



cnrs formation entreprises

Fundamentals of Machine Learning

Direction de l'innovation et des relations avec les entreprises

Day 4: Model selection and ensemble methods

Titouan Vayer

Plan

1 A bit of learning theory

1.1 Recap

1.2 ERM and Overfitting

2 Model selection

3 Boostrap and Bagging

4 Random Forests

5 Boosting

What you've seen so far

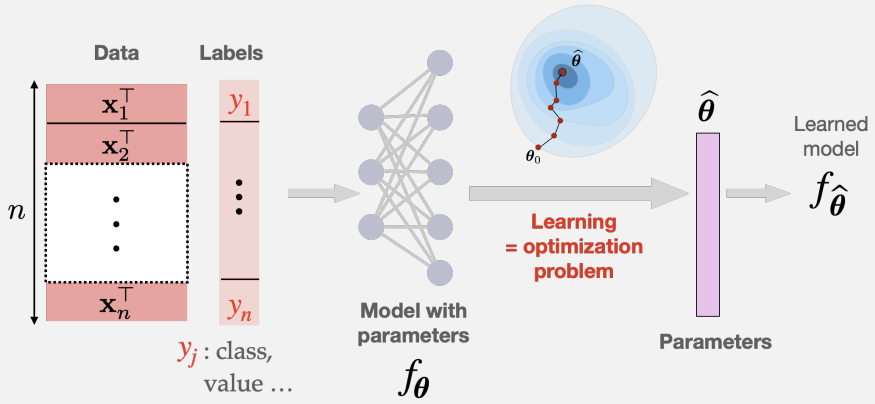
- Learn how to recognize a learning problem.
- Different type of data and variables.
- Dimension reduction and clustering.
- Simple supervised learning algorithms.

Recap

Machine learning: mapping $\mathbf{x}$ to y through some function f_θ

- x: input
- y: output
- f_θ: model
- θ: model parameters
- training samples $(\mathbf{x}_i, y_i)_{i \in [n]}$ representative and independent observations.
- loss $\ell(y, y') \in \mathbb{R}_+$.

Recap



The goal in the **learning step** will be to find the parameters $\widehat{\theta}$ that **minimizes the error on the training data**

Empirical risk minimization

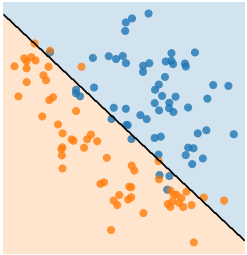
Train by minimizing the train error

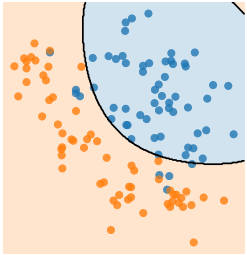
To find $f_{\boldsymbol{\theta}}$ the idea is to **minimize the averaged error** on the training samples:

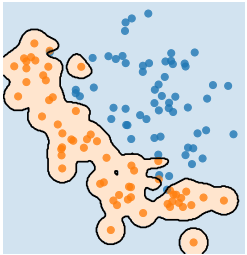
$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)) \tag{ERM}$$

- It is called **empirical risk minimization (ERM)**
- Given the loss, finds the "best" $f_{\boldsymbol{\theta}}$ on the training data.
- Same idea applies for unsupervised problem (minimizing reconstruction error).
- E.g. linear regression: $f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x}$, $\ell(y, y') = (y - y')^2$.

Underfitting and overfitting

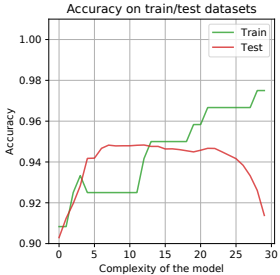






Complexity of a model

- Under-fitting when the model is too simple.
- Over-fitting occurs when the model is too complex
- Training data performance is not a good proxy for testing performance.
- We want to predict well on new data!



Regularization

Parametrized model

We want to find the parameters $\boldsymbol{\theta}$ that minimize:

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \lambda \operatorname{Reg}(\boldsymbol{\theta}) \qquad \text{(ERM)}$$

Limit the expressivity of the function

- We want f to be good outside the training samples
- Add regularization ! (also called penalty).
- E.g. ℓ_2 reg $\operatorname{Reg}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2$.
- λ is the regularization strengh: how to choose it ?

