# Week04 - Summary

## Machine Learning

- Creating and using models that are learnt from data
  - Predicting whether an email is spam or not
  - Predicting whether a credit car transaction is fraudulent
  - Predicting which Football team will win the Champions League
- We'll focus on an *overview* of **unsupervised** and **supervised** machine learning techniques:
  - Clustering
  - Simple Linear Regression
  - Multiple Linear Regression
  - Logistic Regression
  - Classification/Decision Trees

## Unsupervised ML: Clustering

- Given a set of n objects, group into k coherent clusters
- Distance function that specifies the “closeness” of two objects
- Fundamental problem - divide objects into cluster so that points in different clusters are far apart

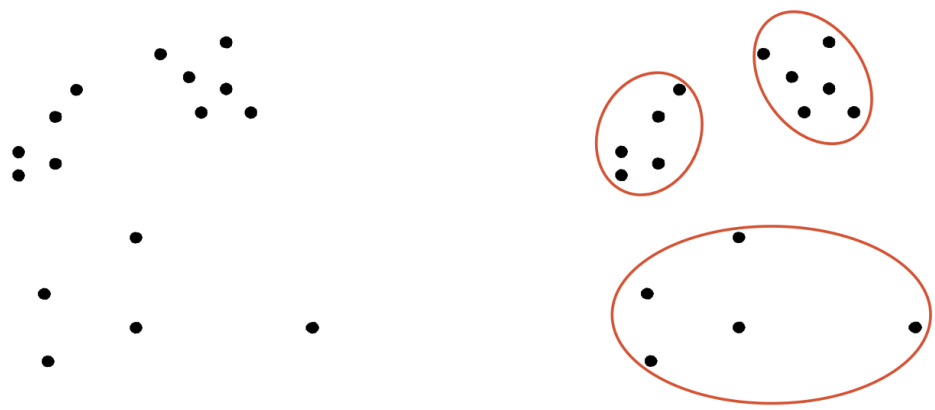
### Clustering for Summarisation

- Reduces the size of large datasets
- Summarise data before further analyssi

- Vector quantisation for e.g., images, audio, video

## Types of Clusterings

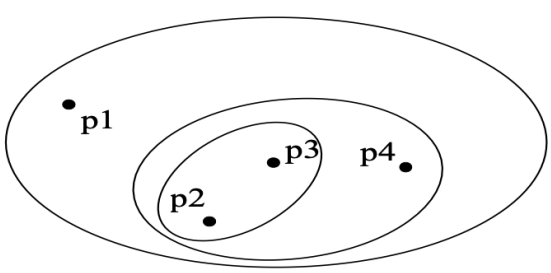
- A clustering is a set of clusters
- Important distinction between **hierarchical** and **partitioned** sets of clusters
- ***Partitional clustering***
    - A division data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset



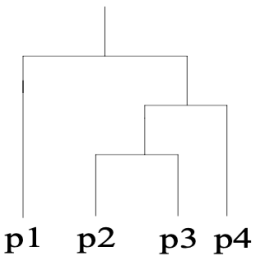
Original Points

A Partitional Clustering

- ***Hierarchical clustering***
    - A set of nested clusters organised as a hierarchical tree




Hierarchical clustering




Dendrogram

## k-Means Clustering

- Partitional clustering approach
- Each cluster is associated with a centroid (centre point)
- Each point is assigned to the cluster with the closest centroid
- Number of clusters, k, must be specified
- The basic algorithm is very simple:

```
Select K points as the initial centroids.
repeat
    Form K clusters by assigning all points to the closest centroid.
    Recompute the centroid of each cluster.
until The centroids don't change
```

## Evaluating Clustering

- Numerical measures that are applied to judge various aspects of cluster validity, are classified into he following three types:
  - **External Index Measure:** Measure the extent to which cluster labels match externally supplied class labels (e.g., v-measure)
  - **Internal Index Measure:** Measure the goodness of a clustering structure without respect to external information (e.g., SSE)
  - **Relative Index Measure:** Compare two different clusterings or clusters (often an external or internal index is used)

