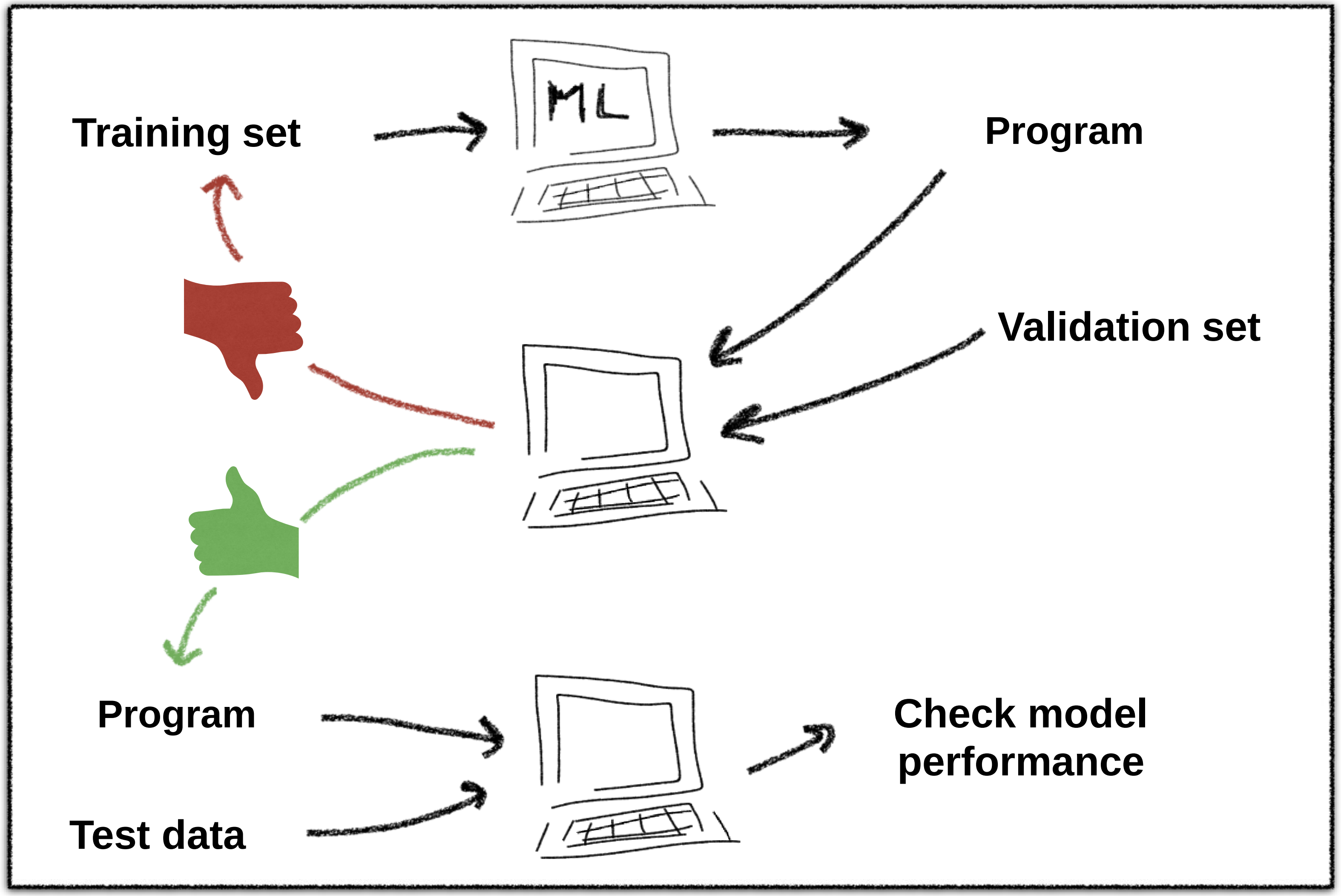
UNIVERSITÀ DI TRENTO

Introduction to Machine learning

Model Evaluation & K-NN

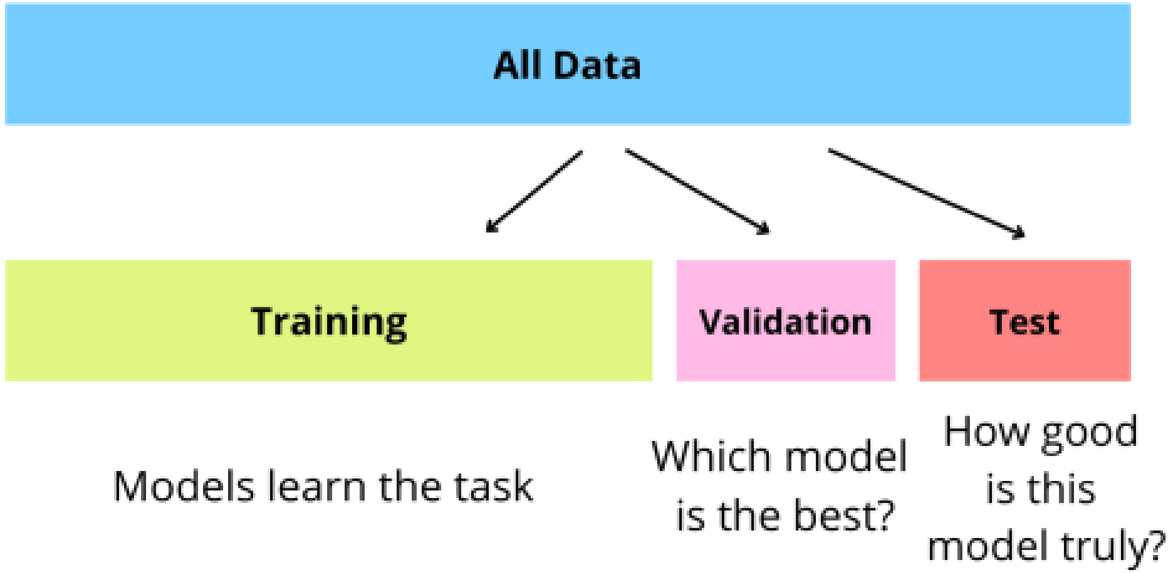
Cigdem Beyan

Recap--train / validation / test sets



The importance of Validation set

- Separate from the training set! *(so-called mini test set)*
  - Is used to pick (a better performing) algorithm
  - Is used to decide the (hyper-)parameters of an algorithm
- ATTENTION: Splitting the datasets into training and validation sets can be done randomly to avoid BIAS. However,
  - there are some specific rules to apply.



Specific rules

- When you split the dataset: *training, validation, testing,* you can have conflicting priorities.
  - Estimate future error (i.e., validation/testing error) as accurately as possible
    - How to? By making the validation set as big as possible. *(High confidence interval)*
  - Learn classifier as accurately as possible
    - How to? By making the training set as big as possible. *(Better estimates, maybe better generalization)*

  - Training and validation/testing instances CANNOT OVERLAP !!!!

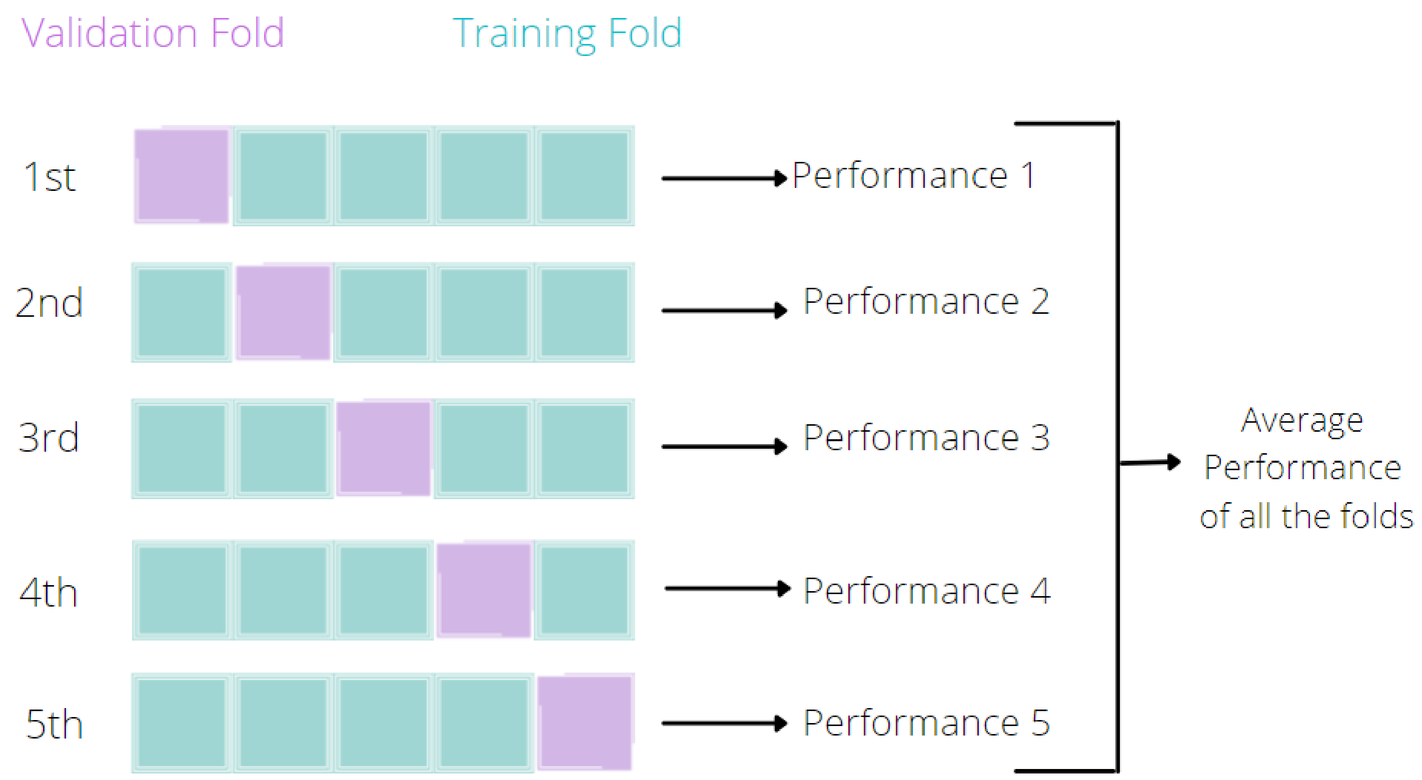
Cross validation

- Training and validation cannot overlap,
  but $n_{train}+n_{validation}$ = constant

- *Cross validation* idea:
  - Train → Validation, then Validation→ Train, average the results of both
  - At each fold (step) you use each instance only in one set, so no overlapping.
  - If there are 2 folds, you have 2 models trained.
  - Every sample is in both training and testing but not at the same time.
    - Reduces the chances of getting an biased training set.

