UNIVERSITÀ di VERONA

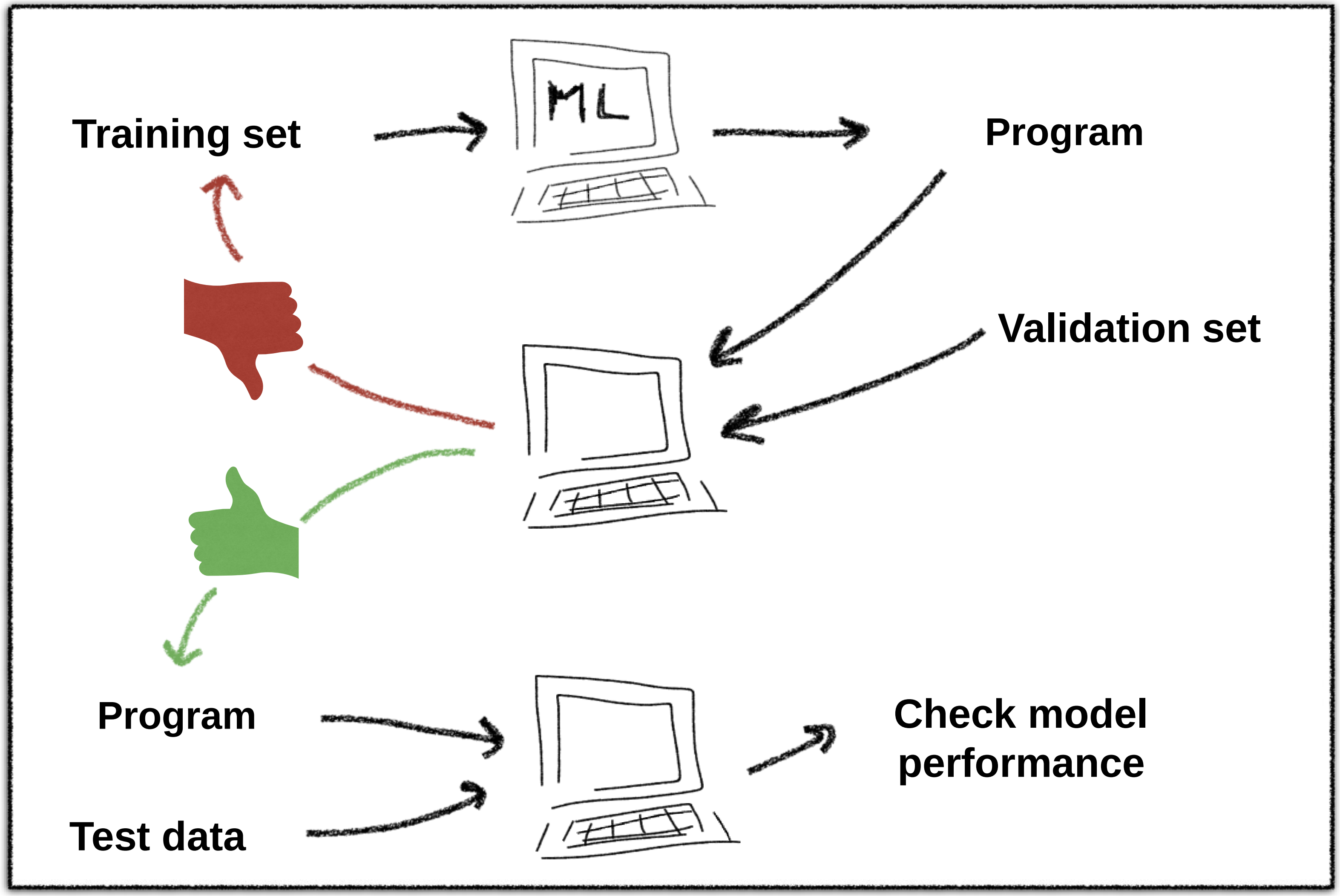
Machine Learning

a)Model Evaluation For Classification & Regression
b) regression Methods
c)Bayes Decision theory

Cigdem Beyan

A. Y. 2024/2025

Recap--train / validation / test sets



Training set

Program

Validation set

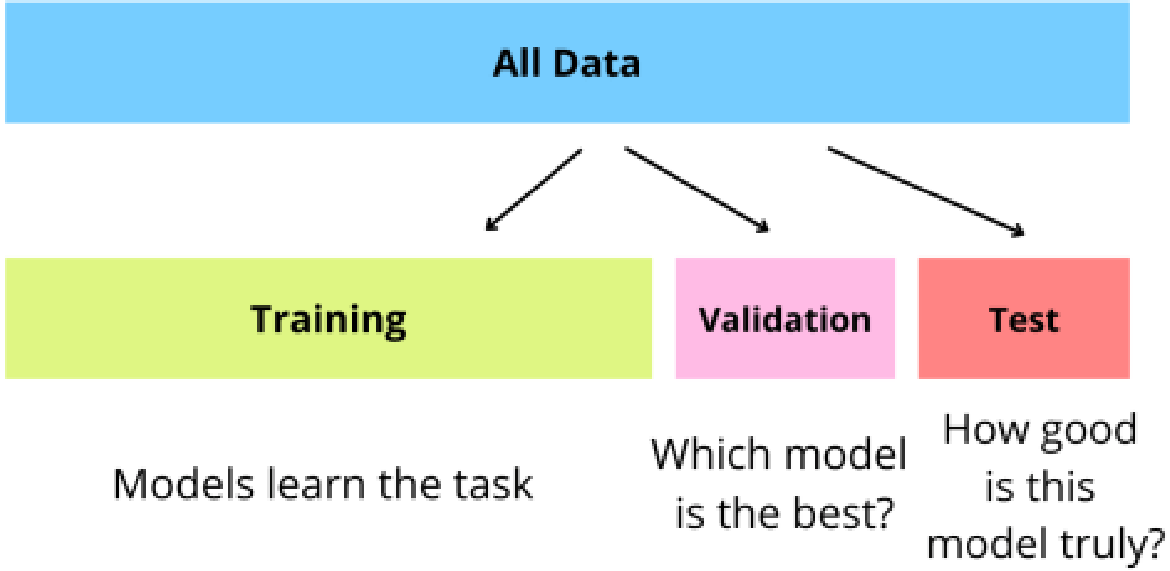
Program

Test data

Check model performance

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

- In cases where we don't have enough data to randomly split between training (e.g., 60% of the total data), validation (e.g., 20% of the total data), and testing (e.g., 20% of the total data), we need to use cross-validation.

- Cross-validation involves reusing a portion of the training set itself to validate performance by retraining not just one, but *N models*.

- Performing cross-validation therefore requires *N training sessions* and can be more computationally demanding.

Cross validation

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

- If there are 2 folds, you have 2 models trained.
  - 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.

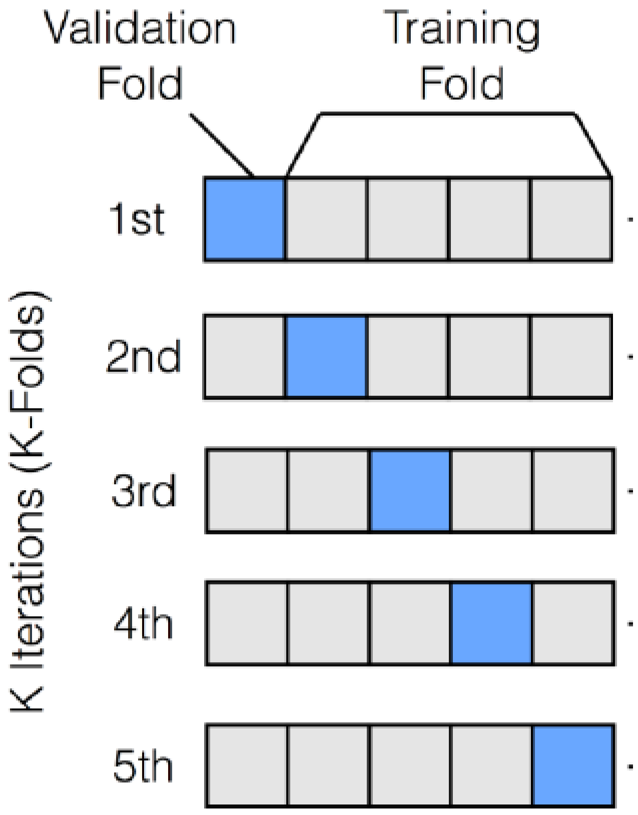
- Every sample is in both training and testing but not at the same time.
  - Reduces the chances of getting a biased training set.

K-fold Cross Validation

- In the case of k-fold cross-validation, we split our dataset into k groups, perform training k times on $(N/k)(k-1)$ data, and measure performance on the last N/k data.

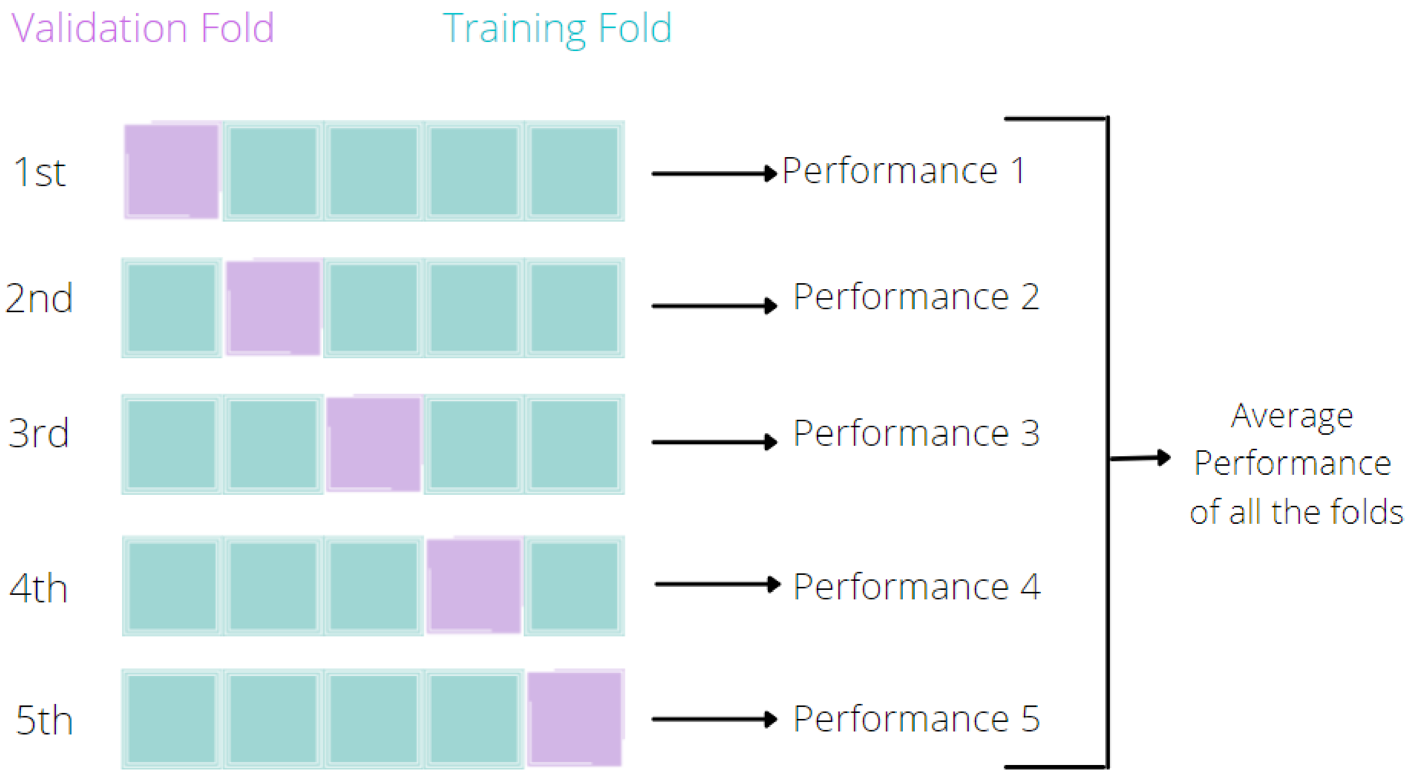
- Our cross-validation performance will be the average of the performance of the k tests.

