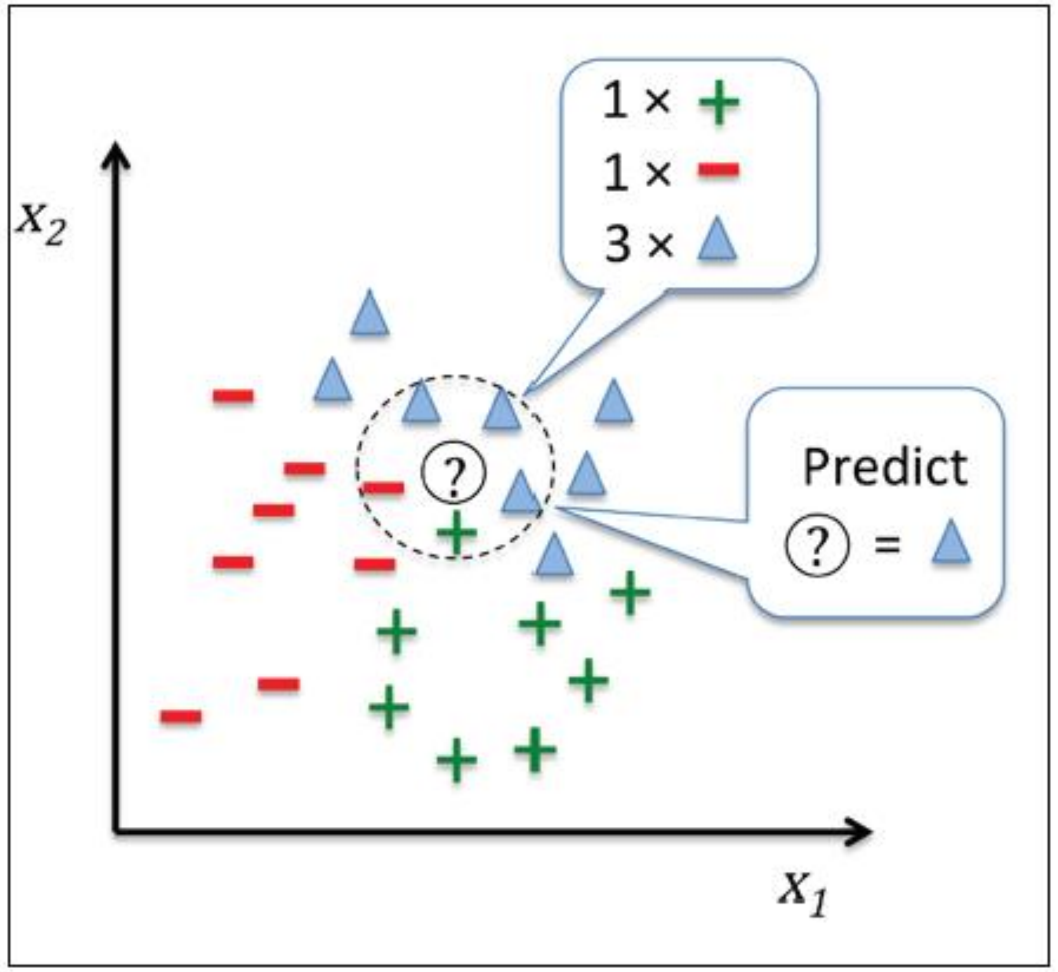
# k-Nearest Neighbors

- Sonal Ghanshani

# K-Nearest Neighbours (kNN)

- K-nearest neighbours is an algorithm that classifies data points by a majority vote of its k neighbours.
- It is used to assign a data point to clusters based on similarity measurement.
- A new input point is classified in the category such that it has the most number of neighbours from that category.