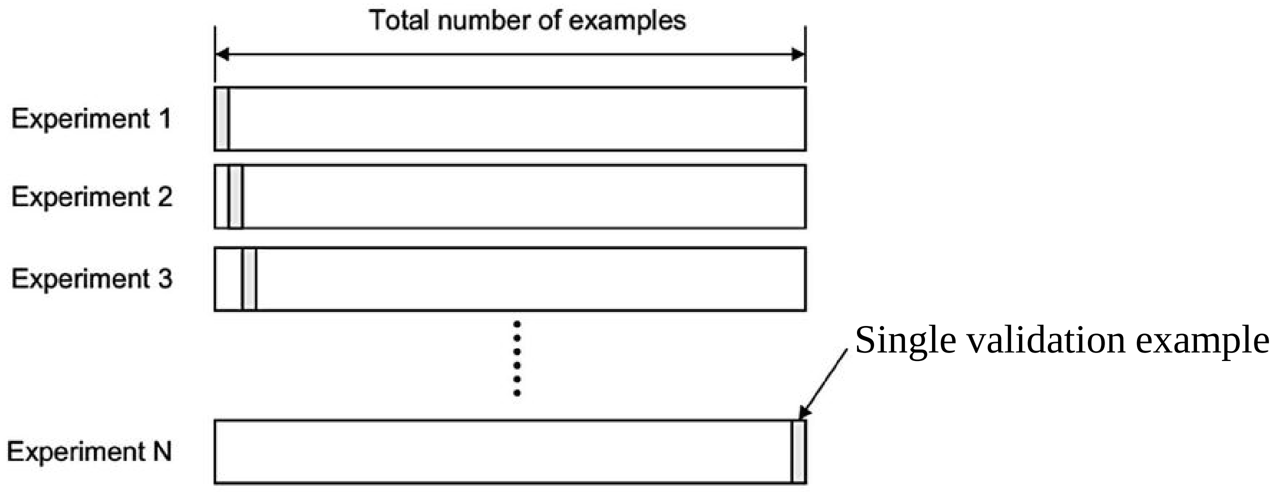
5-FOLD CROSS VALIDATION

- Randomly split the data into 5 folds
- Test on each fold while training on 4 other folds (80% train, 20% test)
- Average the results over 5 folds



LEAVE-ONE-OUT CROSS VALIDATION

- *n*-fold cross validation *n: total number of samples*
  - Training on all (n-1) samples, while test on 1 instance
- Pros. & Cons.
  - Best possible classifier learned from *n-1* training examples
  - High computational cost: re-learn everything for *n* times
  - Classes not balanced in training/ testing sets
    - Solution: Stratification



STRATIFICATION

- Keep class labels balanced across training and validation sets
- How?
  - Instead of taking the dataset and dividing it randomly into K parts
  - Take the dataset, divide it into individual classes
    - Then for each class, divide the instances in K
    - Assemble *ith* part from all classes to make the *ith* fold
    - Attention! It is still random

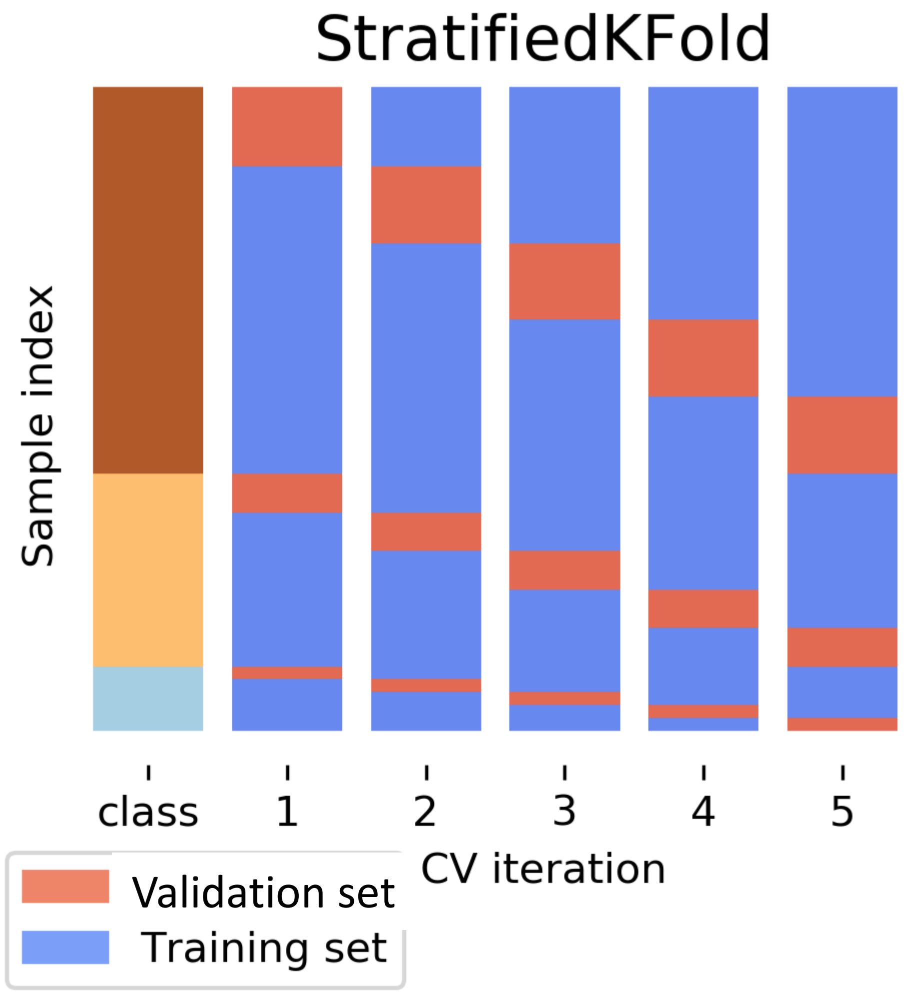


Image credit: https://amueller.github.io/aml/04-model-evaluation/1-data-splitting-strategies.html

Evaluation measures

- To decide if our model is performing well.
- To decide out of many models, which one performs better than the other.

- Classification:
  - How often our model classify something right/wrong

- Regression:
  - How close is our model to what we are trying to predict

- Clustering:
    - How well does our model describe our data (in general, very hard)

Classification evaluation measures--basics

Confusion matrix for binary classification

		Predicted Label	
		Positive	Negative
Actual Label	Positive	TRUE POSITIVE TP	FALSE NEGATIVE FN
	Negative	FALSE POSITIVE FP	TRUE NEGATIVE TN

We want to have large values in TP and TN, while smaller values in FP and FN

Classification evaluation measures--basics

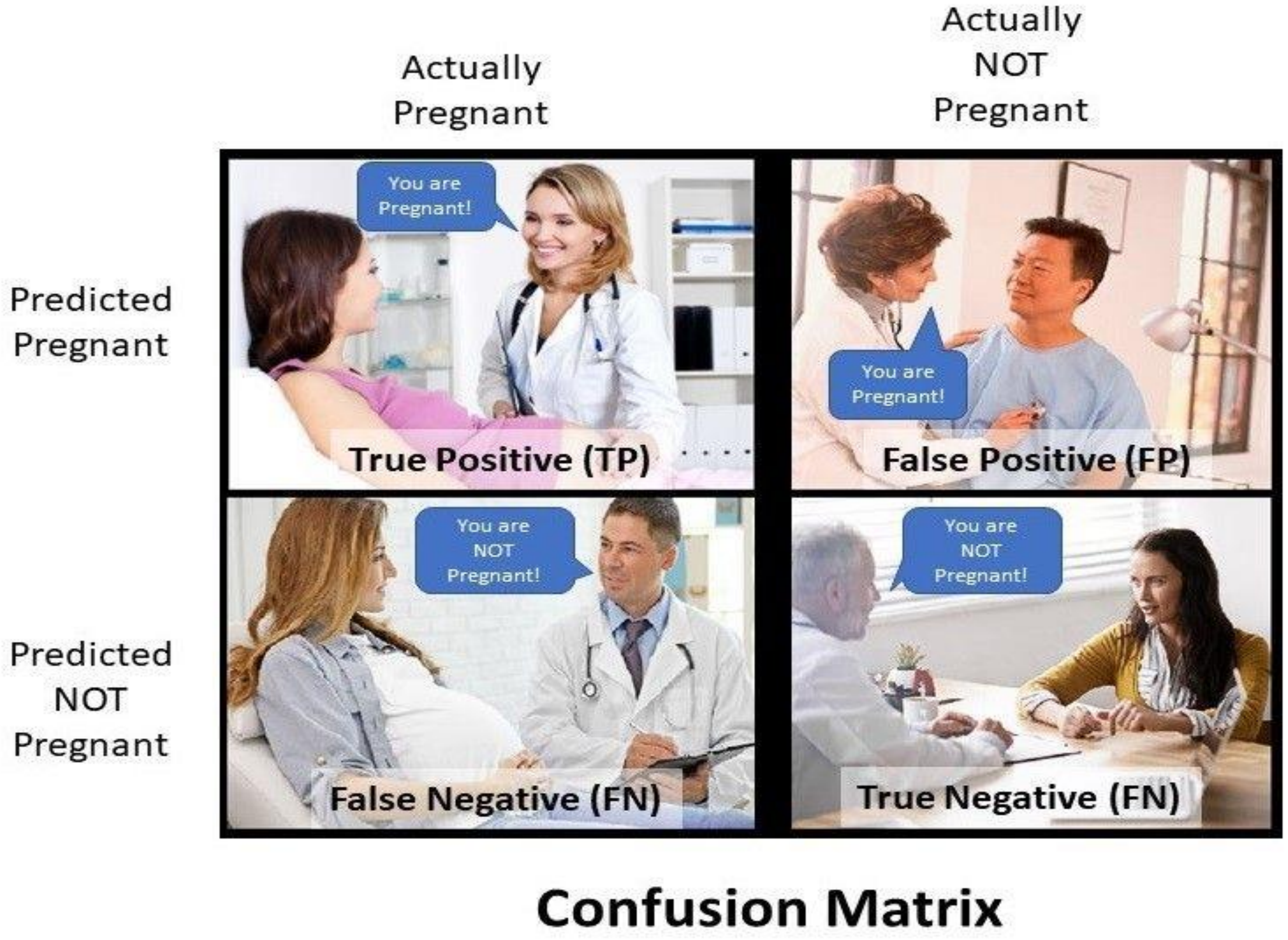


Image Credit: https://medium.com/analytics-vidhya/decoding-confusion-matrix-2b5912cabc6a

Classification evaluation measures

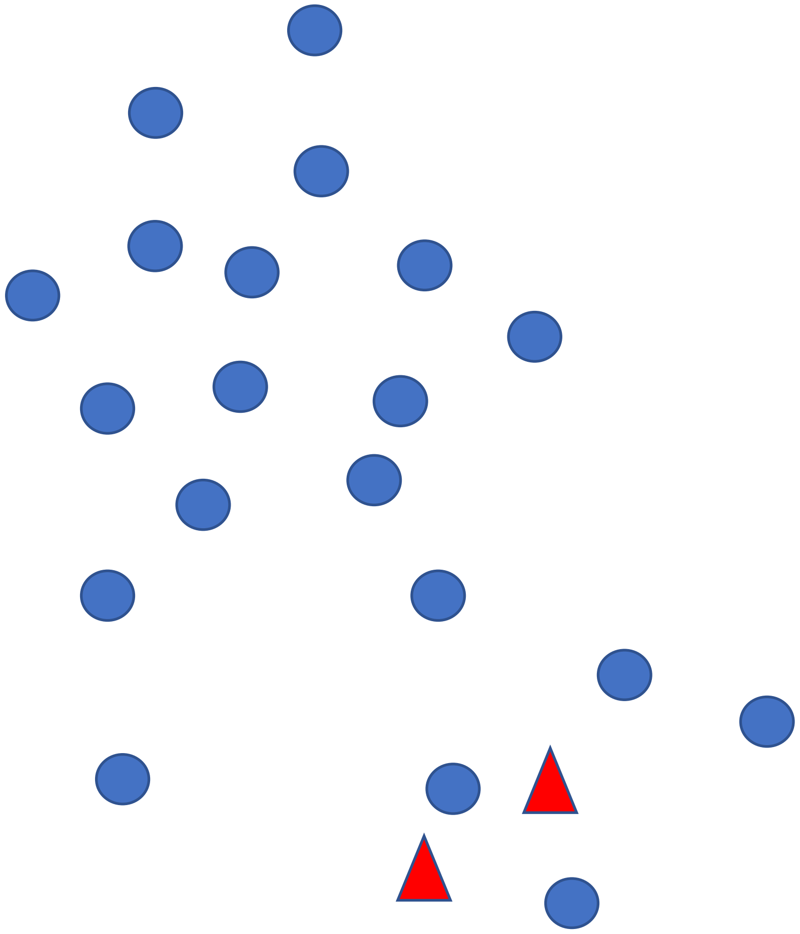
Classification Error= (FP+FN) / (TP+TN+FP+FN)