- Training set and validation set cannot overlap but the sum of training and validation data is a constant.



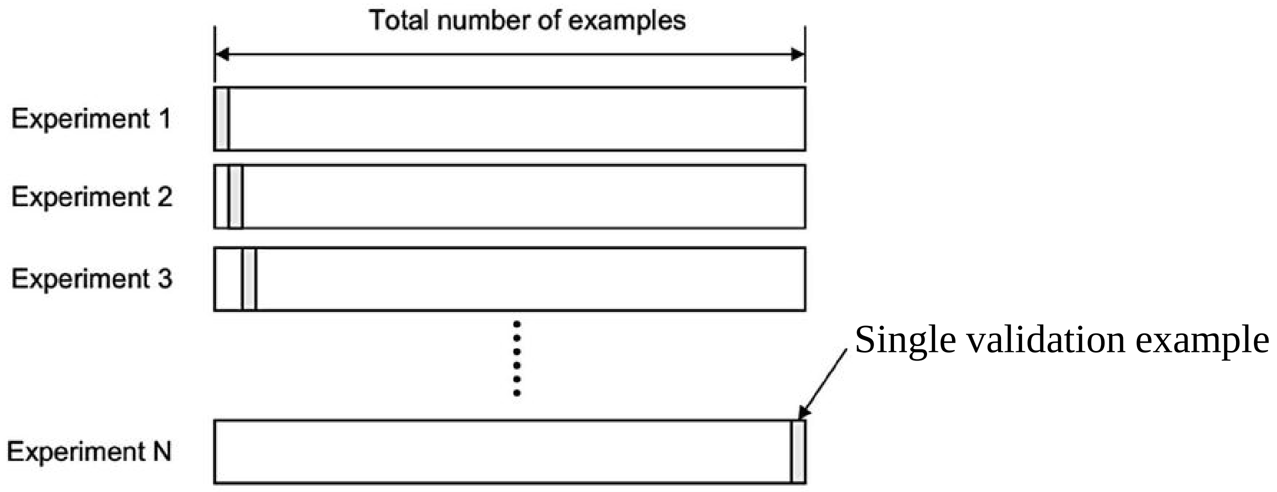
Example: 5-fold cross validation

- 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

- In case we have very few data
- *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



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

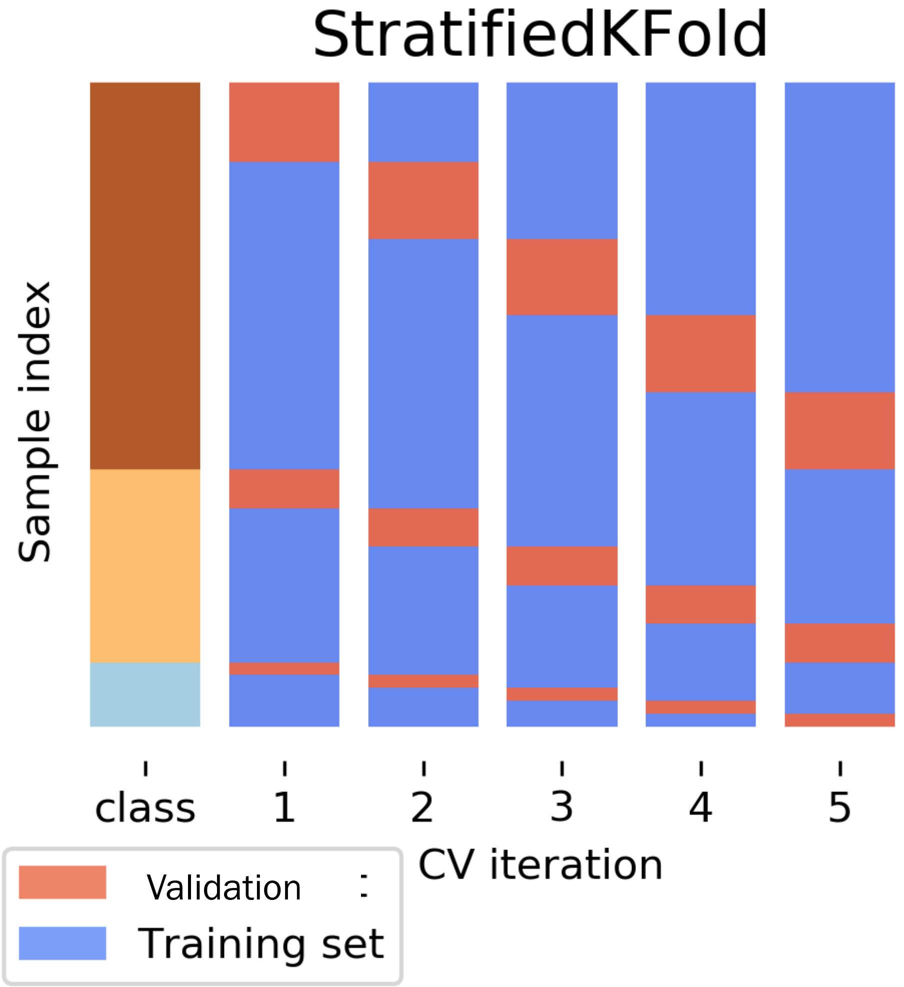
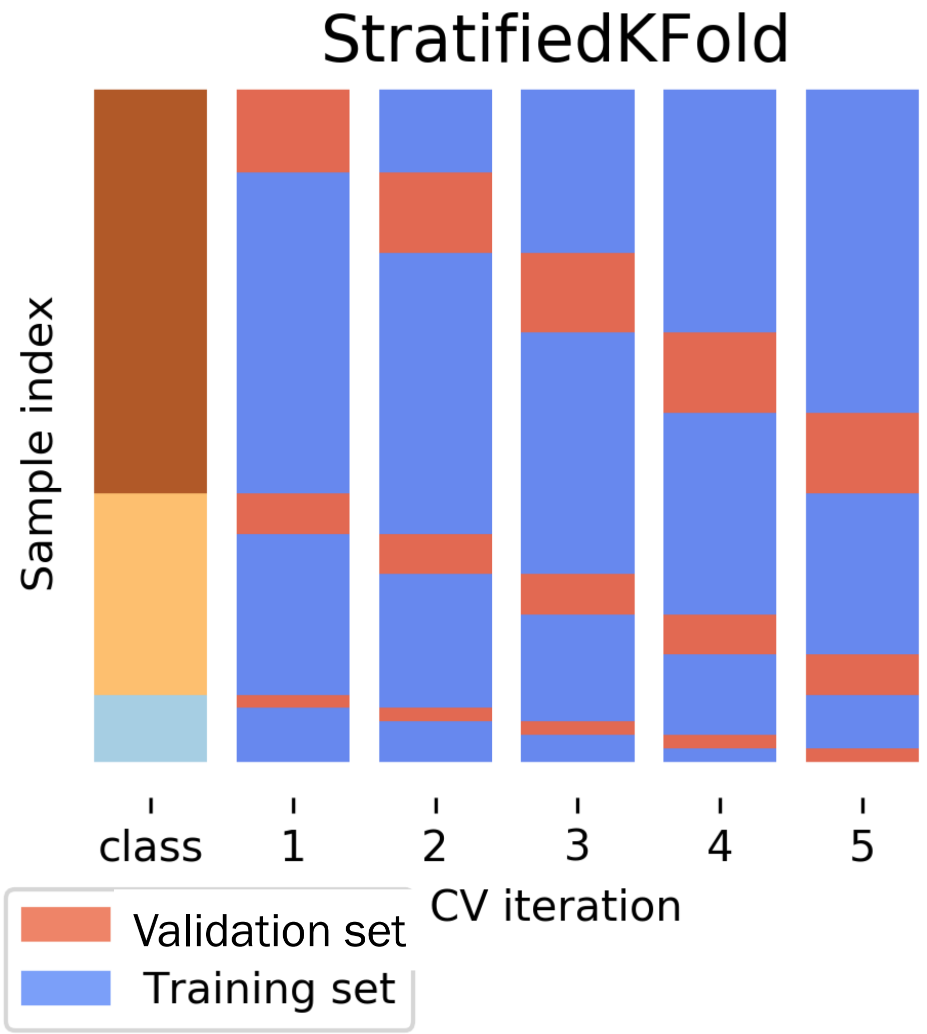


Image credit: https://amueller.github.io/aml/04-model-evaluation/1-data-splitting-strategies.html

Stratified Cross Validation



Let's consider a scenario with an imbalanced dataset containing 300 samples, divided into three classes (A, B, and C), where Class A has more samples than the other two. We'll still perform Stratified Cross Validation with k=5.

Imbalanced Dataset:

Class A: 200 samples
Class B: 50 samples
Class C: 50 samples

Split the dataset into five folds with Stratification:

Fold 1: 60 samples (Class A: 40, Class B: 10, Class C: 10)
Fold 2: 60 samples (Class A: 40, Class B: 10, Class C: 10)
Fold 3: 60 samples (Class A: 40, Class B: 10, Class C: 10)
Fold 4: 60 samples (Class A: 40, Class B: 10, Class C: 10)
Fold 5: 60 samples (Class A: 40, Class B: 10, Class C: 10)

Train and validate:

Iteration 1: Train on Folds 1, 2, 3, and 4, Validate on Fold 5
Iteration 2: Train on Folds 2, 3, 4, and 5, Validate on Fold 1
Iteration 3: Train on Folds 1, 3, 4, and 5, Validate on Fold 2
Iteration 4: Train on Folds 1, 2, 4, and 5, Validate on Fold 3
Iteration 5: Train on Folds 1, 2, 3, and 5, Validate on Fold 4

Average the performance metrics:

Calculate performance metrics for each iteration.
Average these metrics to evaluate the model's performance robustly.