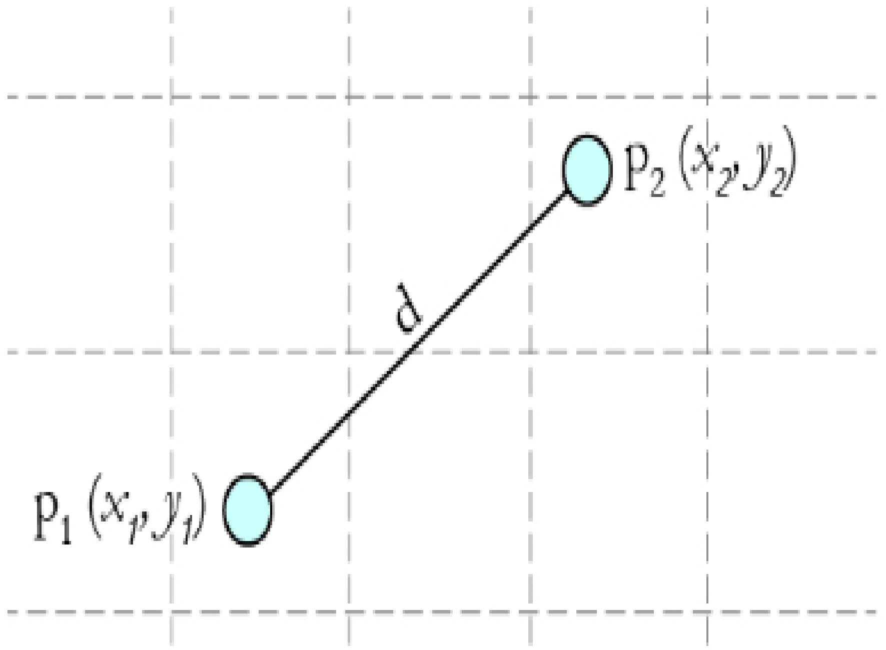
# K-Nearest Neighbours Algorithm

- Calculate the distance of the unknown data points with other training data points i.e. choose the number of k and a distance metric.
- Identify k-nearest neighbours.
- Use category of nearest neighbours to find the category of the new data points based on majority vote.

# Computing Distance and Determining Class

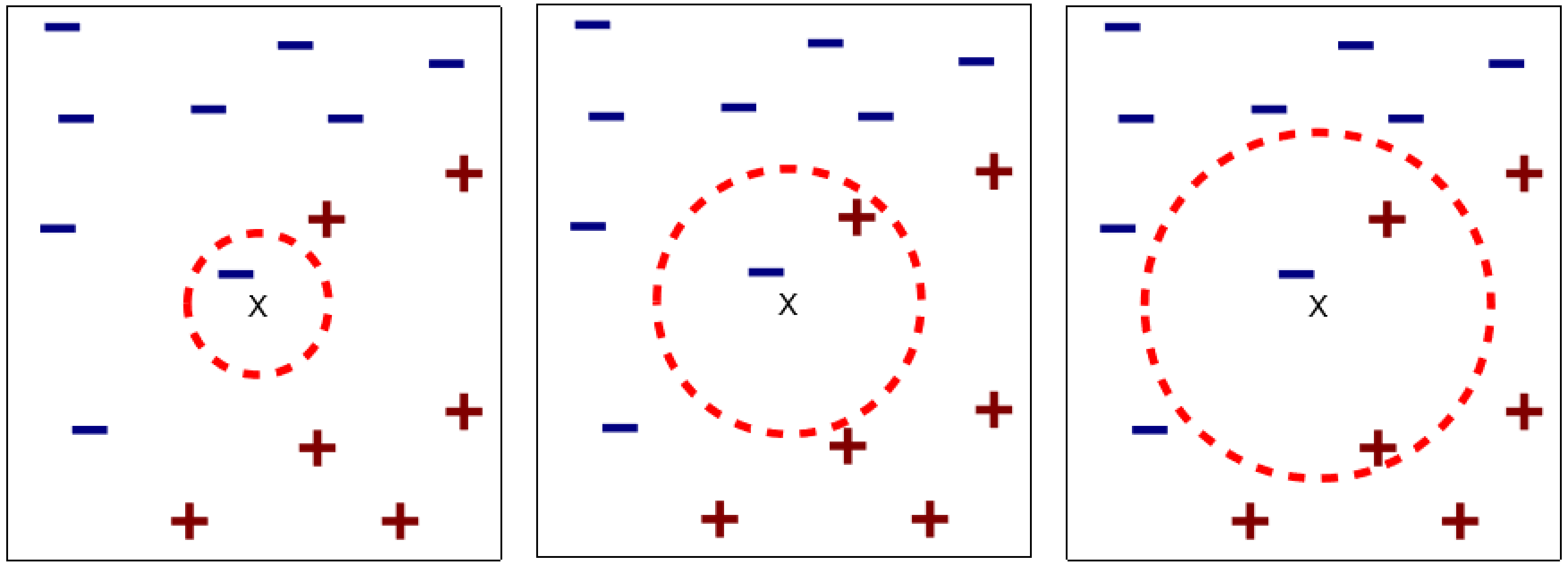
- For the Nearest Neighbour Classifiers, the distance between two points is expressed in the form of Euclidean distance, which is calculated by:

$$(d) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$




- You can determine the class from the nearest neighbour list by:
- Taking the majority number of votes of class labels among the k-nearest neighbours

# Nearest Neighbour




(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

# Computing Distance and Determining Class

- For a customer with

| Age | Salary |
|---|---|
| 30 | 50 |

Predict the Credit Rating for k = 3 and k = 5

Data

| Age | Salary | Credit Rating |
|---|---|---|
| 25 | 36 | Fair |
| 25 | 69 | Excellent |
| 27 | 69 | Fair |
| 28 | 63 | Excellent |
| 28 | 80 | Poor |
| 30 | 54 | Excellent |
| 31 | 65 | Fair |
| 32 | 48 | Fair |
| 32 | 77 | Excellent |
| 33 | 55 | Poor |
| 33 | 35 | Poor |
| 33 | 65 | Fair |
| 33 | 75 | Fair |
| 37 | 51 | Fair |
| 37 | 61 | Excellent |
| 38 | 31 | Poor |
| 39 | 39 | Excellent |
| 40 | 35 | Excellent |
| 40 | 69 | Poor |
| 40 | 70 | Excellent |

Calculating the distances

| Age | Salary | Credit Rating | distance |
|---|---|---|---|
| 25 | 36 | Fair | 25.2389 |
| 25 | 69 | Fair | 28.3196 |
| 27 | 69 | Fair | 29.8329 |
| 28 | 63 | Excellent | 27.2947 |
| 28 | 80 | Poor | 38.4187 |
| 30 | 54 | Excellent | 26.3059 |
| 31 | 65 | Fair | 30.8869 |
| 32 | 48 | Fair | 28.0713 |
| 32 | 77 | Excellent | 38.8973 |
| 33 | 55 | Poor | 29.4279 |
| 33 | 35 | Poor | 32.6497 |
| 33 | 65 | Fair | 32.6497 |
| 33 | 75 | Fair | 38.2884 |
| 37 | 51 | Fair | 33.0151 |
| 37 | 61 | Excellent | 34.7851 |
| 38 | 31 | Poor | 38.9487 |
| 39 | 39 | Excellent | 36.6879 |
| 40 | 35 | Excellent | 39.0000 |
| 40 | 69 | Poor | 40.7063 |
| 40 | 70 | Excellent | 41.1825 |