## External Evaluation Measures

- **Homogeneity** ranges from 0 to 1, measuring whether clusters contain data points that are part of a single class (analogous to precision)
- **Completeness** ranges from 0 to 1, measuring whether classes contain data points that are part of a single cluster (analogous to recall)

- **V-measure** is the harmonic mean of homogeneity and completeness (analogous to F1 score)

### Internal Measure: Sum of Squares Error (SSE)

- For each point, the error is the distance to the nearest cluster
- To get SSE, we square these errors and sum them:

$$\boxed{SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist^2(m_i, x)}$$

k is the number of clusters, x is the data point in cluster C, and $m_i$ is the representative point (mean) for cluster $C_i$

### Internal Measure: Silhouette Coefficient

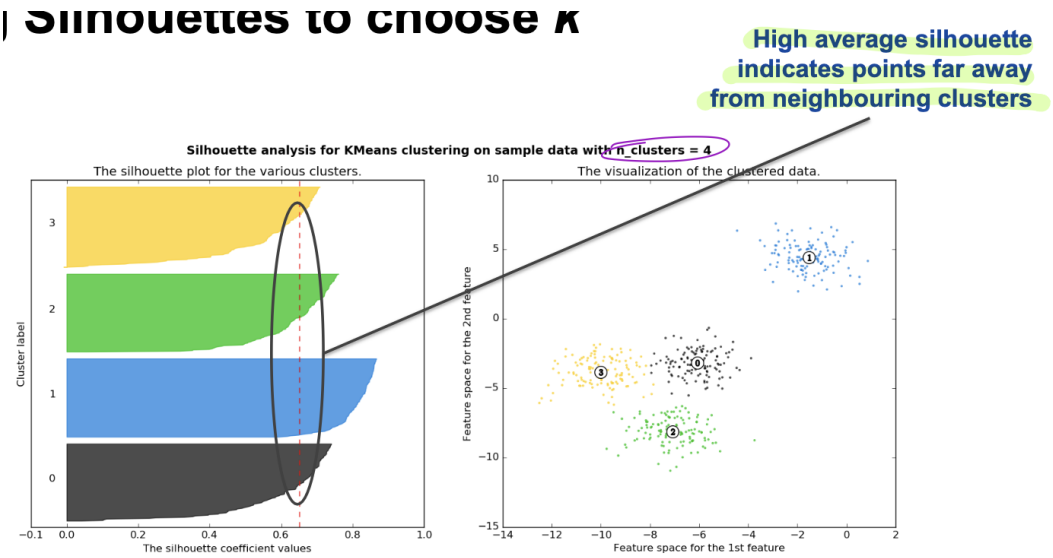
- For an individual point I
  - Calculate a = average distance of i to points in its cluster
  - Calculate b = average distance of I to points in the next nearest cluster
  - The silhouette coefficient for point is then given by

  s = 1 - a/b if a < b, (or s = b/a - 1, if a ≥ b, not the usual case)

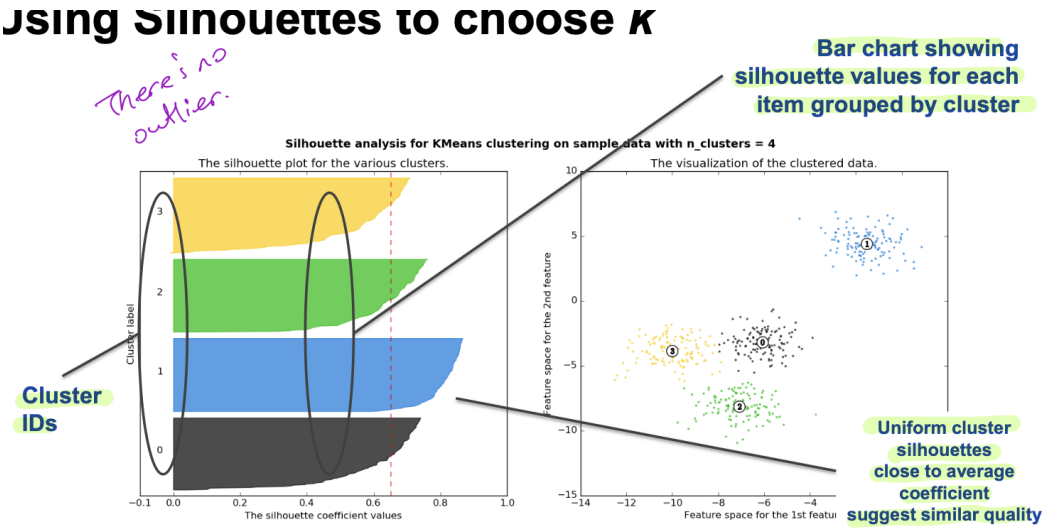
  - The closer to 1, the better
- Silhouette coefficient for dataset is average across all I

