"UniTs" - University of Trieste

Faculty of Data Science and Artificial Intelligence

Department of mathematics informatics and geosciences

Introduction to Machine Learning

Lecturer:
Prof. Eric Medvet

Author:
Christian Faccio

January 8, 2025

github.com/christianfaccio christianfaccio@outlook.it

Abstract

As a student of the “Data Science and Artificial Intelligence” master’s degree at the University of Trieste, I have created these notes to study the course “Introduction to Machine Learning” held by Prof. Eric Medvet. The course aims to provide an introduction to the field of machine learning, covering the fundamental concepts in theoretical terms. The topics are the followig:

- Supervised learning techniques
- Unsupervised learning techniques
- Dimensionality reduction (not covered in the lectures)
- Introduction to NLP

While these notes were primarily created for my personal study, they may serve as a valuable resource for fellow students and professionals interested in this field.

Contents

1 Basic Concepts

What is Machine Learning?

Definition: *Machine Learning*

Machine Learning is the science of getting computers to learn something without being explicitly programmed.

- What science?
- Who is doing that?
- To learn what?
- Who is not being explicitly programmed?

Let's formalize the decision making process:

$$y = f(x)$$

- x is the entity about which the decision has to be made
- y is the decision
- f is the procedure that, given an "x" results in a decision "y"

X is an **observation**, while Y is a **response**.

So, the function f is what has to be learned, and is also denoted as $f_{predict}$, since it predicts a response ($\hat{y}$) given an observation.

$$f : R^2 \to R$$

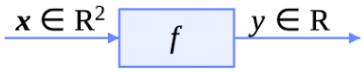


Figure 1.1: Abstract definition of the learning process

The learning process is not written by a human, so we can say that the machine learns `autonomously`.

- condsider the universe of functions $F_{X \to Y} = \{f, f : X \to Y\}$
- choose the function that does what expected

Definition: *New Definition*

Machine Learning is the science of getting computers to learn $f_{predict} : X \to Y$ **autonomously**.

There has to be some sort of `supervision` facilitating the search of a good f and when it is in the form of some **examples** and the function should process them correctly we are in the field of **Supervised Machine Learning**.

- **Supervised Learning**: the function is learned from a set of labeled examples
- **Unsupervised Learning**: finds structures in the data without any labels
- **Reinforcement Learning**: the function is learned from a set of examples and a reward signal

Examples are pairs of observations and responses (x, y), and the set of examples is called **training set** (or learning set). The `dataset` is a bag of pairs of observations and responses.

The learning set has to be consistent with the domain and codomain of the function $f_{predict}$ to be learned.

$$f_{predict} = f_{learn}(D_{learn})$$

- **learning phase**: the function f_{learn} is learned from the learning set(i.e. is applied to obtain $f_{predict}$ from D)
- **prediction phase**: the function $f_{predict}$ is used to predict the response (y) of new observations

Definition: *Supervised Learning Technique*

A **Supervised Learning Technique** is a way to learn an $f_{predict} \in F_{X \to Y}$ given a $D_{learn} \in P^*(X \times Y)$

Different SL techniques differ in applicability, efficiency and effectiveness.

- **Applicability**: with respect to X andor Y, e.g. some require $X = R^p$, some require $Y = R$
- **Efficiency**: the computational resources needed to learn the function wrt $|D_{learn}|$, e.g. some are fast and other slow
- **Effectiveness**: the quality of the learned functions $f_{predict}$

There is another view of it, and I like it more:

Definition: *SL Technique as Optimisation*

A **Supervised Learning Technique** is a way to solve the optimization problem:

$$f_{predict} = argmin_{f \in F_{X \to Y}} L(f, D_{learn})$$

where L is a loss function that measures the quality of the function f wrt the learning set D_{learn} and $F_{X \to Y}$ is the set of functions that can be learned.

The most practical solution is to reduce $F_{X \to Y}$ size by considering only the f of some nature.

`Templating` f means defining a relationship between data as we do in statistics, specifying it with some coefficients that we can evaluate.

We can then explicit the unknown parameters and specify the template as a reduced F'.

$$f_{predict} = f'_{predict}(x, m)$$

Where m is the `model` of how y depends on x.

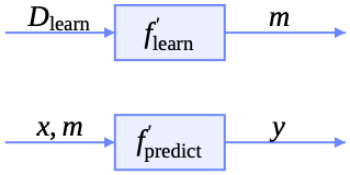


Figure 1.2: Learning a model with template function

Definition: ***ML system***

An **information processing system** in which there is a SL technique and a pair of **pre-processing** and **post-processing** steps.

The designer of the ML system has to choose the learning technique, the pre/post processing steps and the relative parameters. The steps are:

- Decide if to use a ML system
- Supervised vs Unsupervised vs Reinforcement
- Define the problem (X, Y, way of assessing solutions)
- Design the ML system (choose the ML technique, pre/post processing steps)
- Implement the ML system (learning/prediction phases, obtaining the data)
- Assess the system

2 Supervised Learning

2.1 Assessment

The main axes of assessment are:

- `Effectiveness`: how well the model performs on the test set.
- `Efficiency`: how much time and resources are needed to train the model (achieving the goal).
- `Interpretability`: how well the model can be understood by humans.

There are two purposes for assessment:

- **Absolute Assessment:** does something meet the expectation with respect to a determined axis?
- **Comparison:** is one thing better than one other thing in terms of the determined axis?