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

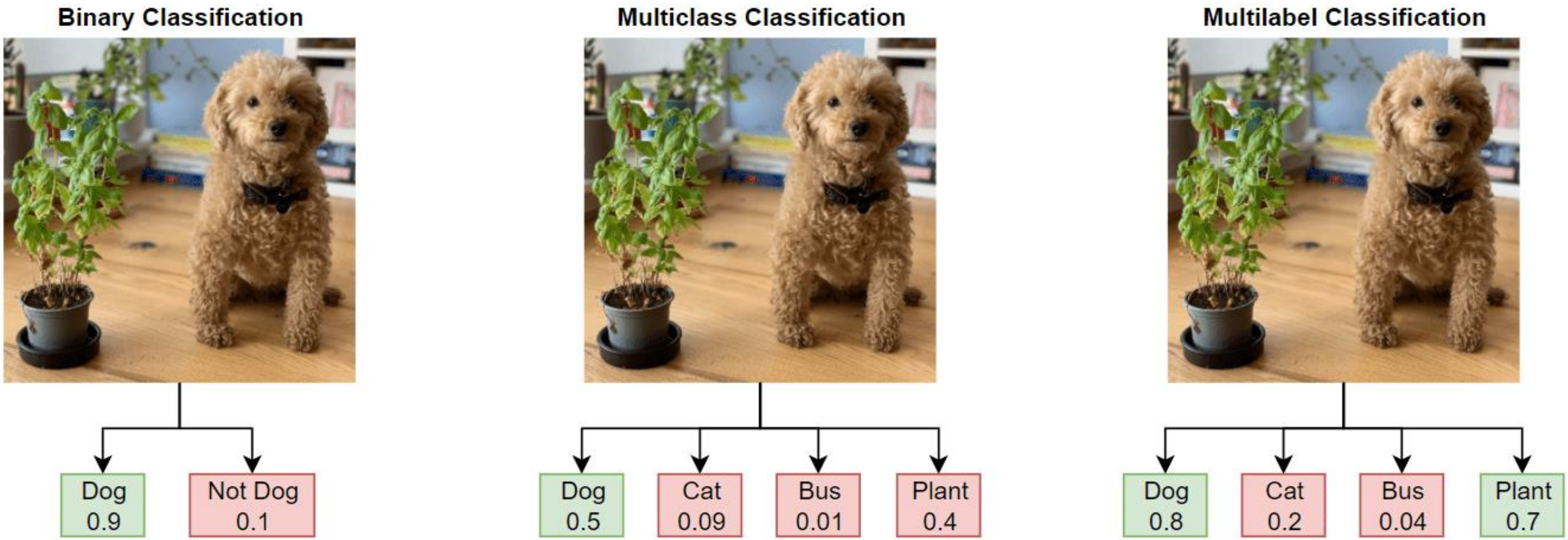
- Regression:
  - How close is our model to what we are trying to predict

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

Evaluation measures - classification

- A classification model will have a set of features as the input and as output one or more classes.
- Classification can be
  - **Binary:** in the case of two classes (0/1) or one vs rest (e.g., apples vs other fruits)
  - **Multi-class:** in the case of having *N classes* to choose from (e.g., cat, dog, wolf, etc.)
  - Moreover, a problem can also be **multi-label** in the case where our example can belong to multiple classes simultaneously.

Evaluation measures - classification



Binary Classification

Dog 0.9

Not Dog 0.1

Multiclass Classification

Dog 0.5

Cat 0.09

Bus 0.01

Plant 0.4

Multilabel Classification

Dog 0.8

Cat 0.2

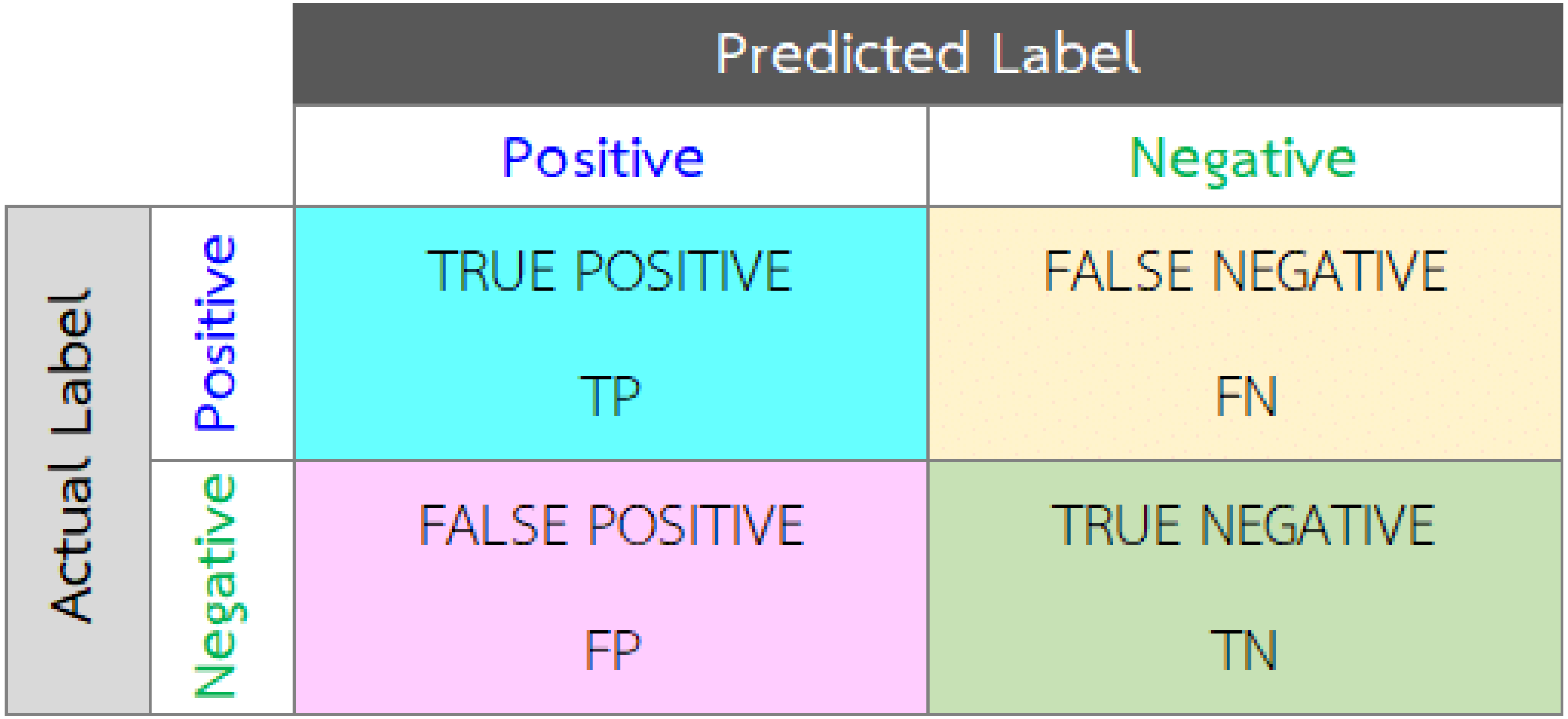
Bus 0.04

Plant 0.7

Image credit: Mathworks

Classification evaluation measures

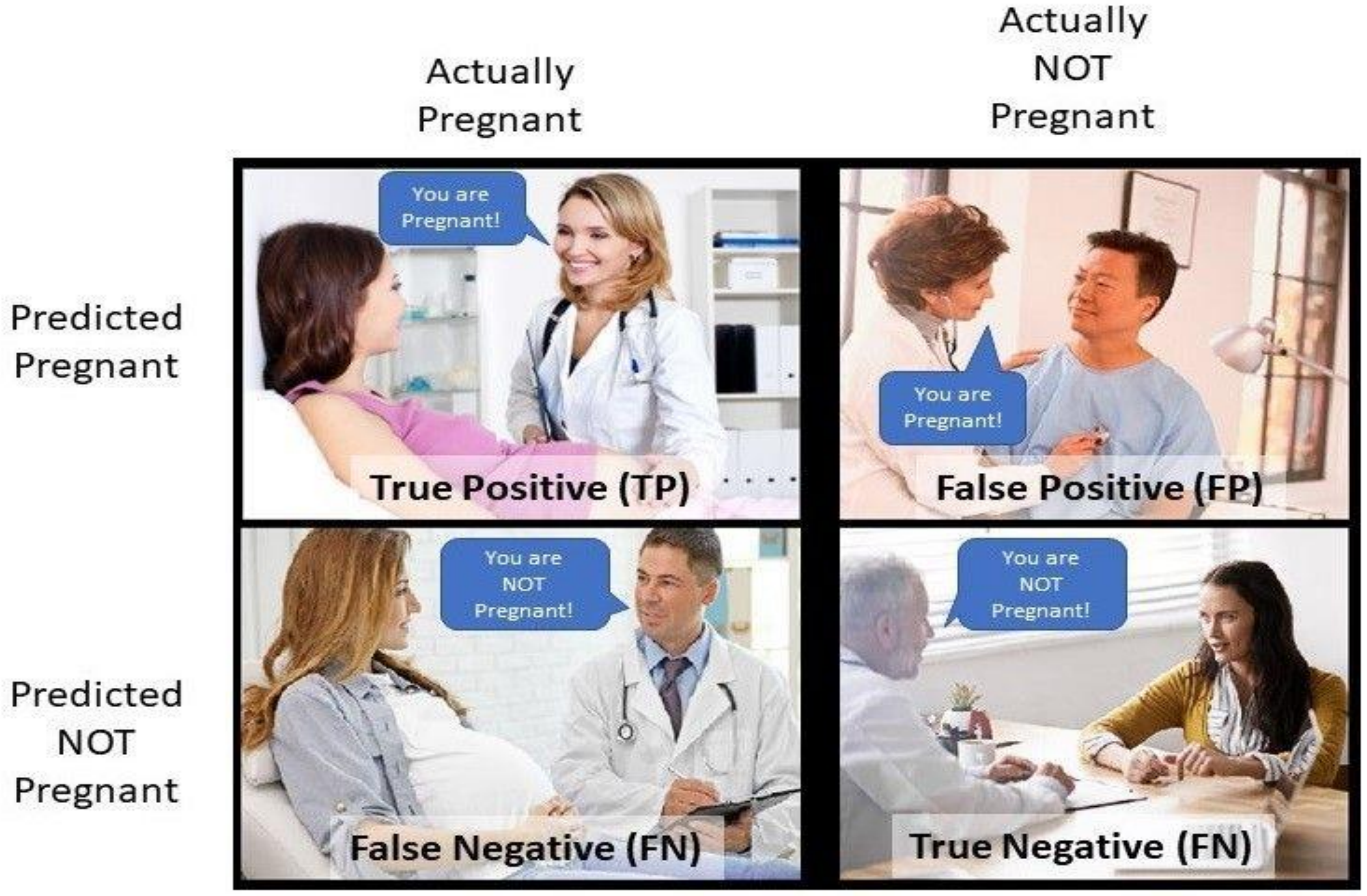
Confusion matrix for binary classification