Accuracy= (1-Error)= (TP+TN) / (TP+TN+FP+FN)

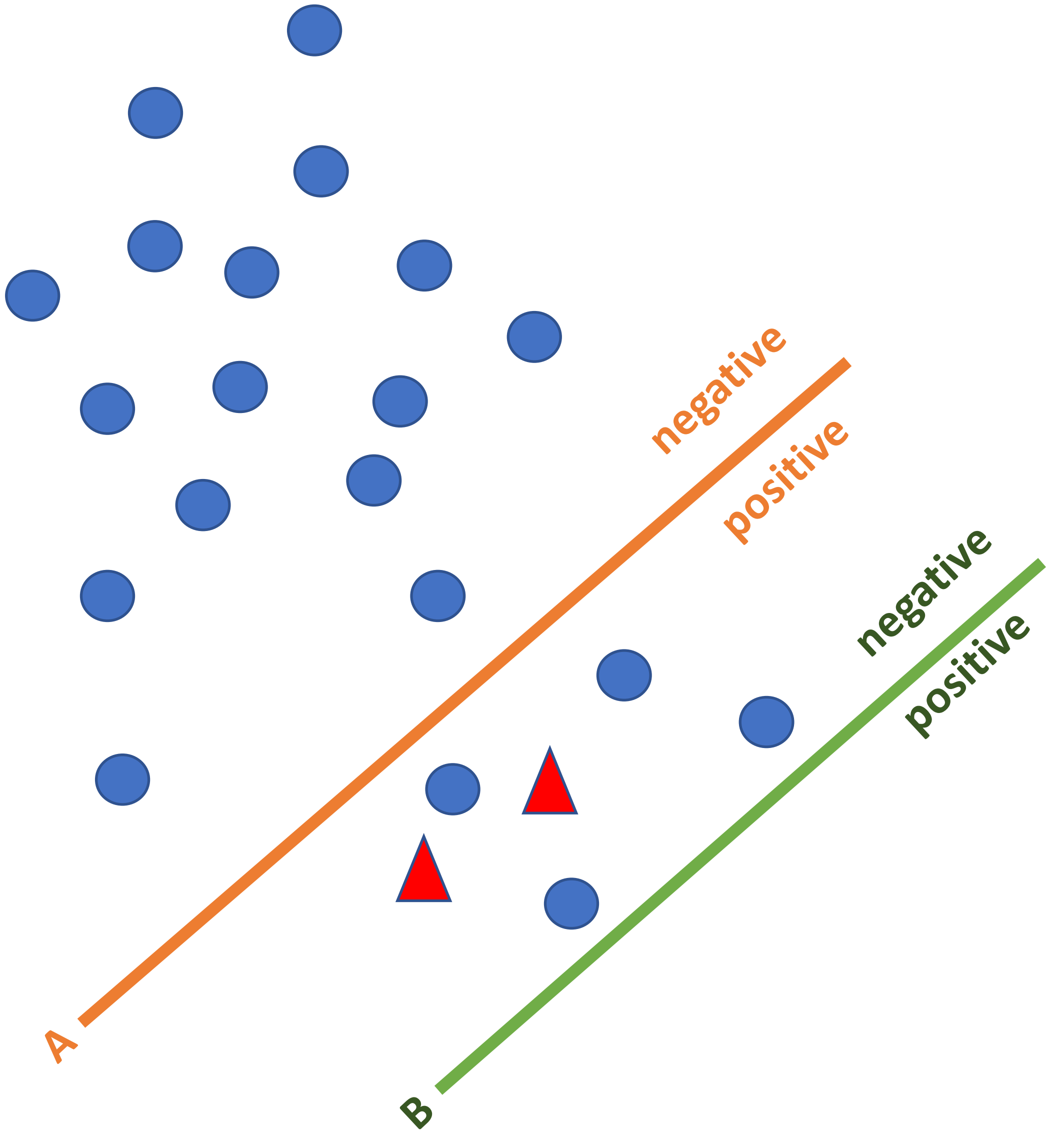
		Predicted Label	
		Positive	Negative
Actual Label	Positive	TRUE POSITIVE TP	FALSE NEGATIVE FN
	Negative	FALSE POSITIVE FP	TRUE NEGATIVE TN

- Basic measure of "goodness" of a classifier.
- ATTENTION! Very misleading if we have imbalanced classes
  - E.g., if you have 90 samples for "NO" and 10 samples for "YES", just by classifying every samples as "NO" would make our accuracy 90%, however, in fact our model is not able to classify any data belonging to "YES" class.

Accuracy -- imbalanced classes



Accuracy -- imbalanced classes



ML experts would prefer system A. However, accuracy metric prefers system B. Because B makes fewer errors (only 2) while A makes 4 errors.

Misses and False alarms

False Alarm Rate= False Positive Rate = FP / (FP+TN)
(% of negative we misclassified as positive)

Miss rate = False Negative Rate= FN / (TP+FN)
(% of positives we misclassified as negative)

Recall = True Positive Rate = TP / (TP+FN)
(% of positives we classified correctly (1-miss rate))

Precision = TP / (TP+FP)
(% positive out of what we predicted was positive)

		Predicted Label	
		Positive	Negative
Actual Label	Positive	TRUE POSITIVE TP	FALSE NEGATIVE FN
	Negative	FALSE POSITIVE FP	TRUE NEGATIVE TN

Cost of the task

- We typically do not use the evaluation metrics: accuracy, recall, precision etc. alone but instead declare a couple of them together.
- However,
  - However, in order to optimize a learner automatically (i.e., training), we need a single evaluation measure
  - How do we decide that single metric?
    - Domain specific !!! – depends on the task
- Depending on the cost of the task we can decide whether we take care more on having *less false positives* or *less false negatives through weighting them.*

$$\text{Cost} = C_{FP} * FP + C_{FN} * FN$$

F-measure

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Harmonic mean of precision and recall

One of the most frequently used metric.
(+) If you do some mathematics, you will see that F1 measure is sort of an accuracy without TN.
(+) Used frequently in information retrieval systems

Multiclass classification -- Evaluation

- Confusion matrix is a generalized version of the binary one.
- *nij* is the number of examples with actual label *yi* and predicted as *yj*.
- The main diagonal contains true positives for each class.
- The sum of off-diagonal elements along a column is the number of false positives for the column label.
- The sum of off-diagonal elements along a row is the number of false negatives for the row label.

		Predicted Label	
		Positive	Negative
Actual Label	Positive	TRUE POSITIVE TP	FALSE NEGATIVE FN
	Negative	FALSE POSITIVE FP	TRUE NEGATIVE TN

T\P	y_1	y_2	y_3
y_1	n_{11}	n_{12}	n_{13}
y_2	n_{21}	n_{22}	n_{23}
y_3	n_{31}	n_{32}	n_{33}

$$FP_i = \sum_{j \neq i} n_{ji}$$

$$FN_i = \sum_{j \neq i} n_{ij}$$

Multiclass classification -- Evaluation

- Accuracy, precision, recall and F1-mesure carry over to a per-class measure considering as negative examples from other classes.

$$Pre_i = \frac{n_{ii}}{n_{ii} + FP_i} \qquad Rec_i = \frac{n_{ii}}{n_{ii} + FN_i}$$

- *Multiclass accuracy* is the overall fraction of correctly classified examples:

$$MAcc = \frac{\sum_i n_{ii}}{\sum_i \sum_j n_{ij}}$$

Regression evaluation metrics

- (Root) Mean Squared Error:
  - Popular, nicely differentiable
  - Sensitive to single large errors
  (i.e., outliers)

$$\sqrt{\underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{testing set}}\Big(\underbrace{f(x_i)}_{\text{predicted}}-\underbrace{y_i}_{\text{true}}\Big)^2}$$

- Mean Absolute Error:
  - Less sensitive to outliers
  - *Why we take the absolute value?*
  Some differences are positive, some others are negative, and we don't want them to cancel each other.

$$\frac{1}{n}\sum_{i=1}^{n}\left|f(x_i)-y_i\right|$$

Regression evaluation metrics

- *Classification:* We can count how often we are correct or wrong in our predictions.

- *Regression:* We cannot do that counting!
  - Predicting *yi* (not discrete, continues value) from inputs *xi*
  - Here the question is not "*how many times your method is wrong*" but "*how much your method is wrong wrt. to the groundtruth-labels*".

Mean squared error

- Very sensitive to the mean and scale of the prediction.
  How to handle this?

- Relative Squared Error: MSE divided by what would happen if you used just the mean as a predictor.

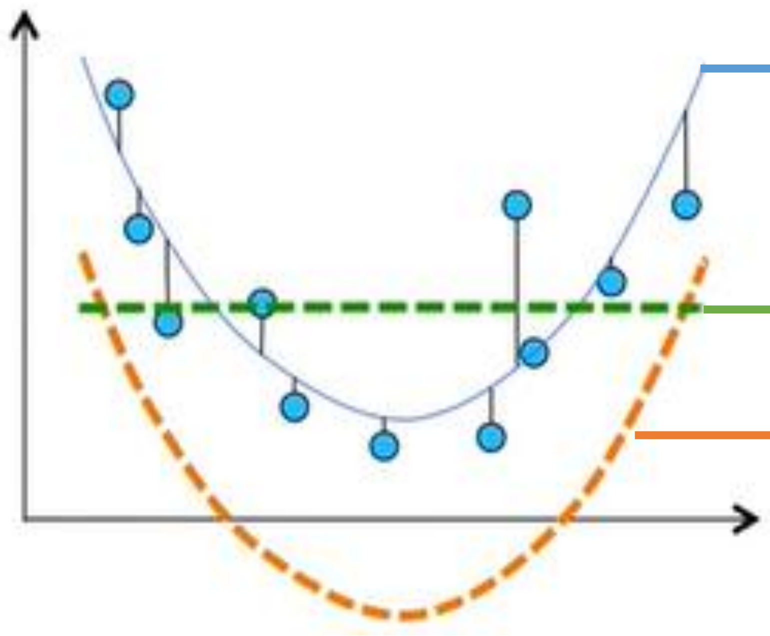
$$\sqrt{\underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{testing set}}\Big(\underbrace{f(x_i)}_{\text{predicted}}-\underbrace{y_i}_{\text{true}}\Big)^2}$$

$$\sqrt{\frac{\sum_{i=1}^{n}\left(f(x_i)-y_i\right)^2}{\sum_{i=1}^{n}\left(\mu_y-y_i\right)^2}}\quad\begin{array}{l}\left.\right\}\text{MSE of predictor}\\[1em]\left.\right\}\text{MSE when using the mean as a predictor}\end{array}$$

Mean squared error

$$\sqrt{\underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{testing set}}\Big(\underbrace{f(x_i)}_{\text{predicted}}-\underbrace{y_i}_{\text{true}}\Big)^2}$$

- Very sensitive to the mean and scale of the prediction.
- Having the correct mean matters!