If the output of the assessment is a `quantity`, than a simple check for < or > is enough, so we want a number as output.

A **ML system** can be seen as a composite learning technique. It has two running modes: one in which it tunes itself and the other in which it makes decisions. The goals are:

- **Training:** tune the model to minimize the error on the training set (tuning properly).
- **Testing:** evaluate the model on the test set (making good decisions).

The SLT has then this two goals, while a **model** has the only goal to make good decision when used in an $f'_{predict}$.

To measure the **effectiveness** of a model and to have a number as output, we need to compare our system with the real system that generated the data (assuming there is one).

Figure 2.1: Model vs Real System

Definition: ***Model vs Real System***

- Collect examples of **s** behavior
- Feed **m** with examples
- Compare responses of **s** and **m**

Effectiveness: to which degree the comparison step measures if m behaves like s.

Warning: $f_{collect}$

The data collection is really important, since:

- small n → **poor** effectiveness, **great** efficiency
- large n → **good** effectiveness, **poor** efficiency

(effectiveness = accuracy, efficiency = resources)
and also:

- poor coverage → **poor** effectiveness
- good coverage → **good** effectiveness

where coverage = how well the data represents the real system.

Figure 2.2: Comparing the responses

where $(y^{(i)}, \hat{y^{(i)}})_i \in P^*(Y^2)$ is a multiset of pairs of y.

Example: ***Performance Indexes***

Classification:

- all types → error, accuracy
- binary → EER, AUC, FPR, FNR and variants
- multi-class → weighted accuracy

Regression:

- MSE, MAE, R2

2.1.1 Classification

In `Classification` Y is a finite set with no ordering.

The **Classification Error** is then

$$f_{err}(y^{(i)}, \hat{y^{(i)}}) = \frac{1}{n} \sum_{i=1}^{i=n} \mathbf{1}(y^{(i)} \neq \hat{y^{(i)}})$$

where **1(b)** is the indicator function and takes value 1 when (in this case) the prediction is wrong.

The **Classification Accuracy** is instead:

$$f_{acc}(y^{(i)}, \hat{y^{(i)}}) = \frac{1}{n} \sum_{i=1}^{i=n} \mathbf{1}(y^{(i)} = \hat{y^{(i)}})$$

Reminder: $f_{acc} = 1 - f_{err}$

Extreme cases for accuracy (and errors) are:

- m is s: **perfect model**
- m is random: does not model any dependence

Example: *Random Classifier - Lower Bound*

$f_{random}(x) = y_i$ with $i\ U(1,\ldots,|Y|)$ just pick random uniformly

Example: *Dummy Classifier - Better Lower Bound*

$f_{dummy}(x) = argmax\frac{1}{n}\sum_{i=1}^{i=n}\mathbf{1}(y = y^{(i)})$ just pick the category with more example always

Example: *Perfect Classifier - Upper Bound*

$f_{perfect}(x) = s(x)$

But the world is not deterministic, so a Bayes (stochastic) classifier may be better:

Example: *Bayes Classifier - Better Upper Bound*

$f_{Bayes}(x) = argmaxPr(s(x) = y|x)$

2.1.2 Binary Classification

The main problem here are the `skewed datasets`, meaning the datasets with more example of a category than the other (we are considering only two categories here).

Definition: *FPR - False Positive Rate*

It is the rate of **negatives** that are wrongly classified as positives
$FPR = \frac{FP}{N}$

Definition: *FNR - False Negative Rate*

It is the rate of positives that are wrongly classified as negatives. $FNR = \frac{FN}{P}$

Where

- FP = False Positives
- FN = False Negatives
- P = Total positives
- N = Total Negatives

		Test Result	
		0	1
Ground Truth	0	TN (True Negative)	FP (False Positive)
	1	FN (False Negative)	TP (True Positive)

Figure 2.3: Confusion Matrix

`TPR:` True Positive Rate = 1 - FNR

`FPR:` False Positive Rate = 1 - TNR

`Err:` Error Rate = $\frac{FP+FN}{P+N}$

`Acc:` Accuracy = $\frac{TP+TN}{P+N}$

Definition: ***Balanced Data***

In classification, a dataset is **balanced** with respect to the response variable y, if the frequency of each value of y is roughly the same.

`Precision:` $\frac{TP}{TP+FP}$

`Recall:` $\frac{TP}{TP+FN}$

`F1:` $2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$

`Sensitivity:` $\frac{TP}{TP+FN} = TPR$

`Specificity:` $\frac{TN}{TN+FP} = TNR$

`Type I error:` False Positive Rate = FPR

`Type II error:` False Negative Rate = FNR

Warning: ***Cost of the error***

Once $f_{predict}$ outputs a y, some action is taken, and if the action is wrong, there is some cost to be paid wrt the correct action.
$c = c_{FP} \times FPR \times N + c_{FN} \times FNR \times P$
If one knows the costs and N,P, it is possible to compute the overall cost and find a good trade-off for FPR and FNR.

Given a model, we can ”tune” it to prefer avoiding FPs rather than FNs, just by modifying the threshold that is used to determine the choice to be made wrt the category. Many learning techniques compute a probability distribution over Y before returning one y, and in the case of Binary Classification the threshold is 0.5.

