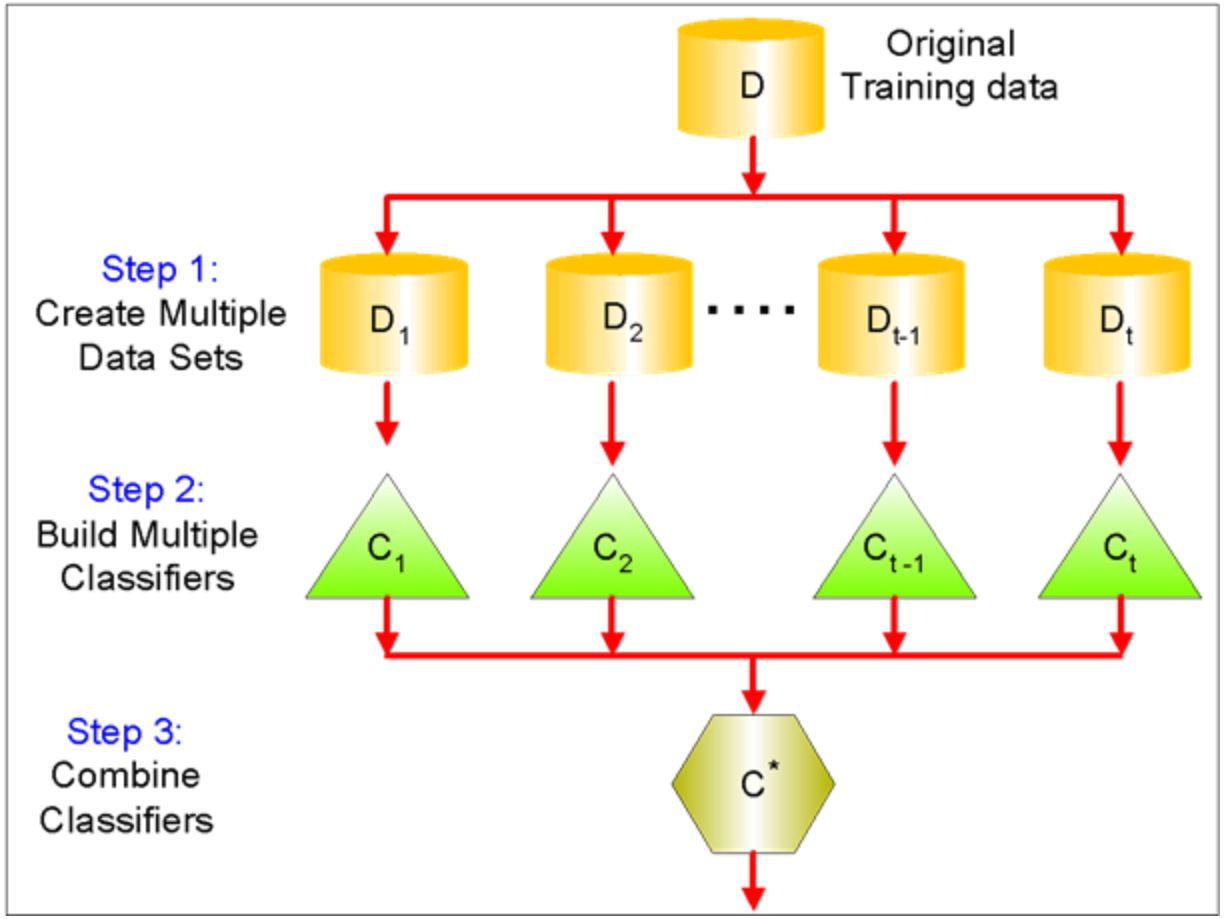
Ensemble Methods

UNIVERSITÀ DI PISA

Ensemble Methods

- Improves the accuracy by **aggregating the predictions** of multiple classifiers.
- Construct a set of **base classifiers** from the training data.
- Predict class label of test records by combining the predictions made by multiple classifiers.



Back to Machine Learning

It will exploit *Wisdom of crowd* ideas for specific tasks

- By combining classifier predictions and
- aims to combine independent and diverse classifiers.

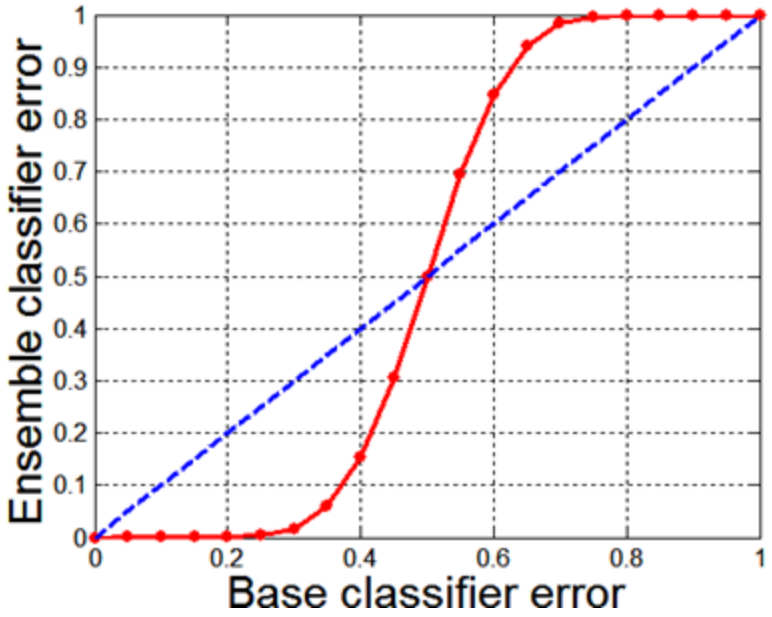
But it will use labelled training data

- to identify the **expert** classifiers in the pool;
- to identify **complementary** classifiers;
- to indicate how to the best **combine** them.

Why Ensemble Methods work?

Suppose there are 25 base classifiers

- Each classifier has error rate, $\varepsilon = 0.35$
- Assume errors made by classifiers are uncorrelated (i.e., their errors are uncorrelated)
- Thus, the ensemble makes a wrong prediction only if more than half of the base classifiers predict incorrectly.
- Probability that the ensemble classifier makes a wrong prediction:



$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

Types of Ensemble Methods

- Manipulate data distribution
  - Example: bagging, boosting
- Manipulate input features
  - Example: random forests
- Manipulate class labels
  - Example: error-correcting output coding

Bagging

Bagging (a.k.a. Bootstrap AGGregatING)

- Sampling with replacement

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Build classifier on each bootstrap sample
- Each sample has probability (1 – 1/n)n of being selected

Algorithm 5.6 Bagging Algorithm

1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size n, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: **end for**
6: $C^*(x) = \arg\max_y \sum_i \delta\big(C_i(x) = y\big)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise.$\}$

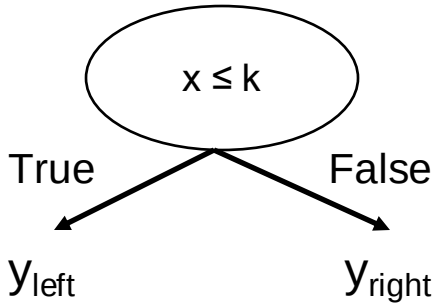
Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: x <= k versus x > k
  - Split point k is chosen based on entropy



Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 ➔ y = 1
x > 0.7 ➔ y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging Example

- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Boosting

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records.
- Initially, all the records are assigned equal weights.
- Unlike bagging, weights may change at the end of each boosting round.

