# Machine learning

3rd April 2019

## K-means clustering

→ Supply the number of clusters. This is a hyperparameter

→ The computer iterarilively finds the center of k clusters

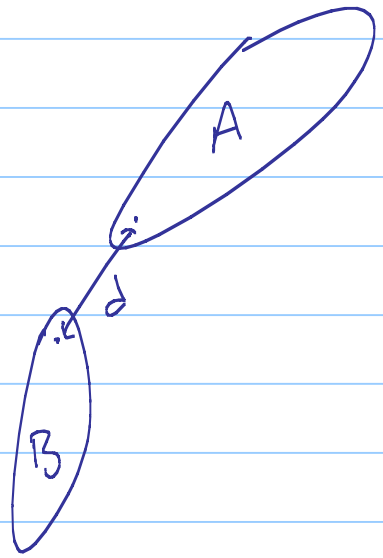
→ At the initial step randomly choose k points as cluster centers, each data point is assigned cluster corresponding to its nearest cluster center. The cluster centers are now moved to the mean-position of their respective clusters.

## Hierarchical clustering

→ The number of clusters is a priori unfixed.

→ We need to supply the value for minimal seperation between two distinct clusters. Call this $d_{min}$.

→ If the closest points between two clusters are less than dmin apart, then the clusters are merged.




→ if d < dmin, then merge A & B;
Else A & B are left as distinct clusters.

→ At the initial stage, each datapoint is identified as an independent cluster. Then the computer iteratively merges the clusters by measuring their seperation.

## Numpy :

→ if A is a numpy array then

A[-k:] gives the last k elements in the array.

# Pandas :

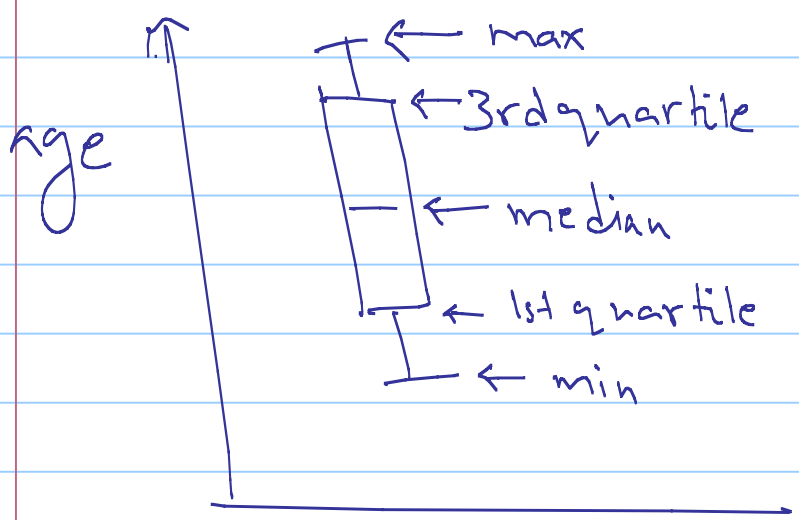
→ if df is a pandas data frame, with columns : 'age', 'gender', ...

→ find average age for each gender

sol : `df.groupby('gender')['age'].mean()`

Can also get median, mod, std etc

→ `df['age'].plot(kind='box')` is the distribution of age




median : (# data points < median) = (# data points > median)

4 th April 2014

→ Make sure to use only the training data and NOT the cross-validation or test data during the preprocessing stages.

↓

This is exactly why using a pipeline that integrates the various preprocessing and machine learning libraries is a good idea, as it automatically takes care of this.

- Scoring on classification tasks:

Confusion matrix

| Prediction → / labels ↓ | Pos. | Neg. |
|---|---|---|
| Pos. | TP | FN |
| Neg. | FP | TN |