		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



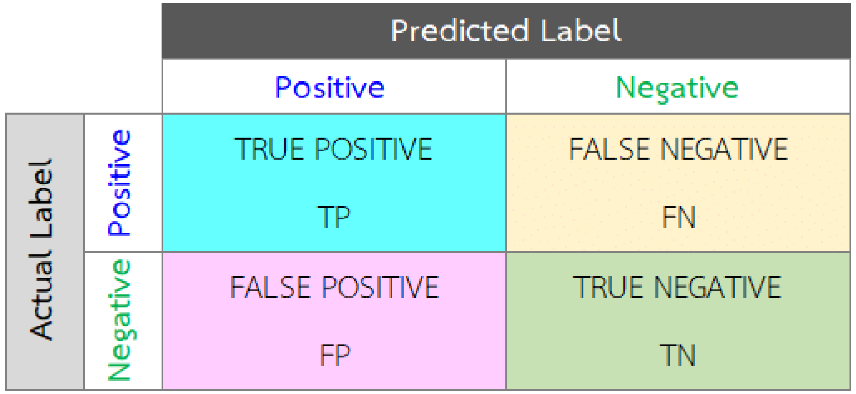
Confusion Matrix

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

Classification evaluation measures

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

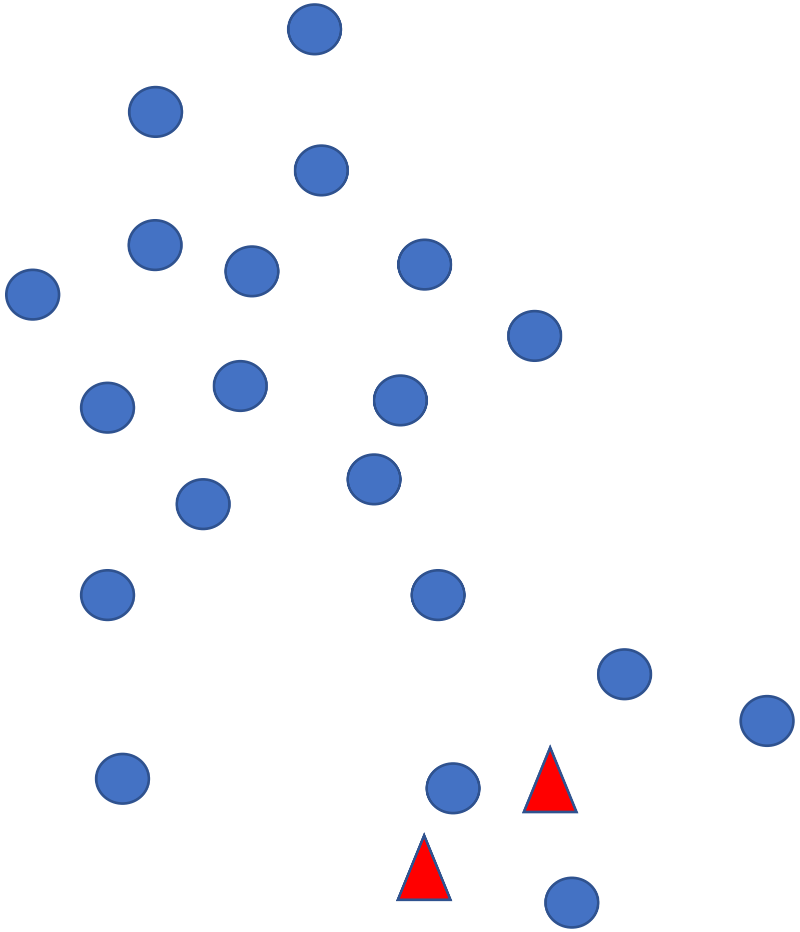
Accuracy= (1-Error)= (TP+TN) / (TP+TN+FP+FN)



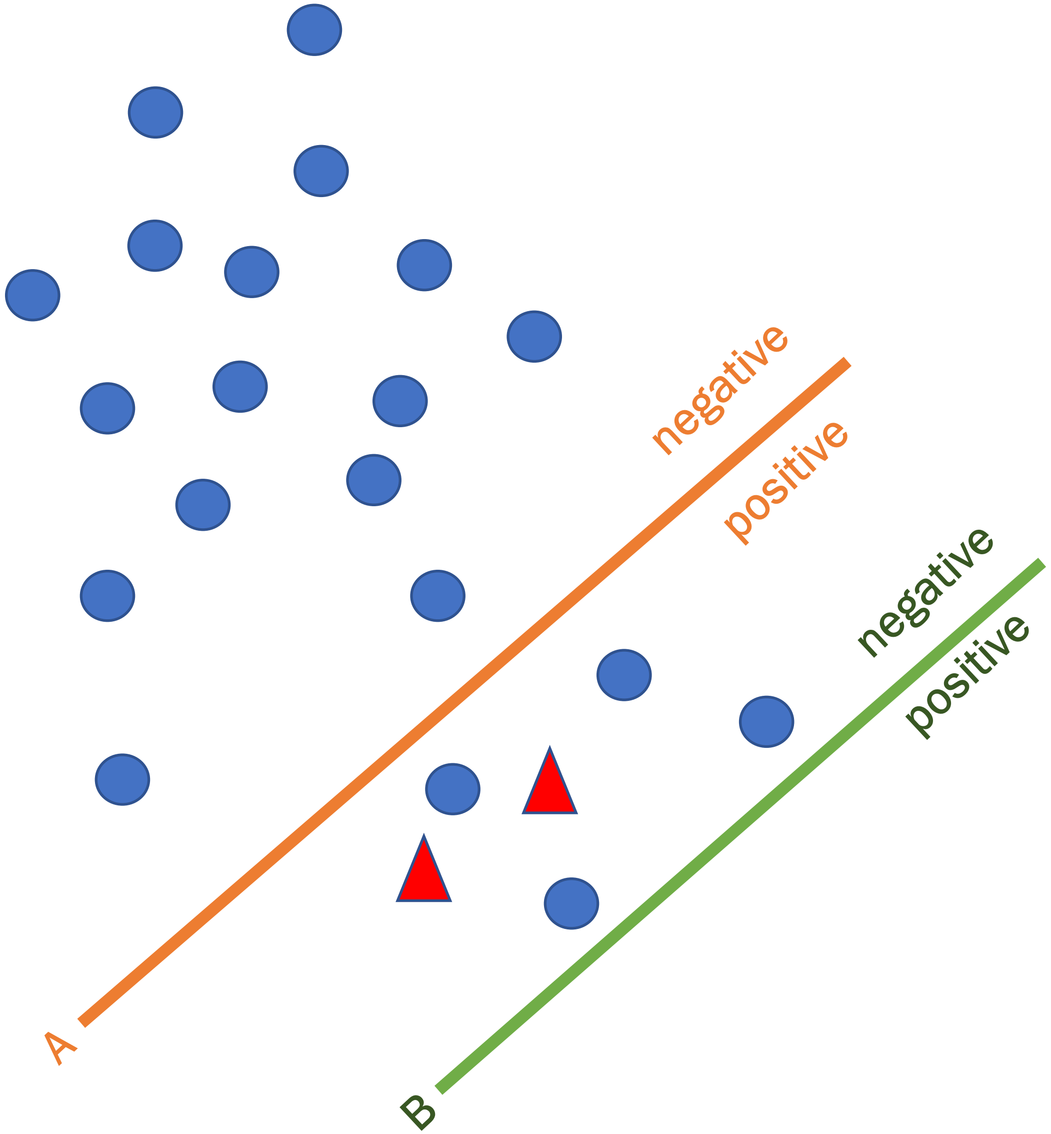
		Predicted Label	
		Positive	Negative
Actual Label	Positive	TRUE POSITIVE TP	FALSE NEGATIVE FN
	Negative	FALSE POSITIVE FP	TRUE NEGATIVE TN

- 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.

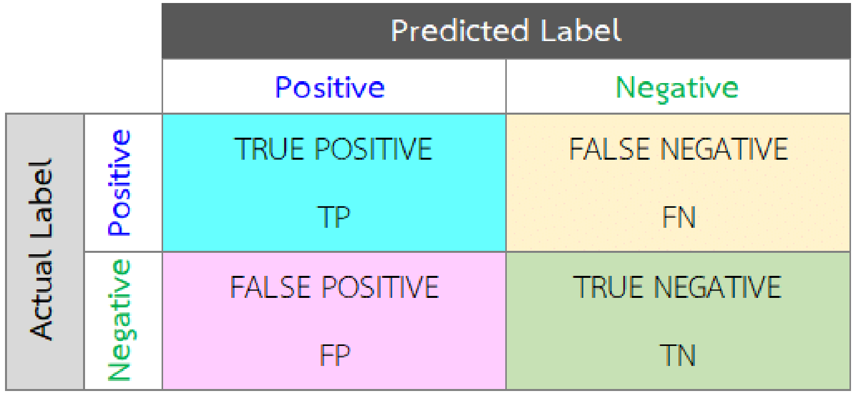
Misses and False alarms

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

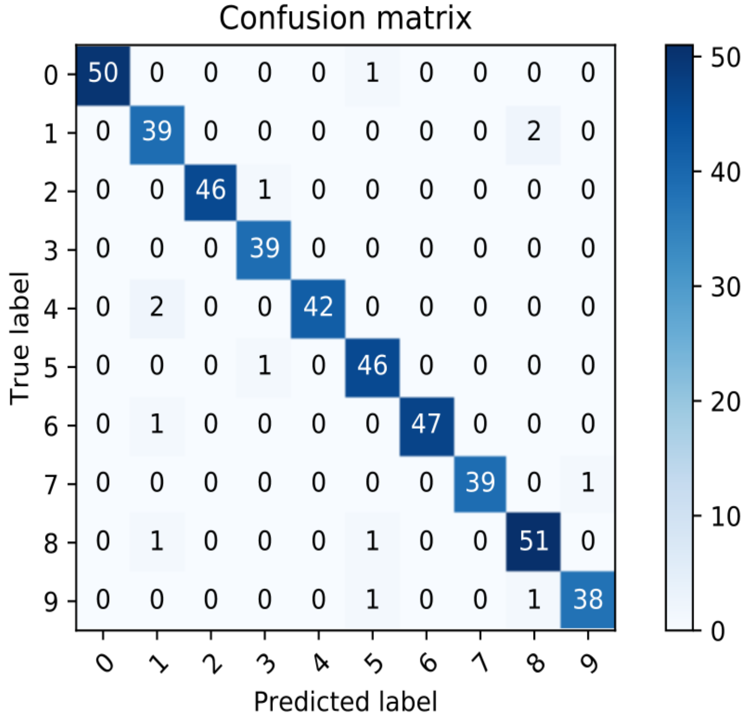
$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

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

- n_{ij} is the number of examples with actual label (true label) y_i and predicted label y_j.
- 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.