Boosting

- Records that are wrongly classified will have their weights increased.
- Records that are classified correctly will have their weights decreased.

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased; therefore it is more likely to be chosen again in subsequent rounds

AdaBoost

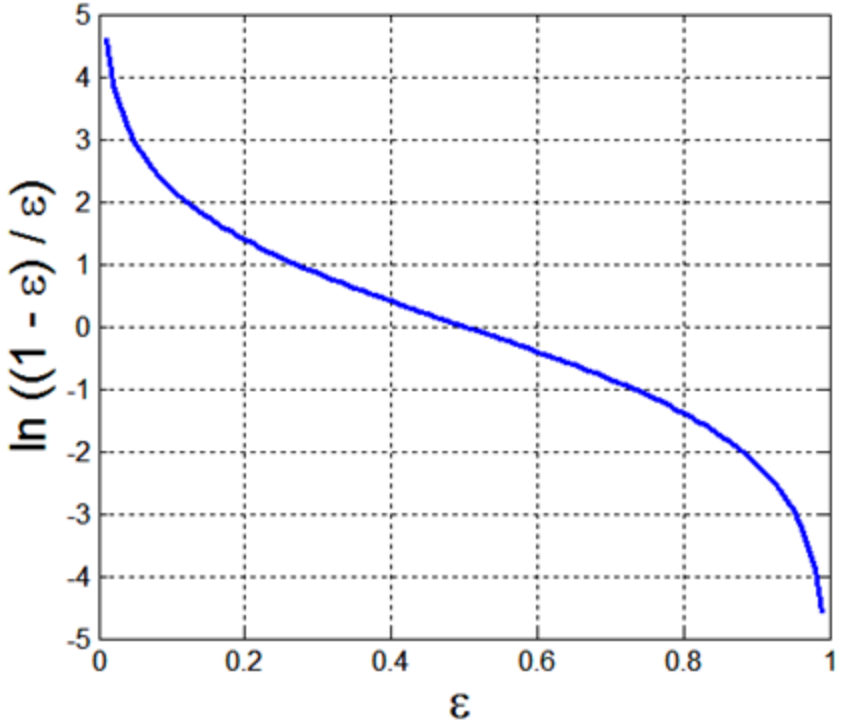
- Base classifiers: C_1, C_2, …, C_T
- Error rate:

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of a classifier depends on its error rate:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

High positive importance when error is close to 0,
High negative importance when error is close to 1



AdaBoost Algorithm

- Weight update:

Weight associated to x_i during the j boosting round

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where Z_j is the normalization factor

- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to 1/n and the resampling procedure is repeated
- Classification:

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

AdaBoost Algorithm

Algorithm 5.7 AdaBoost Algorithm

1: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Initialize the weights for all n instances.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all instances in the original training set, D.
7: $\epsilon_i = \frac{1}{n}\left[\sum_j w_j \, \delta(C_i(x_j) \neq y_j)\right]$ {Calculate the weighted error}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Reset the weights for all n instances.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each instance according to equation (5.88).
14: **end for**
15: $C^*(\mathbf{x}) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y))$.

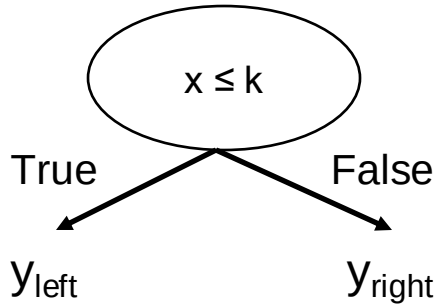
AdaBoost Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \leq k$ versus $x > k$
  - Split point k is chosen based on entropy



AdaBoost Example

- Training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- Weights:

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

AdaBoost Example

- Summary:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

- Classification

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

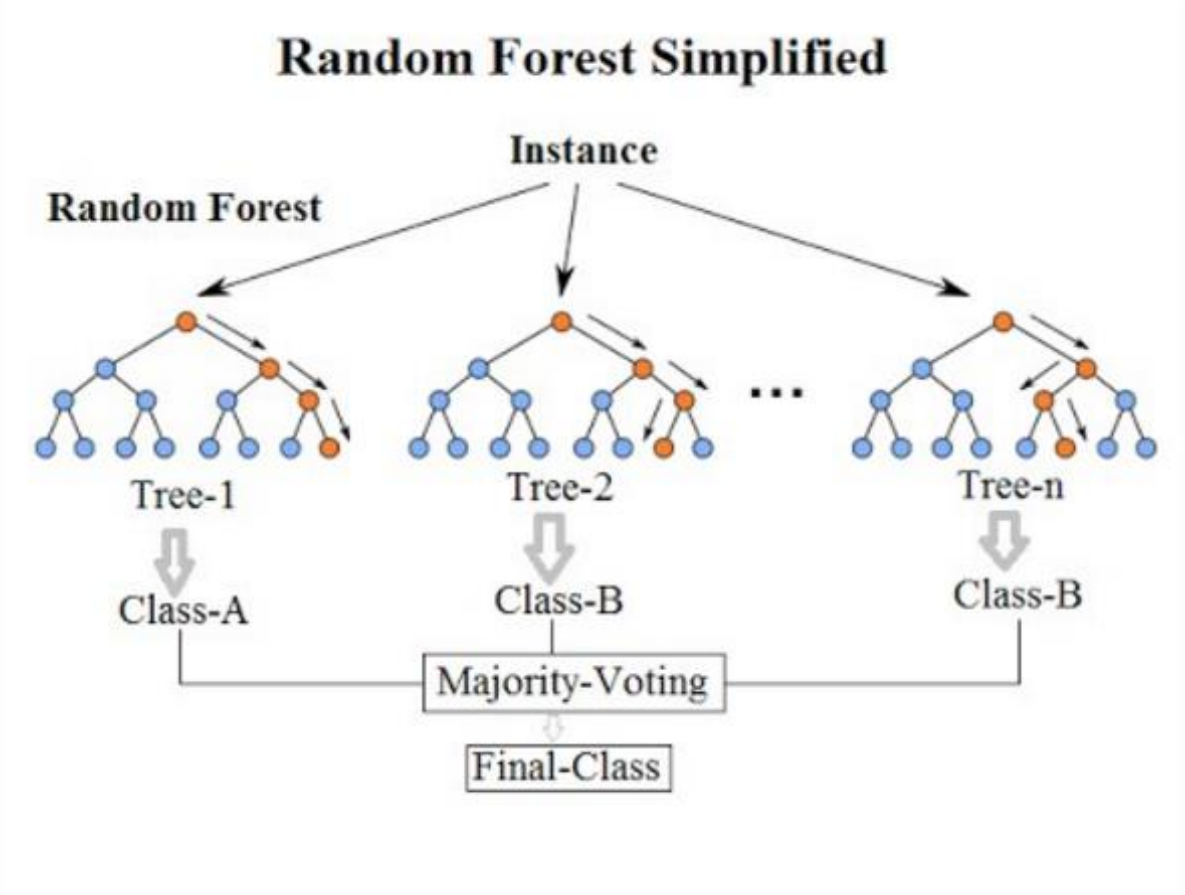
Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Random Forests