Ground-truth

Green system, having the same mean, but the pattern is completely wrong

Orange system, following the trend of ground-truth, with wrong mean

Orange has higher MSE!!!

Mean absolute error $\frac{1}{n}\sum_{i=1}^{n}|f(x_i)-y_i|$

- Less sensitive to outliers
  - *i.e.*, a single large deviations will not overpower many small deviations

- Not differentiable (you cannot take the derivative of it and taking derivative is important if you want to build an algorithm that minimizes the function.)
  - That is why they used a lot less!!!
  - For MSE, taking derivative is very easy.

Coding Hint

- Model Evaluation in **Scikit-learn**

- A tutorial on how to calculate the most common metrics for **regression** and **classification** using scikit-learn:

https://towardsdatascience.com/model-evaluation-in-scikit-learn-abce32ee4a99

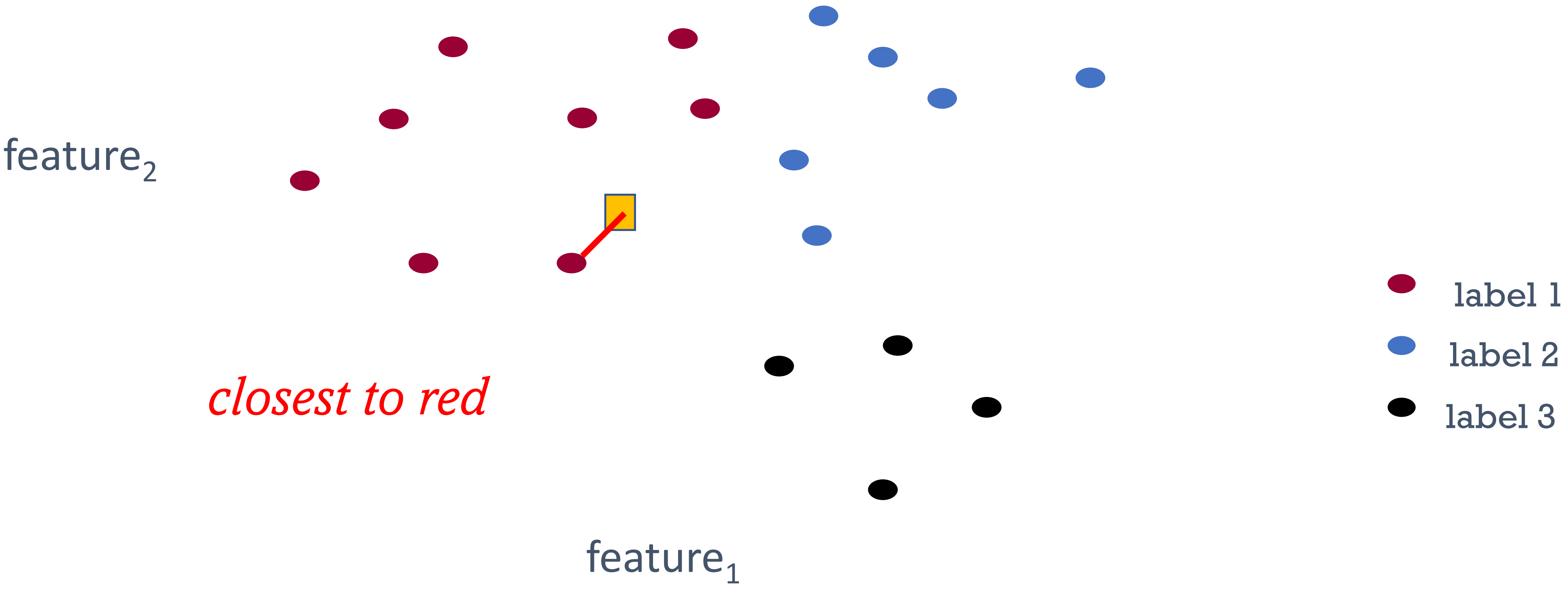
Training samples – scatter plot

TEST SAMPLES – SCATTER PLOT

- Which class is the ■ from?

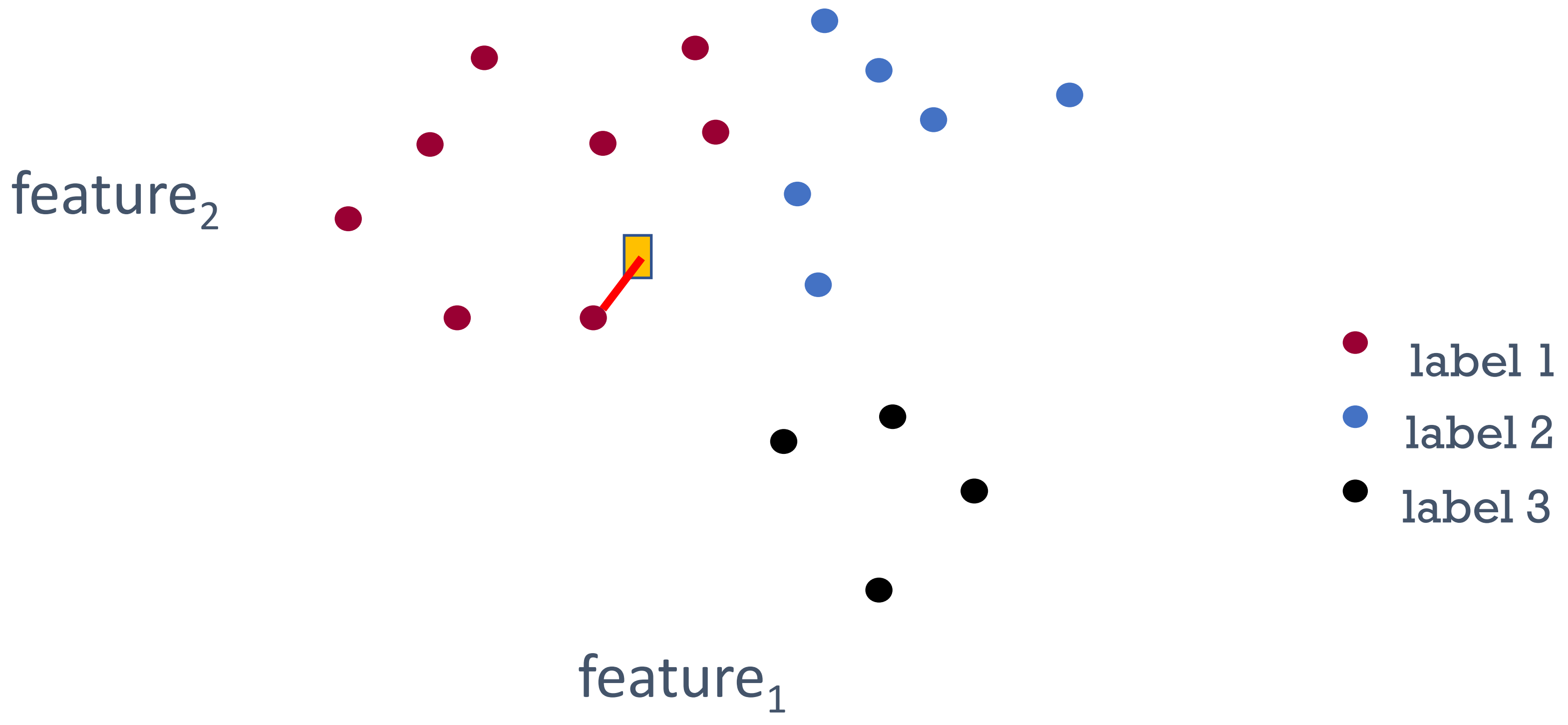
TEST SAMPLES – SCATTER PLOT

- Which class is the ■ from?

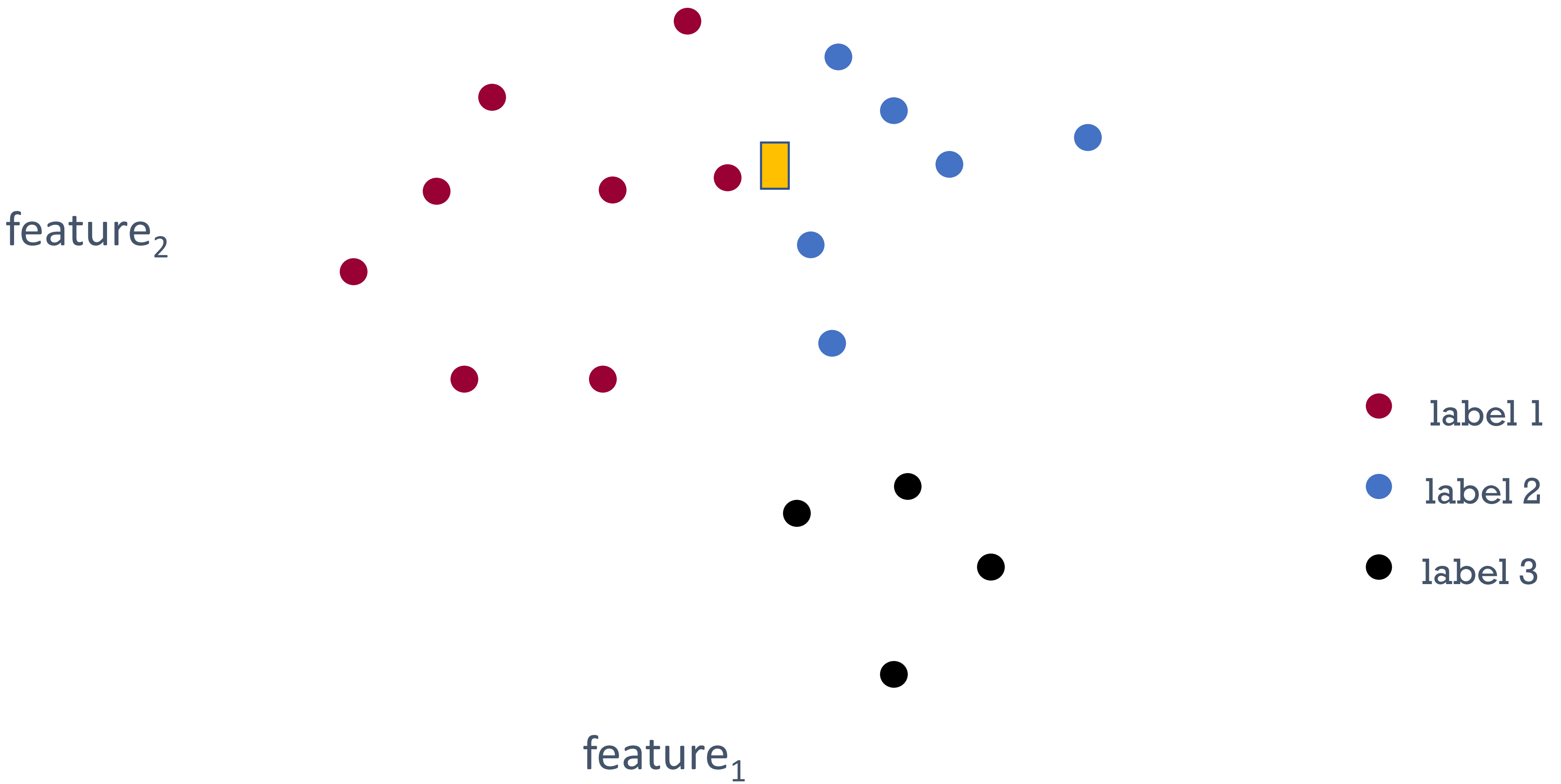


Finding Closest--a classification algorithm?