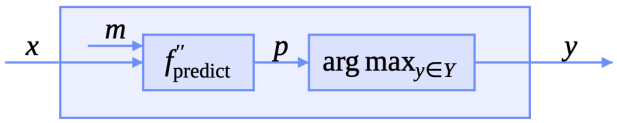


Figure 2.4: Threshold

So, by varying the threshold we can change FPR and FNR:

- the greater the threshold, the lower FPR and the greater FNR
- the lower the threshold, the greater FPR and the lower FNR

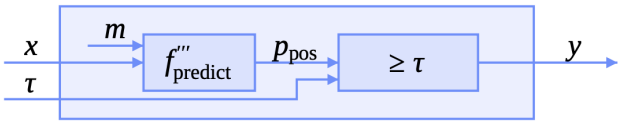


Figure 2.5: Threshold

Definition: *ROC Curve (Receiver Operating Characteristic)*

The ROC curve is a graphical representation of the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) for every possible threshold.

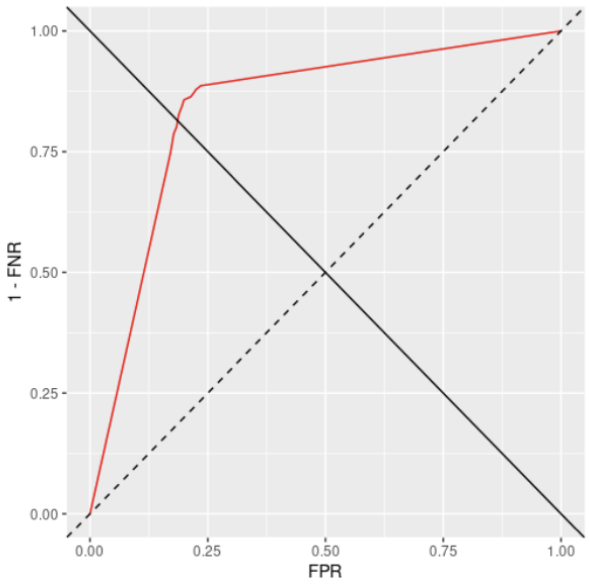


Figure 2.6: ROC Curve

The dotted line represents the random classifier, while the closer the curve is to the top-left corner, the better the classifier is.

How to choose the threshold? Take different values and compute the ROC curve, then choose the one that is closer to the top-left corner. To do this, the best way is to take the **midpoints** of the sorted probabilities.

2.1.3 Multiclass Classification and Regression

Here we use the concept of **weighted accuracy**, which is

$f_{acc} = \frac{1}{|Y|} \sum_{y \in Y} Acc_y$

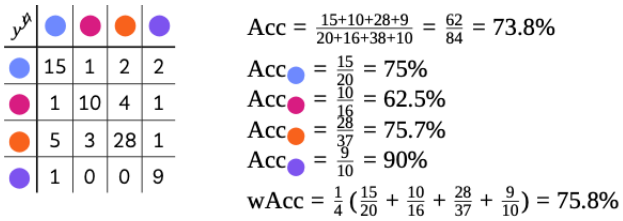


Figure 2.7: Multiclass Confusion Matrix

Moreover, the error there is different from the classification one: here we measure the `distance` from the real value.

- `MAE:` Mean Absolute Error = $\frac{1}{n} \sum_{i=1}^{i=n} |y^{(i)} - \hat{y^{(i)}}|$
- `MSE:` Mean Squared Error = $\frac{1}{n} \sum_{i=1}^{i=n} (y^{(i)} - \hat{y^{(i)}})^2$
- `RMSE:` Root Mean Squared Error = $\sqrt{MSE}$
- `MAPE:` Mean Absolute Percentage Error = $\frac{1}{n} \sum_{i=1}^{i=n} \frac{|y^{(i)} - \hat{y^{(i)}}|}{y^{(i)}}$

2.1.4 Assessing Learning Techniques

- an **effective** learning technique is a pair of f'_{learn}, $f'_{predict}$ that learns a good model m
- a good **model** is one that has the same behavior of the real system s

We, then, want to measure the effectiveness of the learning techniques (learning and prediction). To do so, we need to divide the dataset, since we want to see if the $f'_{predict}$ generalises well.

Definition: ***Static Train/Test Division***

- **Effectiveness:** generalization is assessed, but there is no robustness wrt D division
- **Efficiency:** learning is executed only once

D_{test} is the unseen data and we treat is as said even if it has been collected at once with D_{train}.

Definition: ***Repeated random train/test division***

- **Effectiveness:** generalization is assessed, and there is robustness wrt D division
- **Efficiency:** learning is executed multiple times $\propto$k

Definition: ***Cross-Validation***

- **Effectiveness:** generalization is assessed, and there is robustness wrt D division
- **Efficiency:** learning is executed multiple times $\propto$k

Definition: *Leave-One-Out*

- **Effectiveness:** generalization is assessed, and there is robustness wrt D division
- **Efficiency:** learning is executed multiple times $\propto n$

This is the worst case for efficiency, since we repeat the learning phase n times.

