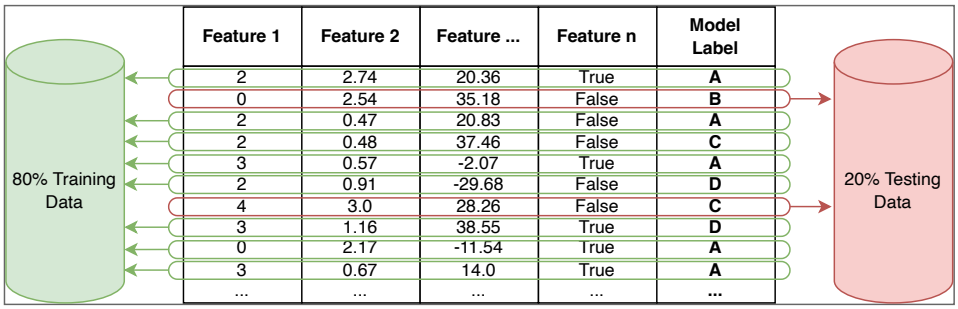
TRAINING AND TESTING ON THE SAME DATA SET

- A model validated on it's training data can:
  - Easily overfit the data and will not generalise to new inputs
  - Overestimate the model's performance
- What do you think is the solution to this?



TRAIN / TEST SPLITTING

- Solution is to split the dataset into:
  - **Training set** - to train the model
  - **Testing set** - to validate the trained model on unseen data
- To avoid bias due this split should be randomly assigned
- Typically splits will be 80/20 or 70/30 for Training/Testing



Feature 1	Feature 2	Feature ...	Feature n	Model Label
2	2.74	20.36	True	**A**
0	2.54	35.18	False	**B**
2	0.47	20.83	False	**A**
2	0.48	37.46	False	**C**
3	0.57	-2.07	True	**A**
2	0.91	-29.68	False	**D**
4	3.0	28.26	False	**C**
3	1.16	38.55	True	**D**
0	2.17	-11.54	True	**A**
3	0.67	14.0	True	**A**
...	...	...	...	**...**

PYTHON IMPLEMENTATION IN `SCIKIT-LEARN`

In [61]:

```
from sklearn.model_selection import train_test_split

# Split the data into train and test sets
#   - with 20% in the test
#   - with 123 as the randomisation seed
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=123

print(f'Samples in full data set:     {len(X)}')
print(f'Samples in training data set: {len(X_train)} ({100*len(X_train)/len(X)} %)')
print(f'Samples in testing data set:  {len(X_test)} ({100*len(X_test)/len(X)} %)')
```

```
Samples in full data set:     4379
Samples in training data set: 3503 (79.99543274720256 %)
Samples in testing data set:  876 (20.004567252797443 %)
```

TRAIN, PREDICT AND EVALUATE

In [63]:

```
# Train the model on the training set
model.fit(X_train,Y_train)

# Use the model to predict values for the testing set
Y_pred = model.predict(X_test)

# Evaluate the metrics
mse = mean_squared_error(Y_test, Y_pred)
print(f'Mean Squared Error:      {mse} kg2')

rmse = root_mean_squared_error(Y_test, Y_pred)
print(f'Root Mean Squared Error: {rmse} kg')
```

```
Mean Squared Error:      532.7518734142269 kg2
Root Mean Squared Error: 23.08141835793951 kg
```

These are slightly higher than they were for the self train test example earlier

DOES TRAIN/TEST SPLITTING GUARANTEE NOT TO OVERFIT THE DATA?