Random Forests

- Is a class of ensemble methods specifically designed for **decision trees**.
- It combines the predictions made by multiple decision trees and outputs the class that **is the mode of the class's output** by individual trees.



Only Bagging?

- With bagging we have an ensemble of structurally similar trees. This causes **highly correlated trees**.

- Random Forest aims at creating trees that have **no correlation** or **weak correlation**.

RF solution

- To de-correlate the trees:
  1. Take a random sample of size N with replacement from the data.
  2. Learn a decision tree where for each split a random sample without replacement of *m* attributes is considered

- m ~ $\sqrt{d}$ or m ~ $\log d$+1 turns out to be a good choice

- *Example*: if we have 100 features, each split will be allowed to choose from among 10 randomly selected features

Random Forest - Advantages

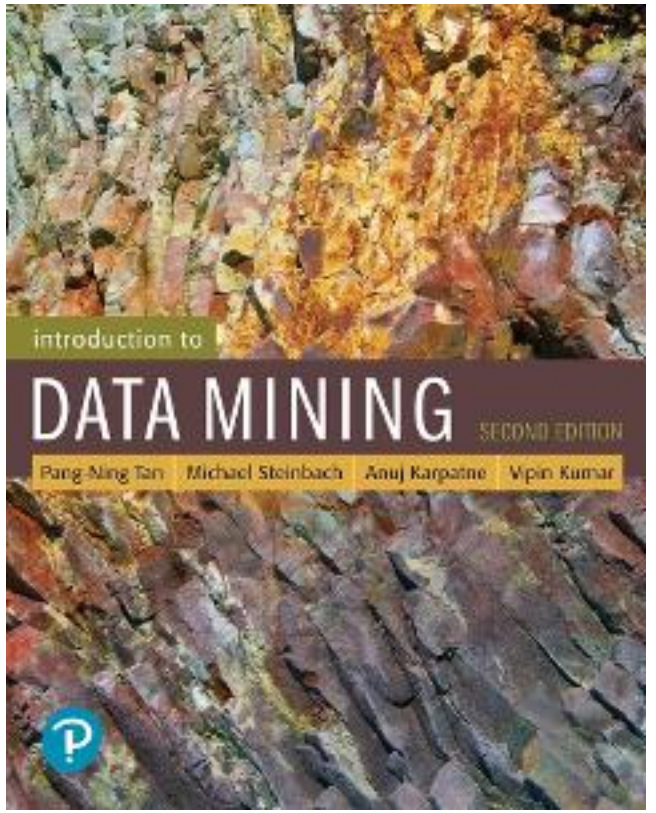
- It is one of the most accurate learning algorithms available. For many data sets, it produces a high accurate classifier.
- It runs efficiently on large databases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.

Final Thoughts on Random Forests

- When the number of attributes is large, but the number of relevant predictors is small, random forests can perform poorly.
- In each split, the chances of selected a relevant attribute will be low and hence most trees in the ensemble will be **weak models**.
- Increasing the number of trees in the ensemble generally does not increase the risk of overfitting.
- Decomposing the generalization error in terms of bias and variance, we see that increasing the number of trees produces a model that is at least as robust as a single tree.
- However, if the number of trees is too large, then the trees in the ensemble may become more correlated, increase the variance.

References

- Ensemble Methods. Chapter 5.6. Introduction to Data Mining.



LightGBM

1. Based on the concept of gradient boosting
2. Designed for large datasets
3. Fast and scalable
4. Based on leaf-wise tree growth strategy

What is gradient boosting?

It is based on the concept of gradient descent.

We train the model by minimizing the loss function.

Hence we need (again!):

1. Loss function
2. Learning rate

What is gradient boosting?

Gradient because it exploits the gradients of the loss function to find the direction in which we improve our model.

Boosting because it combines weak learners, focusing on the previous errors.

LightGBM: step by step

1. Start with an initial prediction (prior probability)
2. Evaluate the residuals (difference between the prediction and the correct outcome). Mathematically:

$$r_i^{(t)} = -\frac{\partial L(y_i, F_t(x_i))}{\partial F_t(x_i)}$$

1. Train a DT on the training set and its residuals. The target is not anymore the classes, but the residuals. The objective is to reduce the model errors, hence we can focus on the biggest residuals

LightGBM: step by step

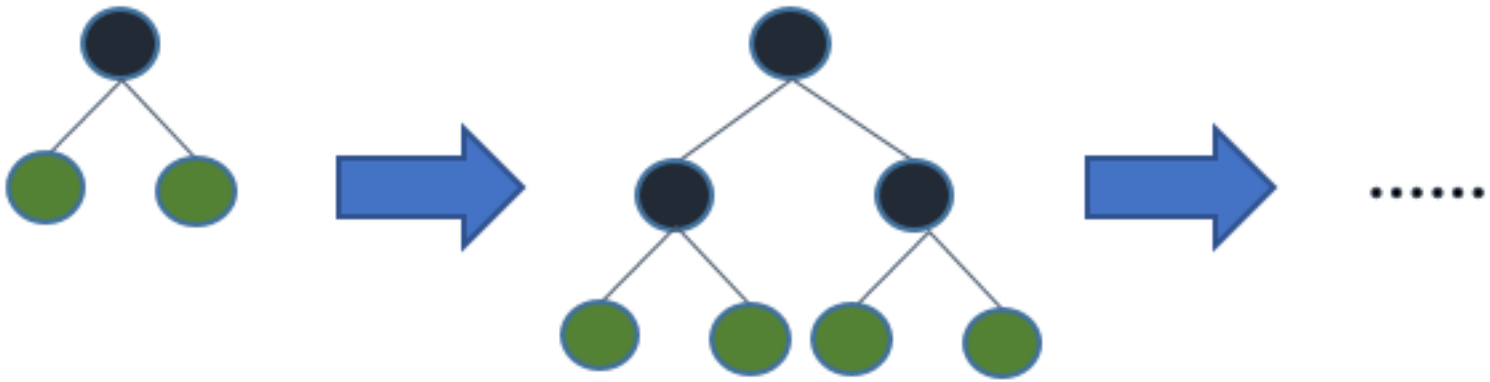
1. The tree predictions are used to update the model:

$$F_{t+1}(x) = F_t(x) + \eta \cdot f_t(x)$$

We scale it by a learning rate

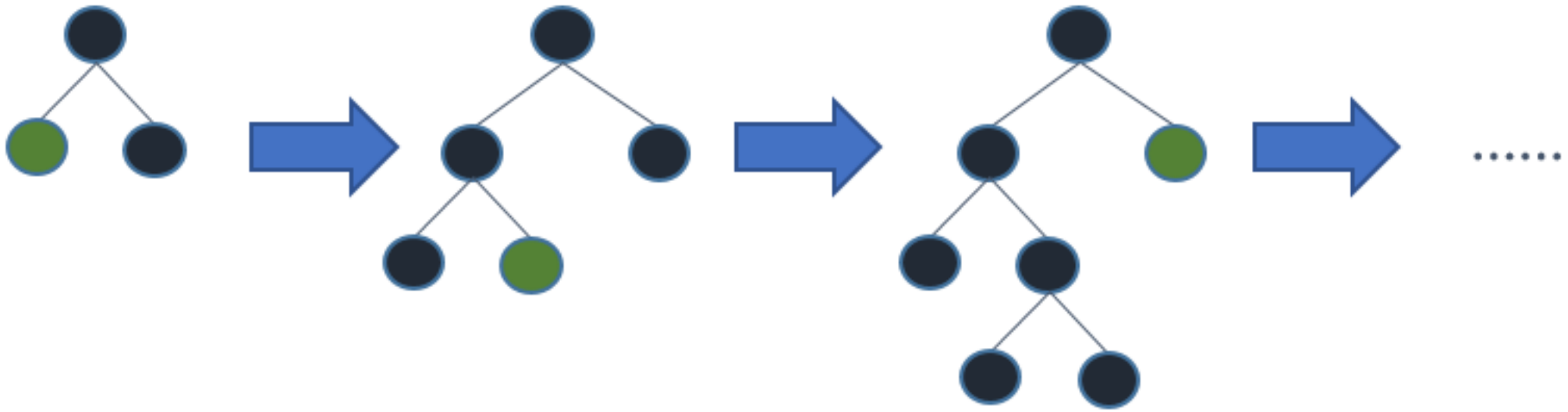
The intuition is that, adding tree by tree, you correct the errors and obtain a better model.

LightGBM: leaf-wise growth strategy



Level-wise tree growth

LightGBM: leaf-wise growth strategy



Leaf-wise tree growth

LightGBM: step by step

LightGBM grows the leaf based on the split that gives the highest reduction in the loss function (the maximum gain). For this reason the final tree has a strange structure, with different levels.

LightGBM: step by step

1. Fast
2. Fewer trees in the end: since the trees can grow deep where needed (instead of uniformly across all paths), LightGBM often requires fewer trees to achieve the same performance as level-wise growth.
3. Good accuracy
4. If we have small dataset, this method can overfit

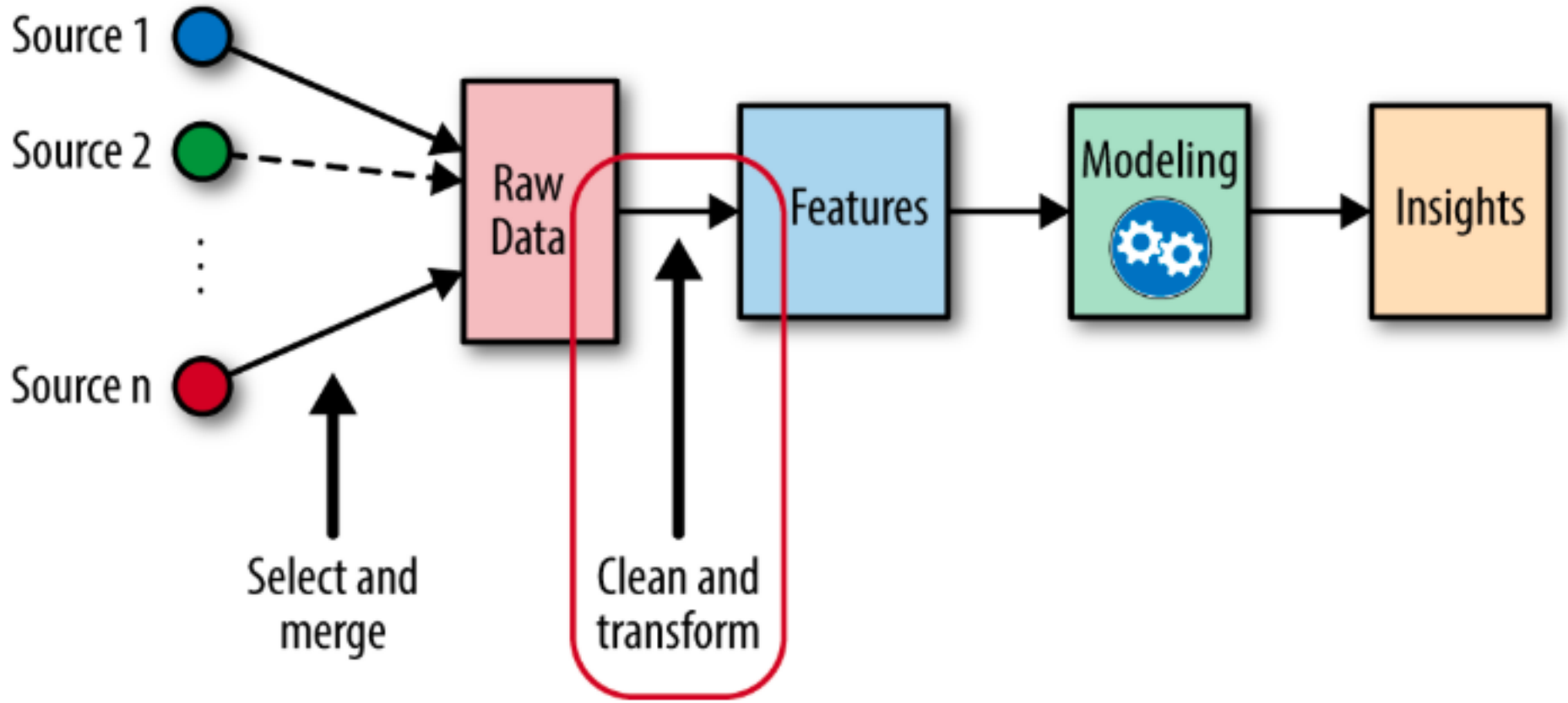
LightGBM: pros

1. Good Accuracy
2. Fewer trees thanks to leaf-wise growth
3. Fast, low memory usage
4. Good with big datasets
5. Handle imbalance datasets
6. To control overfitting, it uses regularization terms