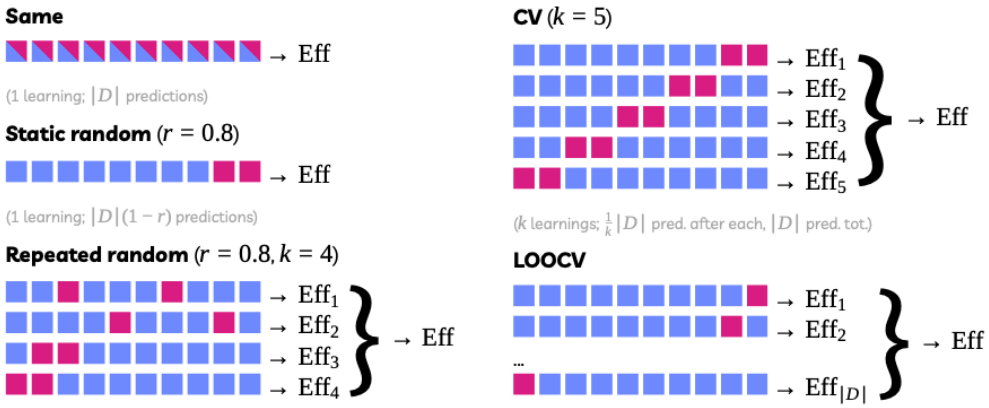


Figure 2.8: Train/Test Division Methods

For comparison btw learning techniques, we can use the mean and standard deviation of the performance indexes obtained with the previous division methods. **Boxplots** are useful in this case.

2.2 Tree-based Learning Techniques

2.2.1 Decision Trees

A **Decision Tree** is a tree where each node is a decision on the value of a feature, and each branch is a possible value of the feature. The leaves are the categories. It can be understood in simpler terms by considering a basic `if-then-else` nested structure.

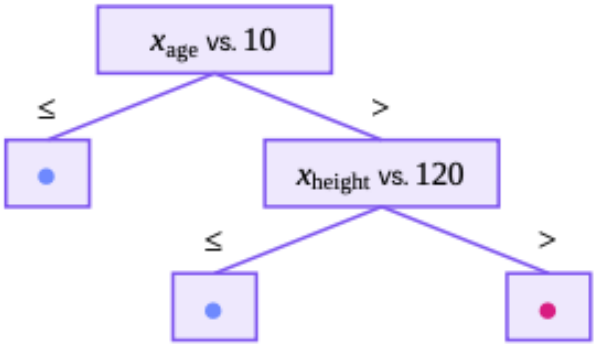


Figure 2.9: Decision Tree example

This is a binary tree, but it can be n-ary as well. The tree is built by recursively splitting the dataset into subsets, and the splitting is done by selecting the feature that best splits the dataset into the purest subsets. The purity is measured by the `Gini Index` or the `Entropy`, usually.

One can also represent the tree in a compact way:

$$t = [l, t', t'']$$

where $l \in L$ is the **label** and $t', t'' \in T_L \cup \{\emptyset\}$ are the left and right children.

Also, each **non-terminal** node is a pair (j, τ), where $j \in \{1, \ldots, p\}$ is the index of the independent variable and $\tau \in R$ is the threshold for comparison.

The above figure is then represented as:

$$t = [(1, 10); [\bullet]; [(2, 120); [\bullet]; [\bullet]]]$$

Figure 2.10: Compact representation of the Decision Tree

```
function predict(x, t)
    if ¬has-children(t) then return label(t)
    else {
        (j, τ) = label(t)
        if x[j] ≤ τ then return predict(x, left(t))
        else return predict(x, right(t))
    }
```

Algorithm 1: Templated $f'_{predict}$

This is a recursive function that works with every tree and always returnns a $y \in Y$.

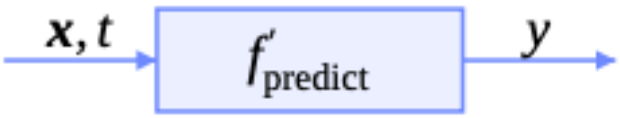


Figure 2.11: Templated $f'_{predict} : (R^p \times T_{p,Y}) \to Y$

Warning: *When to stop?*

The tree can grow indefinitely, so we need to stop it at some point. We can do this by:

- **Depth Limit:** stop when the depth is reached
- **Leaf Size:** stop when the leaf size is reached
- **Impurity:** stop when the impurity is below a certain threshold

Also, how to find where to split the data? The algorithm searches for the best point wrt the accuracy, meaning it will find the point where accuracy (or misclassification to the contrary) is the best, that's why it is called `greedy`.

Other approaces are:

- **Gini Index:** $Gini(t) = 1 - \sum_{y \in Y} (Pr(y|t))^2$
- **Entropy:** $Entropy(t) = -\sum_{y \in Y} Pr(y|t) log_2(Pr(y|t))$

They are both measures of impurity, and the best split is the one that minimizes the impurity. **The lower the better**.

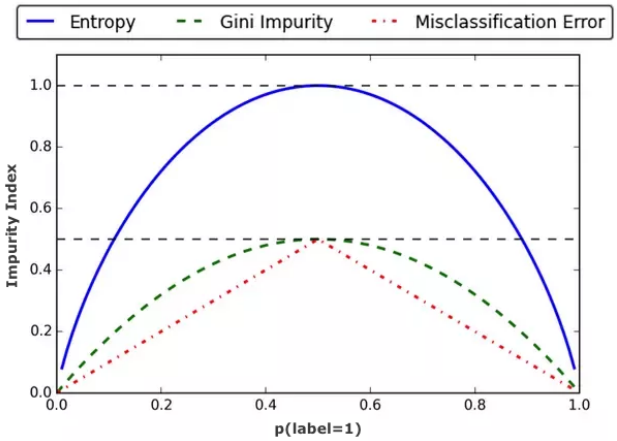


Figure 2.12: Gini Index, Entropy and Error

Gini and Entropy are both smoother than the error, and they are similar, but Gini is faster to compute.