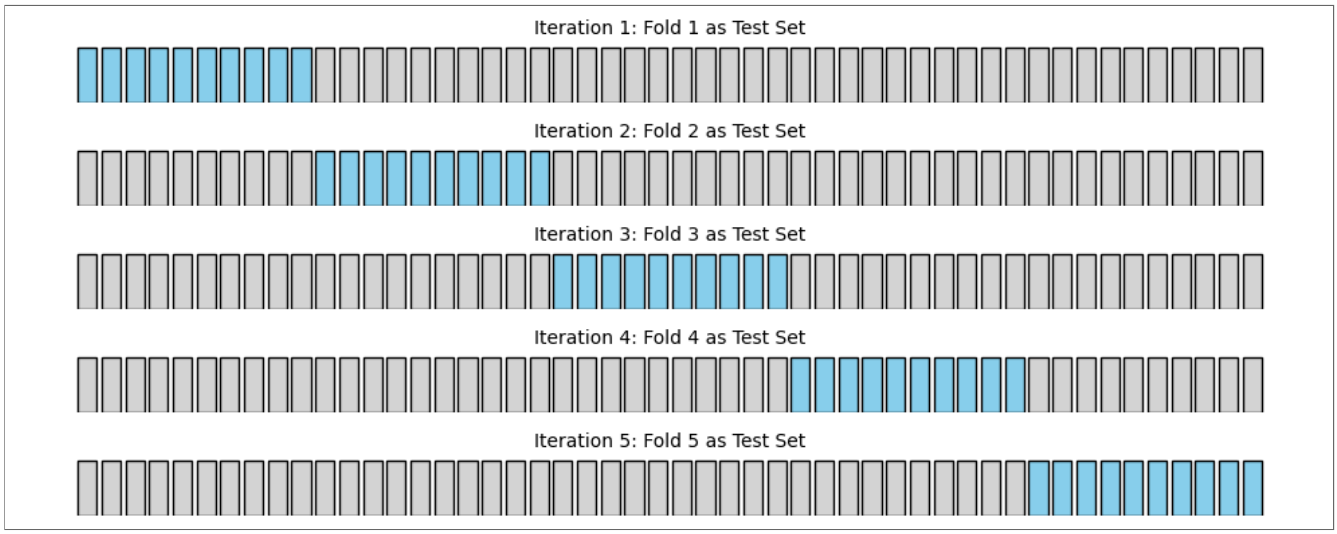
IF NOT, WHAT COULD WE DO TO MITIGATE THIS?

CROSS VALIDATION

- To mitigate the risk of non-representative splits in train/test data we can use 'cross validation'
- Simplistically, cross validation involves repeatedly train/testing the data on several different segregations of the data

K-FOLD CROSS VALIDATION

- k-fold cross validation is a very widely uses cross validation approach
- The method is:
  - Split your data into k different sub sets - called folds
  - Train your model on all but one fold and test/validate on the remaining fold
  - Repeat this process k times so that each fold is used as the testing set once
  - Combine your validation metrics



PYTHON IMPLEMENTATION IN `SCIKIT-LEARN`

In [83]:

```
from sklearn.model_selection import cross_val_score

# Perform the cross validation
#   - pass it the model you want to validate (e.g. the LinearRegression())
#   - pass the full feature and output datasets
#   - specify the number of folds (e.g. 5)
#   - specify the scoring method you want (e.g. RMSE)
scores = cross_val_score(model, X, Y, cv=5, scoring='neg_root_mean_squared_error')

print(f'Cross-Validation Scores: {scores}')
print(f'Mean Score:              {scores.mean()}')
```

```
Cross-Validation Scores: [-27.34798278 -22.96258907 -22.01208489 -20.184
02581 -19.84045977]
Mean Score:              -22.469428464632717
```

Note - `scikit-learn` generally tries to maximise a metric/score, therefore we are using the NEGATIVE of the RMSE

ALTERNATIVE APPROACHES

- Stratified k-fold Cross-Validation
  - Used for imbalanced classification problems (e.g. trying to classify mortality where the rate is low)
  - Modified k-fold where folds are adjusted to ensure each fold has a proportional representation of classes
- Leave-One-Out Cross-Validation (LOOCV)
  - Special case where k is the number of samples in the dataset
  - Therefore, each sample is used once as a test set, and the rest as the training set

COMMENTS ON VALIDATION

- Always ensure that you are splitting your data fairly and independently to prevent training data leakage
- Prospective multi-centre validation is the gold-standard, that very few models achieve

REGULARISATION

What is Regularisation?