### Using Silhouettes to choose K

## Silhouettes to choose $K$




## Using Silhouettes to choose $K$




## Data Normalisation and Curse of Dimensionality

- **Dimensionality Reduction** / choice or projection of dimension
  - closely related to choice of **distance metric**
  - we used Euclidean Metric so far ($L_2$-Norm)
  - but others possible too, e.g., Manhattan Distance ($L_1$-Norm)

## Data Normalisation (Feature Scaling)

- normalise the range of independent variables or features of the original data:
  - **Rescaling (min-max normalisation)**
    - linear transformation to rescale the range of features to the range [0, 1]
  - **Mean normalisation**
  - **Standardisation (Z-score Normalisation)**
    - widely used for normalisation in machine learning
  - **Scaling to unit length**
    - scale the components of a feature vector by dividing each component by the Euclidean length of the vector
  - **Log Transformation**

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

min/max normalisation

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

mean normalisation

$$\boxed{x' = \frac{x - avg(x)}{\text{stdev}(x)}}$$

Z-score normalisation

$$\boxed{x' = \frac{x}{||\, x \,||}}$$

scaling

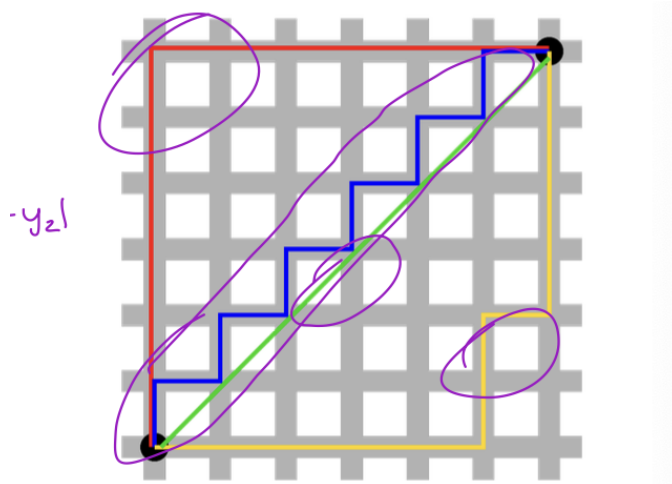
$$\boxed{x' = \log(x)}$$

Log Transformation normalisation

## Euclidean vs. Non-Euclidean Distance Measures

- Euclidean

- $L_2$ norm (Euclidean distance)
  - Square root of sum of squared differences
- $L_1$ norm (Manhattan distance)
  - Sum of absolute differences
- $L_\infty$ norm (Infinity norm)
  - Maximum of absolute differences




Manhattan versus Euclidean distance. The red, blue, and yellow lines all have the same length, whereas the green line is the $L_2$ norm distance.