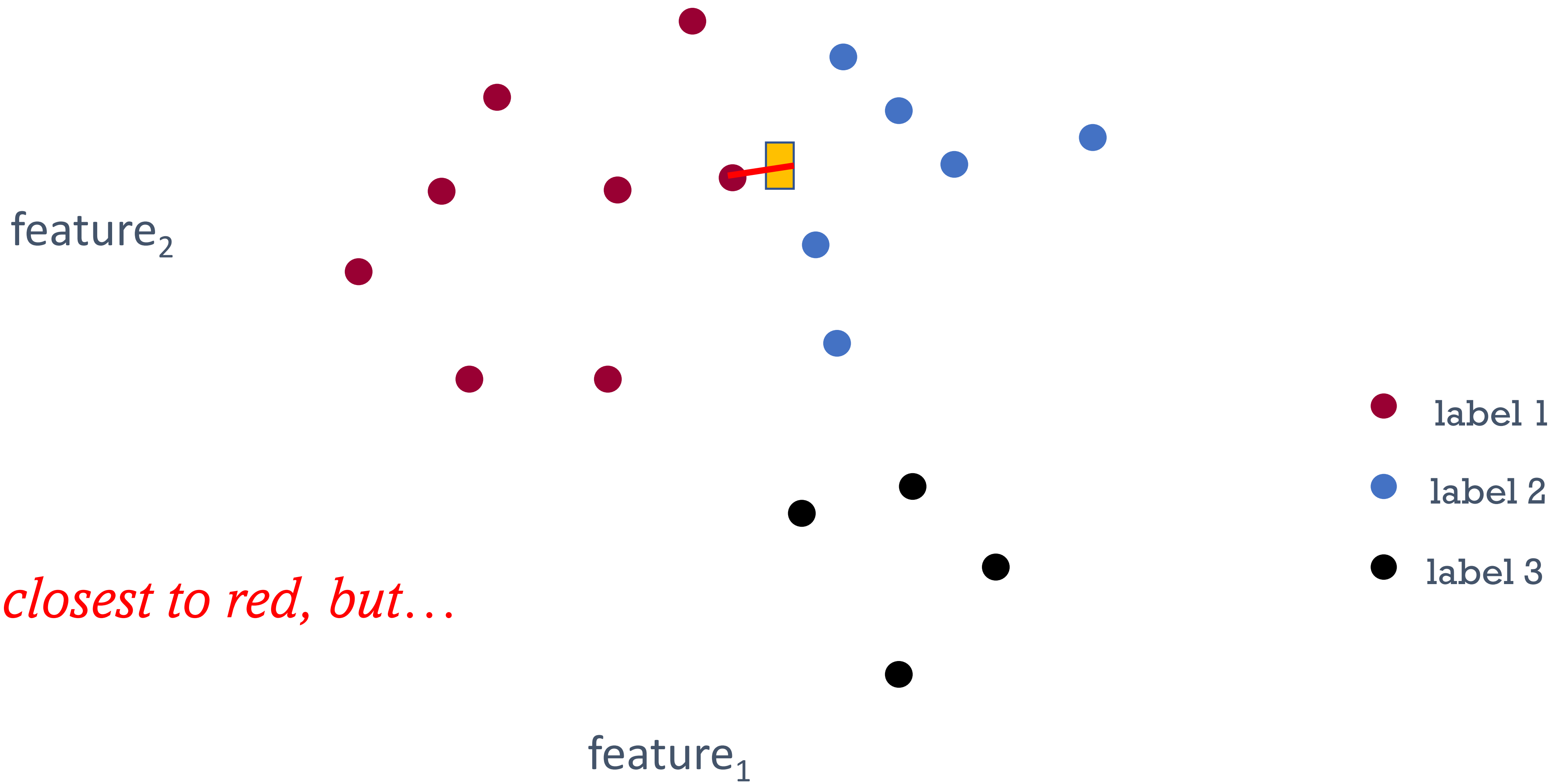
- To classify an example **d**:
  Label **d** with the label of the closest example to d in the training set



More challenging scenarios….

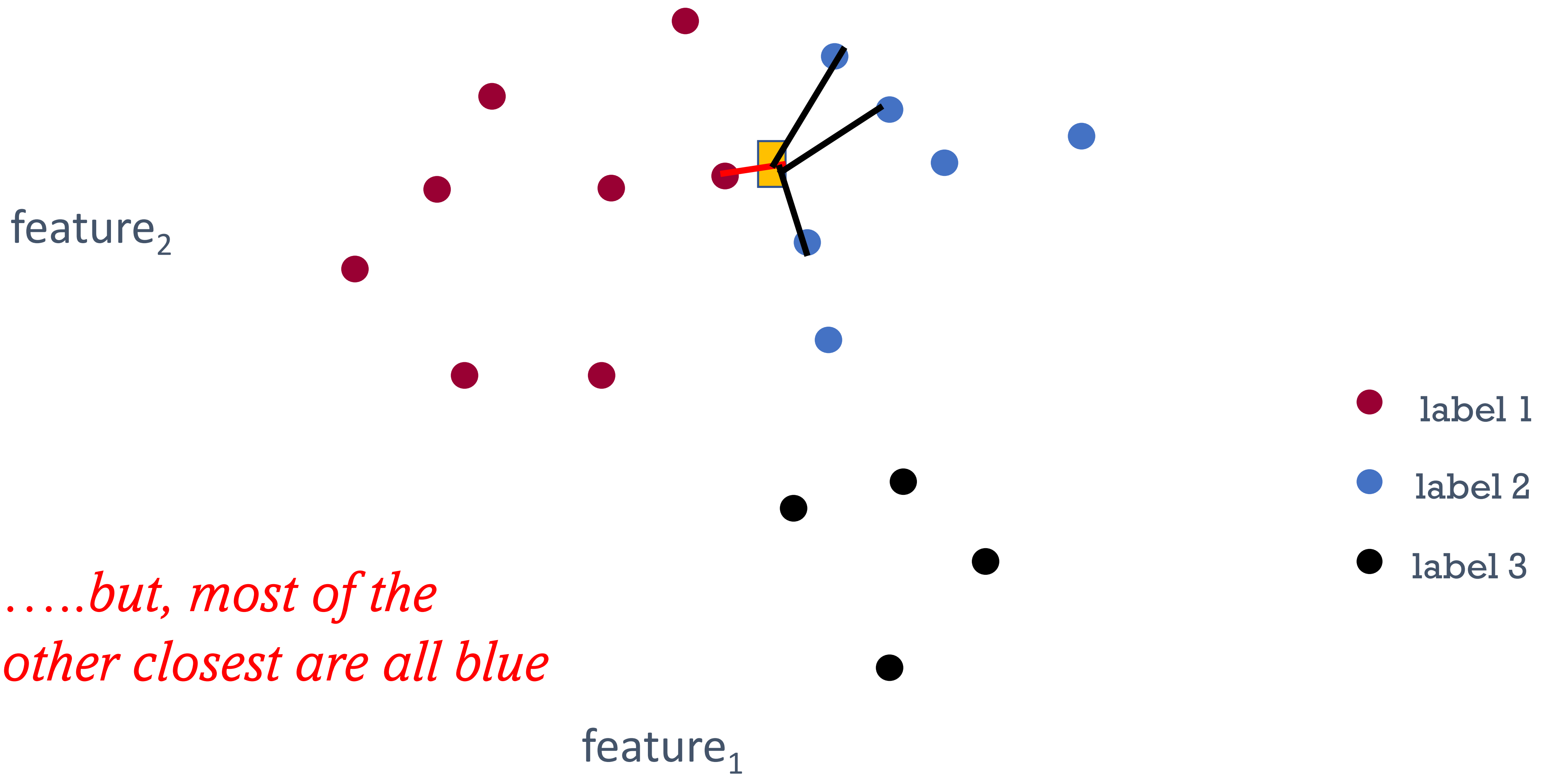


More challenging scenarios….



closest to red, but…

More challenging scenarios….



…..*but, most of the other closest are all blue*

k-Nearest Neighbor (k-NN)

- To classify an example **d**:
  - Find ***k*** nearest neighbors of **d**
  - Choose **d**'s label as the label which is the majority label within the k nearest neighbors

k-Nearest Neighbor (k-NN)

- To classify an example **d**:
  - Find ***k*** nearest neighbors of **d**
  - Choose **d**'s label as the label which is the majority label within the k nearest neighbors

How to calculate "nearest"?

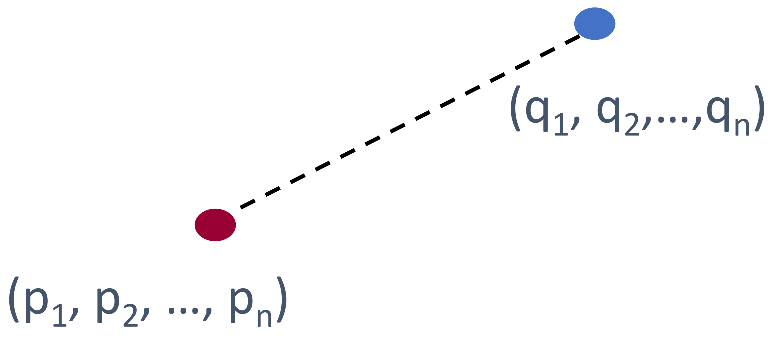
DISTANCE (SIMILARITY) FUNCTIONS

- Euclidean distance
- Mahalanobis distance
- …….

Measuring distance/similarity is a domain-specific problem and there are many, many different variations!