- Regularisation is a technique used in regression to penalise models with many large coefficients
- Effectively, it tries to make the model more efficient with its features
- Reducing the complexity of the model reduces the likelihood of overfitting
- This can help find the model a balance of underfitting and overfitting

Common Methods

- Lasso Regression (L1 Regularisation) - reduces the magnitude of coefficients
- Ridge Regression (L2 Regularisation) - reduces the sum of squared coefficients
- Elastic Net Regression - combines Lasso and Ridge regression

Further reading on this is included in the reading list

OTHER REGRESSION METHODS

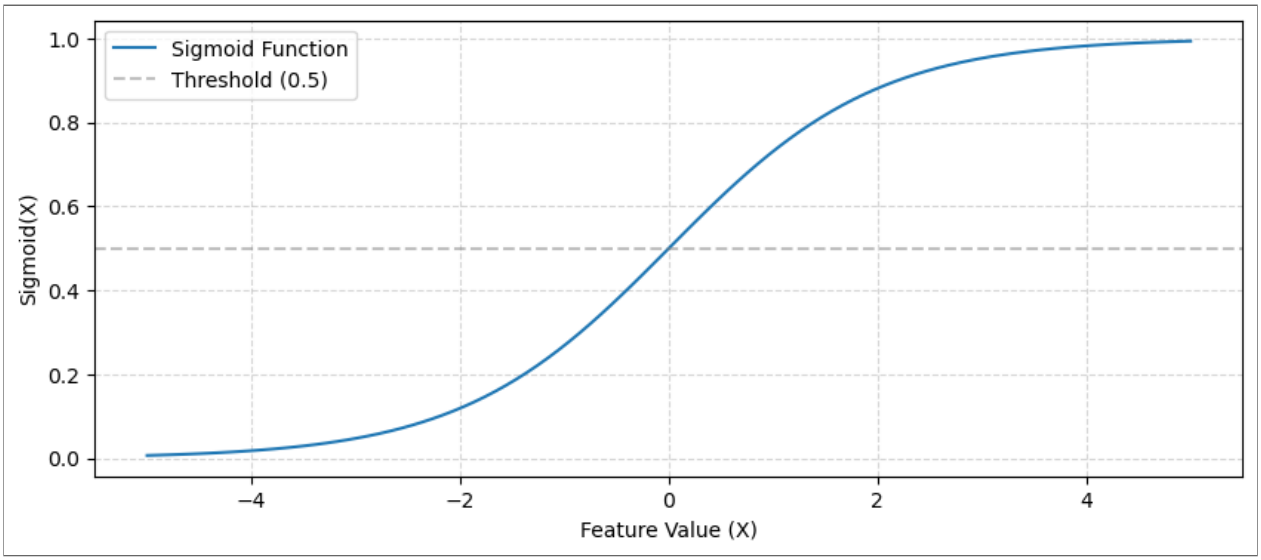
- There are several other models that can be used for regression
- If you want to model something with regression that is non-linear, you should explore these
  - Polynomial regression - where the relationship is modelled as an nth degree polynomial
  - Neural networks - which we will cover next week
  - Decision tree/random forest
  - Bayesian regression

CLASSIFICATION MODELS

- Supervised learning that predicts a discrete/categorical value output based on one or more features
- Learns the relationship between feature values and an output category

LOGISTIC REGRESSION

- A classification model used to predict the probability of a **binary** outcome
- It is based on Linear Regression but uses the logistic function to limit the output between 0 and 1
- By applying a probability threshold (typically 0.5), the logistic function output can give the binary prediction



PYTHON IMPLEMENTATION IN `scikit-learn`

In [ ]:

```
from sklearn.linear_model import LogisticRegression

# # Prepare our data
# #  - first remove any nan values as LR cannot work with them (if not already cleaned)
data = df_day1_vitalsign.dropna(subset=['admission_age', 'mortality'])
X = data[['admission_age']]
Y = data['mortality']

# Create the model
model = LogisticRegression()

# Train (fit) the model
model.fit(X, Y)
```