Recall : $\frac{TP}{\text{labelled positive}}$

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

Precision : $\frac{TP}{\text{predicted pos.}}$

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

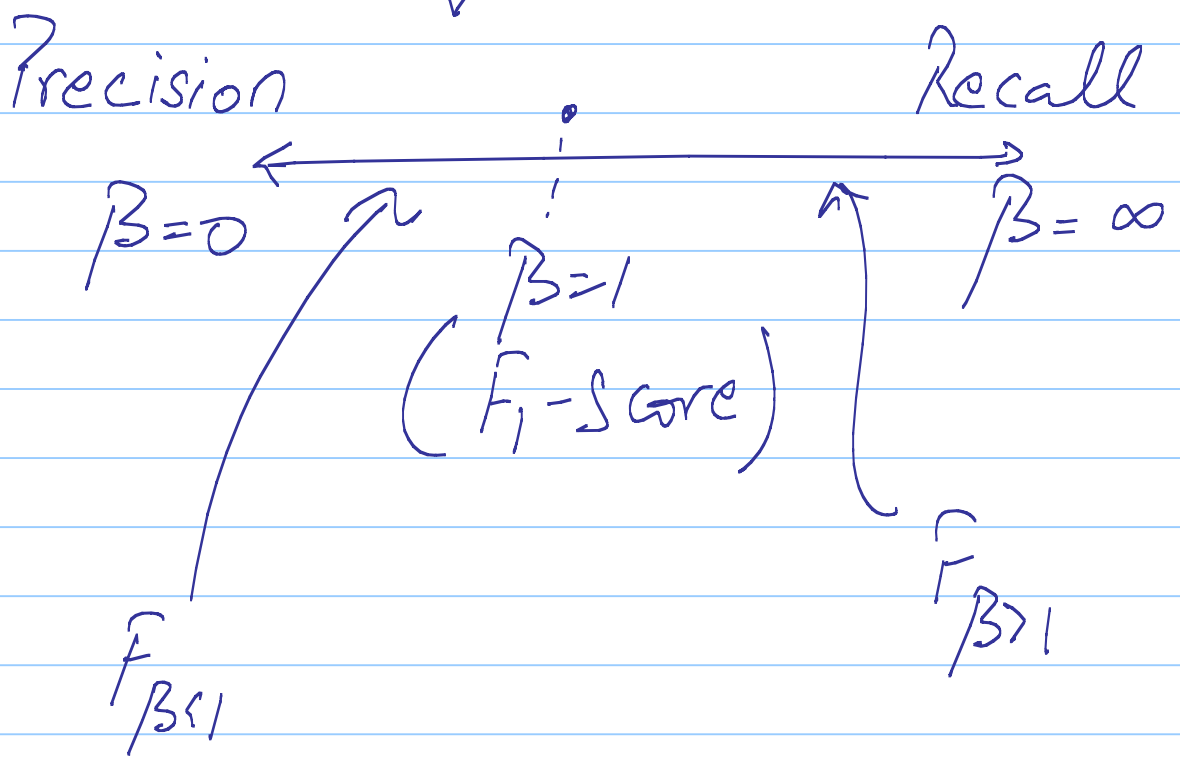
Mnemonic:

Recall = $\frac{TP}{\text{Real pos.}}$

Precision = $\frac{TP}{\text{Predicted pos.}}$

$F_1\text{-Score} = \frac{\text{Prec.} \times \text{Rec.}}{\left(\frac{\text{Prec.} + \text{Rec}}{2}\right)}$

$$F_\beta\text{-Score} = \frac{\text{Prec.} \times \text{Rec.}}{\left(\frac{\beta^2\,\text{Prec} + \text{Rec}}{1+\beta^2}\right)}$$




$F_{\beta<1}$ → more imp. to Precision

(for e.g we'd rather have a spam email not get filtered than have an imp. email sent to spam. ⇒ prefer precision over recall)

$F_{\beta>1}$ → more imp. to Recall

(e.g. We'd rather diagnose a healthy patient as mildly sick, then diagnose a sick patient as healthy => want less FN

=> want high recall)

## ROC curve

→ Receiver operating characteristic

↳ Curve of Signal vs Noise

Signal := True pos. rate

$$TPR = \frac{TP}{\text{Real pos.}} = \frac{TP}{TP + FN}$$

Noise = False pos. rate

$$FPR = \frac{FP}{\text{Real neg.}} = \frac{FP}{FP + TN}$$

→ Used to gauge the ability of a binary classifier's ability to distinguish between the positive class and the negative class.

→ To plot ROC we do the following : For each instance, the classifier will output a number. We then decide a threshold & if that number is greater than the threshold, the instance is predicted to in the positive class.

OR equivalently the prob. of being in positive class

ROC curve is obtained by varying the threshold from very low to very high values and then

Computing (TPR, FPR) pairs for each value of threshold.

plotting the list

$[(TPR, FPR)_1, (TPR, FPR)_2, \ldots]$

gives us the ROC curve.

$(TPR, FPR)_i$ = TPR & FPR for i-th threshold value

→ If the classifier is able to distinguish well between pos. class & negative class, then area-under the curve (AUC) of ROC will be ~1.

→ For a classifier than randomly guesses the pos. & neg. classes, its ROC-AUC will be ~ 0.5.

Numpy as np

- np.clip(array, min, max)
- np.arange(start, stop, step)
- let 'a' be a numpy array
  => a.flatten() returns the result of flattening 'a'
- np.linspace(start, stop, num)
- if 'a' is a numpy array,
  a.sort(axis = axis)
  ↳ returns the result of sorting a along the specified axis.

→ Can also do this by np.sort(a)

- np.argsort( )
  → returns the indices that would sort an array.

- np.argwhere( some_condition )
  ↳ returns the indices of the arrays elements where the given condition is true

- matplotlib.pyplot as plt
- use `plt.imshow( )` to plot images.