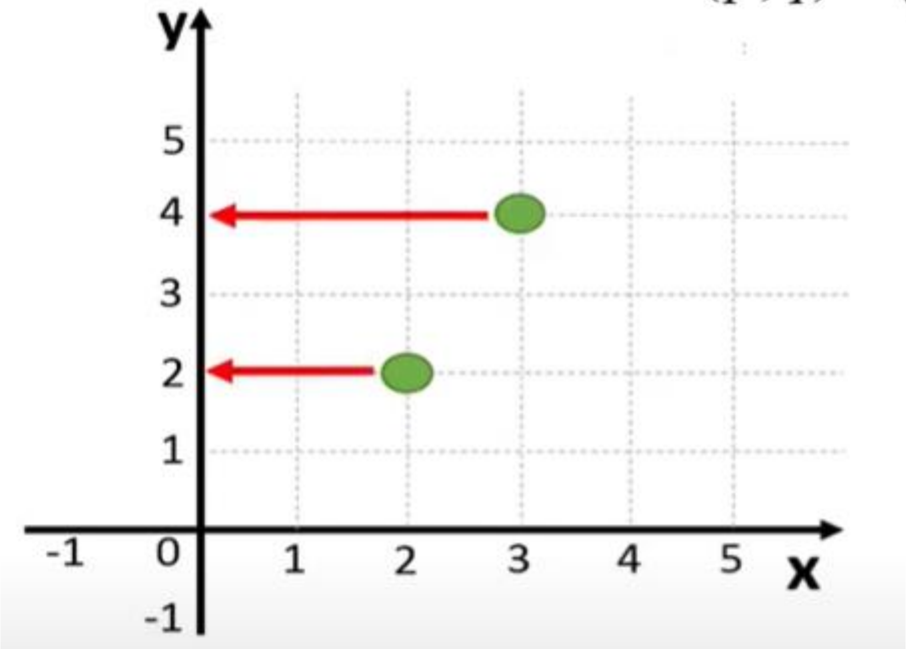
Euclidean distance

- Given two samples *p* and *q*, in n-dimensional feature space



$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + .. + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Euclidean distance- example



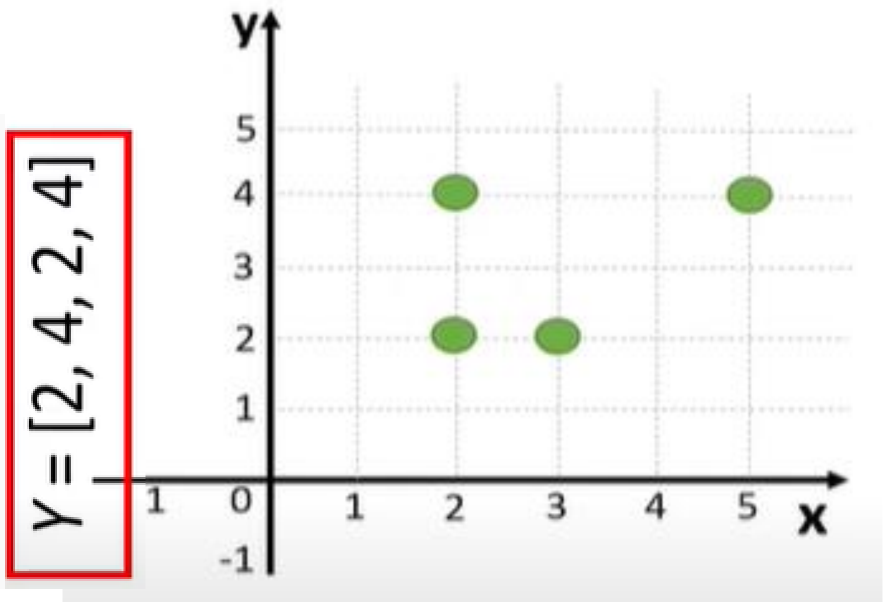
$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + .. + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

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

$$d = \sqrt{(2-3)^2 + (2-4)^2} = \sqrt{5} = \boxed{2.2}$$

Mahalanobis distance

- Centroid is the mean position of all data points in all directions.



$$X = [2,\ 2,\ 3,\ 5]$$

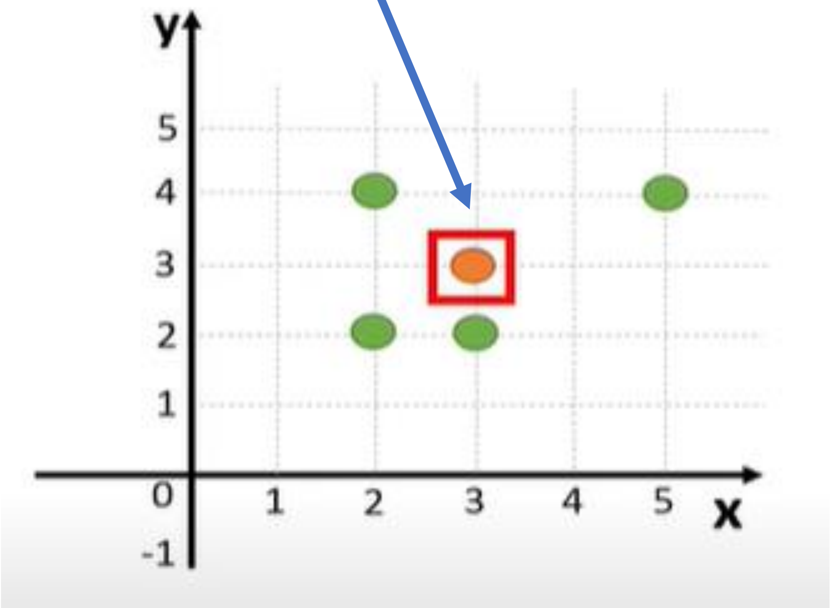
$$\boxed{\overline{X} = \frac{2+2+3+5}{4} = 3}$$

$$\boxed{\overline{Y} = \frac{2+4+2+4}{4} = 3}$$

$$\boxed{\text{Centroid} = \begin{pmatrix} \overline{X} \\ \overline{Y} \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}}$$

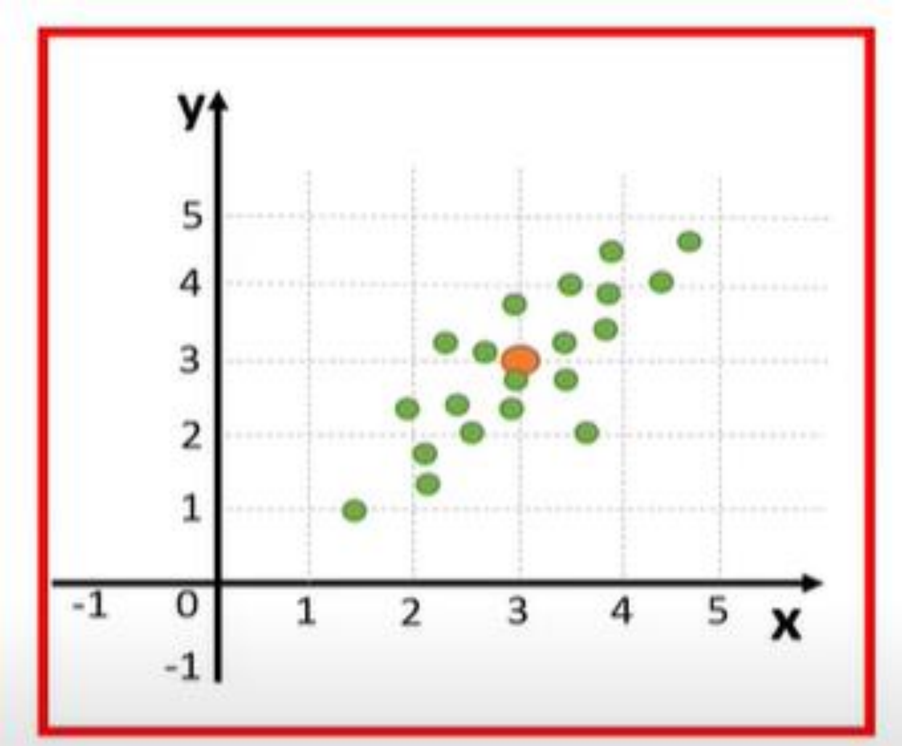
Mahalanobis distance

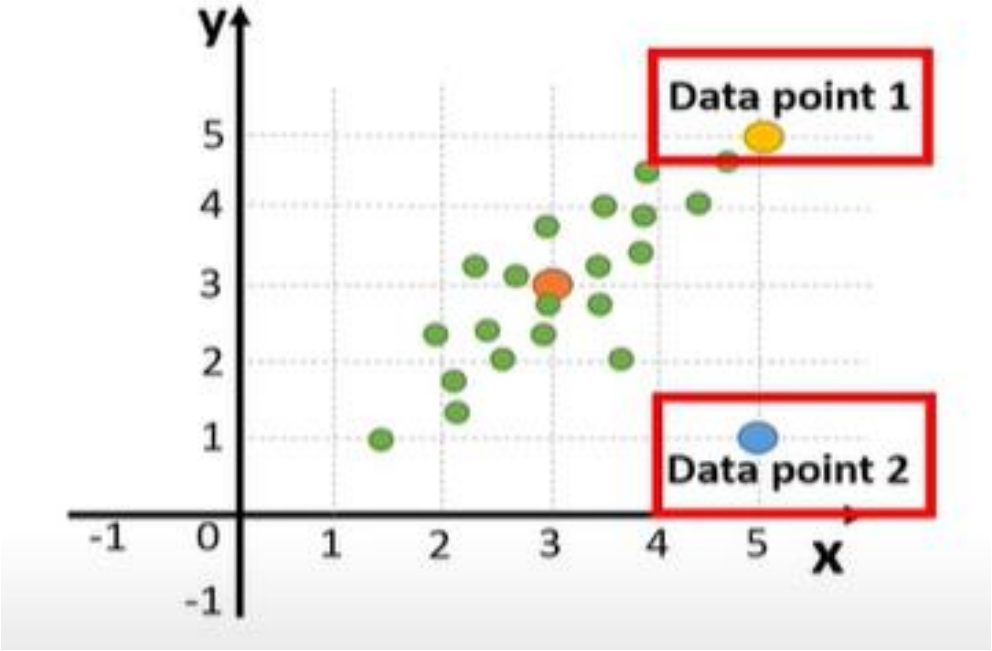
- Centroid is the mean position of all data points in all directions.



- The Mahalanobis distance is a distance measure between a point and a distribution.
- It takes into account the correlation between variables.

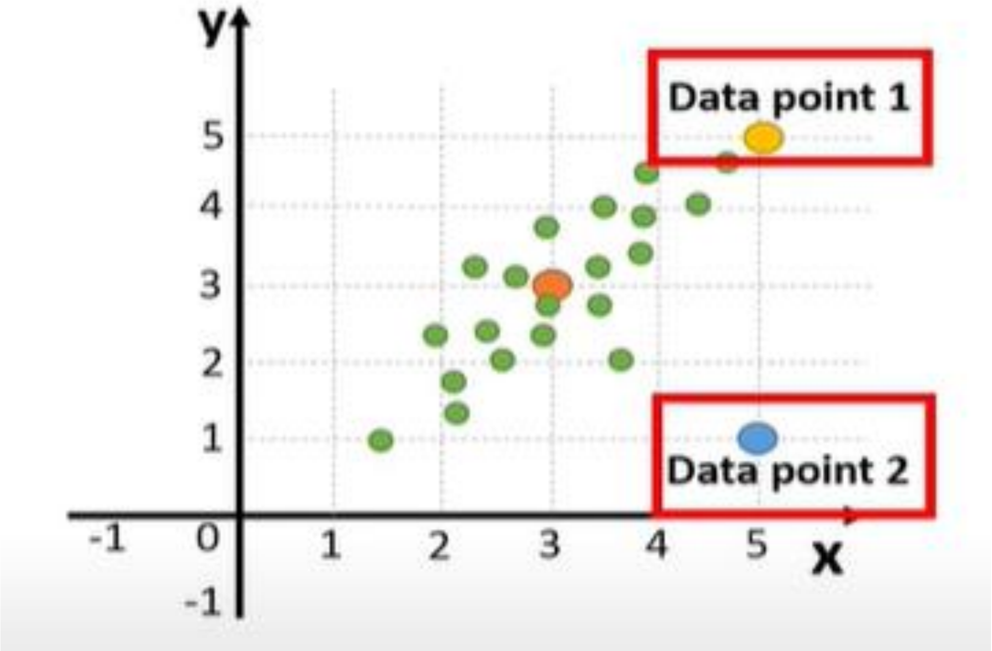
Mahalanobis distance





$$\text{Centroid} = \begin{pmatrix} \overline{X} \\ \overline{Y} \end{pmatrix} = \begin{pmatrix} 3.1 \\ 3.0 \end{pmatrix}$$