- Non-Euclidean
  - Jacquard distance
    - Similarity between two sets
    - 1 - Jacquard similarity
  - Cosine distance
    - Alignment of two vectors
    - 1 - Cosine similarity
  - Edit distance

        - Number of inserts & deletes to change a string into another

### The Curse of Dimensionality

- From a theoretical point of view, increasing the number of features should lead to better performance
- In practice, the inclusion of more features leads to worse performance (i.e., curse of dimensionality)
- The more dimensions, the less difference in the distances between data points
    - This complicates clustering or k-nearest neighbour classification
- Dimensionality reduction can improve the performance of machine learning

### Dimensionality Reduction

- Transformation of data from high-dimensional into a low-dimensional space
    - low-dimensional representation retains the meaningful properties of the original data
- Feature Selection
    - Best subset selection (NP problem)
    - Forward/backward stepwise
- Dimension Reduction (Feature Projection)
    - Principal Components Analysis (PCA)
        - Applied only on predictor variables i.e., unsupervised
    - Partial Least Squares
        - chooses predictors related to predicted variables i.e., supervised

## Supervised Machine Learning

Complex decision making: $\boxed{Object \rightarrow Outcome}$

Entity → Category
Entity → Popularity
Entity → IsChainElement

input/independent variable → $\boxed{X \rightarrow Y}$ ← prediction (response/dependent variable). Can be qualitative/quantitative (classification/regression).

classifier

$$\boxed{\vec{X} = (x_0, \dots, x_d) \rightarrow Y}$$

object encoded with features (think DB attributes/ OO member fields of primitive types)
$d$ is the feature dimensionality.

We may know the relation for certain values of $X$ and $Y$:

$$\boxed{(\vec{x}, y)}$$

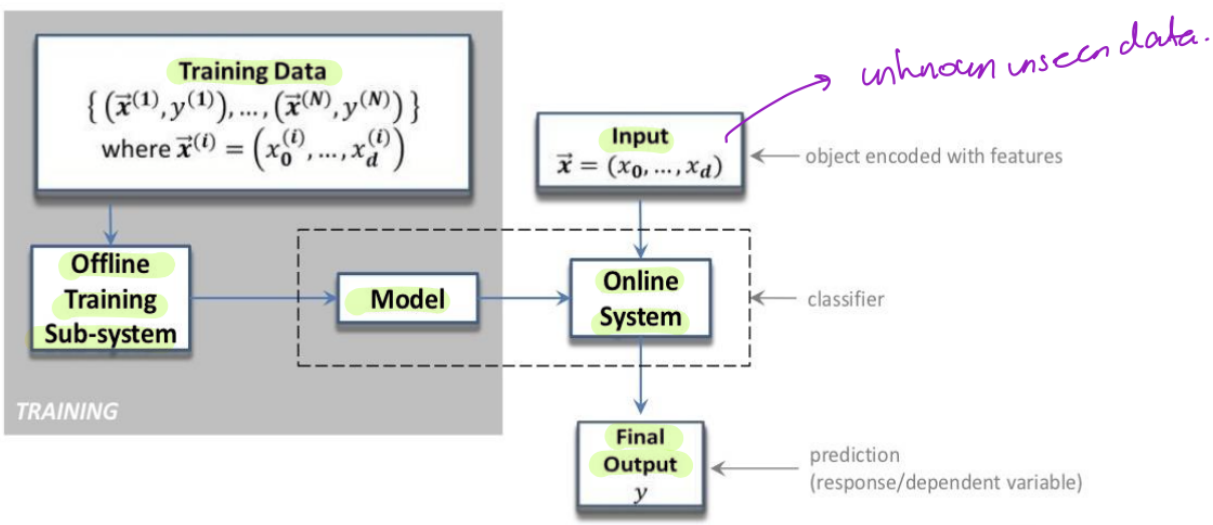
In fact, we may know the relation for many $\vec{x}$s and $y$s:

$$\boxed{\{(\vec{x}^{(1)}, y^{(1)}), \dots, (\vec{x}^{(N)}, y^{(N)})\}}$$

The $i$-th $x$ is: $\vec{x}^{(i)} = \left(x_0^{(i)}, \dots, x_d^{(i)}\right)$

38

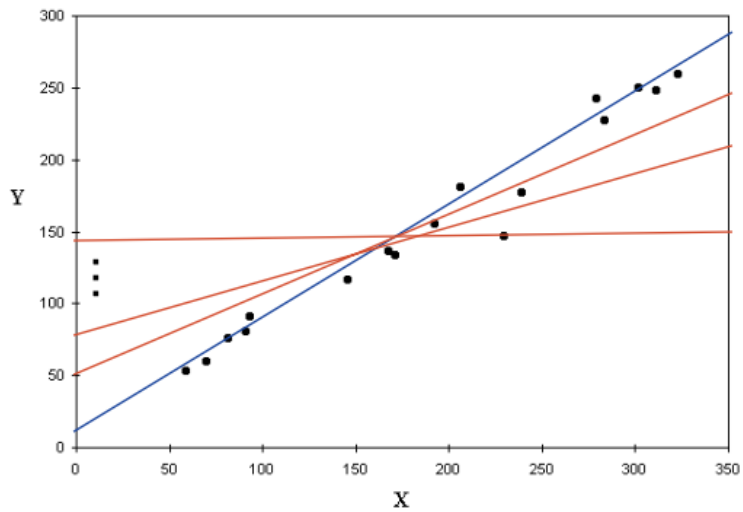
$X \rightarrow Y$ $\qquad$ $f(X) = Y$




## Linear Regression

### "Least Squares" Regression

- Intuitively: Find the line that minimises the squared errors (i.e., has the smallest residuals)




- More formally, our goal is:

  Find $\alpha$, $\beta$ **that minimises:**

$$\sum \varepsilon_i^2 = \sum (y_i - \hat{y}_i)^2 = \sum (y_i - (\alpha + \beta x_i))^2$$

## Fitting SLR: Least Squares

- We can resolve the previous least-square equation o:

$$\boxed{\alpha = \bar{y} - \beta \bar{x}} \quad \text{with} \quad \boxed{\beta = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{\text{corr}(x,y)\ \text{stdev}(y)}{\text{stdev}(x)}}$$

```
def least_squares_fit(x,y):
  """give training values for x and y,
  find the least-squares values of alpha and beta
  """
  # Slope of standardised data points with mean 0 and stdev 1
  beta = correlation(x, y) * standard_deviation(y) / standard_deviation(x)
  # Adjust slope for variation in Y and X
  alpha = mean(y) - beta * mean(x) # Intercept is the difference between means of
```

```
    # observed and predicted y
    return alpha, beta
```

### Coefficient of Determination ($R^2$)

- R^2: ratio of **explained variation in y** to **total variation in y**

$$\boxed{R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} = \frac{SST - SSE}{SST}}$$

- Ranges from 0 to 1, with higher values indicating better fit
- Conveys goodness of fit but not problem

(SSE: Sum of Squares Error - sum of the residuals;

SST: Sum of Squares Total - distance of each $y_i$ from the mean)

### Standard Error (S)

- Square root of the sum of squared errors divided by N

$$S = \sqrt{\frac{SSE}{N}}$$

- Measure of the prediction accuracy
- Expressed in units of the response variable

### Calculating a Prediction Interval from S

- Prediction interval: range that should contain the response value of a new observation
- If sample size is large enough then useful rule-of-thumb:

approximately 95% of predictions should fall within

$$\hat{y}_i \pm 2 * S$$

prediction interval, not confidence!

There are functions and libraries available for us to compute this. However, this is just a rule of thumb

## Multiple Linear Regression

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
_ = lm.fit(X_train, Y_train)
Y_test = lm.predict(X_test) # we predict on the test dataset, but fit Lin Reg on training
```

## Assessing Fit and Standard Error

- We can use the same methods than with Simple Linear Regression to assess goodness-of-fit:

  R-squared (R^2), and

  the standard error S of the regression

```
# We use the score method to get r-squared
print('\nR-squared:', lm.score(X_train, Y_train))