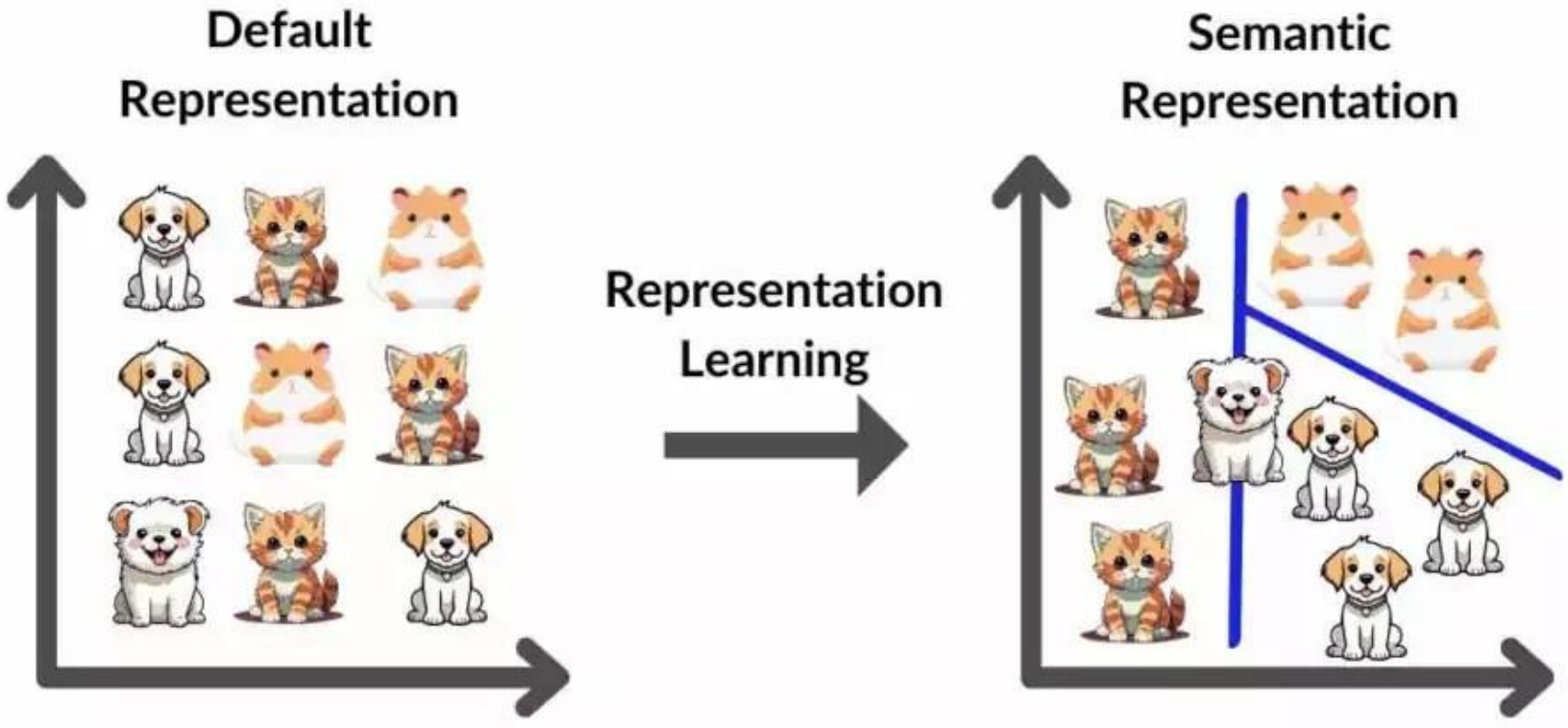
LightGBM: cons

1. Overfitting when small datasets are used
2. Needs a good parameter tuning
3. Difficult to handle sparse data (XGBoost may be a good option)
4. Difficult in handling categorical (CatBoost may be a good option)

A recap: ML pipeline



Representation learning



Representation learning

Representation learning is a method of training a ML model to discover and learn the most useful representations of input data automatically.

Representation learning offers a way for machines to autonomously grasp and condense the information stored in large datasets, making the steps in ML that follow after more informed and efficient.

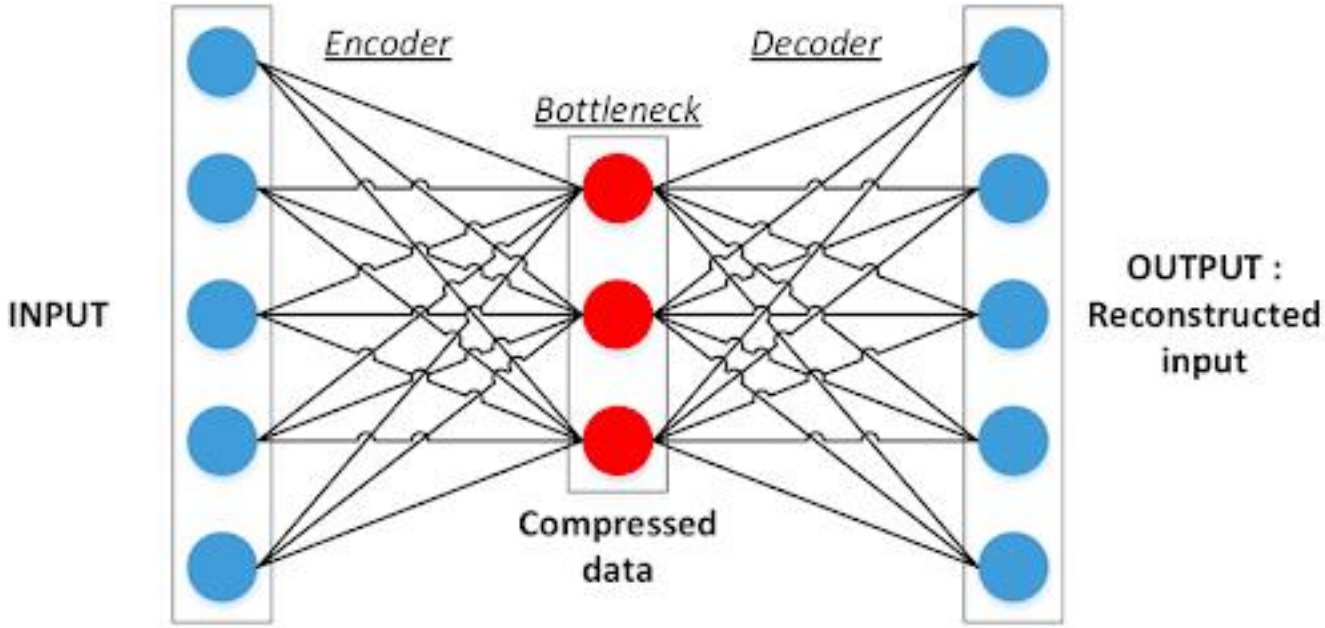
Autoencoders, CNN, DNN

To achieve representation learning we use Deep Neural Networks.