2.2.2 Tree Learning with probabilities

In this context terminal nodes **return discrete probability distributions**:

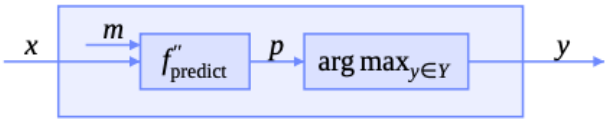


Figure 2.13: Tree Learning with probabilities

$$f''_{predict} : X_1 \times \cdots \times X_p \times T_{(\{1,\ldots,p\} \times R) \cup P_y}$$

So the output will be the `frequency` of the categories.

Warning: *Model Complexity*

Sometimes the tree could be too large wrt the real system. Almost every learning technique has at least one parameter that affects the maximum complexity of the learnable models, often called **flexibility**:

- more flexibility $\rightarrow$ more complex models
- less flexibility $\rightarrow$ simpler models

The point is: the model tries to reach the maximum accuracy possible, but there is `noise` in the data, that leads to an impossible perfect score.

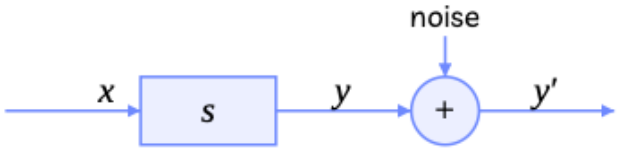


Figure 2.14: Noise in the data

However, out goal is to **model s**, not s+noise.

Definition: *Overfitting*

When we have a **noisy dataset** and we allow for **large complexity**, by setting a flexibility parameter to a high flexibility, the learning technique **fits the noisy data** instead of the real system s, and **overfitting** occurs.

Underfitting is the opposite, when the model is too simple to capture the real system.

- Underfitting exibits **high bias** and **low variance**, since it tends to generate models that incorporate a bias towards some y values
- Overfitting, instead, exibits **low bias** and **high variance**, since it tends to generate different models if the learning is repeated with different datasets

Example: *Practical Procedure*

- consider different values of the flexibility parameter
- choose a suitable **effectiveness index**
- choose a suitable **learning/testing division method**
- learn a model and compute the performance index on the train and test set

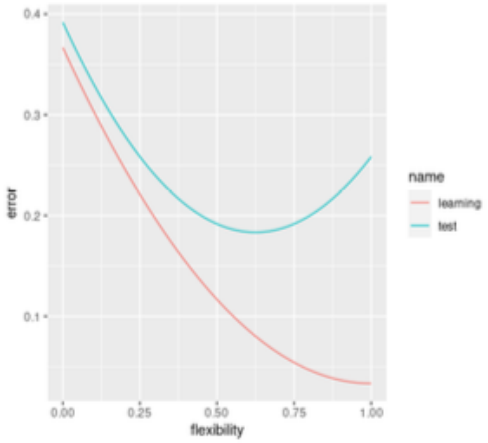


Figure 2.15: Bias-Variance Trade-off

Definition: *Hyperparameter tuning*

It is the task of finding the tuple of hyperparameters $p_1^*, \dots, p_h^* \in P_j$ that maximizes the effectiveness index.

Hyperparameters are the parameters that are not learned by the learning technique, but are set by the user.

Example: *Grid Search*

It is a simple form of hyperparameter tuning, where the user specifies a grid of values for each hyperparameter, and the learning technique is executed for each combination of hyperparameters. To be feasible, the set of hyperparameters must be small.

So why we can't just always use hyperparameter tuning instead of choosing a value? Well, it is expensive and probably lacks of generalization since it is specific fot that dataset.

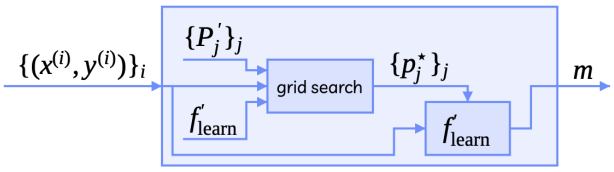


Figure 2.16: Hyperparameter-free Learning

2.2.3 Categorical Independent Variables and Regression

Here we find a variable x_j and a set of values $X'_j \in X_j$ that well separates the data. It's a simple binary split, where the x_j is contained either in a group or the other.

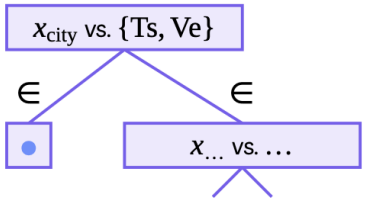


Figure 2.17: Tree with categorical variables

For a given **categorical variable** $x_j \in X_j$, we choose $X'_j \subset X_j$ such that:

$$X'_j = \text{argmin}_{X'_j \in P(X_j)} f_{impurity}(\{y^{(i)}\}_i|_{x_j^{(i)} \in X'_j}) + f_{impurity}(\{y^{(i)}\}_i|_{x_j^{(i)} \in X'_j})$$

In a **Regression Tree**, instead, the terminal nodes are no more categorical values, they are instead numerical ones. For this reason concepts like $f_{impurity}$ or minimizing the error to stop do not hold anymore: they must be changed.