# We can also calculate the standard error
stderr = math.sqrt(np.mean((Y_train - lm.predict(X_train))**2))
print('\nStandard error:', stderr)
```

## Example: Predicting House Prices

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import math
import numpy as np
```

```
# load Scikit.learn's example dataset of Boston house prices
from sklearn.datasets import load_boston
boston = load_boston()

# First let's create a train and test split
X_train, X_test, Y_train, Y_test = train_test_split(boston.data, boston.target,
                                                    test_size = 0.33, random_state = 5)
                                  # so we get the same results

# Now let's fit a model
lm = LinearRegression()
_ = lm.fit(X_train, Y_train)

print('Intercept:', lm.intercept_)
print('Coefficients:\n', lm.coef_)
print('\nR-squared:', lm.score(X_train, Y_train))
```

Results:

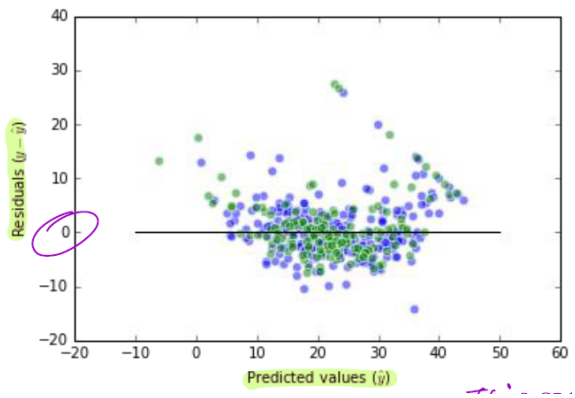
```
Intercept: 32.858932634085924
Coefficients:
 [-1.56381297e-01  3.85490972e-02 -2.50629921e-02  7.86439684e-01
 -1.29469121e+01  4.00268857e+00 -1.16023395e-02 -1.36828811e+00
  3.41756915e-01 -1.35148823e-02 -9.88866034e-01  1.20588215e-02
 -4.72644280e-01]

R-squared: 0.7551332741779998
```

```
:Attribute Information (in order):
    - CRIM     per capita crime rate by town
    - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
    - INDUS    proportion of non-retail business acres per town
    - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    - NOX      nitric oxides concentration (parts per 10 million)
    - RM       average number of rooms per dwelling
    ...
```

## Visually Assessing Good Fit: Residual Plots

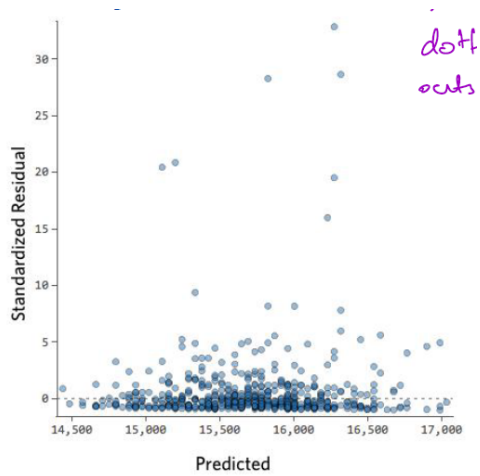
Residual Plots indicate good fit if…

- symmetrically distributed, clustering towards middle of plot
- there aren't any clear patterns
- they cluster around y = 0
- green training points, blue predicted points