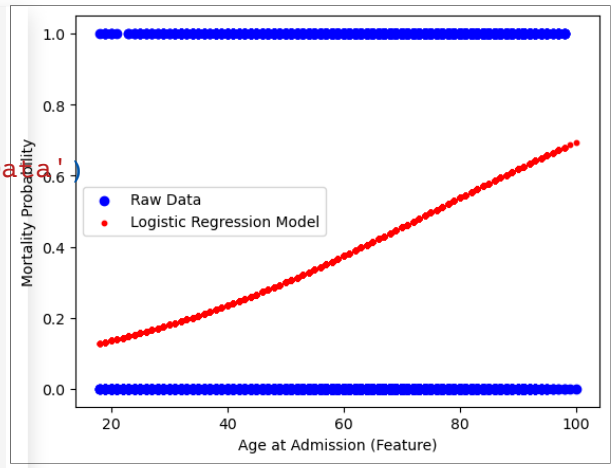
Note - the ceofficients of a logistic regression model cannot be so easily interpreted as they were for linear regression

PREDICTING AND PLOTTING THE MODEL

- The trained model can be used to predict (new) probablities with the `.predict_proba()` method
- Or predict (new) classifications with the `.predict()` method

In [148]:

```
# Calculate the predicted probabilities
Y_prob = model.predict_proba(X)[:, 1]

# Plot the age vs mortality points
plt.scatter(X, Y, color='blue', label='Raw Data')

# Plot the logistic regression model
plt.scatter(
    X, Y_prob, color='red', s=10,
    label='Logistic Regression Model'
)

# Formatting
plt.xlabel('Age at Admission (Feature)')
plt.ylabel('Mortality Probability')
plt.legend()
plt.show()
```



MODEL EVALUATION IN CLASSIFICATION

Unlike in regression, a classification is either 100% correct or wrong; therefore, measuring the residual error is less helpful

What method would you use to evaluate a classification model?

CONFUSION MATRIX

- The confusion matrix is simply a tabulation of the counts for predicted against actual classifications

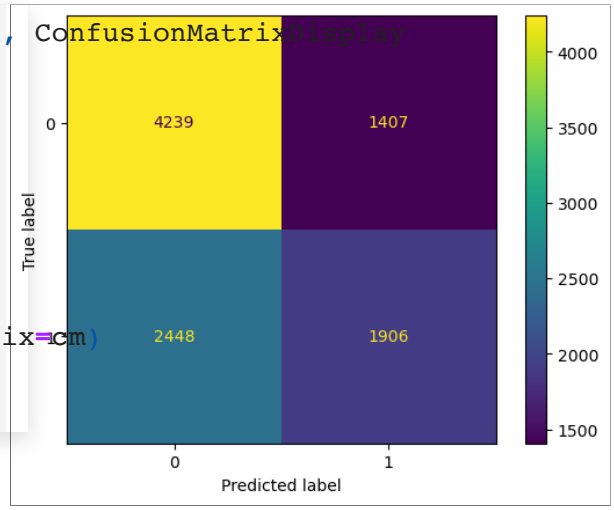
		Predicted Mortality	
		True	False
Actual Mortality	True	19	24
	False	14	42

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

PYTHON IMPLEMENTATION IN `SCIKIT-LEARN`

In [153]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Predict the classes
Y_pred = model.predict(X)

# Compute the confusion matrix
cm = confusion_matrix(Y, Y_pred)

# Visualise the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```



METRICS

To compare and assess classifier predictions, the confusion matrix can be converted into a range of metrics (there are many more)

ACCURACY

- **Definition**: Proportion of correctly classified instances over the total number of instances
- **Formula**: (TP + TN) / (TP + TN + FP + FN)

PRECISION (POSITIVE PREDICTIVE VALUE)

- **Definition**: Proportion of true positives out of all predicted positives
- **Formula**: TP / (TP + FP)

RECALL (SENSITIVITY/TRUE POSITIVE RATE)

- **Definition**: Proportion of true positives out of all actual positives
- **Formula**: TP / (TP + FN)

F1 SCORE

- **Definition**: Harmonic mean of precision and recall, useful for imbalanced datasets
- **Formula**: 2 * (Precision * Recall) / (Precision + Recall)