Sorting the distances in ascending order

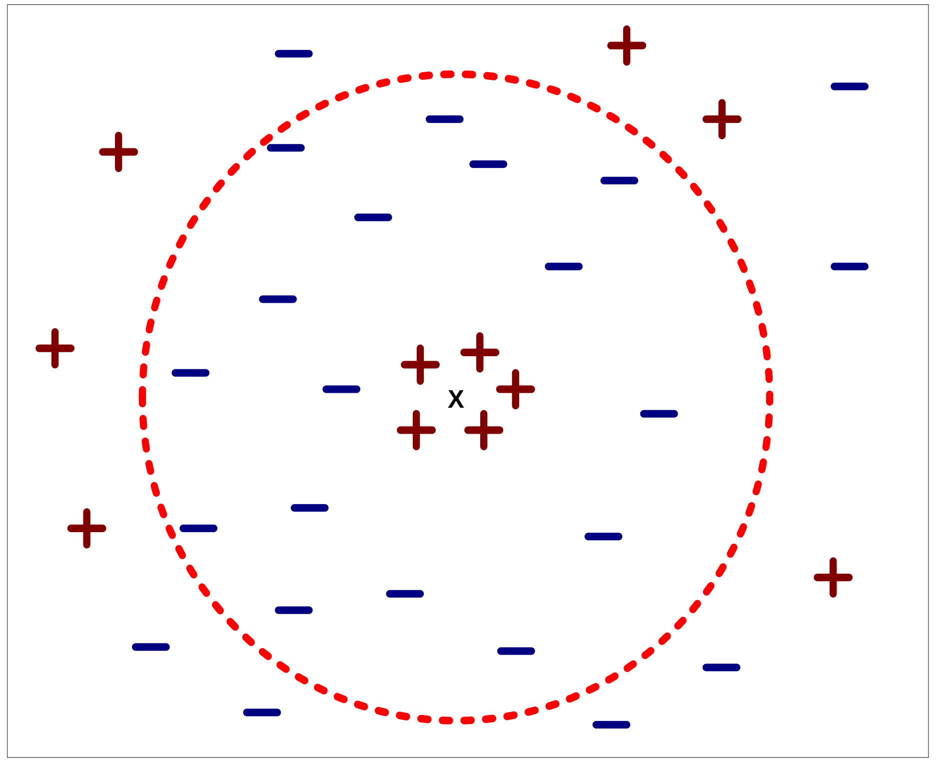
| Age | Salary | Credit Rating | distance |
|---|---|---|---|
| 25 | 36 | Fair | 25.23885893 |
| 30 | 54 | Excellent | 26.30589288 |
| 28 | 63 | Excellent | 27.29468813 |
| 32 | 48 | Fair | 28.0713377 |
| 25 | 69 | Fair | 28.31960452 |
| 33 | 55 | Poor | 29.42787794 |
| 27 | 69 | Fair | 29.83286778 |
| 31 | 65 | Fair | 30.88689042 |
| 33 | 35 | Poor | 32.64965543 |
| 33 | 65 | Fair | 32.64965543 |
| 37 | 51 | Fair | 33.01514804 |
| 37 | 61 | Excellent | 34.78505426 |
| 39 | 39 | Excellent | 36.68787266 |
| 33 | 75 | Fair | 38.28837944 |
| 28 | 80 | Poor | 38.41874542 |
| 32 | 77 | Excellent | 38.89730068 |
| 38 | 31 | Poor | 38.94868419 |
| 40 | 35 | Excellent | 39 |
| 40 | 69 | Poor | 40.70626487 |
| 40 | 70 | Excellent | 41.18252056 |