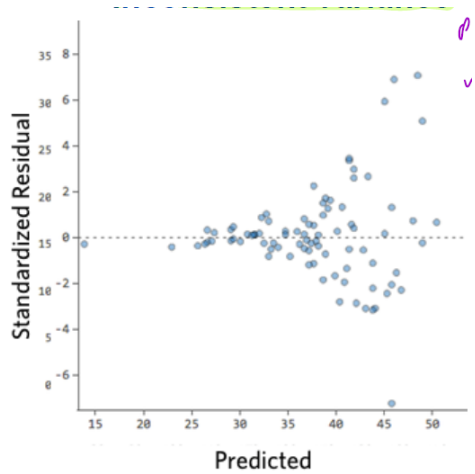



## Bad Residuals

y-axis off balance

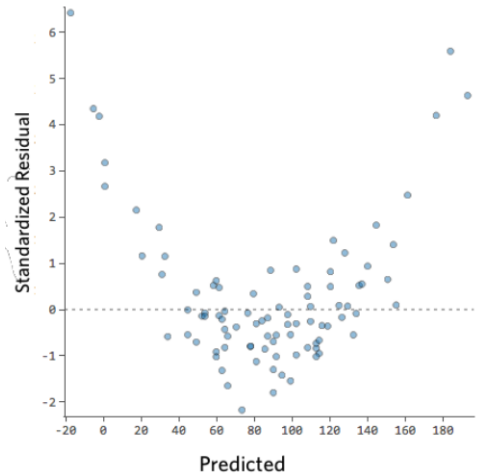



Inconsistent variance

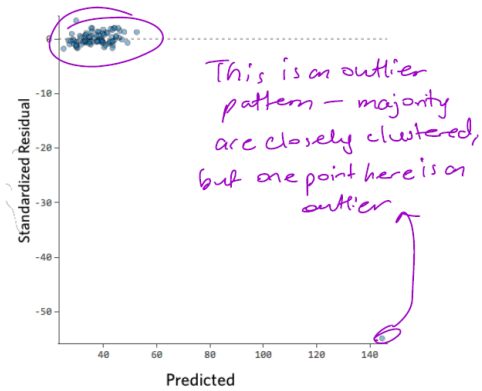



## More Bad Residuals

Non linear




Outlier



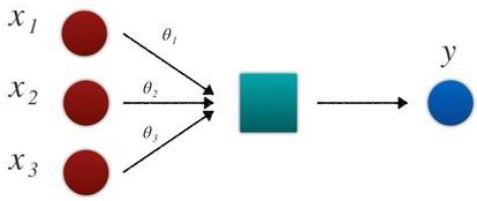

# Logistic Regression

## Classification vs Regression

- Classification assigns a class to each example
- Output is a discrete/categorical variable
- E.g., predict whether tumour is harmful or not harmful

- Regression assigns a numerical value
- Output is a continuous variable (real value)
- E.g., predict house price

## Logistic Regression



- Predict the probability of some categorical label
- E.g., given
  - amount of debt
  - late payment count

  predict the probability of defaulting on a loan

## Split data and train a classifier in scikit-learn

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# First let's create a train and test split
X_train, X_test, Y_train, Y_test = train_test_split(iris.data, iris.target,
                                                    test_size = 0.33)

# Now let's fit a model
logreg = LogisticRegression()
_ = logreg.fit(X_train, Y_train)
```

Result:

```
Intercept: [  8.77541301   1.60989219 -10.38530519]
Coefficients:
 [[-0.40882379  0.8471058  -2.20548191 -0.96488141]
 [ 0.60527175 -0.44107978 -0.16035026 -0.8906139 ]
 [-0.19644796 -0.40602602  2.36583217  1.85549531]]

Predicted type of first five organisms from test split: [1 2 2 0 2]
Actual type of first five organisms from test split: [1 2 2 0 2]
```

matches exactly the same.

(0=setosa, 1=versicolor, 2=virginica)

## Evaluating Classification

```
from sklearn.metrics import classification_report
import pandas as pd
key=', '.join(['{}={}'.format(i,name) for i,name in enumerate(iris.target_names)])
print('Classification report ({}):\n'.format((key))
print(classification_report(Y_test, logreg.predict(X_test))
```

Result:

```
Classification report (0=setosa, 1=versicolor, 2=virginica):

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.94      0.97        17
           2       0.94      1.00      0.97        17