Confusion matrix

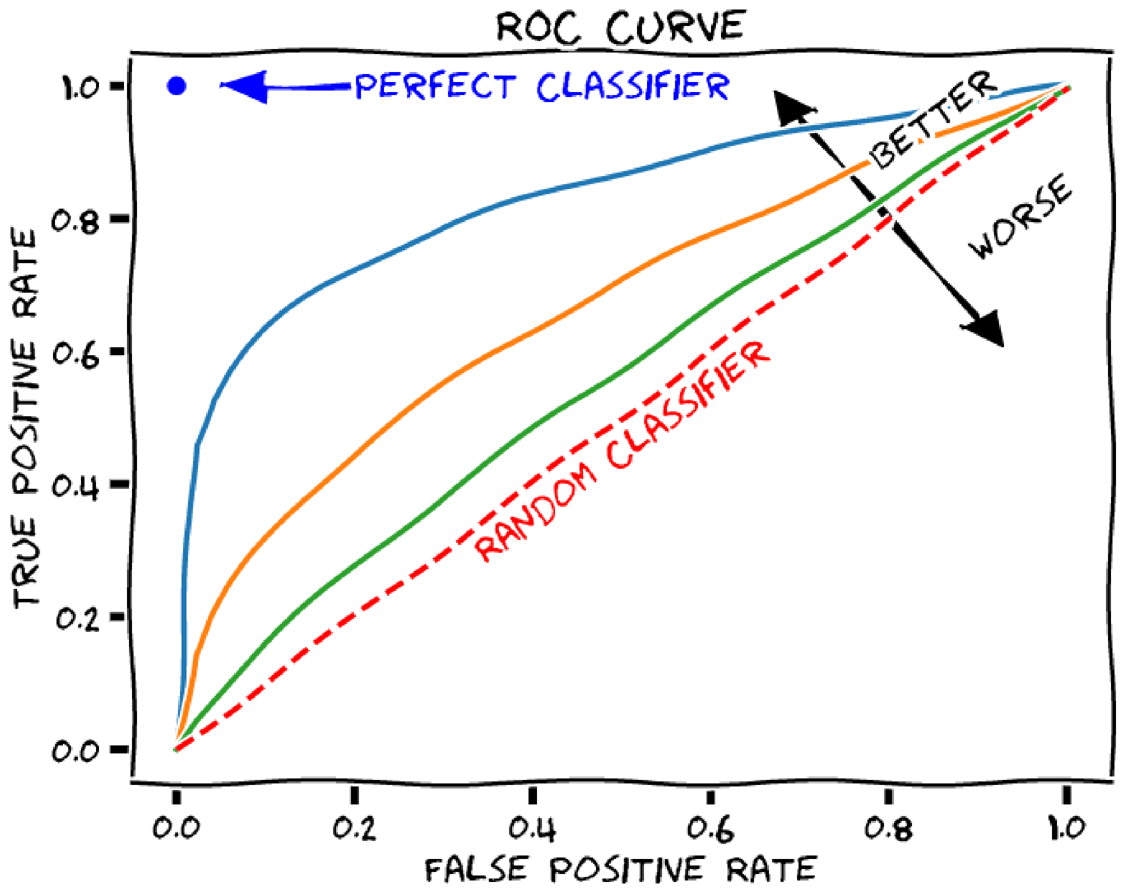
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	50	0	0	0	0	1	0	0	0	0
1	0	39	0	0	0	0	0	0	2	0
2	0	0	46	1	0	0	0	0	0	0
3	0	0	0	39	0	0	0	0	0	0
4	0	2	0	0	42	0	0	0	0	0
5	0	0	0	1	0	46	0	0	0	0
6	0	1	0	0	0	0	47	0	0	0
7	0	0	0	0	0	0	0	39	0	1
8	0	1	0	0	0	1	0	0	51	0
9	0	0	0	0	0	1	0	0	1	38

y_i

y_j

$$FP_i = \sum_{j \neq i} n_{ji} \quad FN_i = \sum_{j \neq i} n_{ij} \quad MAcc = \frac{\sum_i n_{ii}}{\sum_i \sum_j n_{ij}} \quad Pre_i = \frac{n_{ii}}{n_{ii} + FP_i} \quad Rec_i = \frac{n_{ii}}{n_{ii} + FN_i}$$

ROC - Receiver Operating Curves



The ROC curve allows us to

- Compare different classifiers
- Estimate the false positive rate / negative depending on the thresholds that we set to classify
- Our task is to maximize the area under the ROC Curve, to therefore maximize what we define: **Area Under the ROC Curve (AUC).**

Image credit: Wikipedia

Regression evaluation metrics

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

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

Regression evaluation metrics

$$MSE(f) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)^2$$

- **Mean Square Error (MSE)**
  - where *f* is the model that takes a feature vector *x* as input and generates a prediction $f(x_i)$.
  - *i*, ranging from *1 to N*, denotes the index of a sample in the dataset, and y_i is the ground truth.
  - The sum of predictions minus real values over *N* indicates the average. Squaring removes the negative sign and gives more weight to larger differences.
  - The lower the MSE, the better.
  - MSE is influenced by outliers, i.e., data values that are abnormally distant from the true regression line.

Regression evaluation metrics

$$MAE(f) = \frac{1}{N}\sum_{i=1}^{N}|f(x_i) - y_i|$$

- **Mean Absolute Error (MAE)**
  - MAE is similar to MSE in that it returns absolute values of the residuals *f(x)-y* without squaring.
  - It does not consider the direction of the error, which means we will not know whether negative or positive errors weigh more on the overall average.
  - MAE is more robust to outliers precisely due to the absence of squaring of distant forecast error values.
  - However, MAE is not differentiable. It means that we cannot take the derivative of it and taking derivative is important if you want to build an algorithm that minimizes the function. Because of this, it is used a lot less.

Regression

Linear Regression

- Aims to model the relationship between a dependent variable ***Y*** (output) and one or more independent variables ***X*** (set of features)
- The standard linear regression equation for a single variable is:

$$Y = \beta_0 + \beta_j X + \varepsilon$$

Y **:** the dependent variable (e.g. output)
X **:** the independent variable (e.g., a feature)
$\boldsymbol{\beta_0}$: the intercept of the regression line
$\boldsymbol{\beta_j}$: the coefficient
ε : the error term.

- The biggest problem of linear regression is that it is sensitive to **collinearity**, that is **correlation** between the features.

Linear Regression-- Pros

- Easy to understand and interpret
- Computationally efficient, suitable for large datasets
- Works well if the relationship between output (dependent) and independent (features) is approximately linear.
- Exist several statistical tests to evaluate the model's performance and significance (refer to your statistics lessons).

Linear Regression-- Cons

- Assumption of linearity may not hold true for all datasets
- Outliers can significantly affect the regression coefficients, leading to misleading results.
- When independent variables are highly correlated, it can make coefficient estimates unstable and increase the standard errors.

Ridge Regression (L2)

- An extension of linear regression that introduces **regularization** to the cost function. The objective function is modified to include a penalty term proportional to the squared magnitudes of the coefficients:

$$\underbrace{\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2}_{\sum_{i=1}^{N_{training}}\left(y_{real}^{(i)} - y_{pred}^{(i)}\right)^2} + \boxed{\alpha \sum_{j=1}^{p}\beta_j^2}$$

Y dependent variable = target variable to predict

X independent variable = features

β_o intercept,

β_j parameters associated with each feature.

α regularization parameter.

!!! If α goes to zero, then Ridge Regression is the same as Linear Regression and if α is very large, then all weights end up very close to zero resulting in an underfitting model.

Ridge Regression – Pros

- Wrt. to linear regression, ridge regression
  - Helps prevent overfitting, especially for high-dimensional datasets.
  - Effective when features are highly correlated.
  - Enhances the model’s ability to generalize to new data.
  - Can help to identify **important features** through the magnitude of the coefficients. Smaller coefficients indicate less important features

Ridge Regression – Cons

- Cannot perform feature selection (to be introduced later)
- **Less interpretability**
  - Shrinks all coefficients, interpreting the magnitude and importance of individual features is challenging wrt. to linear regression
- **Sensitive to scale:** Features with larger scales can dominate the regularization penalty, so it requires standardization/normalization of variables to ensure they are on the same scale.
- Still assumes a **linear** relationship between the ***X*** and ***Y***. If the underlying relationship is nonlinear, the model will perform poorly ➔ use other types of regression, e.g., polynomial regression
- Performance depends on α, which to be defined though cross validation.
- Not suitable for **sparse** data and **outliers.**

LASSO Regression (L1)

- Similar to Ridge, introduces regularization to the cost function but uses the absolute values of the coefficients.

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \alpha \sum_{j=1}^{p}|\beta_j|$$

Y dependent variable = target variable to predict

X independent variable = features

β_0 intercept,

β_j parameters associated with each feature.

α regularization parameter.

- Has a tendency to shrink some coefficients to zero, effectively performing feature selection.
- Results in more sparse coefficients compared to Ridge.

LASSO Regression– Pros

- **Feature selection**: effectively removing unimportant feature
- Prevents **overfitting** by adding a regularization term.
  - Helpful in high-dimensional datasets, especially if there are more features than the number of samples
- Improves model **interpretability**
- Works well in high-dimensional data
- Better **generalization**

LASSO REGRESSION– CONS

- When features are highly correlated, Lasso may arbitrarily select one of them and set the others to zero.
- Can introduce more bias than ridge regression, particularly if the true model includes many small but nonzero coefficients. By forcing some coefficients to zero, it may oversimplify the model and miss subtle relationships between variables.
- Performance depends on α, which is to be defined through cross-validation.
- Still not a good performance if the dataset is not linear and requires transformations.
- Requires normalization/standardization.