To find the best branch, RSS is the solution.

$$(j^*, \tau^*) = \text{argmin}_{j,\tau} RSS(\{y^{(i)}\}_i|_{x_j^{(i)} \leq \tau}) + RSS(\{y^{(i)}\}_i|_{x_j^{(i)} \leq \tau})$$

Also for the **Stopping Criterion**, we should use RSS instead of the error, then stopping when $RSS = 0$ or when $n \leq n_{min}$. To notice that $RSS = 0$ is much more unfrequent than error$= 0$.

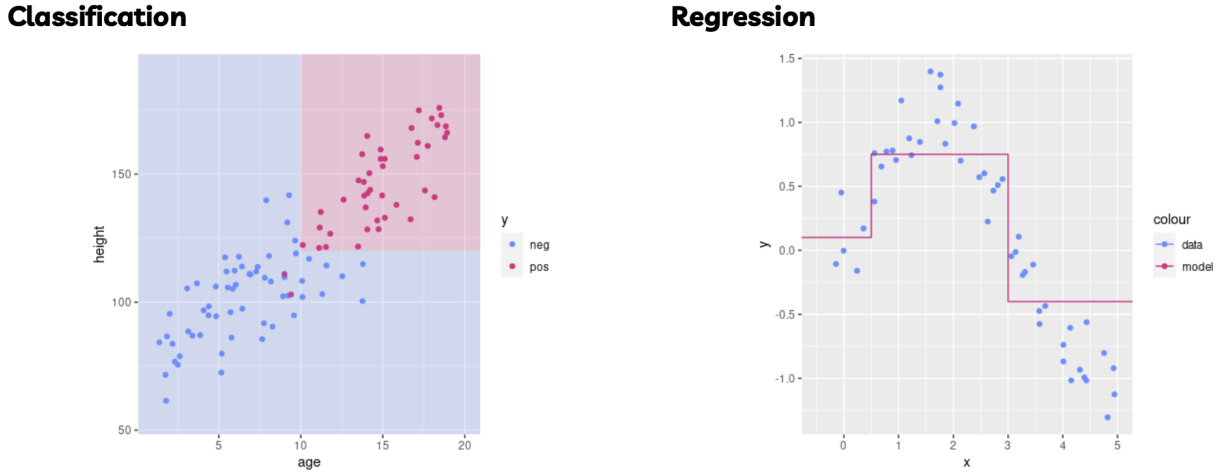


Figure 2.18: Classification Tree vs Regression Tree

2.3 Tree Bagging and Random Forest

2.3.1 Tree Bagging

If we learn a regression tree with **low flexibility**:

- the model will not capture the system behavior
- it will underfit the data and the system

If we learn a regression tree with **high flexibility**:

- the model will likely better capture the system behavior, but…
- it will model also some noise
- it will overfit the data

`What if we combine the power of the high flexibility obtained from different models?`

Definition: *Wisdom of the crowds theorem*

A collective opinion may be better than a single expert's opinion

Warning: *Wisdom of the crowds theorem*

Yes, but only if:

- we have many opinions →(learn many trees)
- they are independent →(each tree is learned on a dataset obtained with sampling with repetition)
- we have a way to aggregate them →(aggregate the predictions)

For the **independency** part, we just have to use `different learning sets`, using for example the Cross-Validation technique or, even better, the Bootstrap technique.

Figure 2.19: Tree bagging

Definition: *Tree Bagging*

- **Learning:** the model is a bag of trees and we can specify the number of trees as an hyperparameter. Remember that the learning technique is not deterministic due to the random sampling part.
- **Predict:** for classification consider the majority of votes for every $y \in Y$ (label), for regression simply return the mean of the relative values (better if also the σ is returned, to have a measure of the uncertainty)

Warning: n_{tree}*hyperparameter*

It is a flexibility parameter, but if increased changes overfitting and generalisation in the same way. Experimentally turns out that with a reasonably large n_{tree} bagging is better than single tree learning, and after that number there is no tendency to overfit. The problem is that the higher the parameter, the less efficient is the model. Simple trade-off.

2.3.2 Random Forest

The question is: can we further increase the independency such that the model will possibly learn different patterns in the data? The answer is yes, and we can do it by including more independency not only in the learning set, but also in the way the learning technique chooses the predictors each time. A sort of **independency in the predictors' sets**.

Idea: when learning each tree, **remove some randomly chosen independent variables** from the observations. This is called **Random Forest**, and is a learning technique that enhances the capabilities of Tree Bagging with variables removal.

- **Random:** because there are two sources of randomness, hence of independency
- **Forest:** because it gives a bag of trees

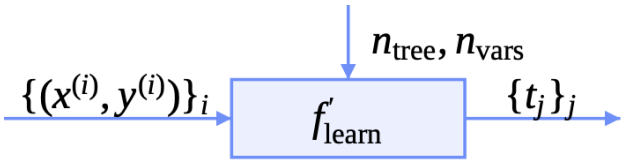


Figure 2.20: Random Forest

The model is a bag of n_{vars} trees, as in bagging, where that parameter is $\leq p$ that is the number of variables. Two parts of this learning technique are **not deterministic**: where it samples observations to create the various learning sets for the various trees and where it chooses the predictors to use.

For prediction just refer to bagging, it is the same.