**For k = 3, the predicted Credit Rating is 'Excellent' and**

**For k = 5, the predicted Credit Rating is 'Fair'**

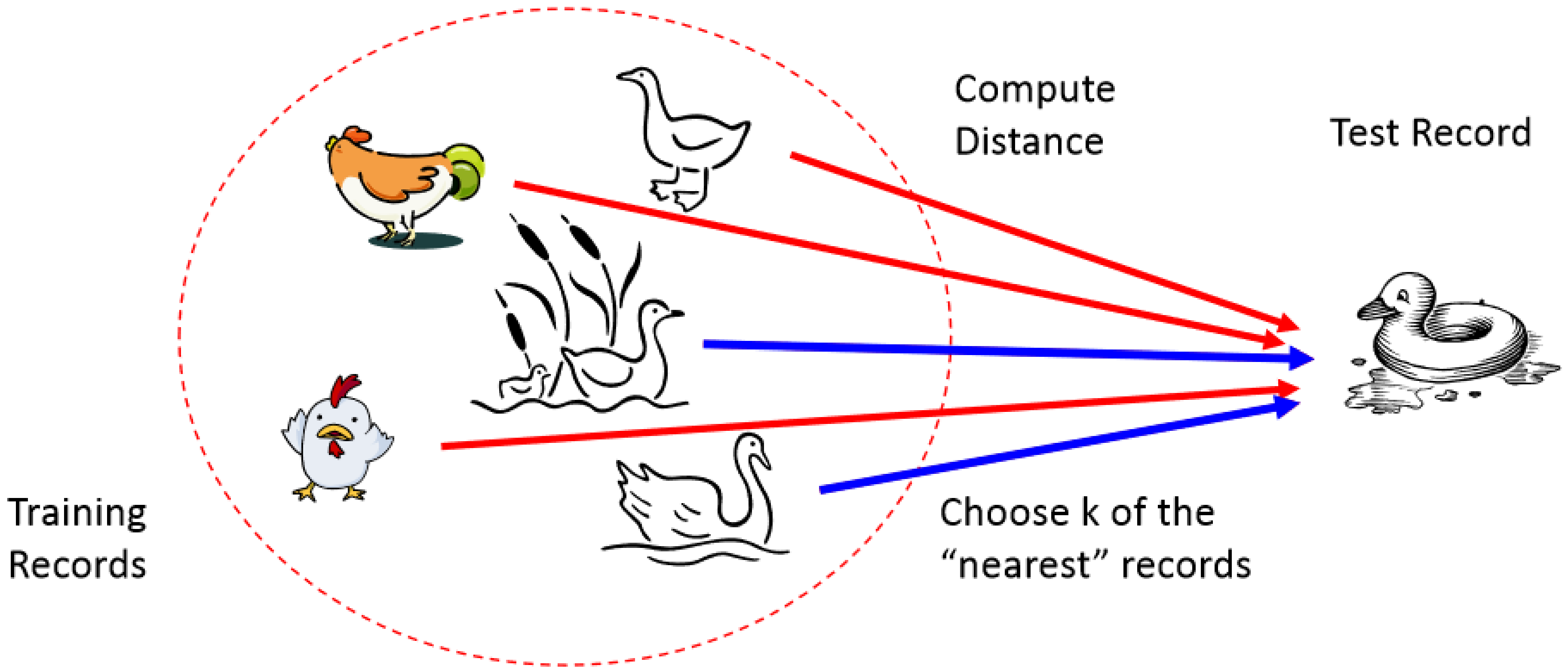
# Choosing the Value of k

- When choosing the value of k, keep the following points in mind:
  - If its value is too small, neighbourhood is sensitive to noise points
  - If its value is too large, neighbourhood may include points from other classes



# Essentially

- **Basic idea**
  - If it walks like a duck, quacks like a duck, then it's probably a duck
- **Rote-learner**
  - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly



# Lazy vs. Eager Learning

# Lazy vs. Eager Learning

- Lazy vs. eager learning
  - Lazy learning (kNN): Simply stores training data (or only minor processing) and waits until it is given a test tuple
  - Eager learning (Decision Tree): Given a set of training set, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training but more time in predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form its implicit global approximation to the target function
  - Eager: must commit to a single hypothesis that covers the entire instance space

# Model Validation

# Accuracy Matrix

| | | Predicted class | |
|---|---|---|---|
| | | $P$ | $N$ |
| Actual Class | $P$ | True Positives (TP) | False Negatives (FN) |
| | $N$ | False Positives (FP) | True Negatives (TN) |