ELASTIC NET

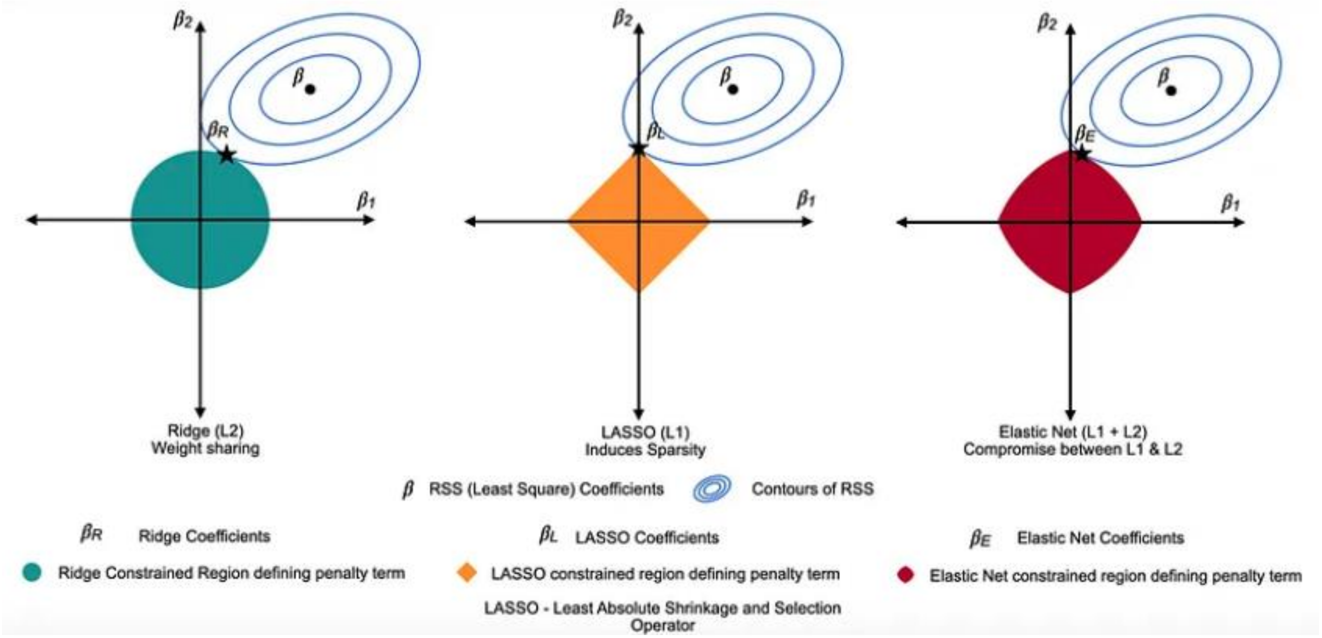
- Combines both L2 and L1 regularization terms in the objective function

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \alpha_1 \sum_{j=1}^{p} |\beta_j| + \alpha_2 \sum_{j=1}^{p} \beta_j^2$$

$$\alpha_1 = \alpha \cdot \rho$$

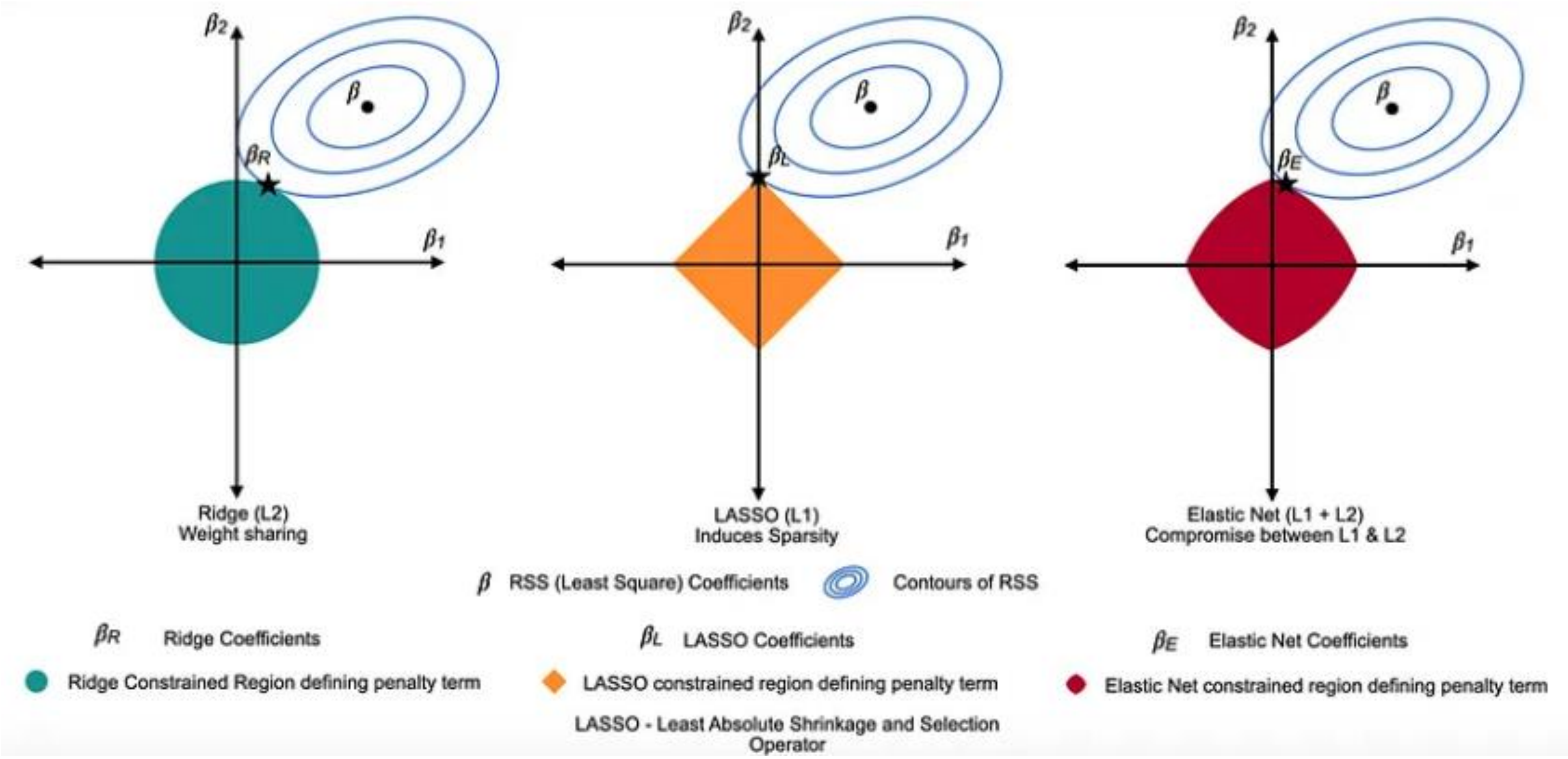
$$\alpha_2 = \alpha \cdot \frac{1-\rho}{2}$$

Y dependent variable = target variable to predict
X independent variable = features
β_0 intercept,
β_j parameters associated with each feature.
α_1 and α_2 regularization parameter


β_2
β_1
$\hat{\beta}$
β_R
β_L
β_E
Ridge (L2)
Weight sharing
LASSO (L1)
Induces Sparsity
Elastic Net (L1 + L2)
Compromise between L1 & L2
$\hat{\beta}$ RSS (Least Square) Coefficients
Contours of RSS
β_R Ridge Coefficients
β_L LASSO Coefficients
β_E Elastic Net Coefficients
● Ridge Constrained Region defining penalty term
◆ LASSO constrained region defining penalty term
◆ Elastic Net constrained region defining penalty term
LASSO - Least Absolute Shrinkage and Selection Operator

- Ridge regression is a good starting point but if there is any chance that only a few features would actually be useful, Lasso and Elastic Nets can be better as they tend to reduce useless features’ weights down to zero.

- Recall that the key difference between Ridge and Lasso regression is that even though both the regression techniques shrink the coefficients closer to zero, only Lasso regression actually sets them to **zero** if the shrinkage parameter is large enough. Thus, resulting in a model having a selected set of features (**sparse model**) making it much easier to interpret and work with.
- If there are fewer features and all seem to be important with regard to the target, then Ridge regression should be the first choice as it tends to give small but well-distributed weights.


β_2
β
β_R
β_1
Ridge (L2)
Weight sharing
β_2
β
β_L
β_1
LASSO (L1)
Induces Sparsity
β_2
β
β_E
β_1
Elastic Net (L1 + L2)
Compromise between L1 & L2
β RSS (Least Square) Coefficients
Contours of RSS
β_R Ridge Coefficients
β_L LASSO Coefficients
β_E Elastic Net Coefficients
● Ridge Constrained Region defining penalty term
◆ LASSO constrained region defining penalty term
● Elastic Net constrained region defining penalty term
LASSO - Least Absolute Shrinkage and Selection Operator

- Elastic Net first emerged as a result of critique on Lasso, whose variable selection can be too dependent on data and thus unstable. The solution is to combine the penalties of Ridge regression and Lasso to get the best of both worlds.
- Elastic Net is preferred over Lasso regression when the number of features is greater than the number of training instances or when several features are strongly correlated.

More on regression

- **Polynomial Regression** ➔ LAB (tomorrow)
  - a type of regression analysis where the relationship between the independent variable ***X*** and the dependent variable ***Y*** is modelled as an **nth-degree polynomial**.

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \epsilon$$

- Best of nonlinear data!!!
- **Cons:** Overfitting, complexity, lack of interpretability

Bayes Decision theory

Introduction

- A fundamental **statistical approach** for classification.
- Hypothesis:
  - The decision problem is cast in probabilistic terms,
  - All relevant probabilities are known.
- Goal:
  - Discriminate the different decision rules using the probabilities and the associated costs
    - Estimating the **joint probability** ***p(x, y)*** from the training data set

An Easy Example

- Let ω (*called state of nature*) is what we want to classify and it is to be probabilistically described
- There are **two classes** $\boldsymbol{\omega_1}$ and $\boldsymbol{\omega_2}$ which correspond to two possible states:

  a) $P(\omega = \omega_1) = 0.7$

  b) $P(\omega = \omega_2) = 0.3$

  → **a-priori** or **prior probability**

  - There are no measurements or observations available to help make a decision about which class ω belongs to.
- Decision rule:
  - Decide ω_1 if $P(\omega_1) > P(\omega_2)$; otherwise decide ω_2
  - Given the probabilities (70% for ω_1 and 30% for ω_2), the decision rule would lead to classifying ω as ω_1 since 0.7 is greater than 0.3.

Another Example

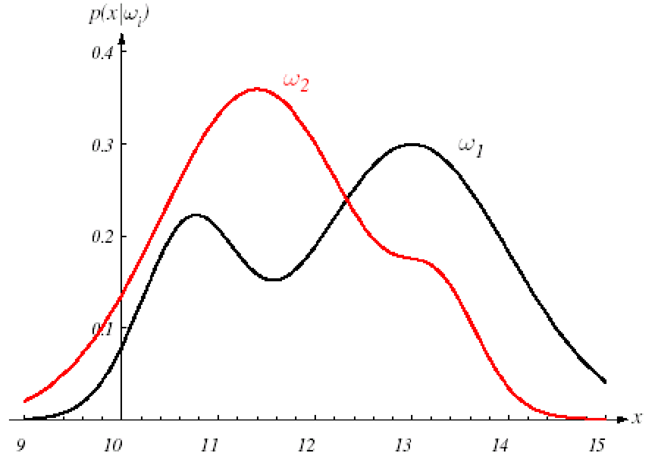
- Having the previous hypothesis and additionally **a single measurement** $\boldsymbol{x}$, which is a random variable that depends on the class ω_j, we can get:

$$p(x \mid \omega_j)_{j=1,2}$$

= **Likelihood** *or* **class-conditional probability density function**

i.e. *the probability of having the measurement x knowing that the state of nature is* ω_j

- When we fix the measurement $\boldsymbol{x}$, the higher $p(x \mid \omega_j)$ is, the more likely ω_j is the correct state.