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```

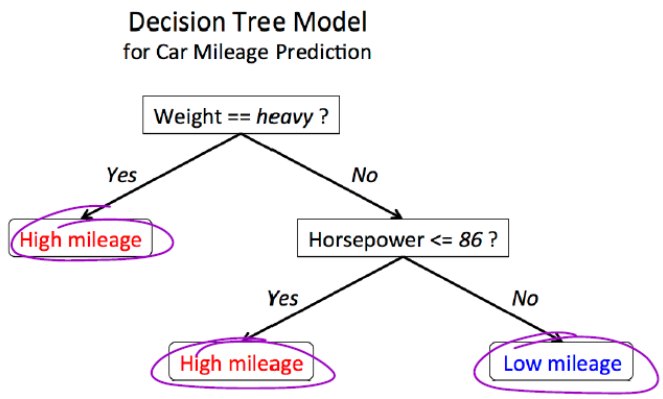
### Selecting Model Parameters with Grid Search

- Parameters like penalty and regularisation strength are not learnt from data by default
- Can be set using exhaustive search through combinations of specified possible values
- Perform n-fold cross validation for each combination
- In scikit-learn:

  `from sklearn.grid_search import GridSearchCV`

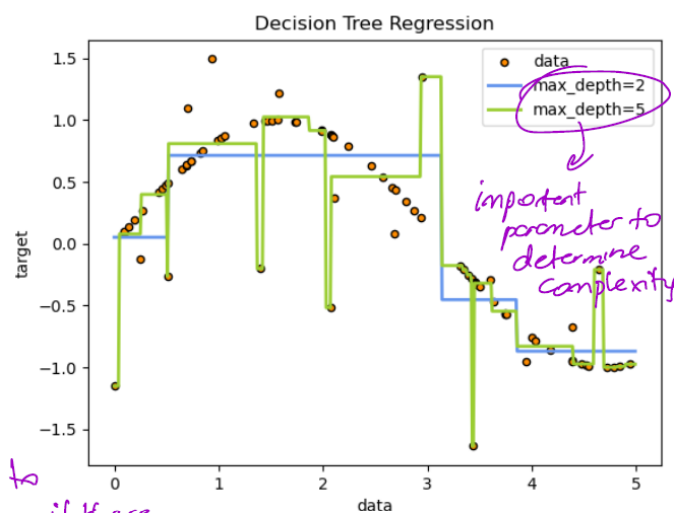
## Decision Trees

- Maps observations to a target value
- Can be viewed as hierarchy of if/else statements
- Resulting model is intuitive and interpretable (plus can be visualised)
- Ensembles of simple trees can do very well




- Basically learn a piece-wise function that approximates data

- Disadvantages
  - Prone to overfitting by creating over-complex models if depth not limited
  - Can be unstable due to small data variations → ensembles of trees
  - Predictions are not continuous
  - If some classes dominate in data, can produce biased trees




## Training a Decision Tree Classifier in scikit-learn

- DecisionTreeClassifier
- Can be parameterised, e.g:
  - max_depth
    - the maximum depth of the tree
  - criterion
    - entropy: choose splits that minimise total uncertainty
    - gini: choose splits that minimise misclassification
  - splitter
    - best: choose the optimal threshold for each feature
    - random: choose the best random threshold for each feature

As, usual split data into training and test sets first. Then:

```
from sklearn.tree import DecisionTreeClassifier

# Let's fit a model
tree = DecisionTreeClassifier(max_depth=2)
_ = tree.fit(X_train, Y_train)
```

### Compare Classifier on a single Test Split

- How to compare two decision tree classifiers?
- Use McNemar's test to compare two classifiers over a single test split
  - Split dataset into training and test data sets
  - Train both classifiers on the training dataset
  - Using the test data, determine the number of instances misclassified by each classifier

- $H_0$: Two classifiers have the same error rate

- $H_A$: One is better (assume whichever has higher accuracy)

### McNemar's Test

- Given:
  - n1: number of instances where Classifier1 is correct, but Classifier2 is not
  - n2: number of instances where Classifier2 is correct, but Classifier1 is not
- Then:

$$\boxed{\frac{(|n1 - n2| - 1)^2}{(n1 + n2)}}$$

chi - squared

follows a $\chi^2$ distribution with 1 degree of freedom if the null hypothesis is correct. If in your evaluation it doesn't you can reject $H_0$