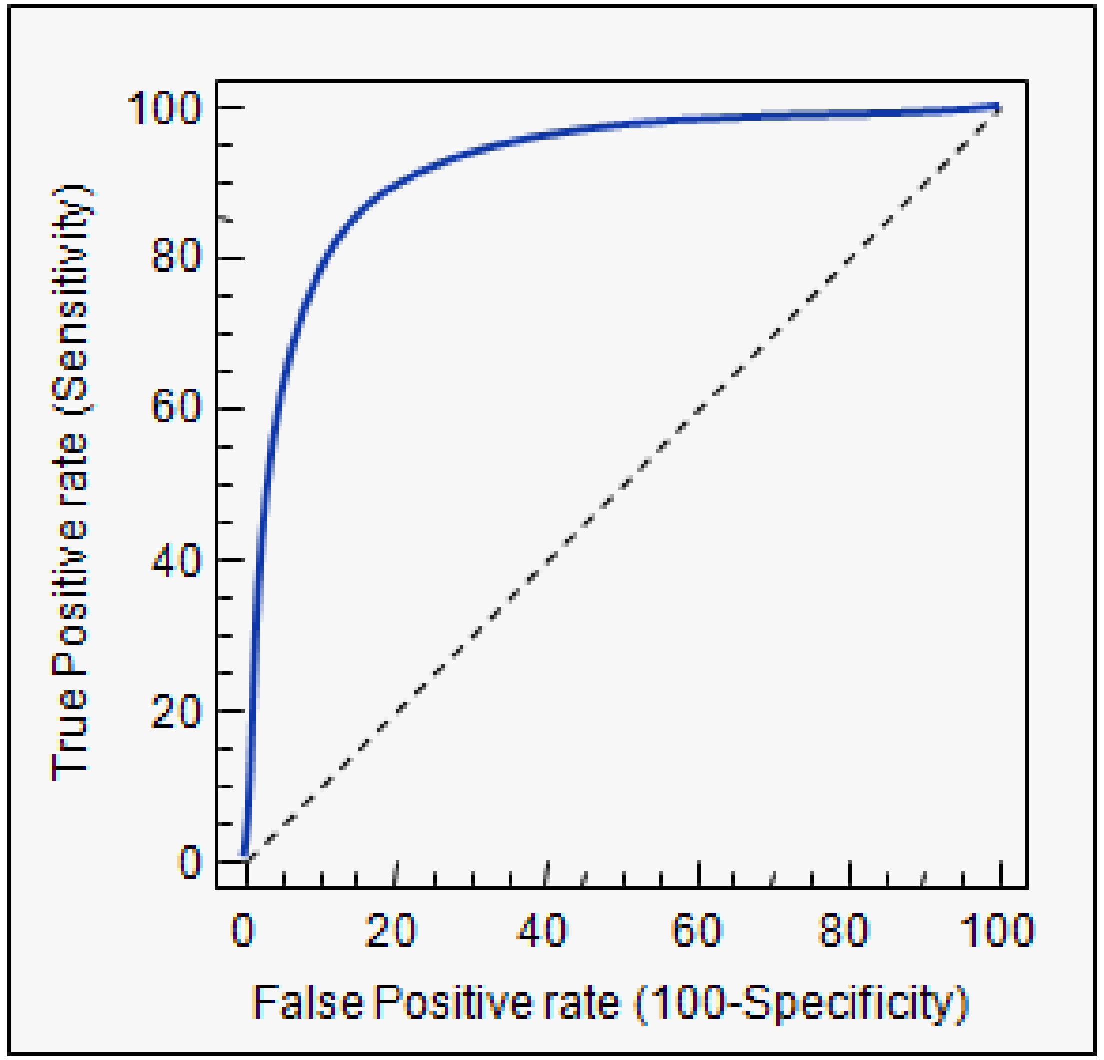
**Sensitivity, recall, hit rate, or true positive rate (TPR)**

$$\mathrm{TPR} = \frac{\mathrm{TP}}{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

**Specificity or true negative rate (TNR)**

$$\mathrm{TNR} = \frac{\mathrm{TN}}{N} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

# ROC - AUC



100
80
60
40
20
0
True Positive rate (Sensitivity)
0
20
40
60
80
100
False Positive rate (100-Specificity)

# Application

# When to Consider k-Nearest Neighbours

- Less than 20 features (attributes) per instance, typically normalized
- Lot of training data
- **Advantages:**
  - Training is very fast
  - Learn complex target functions
  - Do not lose information
- **Disadvantages:**
  - Slow at query time
  - Easily misled by irrelevant features (attributes)