Bayes Formula

- Assuming known ***P*(*ω_j*)** and ***p*(*x* | *ω_j*)** , the decision of *ω (*the state of nature) becomes, for Bayes

$$p(\omega_j, x) = P(\omega_j \mid x)\, p(x) = p(x \mid \omega_j) P(\omega_j)$$

that is

$$P(\omega_j \mid x) = \frac{p(x \mid \omega_j) P(\omega_j)}{p(x)} \propto p(x \mid \omega_j) P(\omega_j)$$

where:

- $P(\omega_j)$ = **Prior**
- $p(x \mid \omega_j)$ = **Likelihood**
- $P(\omega_j \mid x)$ = ***Posterior***
- $p(x) = \sum_{j=1}^{J} p(x \mid \omega_j) P(\omega_j)$ = ***Evidence***

Bayes Decision Rule

$$P(\omega_j \mid x) = \frac{p(x \mid \omega_j)P(\omega_j)}{p(x)} \quad \Longleftrightarrow \quad posterior = \frac{likelihood \times prior}{evidence}$$

- *The posterior or **a-posteriori probability** is the probability that the state of nature is ω_j given the observation x.*
- The most important factor is the product $likelihood \times prior$; the evidence $p(x)$ is simply a scale factor, which ensures that

$$\sum_j P(\omega_j \mid x) = 1$$

- From the formula of Bayes derives the **Bayes' decision rule**:
  Decide w_1 if $P(w_1|x) > P(w_2|x)$, and w_2 otherwise

Naïve Bayes Classifier

- A probabilistic classification algorithm based on based on Bayes' theorem.
- Assumes independence among features.
- Calculates the posterior probability using

$$P(\omega_j \mid x) = \frac{p(x \mid \omega_j) P(\omega_j)}{p(x)}$$

- Assumes features x_1, x_2,…, x_n are independent given the class label ω:

$$P(x|\omega) = P(x_1|\omega) \cdot P(x_2|\omega) \cdots P(x_n|\omega)$$

- Probability of each class is determined from training data.

Naïve Bayes Classifier

- Classify an observation *x* to class ω_j if:

$$P(w_j|x) = \frac{P(x|w_j) \cdot P(w_j)}{P(x)} \text{ is maximized}$$

- **Advantages:**
  - Fast to train and predict, suitable for large datasets.
  - Works well with high-dimensional data.
- **Limitations:**
  - Independence assumption may not hold in real-world scenarios, leading to suboptimal performance.
  - Zero probability problem: requires techniques for unseen feature-class combinations.

Problems

- To create an optimal classifier that uses the Bayesian decision rule you need to know:
  - **Prior probabilities** $P(\omega_i)$
  - **Class-conditional densities** $p(\mathbf{x} \mid \omega_i)$

- The performance of a classifier strongly depends on the **goodness** of these components.

BUT PRACTICALLY ALL THIS INFORMATION IS NEVER AVAILABLE!!

Problems

- More often we only have:
  - A vague knowledge of the problem, from which to extract vague **a-priori probabilities**.
  - Some particularly representative patterns, training data, used to train the classifier (**often too few!**)

- Estimating a-priori probabilities is usually not particularly difficult.

- Estimating conditional densities is **more complex**.

Problems

- Conditional densities can be estimated by estimating the unknown parameters of known function $\boldsymbol{p}(\mathbf{x} \mid \boldsymbol{\omega}_{j})$
  - E.g., estimate the vector

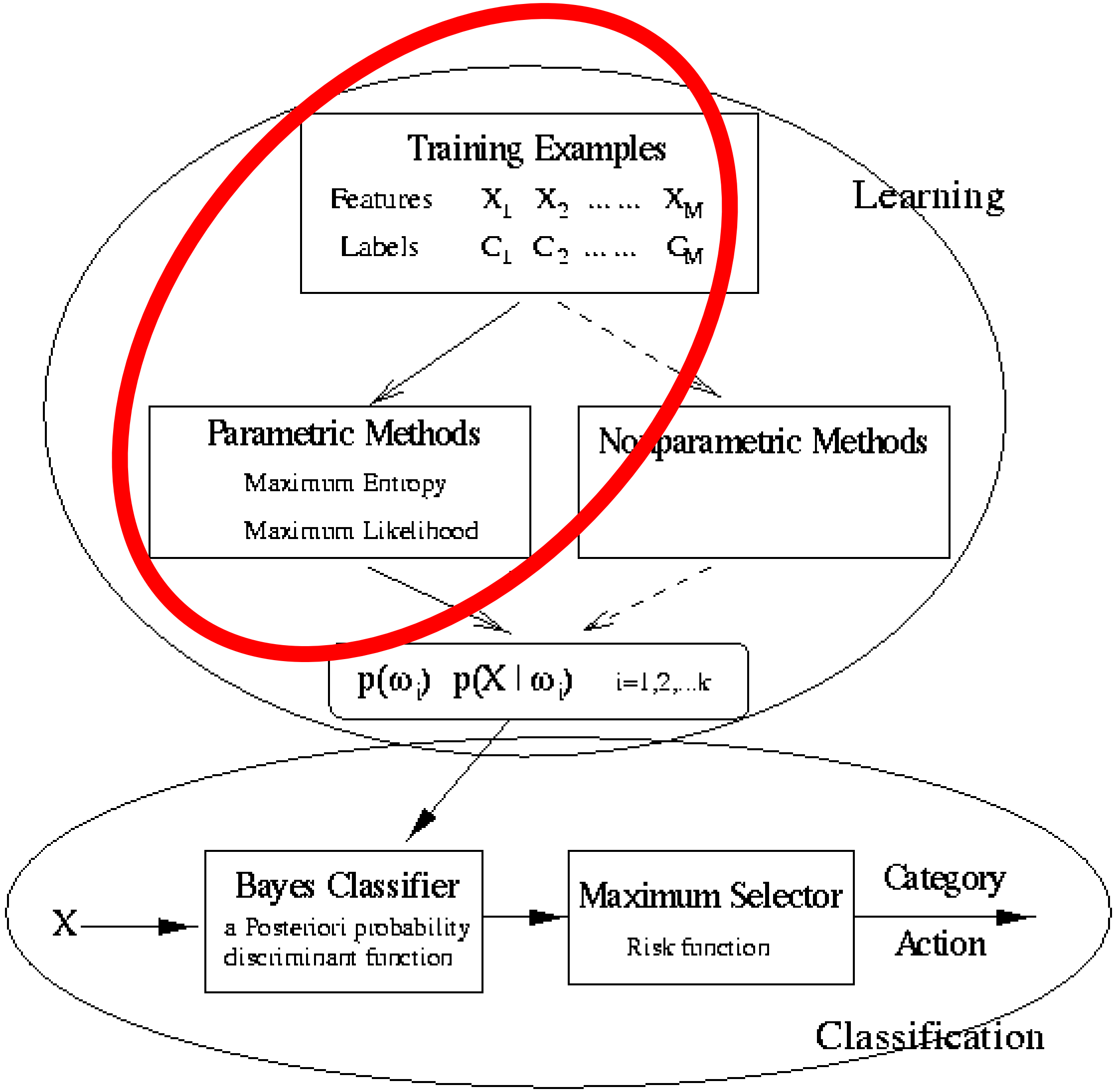
$$\text{when} \quad \boldsymbol{\theta}_j = (\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

$$p(\mathbf{x} \mid \omega_j) \approx N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

PARAMETER ESTIMATION

- Roadmap:
    - Estimate the parameters from the training data.
    - Use the resulting estimated as true values.
    - Use the Bayesian decision theory to build a classifier.

$$p(\mathbf{x}\,|\,\omega_j) \approx N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$



Parameter Estimation

- Suppose we have a set of *n training samples* and each has a class id (w_i).
- Then $P(\omega_i) = \dfrac{n_i}{n}$ where n_i is the number of samples with label w_i .
- Keep in mind that a-priori probabilities are not so useful as much as conditional densitites.
  - Parameter estimation to estimate the conditional densities ➔

Parameter Estimation

- Given a training set D={x_1, x_2, …., x_n}
- $p(\mathbf{x} \mid \omega)$ is determined by $\boldsymbol{\theta}$, which is a vector representing the necessary parameters
  - such as $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$ if $p(\mathbf{x} \mid \omega) \approx N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
  - We need to find the best parameter $\boldsymbol{\theta}$ using the training set.

- To do so, we have two MAIN approaches
  - Maximum likelihood estimation (ML)
  - Bayesian estimation

2 APPROACHES

- **Maximum Likelihood approach**
    - Parameters are seen as quantities whose values are fixed but unknown.
    - The best estimation of their value is defined to be the one that **maximizes the probability** of obtaining the samples actually observed (training data).
    - Good to use it when we have sufficient data and want a frequent estimation approach.
    - Sensitive to overfitting, especially when the data is sparse.

2 APPROACHES

- **Bayesian approach**
    - Uses the full posterior distribution of the parameters rather than a single point like ML.
    - It involves integrating over the posterior distribution to make predictions.
    - Used when you want a full probabilistic model, including uncertainty about parameter values.
    - Provides a complete distribution over possible parameter values, offering more information than point estimates.
    - Computationally expensive!

Maximum Likelihood Approach

- Given the starting hypothesis and the samples of training set **D** are i.i.d. – independent and identically distributed, we have:

$$p(\mathbf{D} \mid \boldsymbol{\theta}) = \prod_{k=1}^{n} p(x_k \mid \boldsymbol{\theta})$$

- is a function of **θ** when $p(\mathbf{D} \mid \boldsymbol{\theta})$ is called the likelihood of **θ** w.r.t. the set of samples **D**.
- The maximum likelihood estimate of **θ** is, the value $\hat{\boldsymbol{\theta}}$ that maximizes $p(\mathbf{D} \mid \boldsymbol{\theta})$;
- Keep in mind that **θ** is fixed but unknown.

Maximum Likelihood Algorithm

- Define the likelihood function $p(\mathbf{D}\,|\,\boldsymbol{\theta}) = \prod_{k=1}^{n} p(x_k\,|\,\boldsymbol{\theta})$
- Calculate the log-likelihood $l(\theta) \equiv \ln p(D\,|\,\theta) = \sum_{k=1}^{n} \ln p(x_k\,|\,\theta)$
  - Notice that the product become a sum, which is easier to handle mathematically and computationally.
- Differentiate the log-likelihood: This step is needed to obtain the equations that we can solve to find the optimal $\boldsymbol{\theta}$
  - Take the derivative and set it to zero.

$$\nabla_{\theta} l(\boldsymbol{\theta}) = \sum_{k=1}^{n} \nabla_{\theta} \ln p(x_k\,|\,\boldsymbol{\theta}) \qquad \nabla_{\theta} l(\boldsymbol{\theta}) = 0$$

Maximum Likelihood Algorithm

- Solve the equation set to zero to find the maximum likelihood estimate $\hat{\theta}$. The solution gives the parameter values that make the observed data most likely.