Mahalanobis distance



$$\text{Centroid} = \begin{pmatrix} \overline{X} \\ \overline{Y} \end{pmatrix} = \begin{pmatrix} 3.1 \\ 3.0 \end{pmatrix}$$

Euclidean distance between the centroid and data point 1:

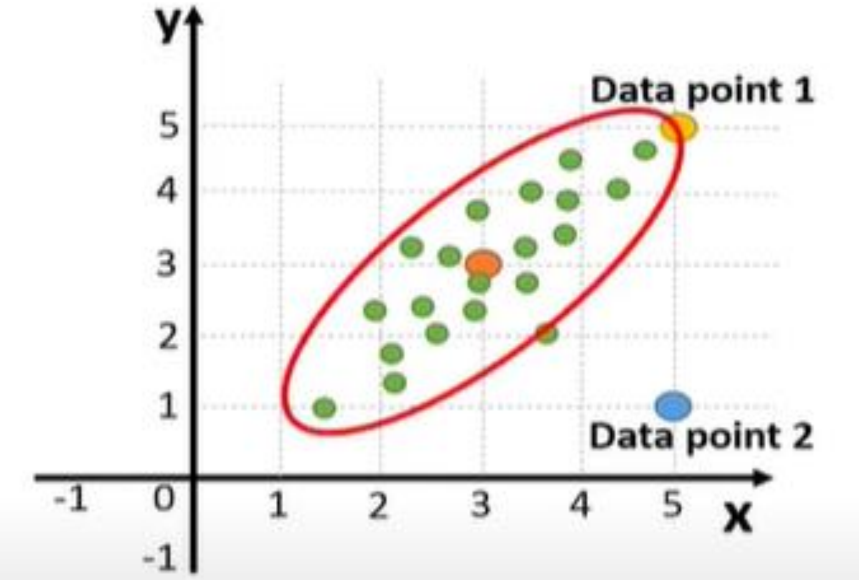
$$\boxed{d = \sqrt{(5-3.1)^2 + (5-3.0)^2} = 2.76}$$

Euclidean distance between the centroid and data point 2:

$$\boxed{d = \sqrt{(5-3.1)^2 + (1-3.0)^2} = 2.76}$$

- Based on Euclidean dist. both points have the same distance to the centroids

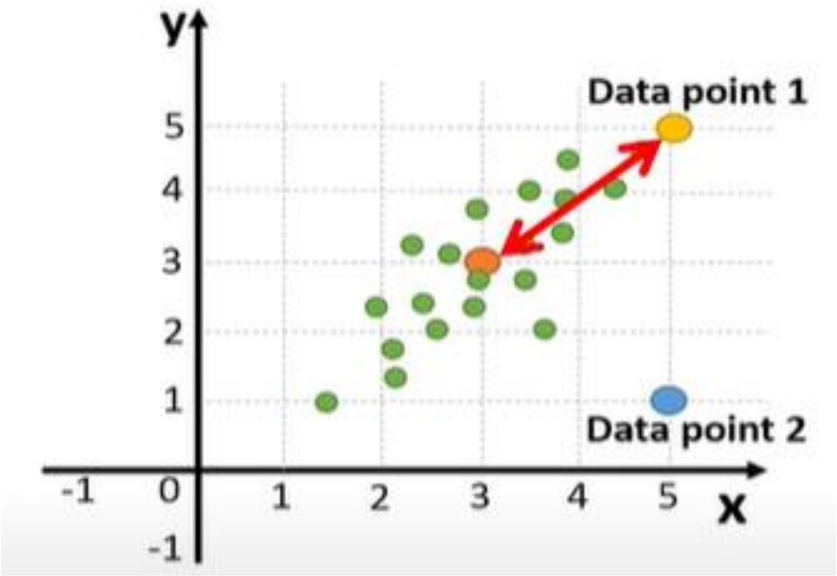
Mahalanobis distance

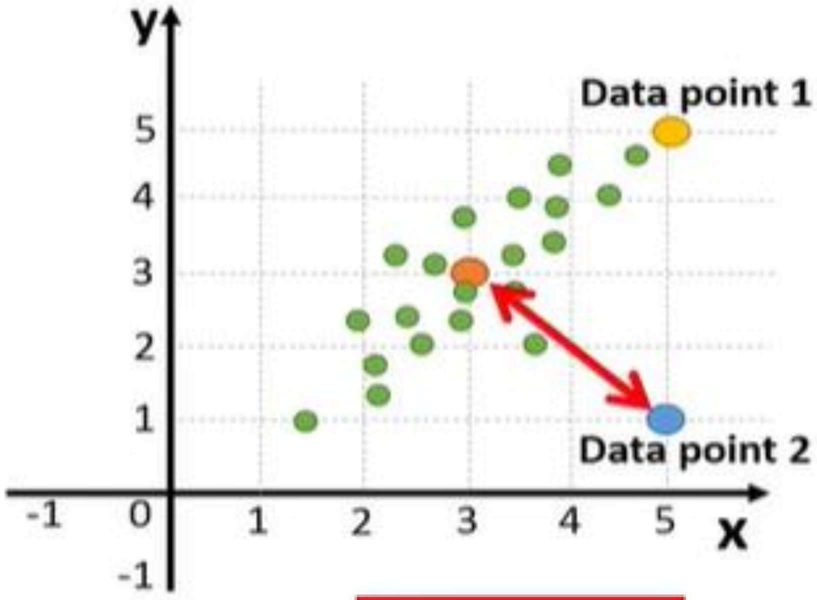


$$\text{Centroid} = \begin{pmatrix} \overline{X} \\ \overline{Y} \end{pmatrix} = \begin{pmatrix} 3.1 \\ 3.0 \end{pmatrix}$$

- However, if we put an ellipse around the data, we see that the data point 1 is much closer to the ellipse than the data point 2.
- Data point 2 does not seem to be a part of the data distribution around the centroid.

Mahalanobis distance





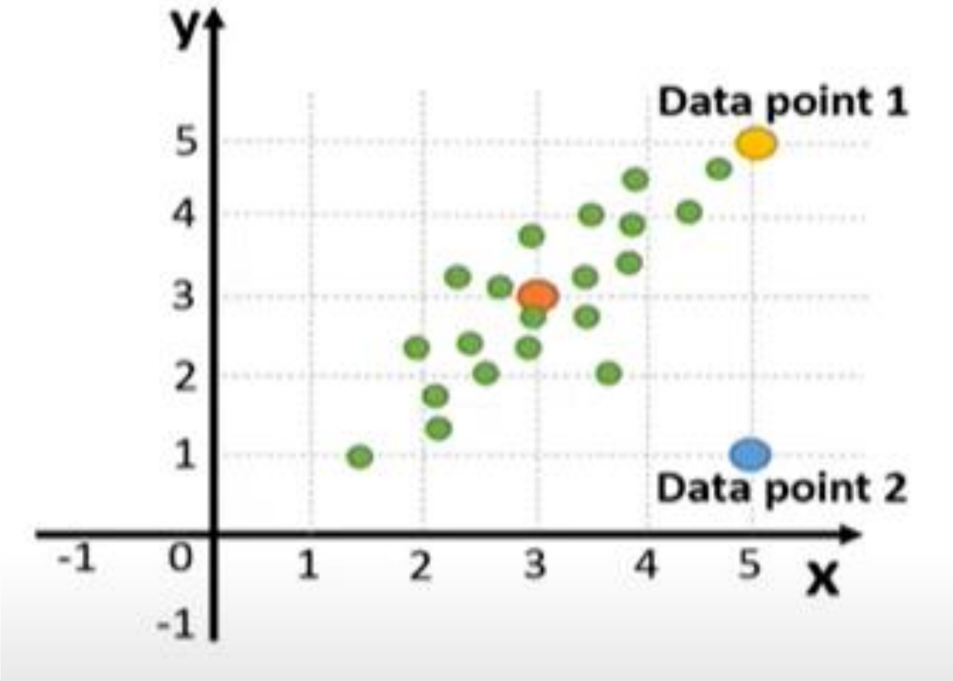
$$MD_1 = 2.26$$

$$MD_2 = 6.45$$

- Why MD_1 is so different from MD_2?

Bcs, MD takes into account the correlation in the data.

Mahalanobis distance



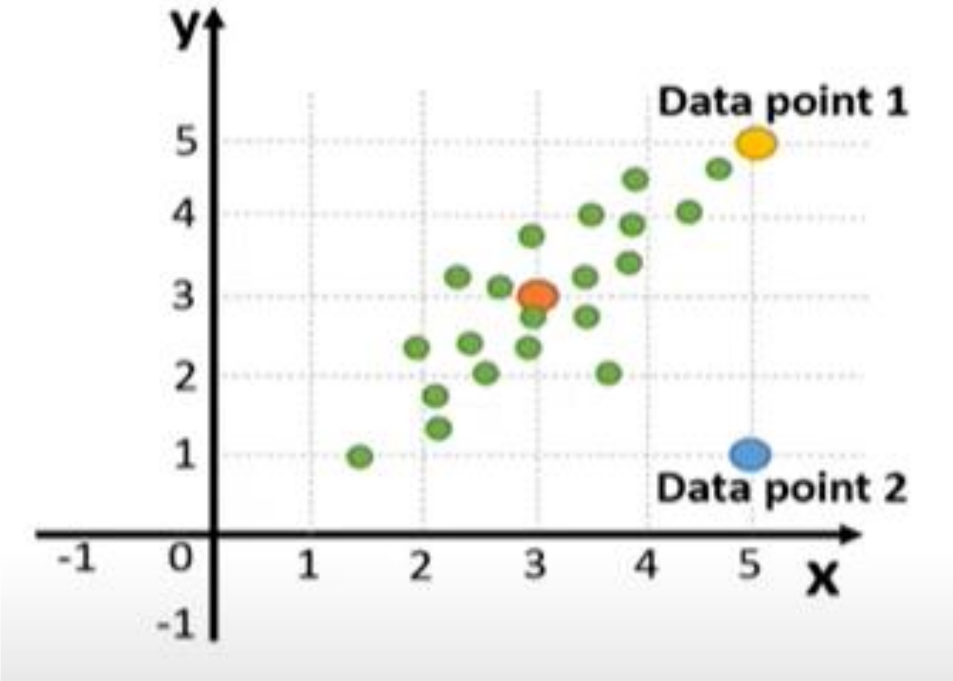
$$MD = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})}$$

mean (centroid)

Data coordinates

Covariance matrix

Mahalanobis distance



$$MD = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})}$$

$$MD = \sqrt{\begin{pmatrix} x - \bar{x} \\ y - \bar{y} \end{pmatrix}^T S^{-1} \begin{pmatrix} x - \bar{x} \\ y - \bar{y} \end{pmatrix}}$$