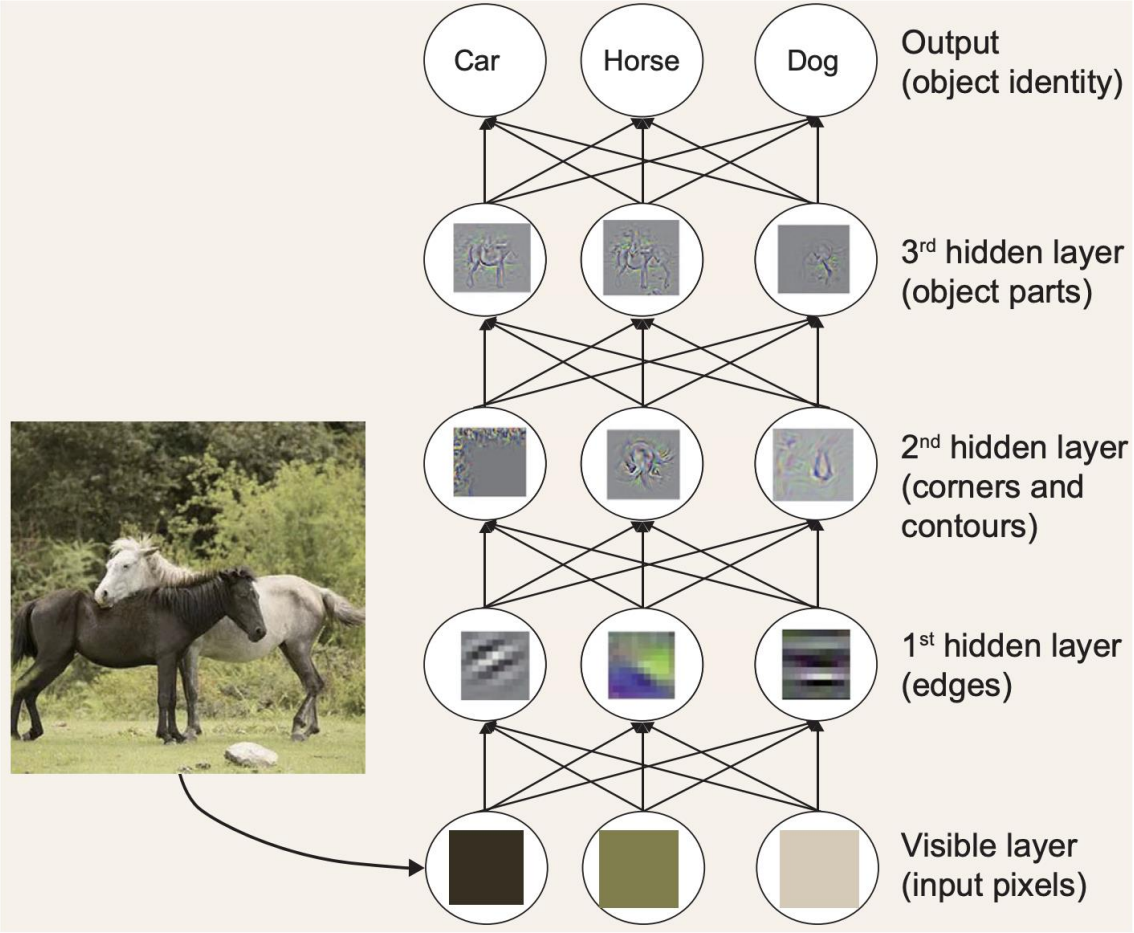
As an example, we can consider Autoencoder.

It is a type of neural network that learns to encode input data into a lower-dimensional, and thus more compact, form. The network then uses this encoded form to reconstruct the original input. The encoding process discovers and extracts essential features in the data, while the decoding process ensures that the extracted features are representative of the original data.

Autoencoders, CNN, DNN



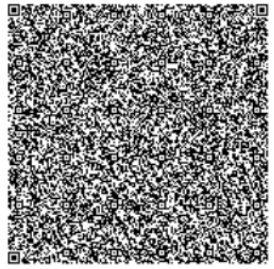
Autoencoders, CNN, DNN



Why are they better?

1. IN MODEL FEATURE REPRESENTATION: They create an internal representation with novel variables, while the models seen so far, instead, work on the original representation of the data (the variables we fed to the model).
2. SUFFICIENT MODEL COMPLEXITY : They can be arbitrarly complex and hence they can work with complex data, while some methods, like logistic regression are much easier.
3. LAYER-BY-LAYER pre-processing: the structure of the NN allows a layer by layer analysis.

Deep Forest



However, recently, a novel model has been proposed. It is NOT a NN, but it achieves great performance and is able to perform also representation learning.