Plan

1 A bit of learning theory

1.1 Recap

1.2 ERM and Overfitting

2 Model selection

3 Boostrap and Bagging

4 Random Forests

5 Boosting

Parameters vs hyperparameters

Hyperparameters

- Are parameters that have to be **selected/chosen** to define a model.
- Used for the configuration of the model.
- They are **not learned** on the data !
- Examples (1): regularization strengh λ, number of neighbors k in k-NN, number of trees, number of iterations for an algorithm.
- Examples (2): k in k-means, output dimension in dimension reduction.
- Example (3): but also all the data pipeline (normalization...)

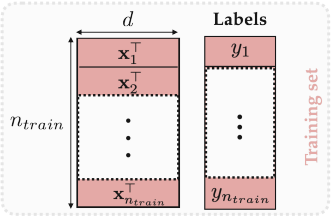
Parameters

- **Are learned** on the data.
- $\boldsymbol{\theta}$ in linear regression, tree cuts, $\mathbf{U} \in \mathbb{R}^{d \times k}$ in dimension reduction (PCA), centroids in k-means

In scikit-learn

```
from sklearn import Model
hyperparameters = ...
clf = Model(hyperparameters) # define the model
cv.fit(X ,y) #train the model
```

Train/validation/test

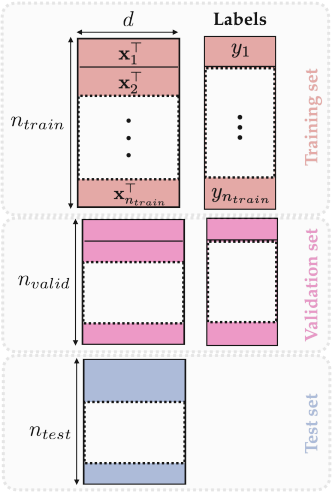




Train, validation and test data

- Data are divided in two sets: a train and a test set
- Learning a ML model = on the train set where you know the output of the model
- The test set is **never available**, used only to deploy the model

Train/validation/test



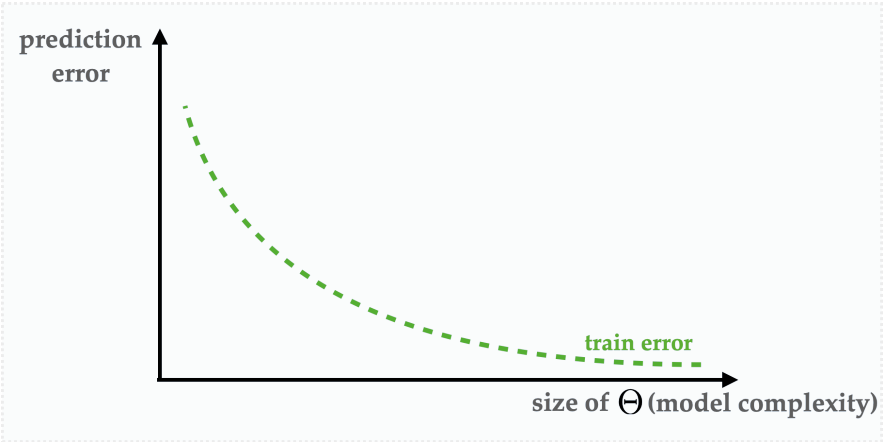
Train, validation and test data

- Data are divided in two sets: a train and a test set
- Learning a ML model = on the train set where you know the output of the model
- The test set is **never available**, used only to deploy the model
- A validation set is often used to **calibrate the model** (choose hyperparameters), and **evaluate the generalization properties** of your model