Is the fact that not all predictors are uses in each tree a problem? Well, no. The model still considers all the predictors but chooses not to use some of them.

Warning: ***Random Forest***

The opposite, i.e. when the variable is in the tree but valued not in x, is a problem.

Experimentally turns out that n_{vars} does not impact the tendency to overfit, hence it is not a proper flexibility parameter. Good default values are:

- $\sqrt{p}$ for classification
- $\frac{1}{3}p$ for regression

For this reason (remember that also n_tree does not impact overfitting) Random Forest is almost a **hyperparameter-free learning technique!** ($\rightarrow$just use default values)

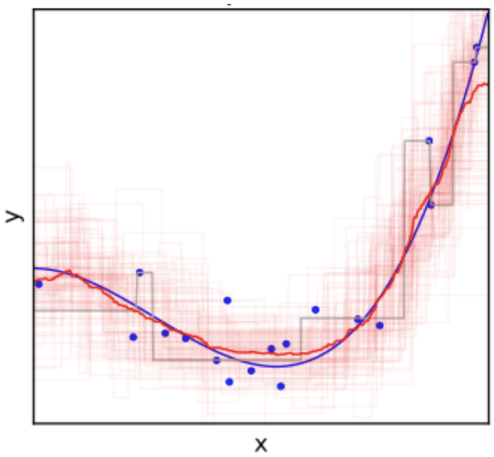


Figure 2.21: Random Forest

Definition: *OOB error*

This value is computed on the unseen observations at each step of the ensemble.

- For each observation, find the out-of-bag trees and obtain their prediction on the observations
- compute the error on the predictions

To remark that this is an **estimate** of the test error, but does not need a test dataset and is computed at learning time.

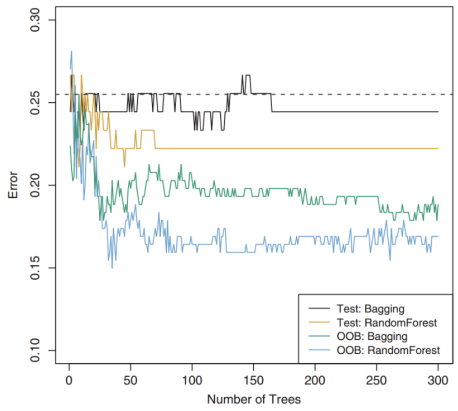


Figure 2.22: Error

Warning: *Interpretability*

Ensemble methods lack of Interpretability, since they consider different scenarios and use the computational power to find human-invisible patterns in the data.

Variable importance: compute teh RSS/Gini before and after each branch-node and assign the increase/decrease to the branch-node variable; this way a ranking of variables can be created (the larger, the more important).

But this process is not so effective, since tends to prefer numerical variables and works with learning data.

A better option is the **mean accuracy decrease**, just after learning.

1. for each j-th variable and each tree t in the bag
   (a) take the observations in D_t not used for t
   (b) measure the accuracy of t on D_t
   (c) shuffle the j-th variable in the observations, obtaining D'_t
   (d) measure again the accuracy
   (e) assigne the decrease in accuracy to the j-th variable
2. build a ranking of variables based on the sum of decreases (the larger, the more important)

Rationale: if the decrease is low, shuffling the variable has no meaning (variable is not so important).

A more general approach comes with **feature ablation**:

1. measure the effectiveness of $f'_{learn}, f'_{predict}$ on the dataset D
2. for each j-th variable x_j
   (a) build a D' by removing x_j from D
   (b) measure the effectiveness of $f'_{learn}, f'_{predict}$ on D'
   (c) compute the j-th variable importance as the decrease of effectiveness in D' wrt D
3. build a ranking of variables based on the decrease in effectiveness (as always, the larger, the more important)

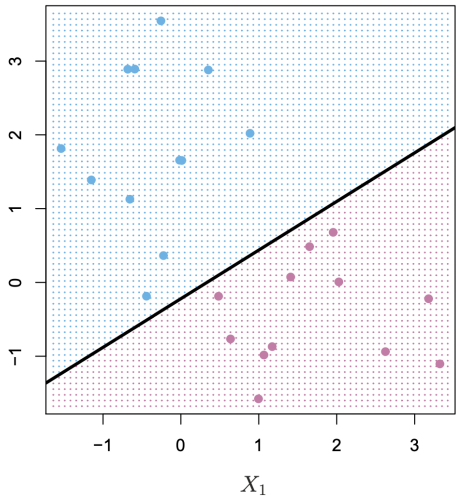
Here some variables are removed to see what happens.

Definition: *No free lunch theorem*

Any two optimization algorithms are equivalent when their performance is averaged across all possible problems

2.4 Support Vector Machines

The learning techniques seen before lack of usefulness in some types of datasets, like the below one:



Here, an **Hyperplane** can be used to tella apart the points with different labels. SVM are intended for the binary classification setting in which there are two classes.

Definition: *Hyperplane*

In a p-dimensional space, a hyperplane is a flat affine subspace of dimension $p-1$.

$$\beta_0 + \beta_x^T = 0$$

Since it divides the space in two parts, if the data is contained in one subspace it will be classified as $y_1 \in Y$, otherwise as $y_2 \in Y$.