PYTHON IMPLEMENTATION IN `SCIKIT-LEARN`

In [159]:

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

accuracy = accuracy_score(Y, Y_pred)
precision = precision_score(Y, Y_pred)
recall = recall_score(Y, Y_pred)
f1 = f1_score(Y, Y_pred)

print(f"Accuracy:  {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall:    {recall:.2f}")
print(f"F1 Score:  {f1:.2f}")
```

```
Accuracy:  0.61
Precision: 0.58
Recall:    0.44
F1 Score:  0.50
```

WHICH OF THESE METRICS IS THE BEST?

FOR EACH, CAN YOU THINK OF ANY OCCURRENCES IN WHICH THEY WOULD GIVE MISLEADING RESULTS?

ROC CURVES

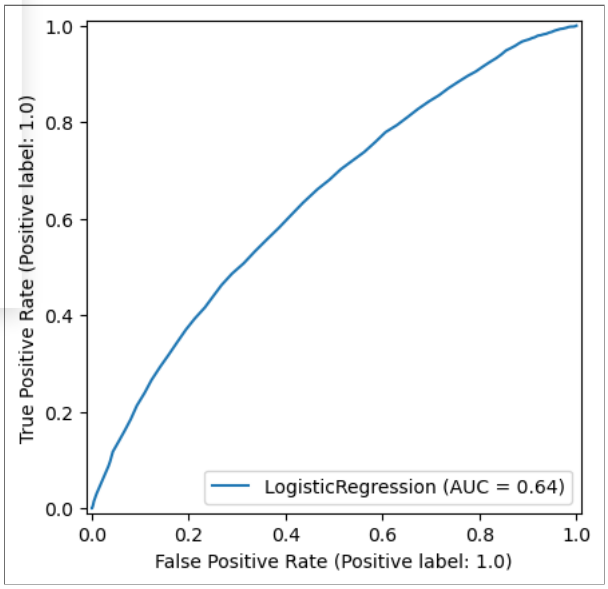
- The challenges of imbalanced data, and the ability to 'hack' metrics by adjusting a model's threshold leads to the use of ROC curves
- This is a plot of the 'true positive rate' (TRP) against the 'false positive rate' (FPR) for varying thresholds
  - TPR = TP / (TP + FN)
  - FPR = FP / (FP + TN)
- The area under the ROC curve (ROC-AUC), is commonly used as an evaluation metric
  - ROC AUC close to 1 identifies a good classifier
  - ROC AUC close to 0.5 indicates a poor classifier (effectively random guessing)

PYTHON IMPLEMENTATION IN `SCIKIT-LEARN`

In [171]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import RocCurveDisplay

# Calculate the ROC-AUC
auc_score = roc_auc_score(Y, Y_prob)
print(f"ROC AUC:  {auc_score:.2f}")

# Plot the ROC curve
RocCurveDisplay.from_estimator(model, X, Y)
plt.show()
```

```
ROC AUC:  0.64
```



- In the bottom left corner, there is a low threshold, thus no patients are classified as True
- In the top right corner, there is a high threshold, thus all patients are classified as True

OTHER CLASSIFICATION MODELS

So far we have only looked at logistic regression, there are many other classification models
The most popular, Neural Networks, we will explore next week

SUPPORT VECTOR MACHINES (SVM)

WHAT IS IT?

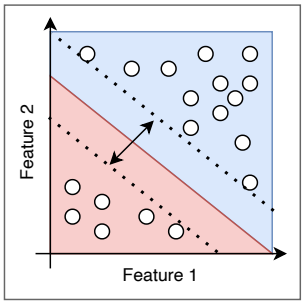
- Training the model optimises the placement for (hyper-) planes that separate the classes
- The model aims to separate the classes as far as possible

WHY USE IT?

- Focus on separation means the model can generalise better than SVM
- Non-linear boundaries can be class modelled by transforming the data into a higher dimensional spaces

SKLEARN FUNCTION

- `sklearn.svm.SVC`



DECISION TREES

WHAT IS IT?