- Let's apply maximum likelihood algorithm for Gaussian Distribution

Maximum Likelihood: Gaussian Distribution

- Suppose the samples are from a multivariate normal distribution with mean **μ** and covariance matrix **Σ**.
- For simplicity, let's first consider the case where only the mean **μ** is unknown.
- Under this condition, we consider a sample point $\mathbf{x}_k$ and find:

$$\ln p(\mathbf{x}_k \mid \boldsymbol{\mu}) = -\frac{1}{2}\ln\left[(2\pi)^d |\boldsymbol{\Sigma}|\right] - \frac{1}{2}(\mathbf{x}_k - \boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu})$$

$$\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)$$

$$\Downarrow$$

$$\nabla_{\boldsymbol{\mu}} \ln p(\mathbf{x}_k \mid \boldsymbol{\mu}) = \boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu})$$

Maximum Likelihood: gaussian Distribution

- Identifying $\boldsymbol{\theta}$ with $\boldsymbol{\mu}$, we see that the Maximum Likelihood estimate $\boldsymbol{\mu}$ must satisfy:

$$\sum_{k=1}^{n} \boldsymbol{\Sigma}^{-1}\left(\mathbf{x}_k - \hat{\boldsymbol{\mu}}\right) = 0$$

- Multiplying for $\boldsymbol{\Sigma}$ and rearranging the sum, we obtain

$$\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{k=1}^{n} \mathbf{x}_k$$

that is the arithmetic **mean** of the training samples, sometimes written as $\hat{\boldsymbol{\mu}}_n$ to clarify its dependence on the number of samples.

Maximum Likelihood: gaussian Distribution

- In the more general typical multivariate normal case, **neither the mean μ nor the covariance matrix Σ is known**.
- Let's consider first the univariate case with $\boldsymbol{\theta} = (\theta_1, \theta_2) = (\mu, \sigma^2)$
- The log-likelihood of a single point is

$$\ln p(x_k \mid \boldsymbol{\theta}) = -\frac{1}{2}\ln\left[2\pi\theta_2\right] - \frac{1}{2\theta_2}(x_k - \theta_1)^2$$

- and its derivative is:

$$\nabla_{\boldsymbol{\theta}} l = \nabla_{\boldsymbol{\theta}} \ln p(x_k|\boldsymbol{\theta}) = \begin{bmatrix} \frac{1}{\theta_2}(x_k - \theta_1) \\ -\frac{1}{2\theta_2} + \frac{(x_k - \theta_1)^2}{2\theta_2^2} \end{bmatrix}$$

Maximum Likelihood: gaussian Distribution

- Set both equations to be zero:

$$\sum_{k=1}^{n}\frac{1}{\theta_2}(x_k-\hat{\theta}_1)=0 \qquad -\sum_{k=1}^{n}\frac{1}{\hat{\theta}_2}+\sum_{k=1}^{n}\frac{(x_k-\hat{\theta}_1)^2}{\hat{\theta}_2{}^2}=0$$

where $\hat{\theta}_1$ and $\hat{\theta}_2$ are the ML estimates for θ_1 and θ_2.

- By substituting $\hat{\mu}=\hat{\theta}_1$ and $\sigma^2=\hat{\theta}_2$ we obtain the ML estimates for mean and variance

$$\hat{\mu}=\frac{1}{n}\sum_{k=1}^{n}x_k \qquad \hat{\sigma}^2=\frac{1}{n}\sum_{k=1}^{n}(x_k-\hat{\mu})^2$$

Bayesian Parameter Estimation

- In this case, we do not assume that θ is fixed but instead it is a **random variable**.
- In this case the training dataset D allows us to convert a prior distribution $p(\theta)$ into a posterior probability density $p(\theta|\mathrm{D})$.

$$p(\theta) \Rightarrow p(\theta|\mathrm{D})$$

- The computation of the posterior probabilities $P(w_i|\boldsymbol{x})$ requires Bayesian classification, meaning that we need to know
  - Prior probabilities $P(\omega_i)$
  - Condiitonal densities $p(\boldsymbol{x}|\omega_i)$
- When these quantities are unknown, the best we can do is to compute $p(\boldsymbol{x}|w_i)$ using ***all of the information at our disposal***.

Bayesian Parameter Estimation

- Given the training set D, the Bayes' formula then becomes:

$$P(\omega_i \mid \mathbf{x}, D) = \frac{p(\mathbf{x} \mid \omega_i, D)P(\omega_i \mid D)}{\sum_{j=1}^{c} p(\mathbf{x} \mid \omega_j, D)P(\omega_j \mid D)}$$

- Assumptions:
  - Reasonably, $P(\omega_i \mid \mathrm{D}) \Rightarrow P(\omega_i)$
  - Since we are treating the supervised learning case, the training set D can be partitioned into c subsets D_1, D_2,…, D_c, with the samples in D_i belonging to ω_i
  - The samples belonging to D_i have no influence on the parameters of $p(\mathbf{x}| \omega_j, D)$ if $i \neq j$.

Bayesian Parameter Estimation

- Considering these assumptions, we obtain:

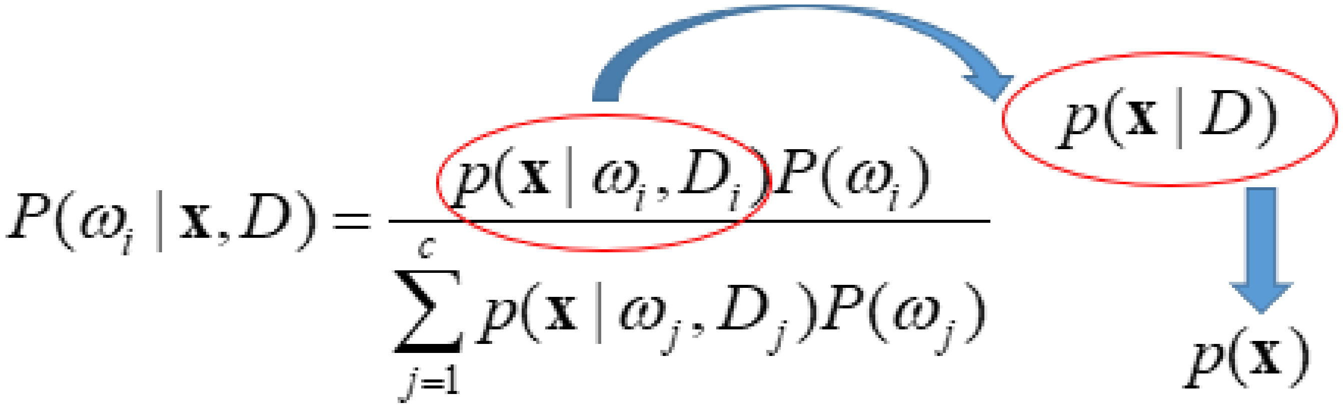
$$P(\omega_i \mid \mathbf{x}, D) = \frac{p(\mathbf{x} \mid \omega_i, D) P(\omega_i \mid D)}{\sum_{j=1}^{c} p(\mathbf{x} \mid \omega_j, D) P(\omega_j \mid D)}$$

$$\Downarrow$$

$$P(\omega_i \mid \mathbf{x}, D) = \frac{p(\mathbf{x} \mid \omega_i, D_i) P(\omega_i)}{\sum_{j=1}^{c} p(\mathbf{x} \mid \omega_j, D_j) P(\omega_j)}$$

Bayesian Parameter Estimation

- Because each class can be treated independently, we do not need distinctions among classes, so we can simplify our notation by **reducing to** c **different instances of the same problem**, i.e.:



$$P(\omega_i \mid \mathbf{x}, D) = \frac{p(\mathbf{x} \mid \omega_i, D_i) P(\omega_i)}{\sum_{j=1}^{c} p(\mathbf{x} \mid \omega_j, D_j) P(\omega_j)}$$

$p(\mathbf{x} \mid D)$

$p(\mathbf{x})$

- *Use a set of samples D, drawn independently according to the fixed but unknown probability distribution p(*$\mathbf{x}$*), to determine p(*$\mathbf{x}$*|D).*

Bayesian Parameter Estimation

- We are realizing the calculation of $p(\boldsymbol{x}|D)$ to estimate $p(\boldsymbol{x})$, converting the problem of estimating a probability density to a problem of estimating a parameter vector.
  - Observe how $p(\boldsymbol{x}|D)$ is obtained via an implicit parameter model $\boldsymbol{\theta}$.
  - Consequently, we have

$$p(\mathbf{x} \mid D) = \int p(\mathbf{x}, \boldsymbol{\theta} | D) d\boldsymbol{\theta}$$

where the integration extends over the entire parameter space.

Bayesian Parameter Estimation

- Then $p(\mathbf{x} \mid D) = \int p(\mathbf{x}, \boldsymbol{\theta} | D) d\boldsymbol{\theta}$

$$= \int p(\mathbf{x} \mid \boldsymbol{\theta}, D) p(\boldsymbol{\theta} \mid D) d\boldsymbol{\theta}$$

- Since, by hypothesis, the selection of **x** is done independently from the training samples in *D*, given θ,

$$p(\mathbf{x} \mid D) = \int p(\mathbf{x} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid D) d\boldsymbol{\theta}$$

- That is, the distribution of ***p(x)*** is known completely once we know the value of the parameter vector θ.

Bayesian Parameter Estimation

$$p(\mathbf{x} \mid D) = \int p(\mathbf{x} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid D) d\boldsymbol{\theta}$$

- Links the desired class-conditional density ***p(x|D)*** to the posterior density $p(\boldsymbol{\theta}|\mathrm{D})$ through the unknown parameter vector $\boldsymbol{\theta}$.
- If $p(\boldsymbol{\theta}|\mathrm{D})$ peaks very sharply about some value $\hat{\boldsymbol{\theta}}$, then we obtain an estimatetion of the most likely vector, thus

$$p(\mathbf{x}|\mathrm{D}) \approx p(\mathbf{x} \mid \hat{\boldsymbol{\theta}})$$

- But this approach allows to **take into account the effects of all other models**, described by the value of the **integral function**, for ***all possible models***.

$$p(\mathbf{x} \mid D) = \int p(\mathbf{x} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid D) d\boldsymbol{\theta}$$

Comparisons: Ml vs. Bayesian estimation

- ML gives us a point estimate $\hat{\boldsymbol{\theta}}$, instead, Bayes approach gives us a distribution on $\boldsymbol{\theta}$.
- ML and Bayes solutions are equivalent in the asymptotic limit of infinite training data.
- Practically, the approaches are different for various reasons:
  - Computational complexity: ML is less
  - Interpretability: ML is easy to interpret
  - Confidence in the prior information: Bayesian heavily relies on prior info.
  - Compromise between estimation accuracy and variance: ML prone to overfitting, leading to estimates with high variance.

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