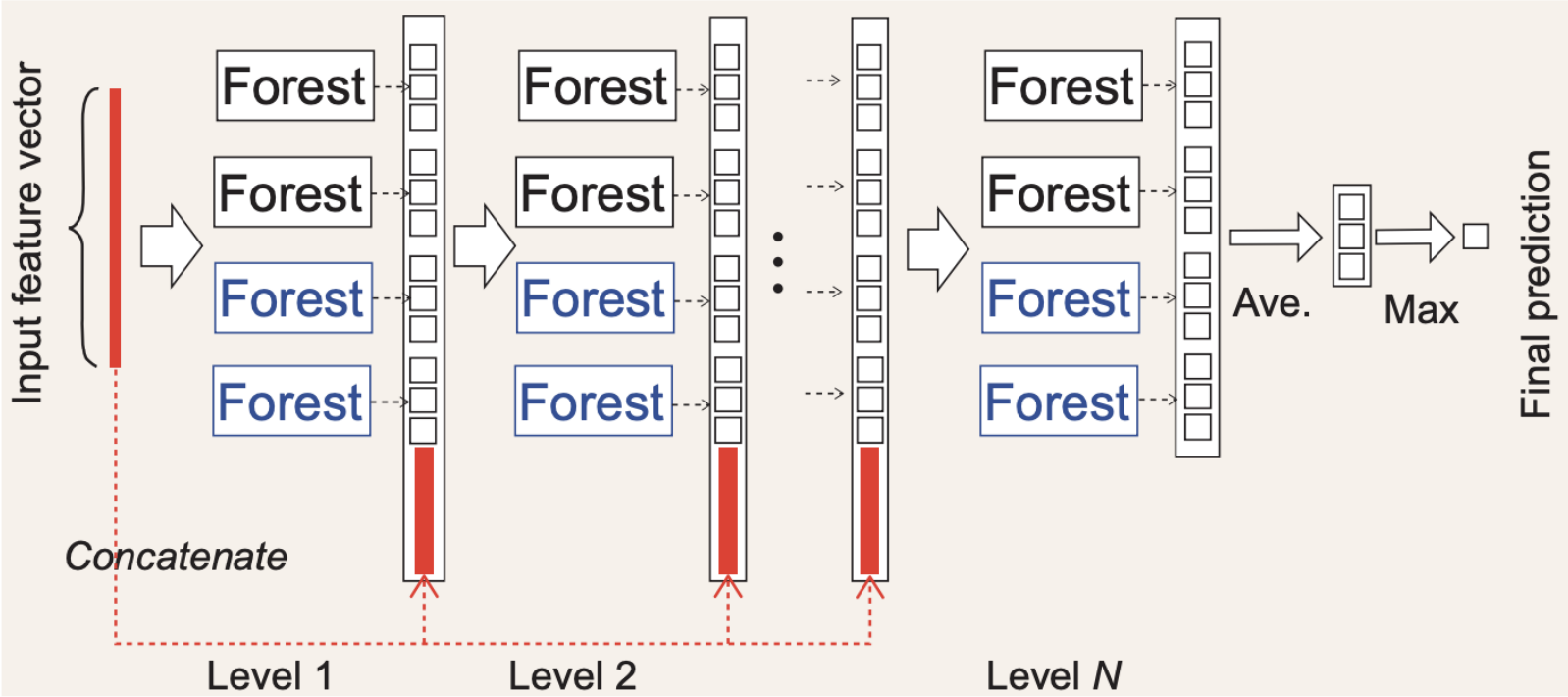
It uses:

1. Random Forests
2. Completely Random Forest (trees are generated without any deterministic heuristic)

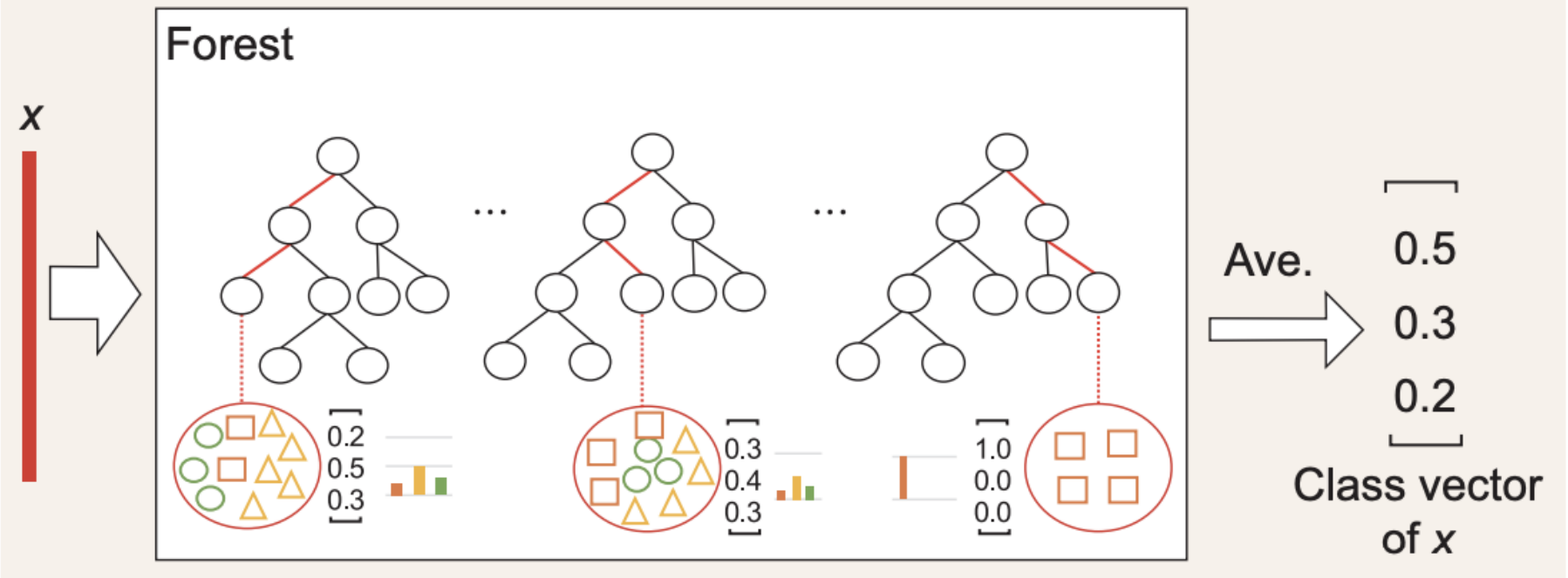
Deep Forest

A tree-based ensemble method, with a cascade structure.