The sources of error

Bias-complexity tradeoff

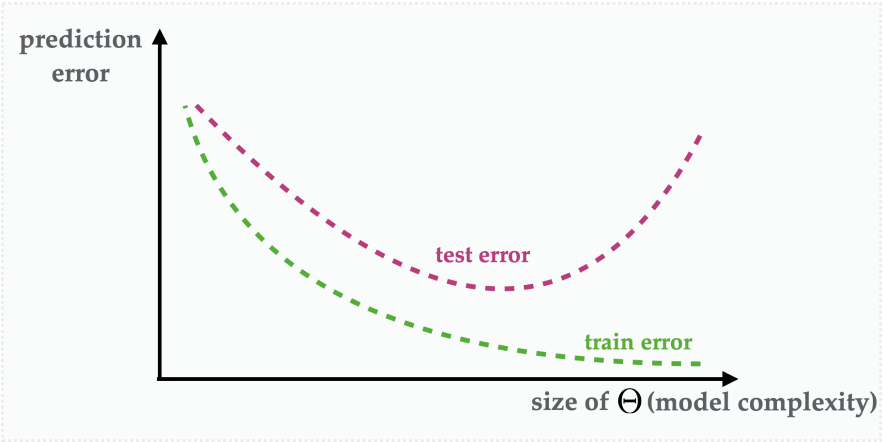
overall error $=$ estimation error $+$ approximation error



The sources of error

Bias-complexity tradeoff

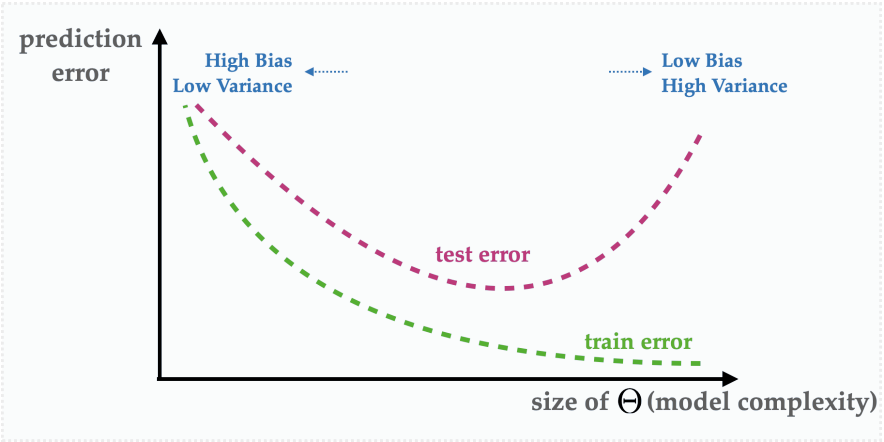
overall error $=$ estimation error $+$ approximation error



The sources of error

Bias-complexity tradeoff

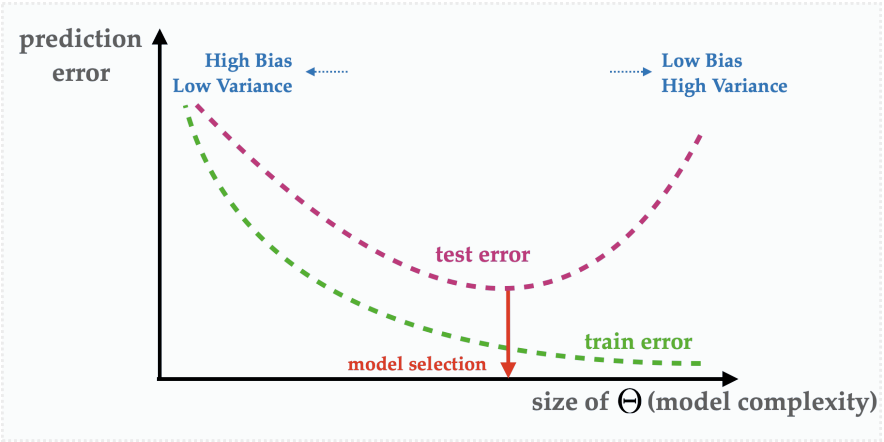
overall error $=$ estimation error $+$ approximation error



The sources of error

Bias-complexity tradeoff

overall error $=$ estimation error $+$ approximation error



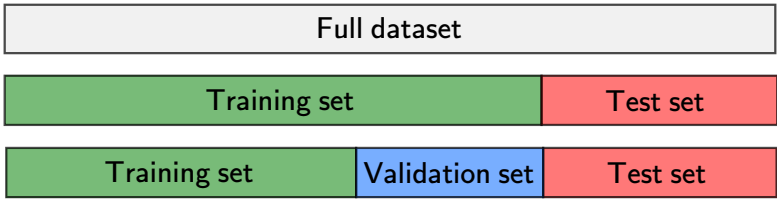
Model selection and validation

Bias-complexity tradeoff

$$\text{overall error} = \text{estimation error} + \text{approximation error}$$

Conlusion: we have to select a model that is not too complex but not too simple !