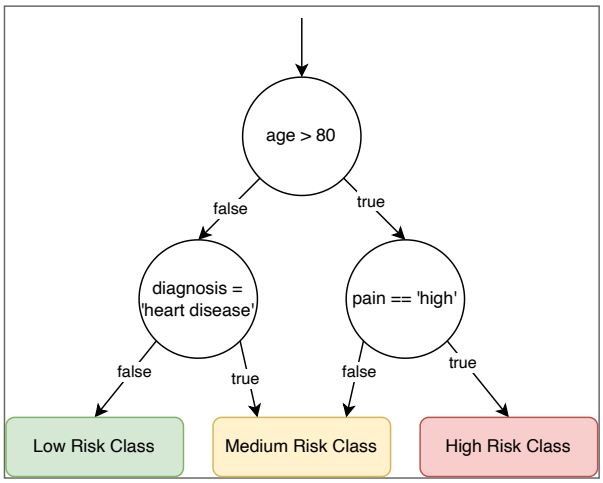
- Decision trees are a tree-like structure where data is split into branches based on feature values, leading to classifications at the leaves

WHY USE IT?

- Can model non-linear features
- Output model is human interpretable

SKLEARN FUNCTION

- `sklearn.tree.DecisionTreeClassifier`



K-NEAREST NEIGHBORS (KNN)

WHAT IS IT?

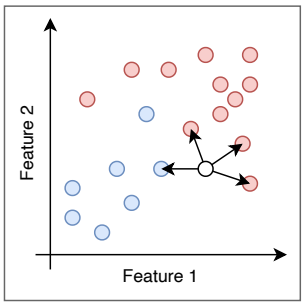
- This classifier predicts an inputs class by looking at the 'k' closest data points (neighbors) in the training data

WHY USE IT?

- Can model non-linear features
- Simple to understand and implement

SKLEARN FUNCTION

- `sklearn.neighbors.KNeighborsClassifier`



TUTORIAL EXERCISES

EXERCISE 4.1 - MULTIVARIATE LINEAR REGRESSION

In the linear regression example in this lecture we only considered a single feature to predict weight. Univariate models are very simple and unlikely to have good performance for problems of any complexity.

Write some code to improve the performance of your weight prediction based on any other (one or two only) columns in the `df_day1_vitalsign` data set that you hypothesise will improve the performance. You can engineer a new feature if you think it necessary.

Validate and compare the performance of your multivariate linear regression model to the univariate one created in the lecture.

EXERCISE 4.2 - MULTIVARIATE CLASSIFICATION AND VALIDATION

For simplicity, we used the same data set to train and test the classification models we built in this lecture. **This is not good practice**. You should always use Train/Test splitting and/or cross validation, as was demonstrated for regression.
The model we developed was also very simple and only used age to predict mortality. As in exercise 4.1, you should use your clinical knowledge to hypothesise about several other features from the `df_day1_vitalsign` data set that might impact mortality. You should then add these to the logisitic regression classifier and perform a full validation of its performance.
Again, you should compare the performance of your multivariate model to the univariate model we demonstrated in the lecture.

EXERCISE 4.3 - CLASSIFICATION MODEL COMPARISON

Once you have selected your best combination of features for predicting mortality in exercise 4.2, you should compare the performance of your features on logistic regression, SVM, decision tree, and KNN classifiers.
Which of the four models we have looked at gives the best performance?
You should do additional reading around these methods to try and understand why each of them performs the way they do and how they are helpful.

BONUS EXERCISE 4.4 - DATA NORMALISATION AND STANDARDISATION

Last lecture we looked at data normalisation and standardisation. Try applying one of these methods to your features and compare the performance of your classifiers with and without it - particularly KNN.
What would explain the difference in performance with normalisation or standardisation?

WRAP UP

- We have focused in this lecture on the end-to-end modelling process encouraging good practice in validation
- To fully understand the models you will need to try applying them yourself

BEFORE NEXT SESSION

- Complete the tutorial exercises

NEW MATERIAL FROM THE READING LIST

- R-squared and Residual Analysis
- Regularisation methods
- Detailed look at SVM, decision tree, and KNN classifiers