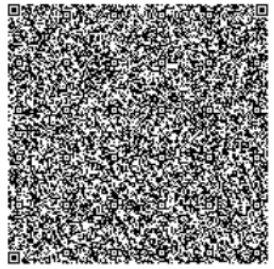
1. Layer-by-layer -> cascade structure



Deep Forest



Deep Forest



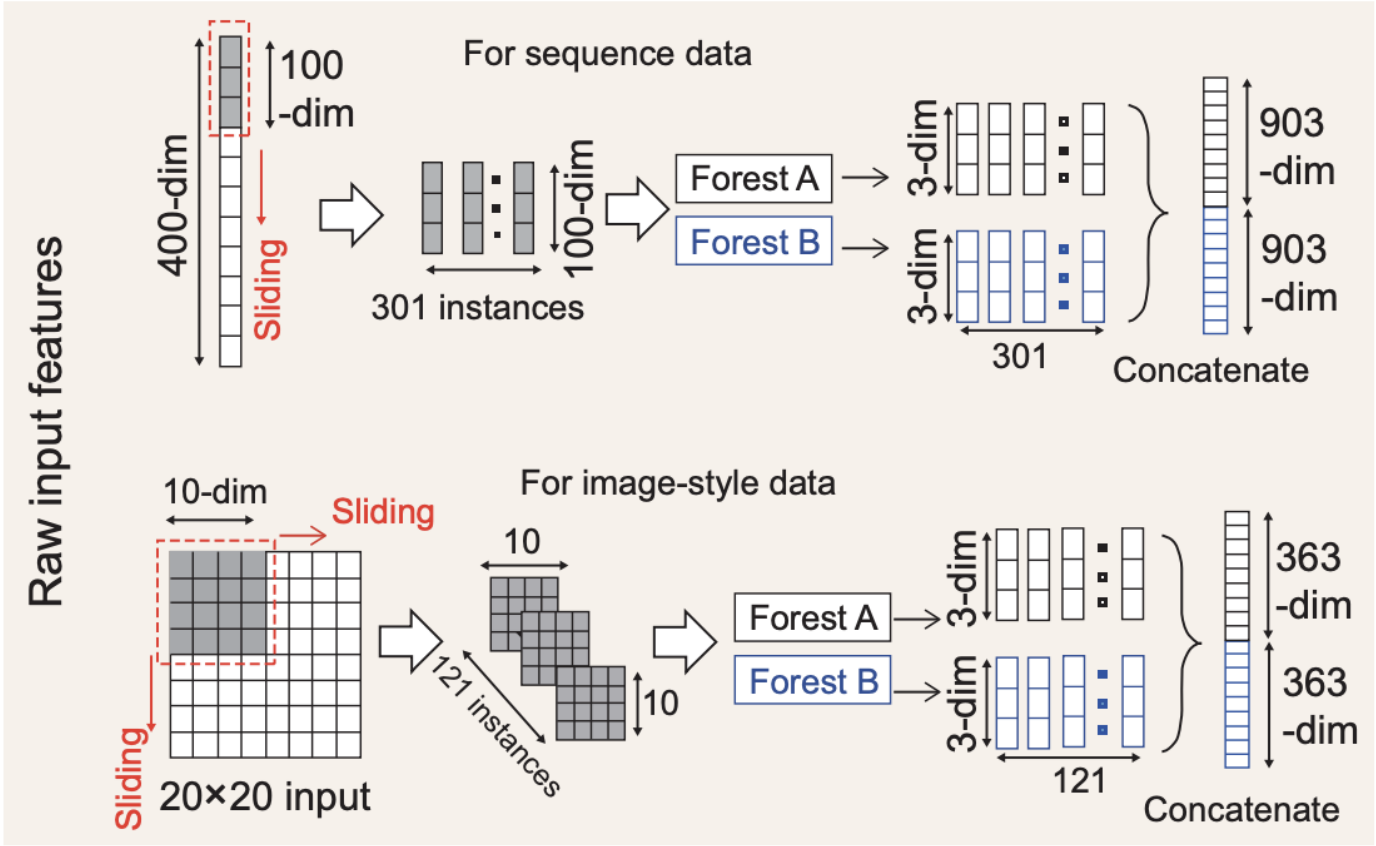
A tree-based ensemble method, with a cascade structure.

2. Complexity -> just add layers to the cascade + attention in overfitting the forests!

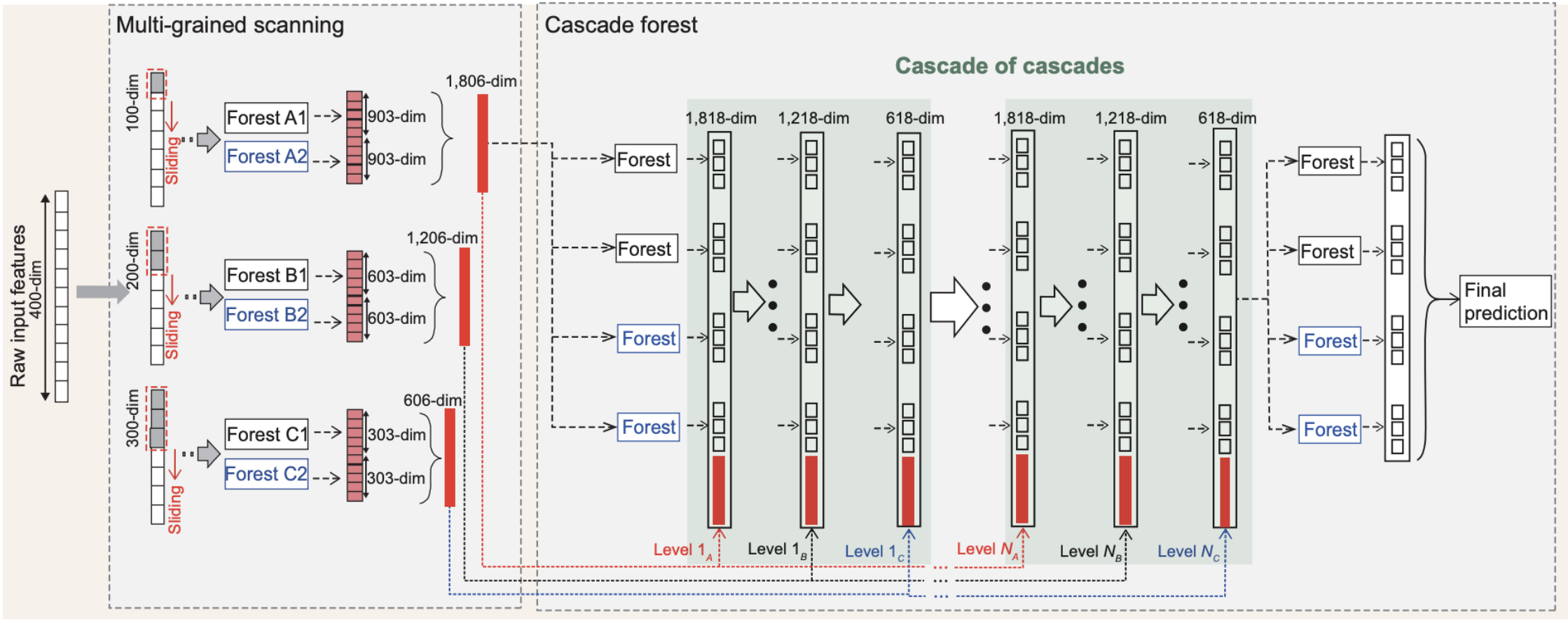
Deep Forest

A tree-based ensemble method, with a cascade structure.

3. Representation learning -> multi-grained scanning



Deep Forest


Multi-grained scanning
Cascade forest
Cascade of cascades
Raw input features
400-dim
100-dim
200-dim
300-dim
Sliding
Forest A1
Forest A2
Forest B1
Forest B2
Forest C1
Forest C2
903-dim
603-dim
303-dim
1,806-dim
1,206-dim
606-dim
Forest
1,818-dim
1,218-dim
618-dim
Level 1A
Level 1B
Level 1C
Level NA
Level NB
Level NC
Final prediction