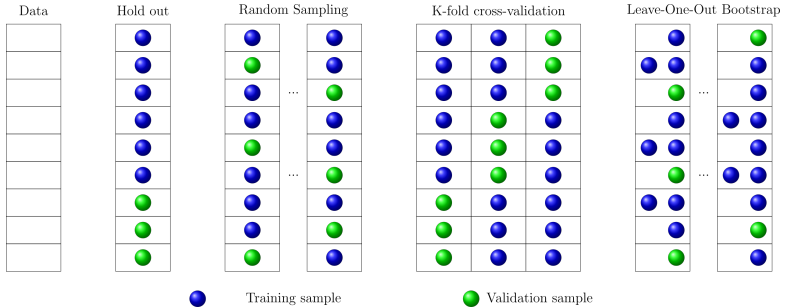
Splitting the data



Principle of Hold-Out cross-validation

- Split the training data in a training and validation sets (non overlapping).
- Train different models (different methods and/or parameters) on the train data.
- Evaluate performance on the validation data and select the method with best performance.

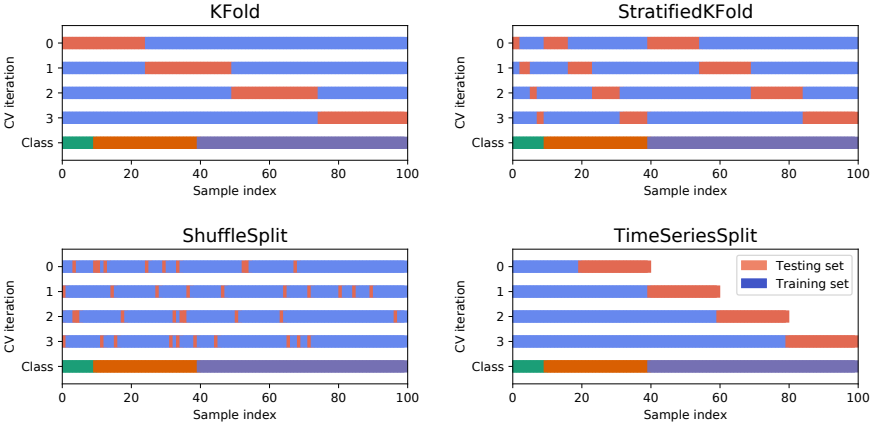
Different ways to split the data



Data splitting for cross-validation

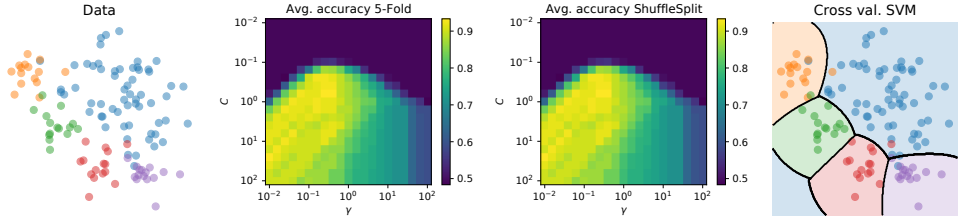
- **Hold-Out** uses a unique split and computes the performance on the validation set.
- More robust cross-validation approaches actually investigate several splits of the data and compute the average performance:
  - **K-fold** (split in K sets and use one split as test for all k)
  - Random sampling (aka **Shuffle split**) draws several random splittings.
- Scikit-learn implementation : sklearn.model_selection.cross_validate

Data splitting with Scikit-learn



- KFold is the classical K-fold cross-validation.
- StratifiedKFold ensures a data split that preserves the proportion of classes.
- ShuffleSplit randomly selects a proportion of the samples for train/validation.

Grid-search



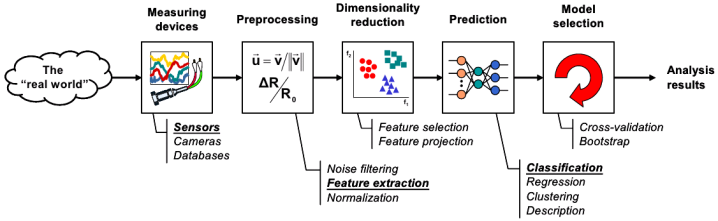
Principle

- GridSearchCV takes a model and a grid of **hyperparameters** as input and performs cross-validation.
- Select the best hyperparameters w.r.t. hold out scores.
- For large number of hyperparameters RandomizedSearchCV can be more efficient.