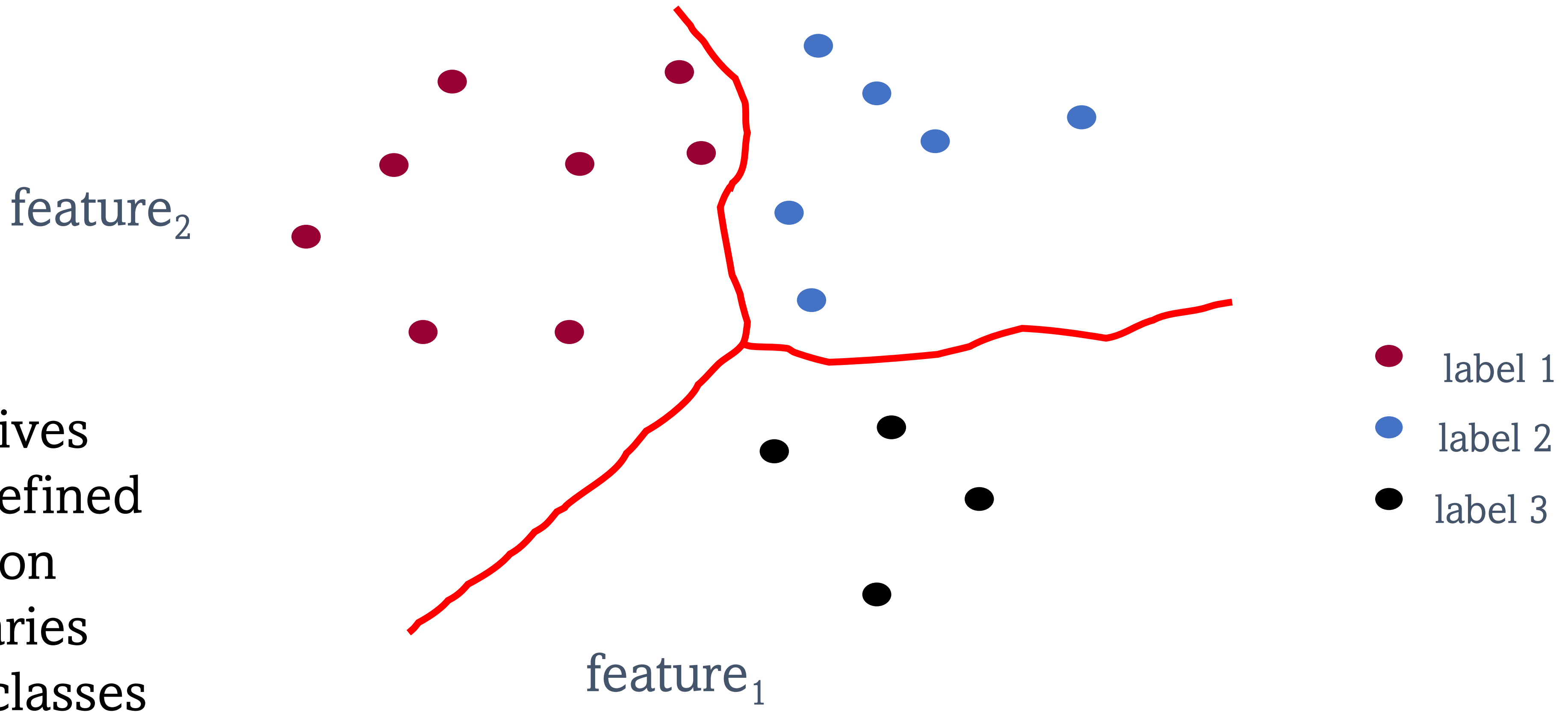
- Covariance matrix is computed based on the green data points (similar to how we computed the centroid).
- Then, calculate the inverse of the covariance matrix.

Decision boundaries

- Are places in the features space where the classification of a point/example changes.

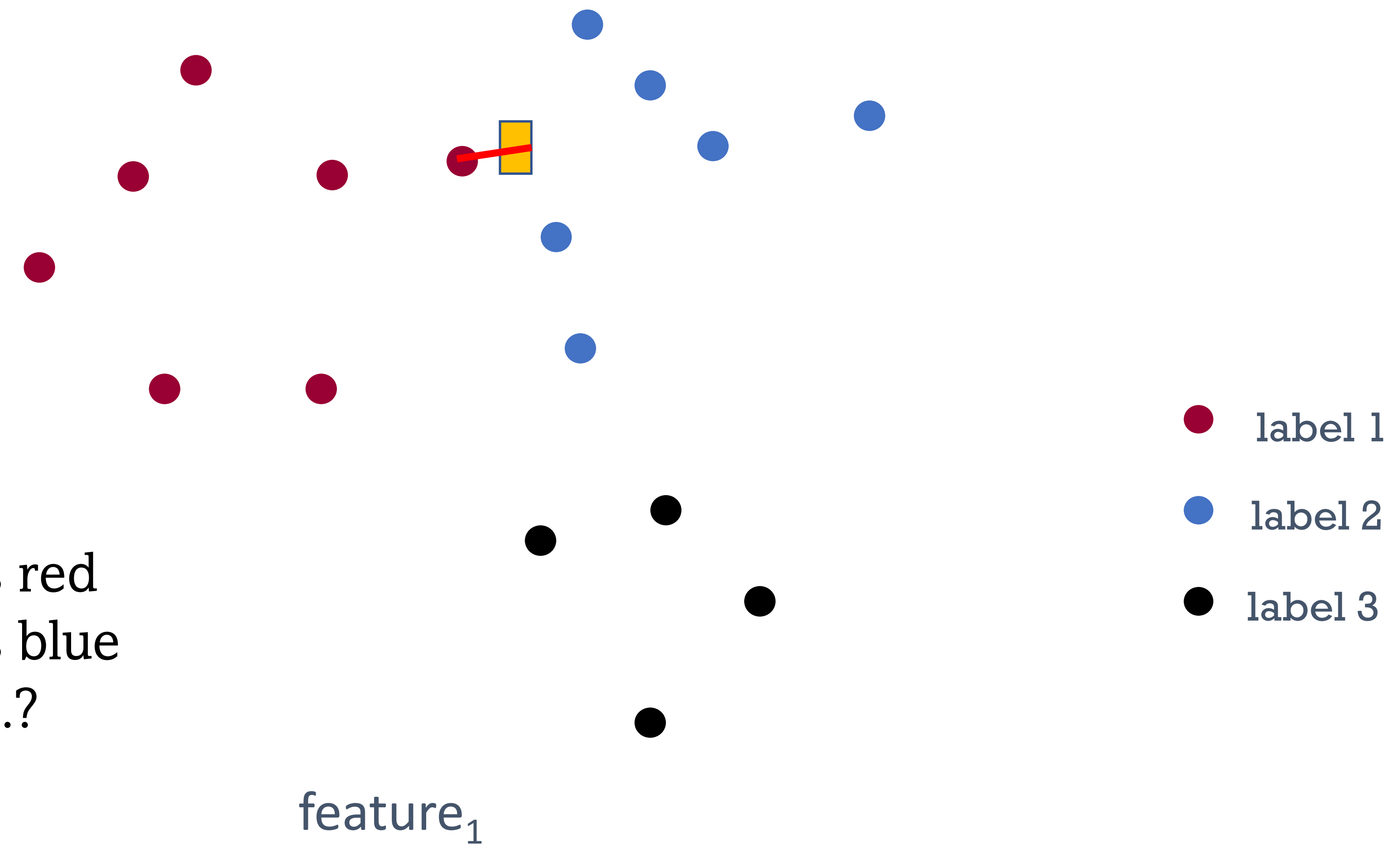
Decision boundaries

- Where are the decision boundaries for k-NN?



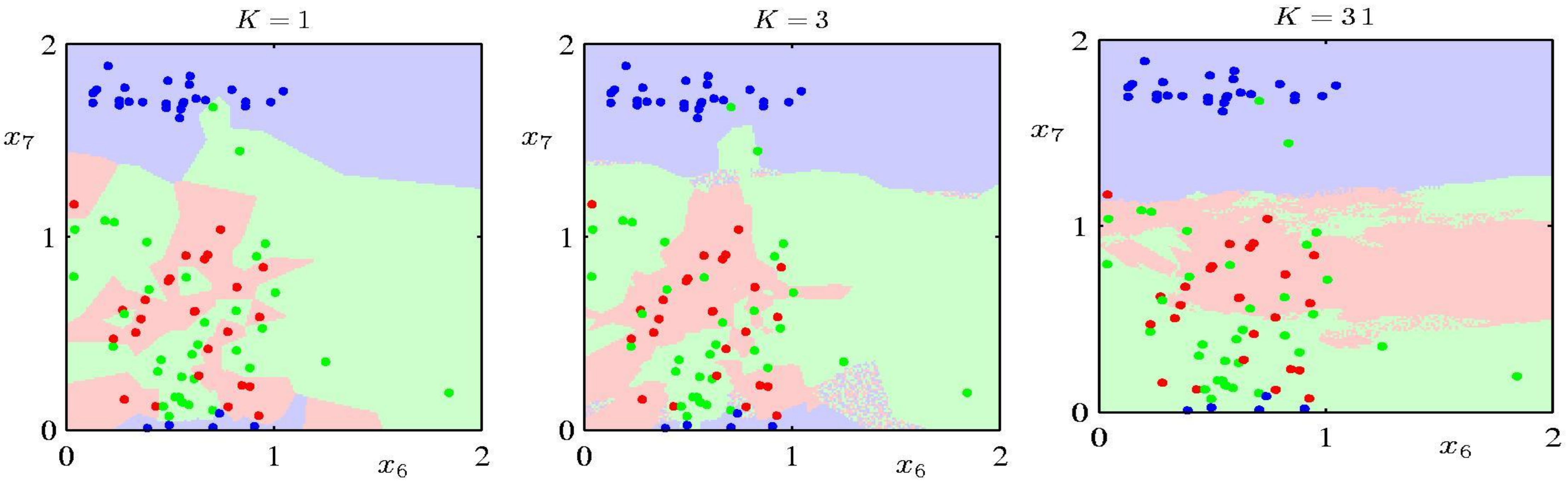
k-NN gives **locally** defined decision boundaries between classes

How to decide K of K-NN?



k=1; test sample is red
k=3; test sample is blue
What about k=8….?
blue

How to decide K of K-NN?



What is the role of *k*?
How does it relate to overfitting and underfitting?

How to decide k of k-NN?

Common heuristics:

- often 3, 5, 7
- choose an odd number to avoid ties

Use your training and validation data to decide *k*.

- Change *k* from *k=1* until the validation performance decreases.
- Pay attention not to underfit and overfit the data.
- Finalize the decision of *k* and use the same *k* to classify the test data.

K-NN– PROS. & CONS.

- Pros:
  - Almost no assumptions about the data
    - Smoothness: nearby regions of space are from the same class
    - Assumptions implied by distance function (only locally!)
    - Non-parametric approach
      - Nothing to infer from the data, except k
      - Easy to update in online setting: just add new item to training set

K-NN– PROS. & CONS.

- Cons:
  - Need to handle missing data: fill-in or create a special distance
  - Sensitive to class-outliers (e.g., mislabeled training instances)
  - Sensitive to lots of irrelevant attributes (affect the distance)
  - Computationally expensive:
    - Space: need to store all training examples
    - Time: need to compute distance to all example: O(nd)
      - n: #number of training examples
      - d: cost of computing distance

Coding Hint

- See the coding examples and tutorials given in

https://www.w3schools.com/python/python_ml_knn.asp

https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn

Cigdem Beyan
cigdem.beyan@unitn.it
https://cbeyan.github.io/