Python code

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
ngrid=21
clf = SVC()
param_grid={'C':np.logspace(-2,2,ngrid),
            'gamma':np.logspace(-2,2,ngrid),}
cv = GridSearchCV(clf,param_grid)
cv.fit(X,y)
# recover best hyperparameters
params_opt =  cv.best_params_
```

Machine learning in practice



- **Data acquisition** : sensor, databases, manual or automatic labeling
- **Pre-processing** : denoising, formating, numerical conversion, normalization
- **Feature extraction** : manual when prior knowledge, feature selection dimensionality reduction
- **Model estimation** : classification, regression, clustering.
- **Validation** : model and parameter selection.
- **Analysis** : performance, uncertainty, interpretation of the model.

Features extraction, selection and model estimation can be done simultaneously (deep learning, sparse models).

Plan

1 A bit of learning theory

1.1 Recap

1.2 ERM and Overfitting

2 Model selection

3 Boostrap and Bagging

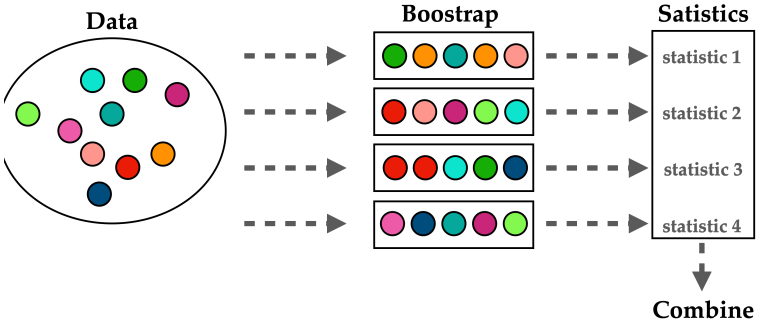
4 Random Forests

5 Boosting

Estimate the variabiliy

Principle of bootstrapping

- Bootstrapping is uses random sampling with replacement.
- Estimates the properties of an estimand (such as its variance) by measuring those properties when sampling from an approximating distribution.
- Powerful idea in statistics (e.g. function estimation)



Bagging = Bootstrap aggregating



- We build N different predictors based on bootstrap sampling.
- We aggregate the predictions.

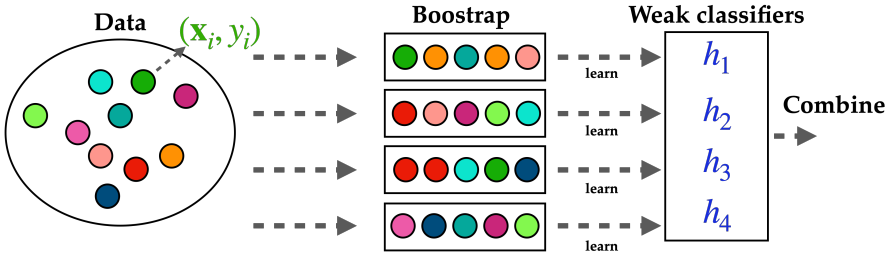
- Principle: Averaging multiple "independent" models helps reduce variance and thus decrease prediction error.
- It is not possible to have truly independent samples, but bootstrapping allows us to approximate independence and obtain useful results.

$\Longrightarrow$ We use computational time to replace data.

- Additionally, we can estimate the error of each estimator using the **out of the bag** samples.
- However, it becomes challenging to interpret the predictions.

In practice

- We choose a base learning algorithm (*weak* classifier).
- For $k \in [N]$, we note h_k the output of the k-th bootstrap subset.
- The final prediction is the aggregation of the h_k :
  - For quantitative output (regression): averaging $h(\mathbf{x}) = \dfrac{1}{N} \displaystyle\sum_{k=1}^{N} h_k(\mathbf{x})$
  - For discrete output (classification): majority vote $h(\mathbf{x}) = \underset{y \in \mathbf{Y}}{\arg\max}\, \mathrm{Card}\left\{k \in [N] : h_k(\mathbf{x}) = y\right\}$



Choice of the weak classifier

- Should be variable / instable
- Should be easily computable, e.g. $N = 10^4$ predictors
  Advantage: this step can be parallelized
- Can't be too expensive: we need to be able to store N models

$\Longrightarrow$ degraded versions of CART are good candidates!

Plan

1 A bit of learning theory

1.1 Recap

1.2 ERM and Overfitting

2 Model selection

3 Boostrap and Bagging

4 Random Forests

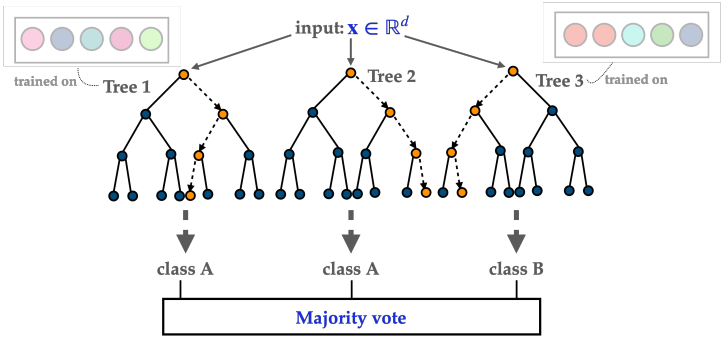
5 Boosting

Random Forests

Principle

Construct N decision tree "CART-like" but:

- Without pruning.
- With limited depth (even 2 leaves...)
- Also with randomness on the **features** (random feature selection).



Interpretability

- lost: black box...
- BUT is able to measure variable importance
  <https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html>.
  - Number of occurrences of each variable in the trees
    - Rudimentary measurement.
    - Gives a rough estimate.
    - The only criterion that gives importance to a variable used to discriminate an infrequent class.
  - Mean Decrease Accuracy (MDA)
    - the *margin* is the difference between the proportion of votes for the correct class, and the maximum of the others $\approx$ how much you "gain".
    - MDA = average over out-of-bag observations of the decrease in their margin when the variable is randomly perturbed.
    - Global but indirect measure of a variable's influence: the more the prediction is disturbed, the more important the variable is.

Plan

1 A bit of learning theory

1.1 Recap

1.2 ERM and Overfitting

2 Model selection

3 Boostrap and Bagging

4 Random Forests

5 Boosting

A race example

Goal = design a program that is capable of accurately predicting the outcome of a race.

- `Bet on the horse that has won the most races recently.`
- `Bet on the horse with the highest number of bets placed on it.`
- `Bet on the horse that prefers heavy track conditions.`

Observation:

- It is easy to find simple rules that work "reasonably well"
- ... but it is difficult to establish a single rule that works really well.

Boosting approach

- Use a "weak classifier" to quickly obtain simple rules that are not very effective.
- Apply them to subsets of examples.
- Correct the misclassified examples with new simple rules.
- Repeat until overall performance is good.

Question:

- How to intelligently choose the examples presented at each iteration?
- How to combine the different simple rules?

$\Longrightarrow$ Another way to aggregate:

- Sequentially (rather than in parallel).
- With unequal weights.
- Through successive corrections.

Boosting

Boosting is a general method that allows us to:

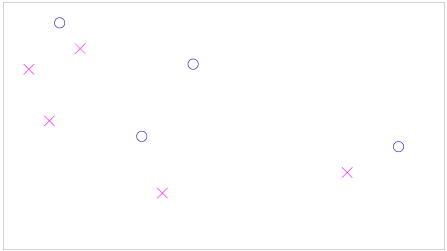
- Transform a "weak classifier" that returns simple and ineffective rules (e.g., 55% accuracy in binary classification).
- Into a strong classifier that achieves very high performance (e.g., 99% accuracy).

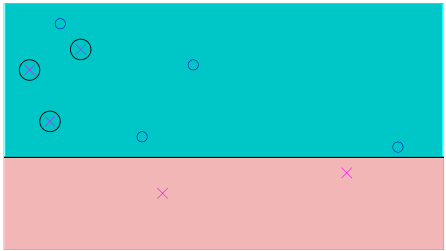
Adaboost provides theoretical guarantees of performance.

Adaboost

- Training samples: $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ avec $y_i \in \{-1, +1\}$
- For $t = 1, \ldots, T$:
  - Updates the sample weights D_t over $\{1, \ldots, n\}$: more importance to misclassified samples.
  - Find a weak classifier which minimizes the classification error ϵ_t with weights D_t.
- Weights $(\alpha_t)_t$ for each classifiers: big weights if small error (and conversly).
- Returns a final classifier h combining the $(h_t)_t$: $h : x \mapsto \text{sign}\left(\sum_{t=1}^{t} \alpha_t \, h_t(x)\right)$.

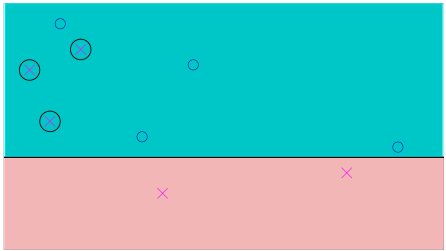
Example : $t = 1$

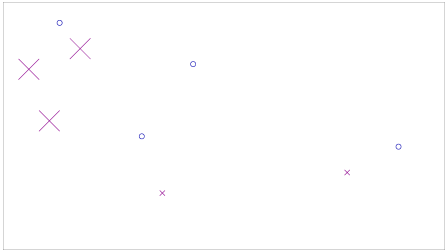




$$\epsilon_1 = 0.33, \quad \alpha_1 = 0.35$$

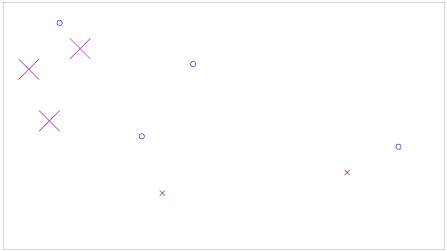
Example $t = 1$





$$\epsilon_1 = 0.33, \quad \alpha_1 = 0.35$$

Example $t = 2$

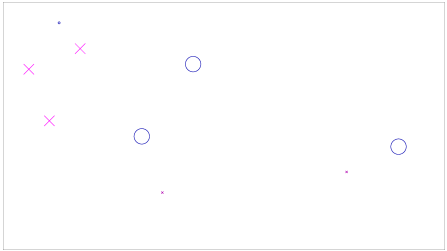




$$\epsilon_2 = 0.21, \quad \alpha_1 = 0.65$$

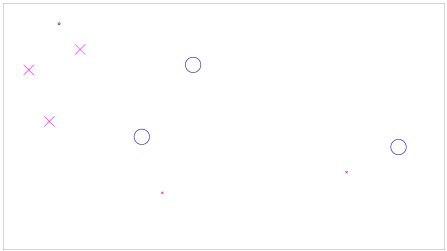
Example $t = 2$

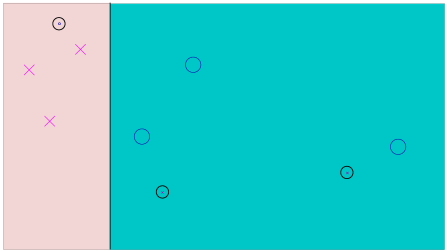




$$\epsilon_2 = 0.21, \quad \alpha_1 = 0.65$$

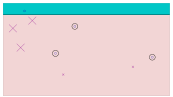
Example $t = 3$

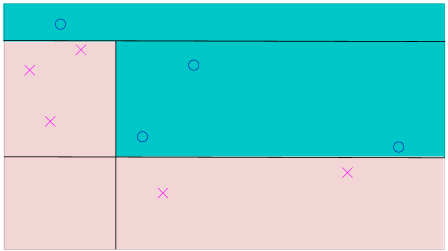




$$\epsilon_3 = 0.14, \quad \alpha_3 = 0.92$$

Example $T = 3$

$$h = \text{signe}\left(0.35\times \right.$$  $$+\ 0.65\times$$  $$+\ 0.92\times$$  $$\left.\right)$$

$$=$$ 

Boosting and overfitting

- As T increases, the training error decreases.
- Eventually, it reaches zero.
- It is expected that for very large values of T, the classifier h may suffer from *overfitting*.
- While overfitting can occur, it is not always the case.
- It can be beneficial to *continue iterations even when the training error is zero*.
- Explanation : we continue to improve the *margin*.
- Note: the theory of boosting is rich and includes other points of view explaining its successes.

Gradient Boosting/ Xgboost

- Slightly different use of boosting.
- Idea: Each new classifier aims directly to *correct the gap between predictions and target labels*.
- Typically employs a simple decision tree for this purpose.