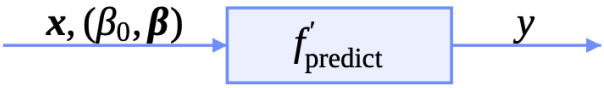


Figure 2.23: SVM $f_{predict}$

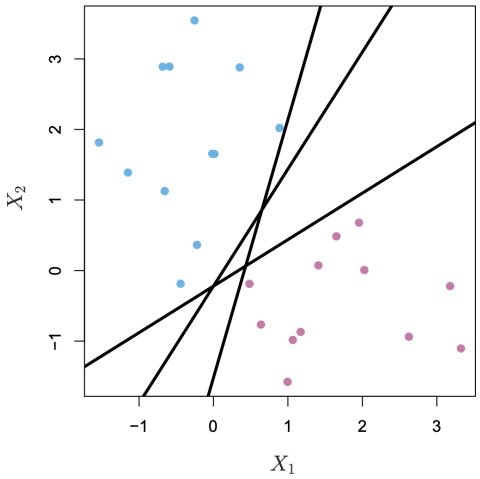
Assumptions:

- Y = Pos,Neg $\rightarrow$ binary classification only!
- X $= R^p \rightarrow$ numerical independent variables only!

It is computationally very fast, since there are just p multiplications and sums.

The `magnitude` of a vector of predictors is the distance from the hyperplane, meaning that the greater the magnitude, the more the point is far from the hyperplane and the more confident is the prediction.

But, how to choose the best hyperplane?



Definition: ***Maximal Margin Hyperplane***

The best hyperplane is the one that maximizes the distance from the closest point to the hyperplane. This distance is called **margin** and is the (perpendicular) distance from each training observation to a given separating hyperplane; the smallest such distance is the minimal distance from the observations to the hyperplane. The maximal margin hyperplane is the separating hyperplane for which the margin is largest,that is, it is the hyperplane that has the farthest minimum distance to the training observations.

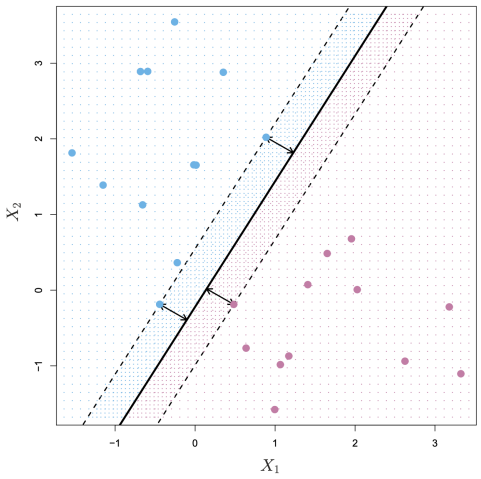


Figure 2.24: Maximal Margin Hyperplane

In fig.2.24 three training observations are equidistant from the maximal margin hyperplane and lie along the dashed lines indicating the width of the margin. These three observations are known as support vectors,since they are vectors in p-dimensional space and they "support" the maximal margin hyperplane in the sense that if these points were moved slightly then the maximal margin hyperplane would move as well.

How to learn an hyperplane:

- must perfectly separates the data
- must maximize the margin

It is an optimization problem, searching for the best m with the constraint that every point must be on the proper size and at a distance $\geq m$ from the hyperplane.

Warning: *Applicability*

What if some noise is introduced in the predictors? The **support vectors** change just a bit and the hyperplane changes as well, it can even not exist anymore! **Tolerance** must be introduced.

Definition: *Soft Margin Classifier*

- argmax$_{\beta_0 \ldots \beta_p, \varepsilon^{(1)} \ldots \varepsilon^{(n)}}$ m
- subject to $\sum_{j=1}^{j=p} \beta_j^2 = \beta\beta^T = 1$
- $y^{(i)}(\beta_0 + \beta^T x^{(i)}) \geq m(1 - \varepsilon^{(i)}) \forall i = 1 \ldots n$
- $\varepsilon^{(i)} \geq 0 \forall i = 1 \ldots n$
- $\sum_{i=1}^{n} \varepsilon^{(i)} = c$

$\varepsilon^{(1)} \ldots \varepsilon^{(i)}$ are positive slack variables: they act as tolerance wrt the margin.
Due to the tolerance (c), the margin can be pushed. $c \in R^+$ is a budget of tolerance.

c is a flexibility parameter:

- $c = \infty \rightarrow$infinite tolerance, you can move a lot the points and the line stay the same hence the model is the same irrespective of learning data (**high bias**)

- $c = 0 \rightarrow$no tolerance, the model is the best wrt the learning data but it is not generalizable (**high variance**)

Variable scale plays an important role here, since we are computing a matrix multiplication and the high coefficients' difference can be subject to noise. For this reason, the data must be **standardized**:

- **min-max scaling**: $x_j^{(i)} = \frac{x_j^{(i)} - \min_i x_j^{(i')}}{\max_i x_j^{(i')} - \min_i x_j^{(i')}}$
- **standardization**: $\frac{1}{\sigma_j}(x_j^{(i)} - \mu_j)$

Since scaling must be done both in learning and in prediction, the coefficients needed for scaling do **belong to the model**.

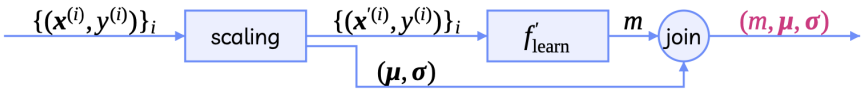


Figure 2.25: Learning with standardization

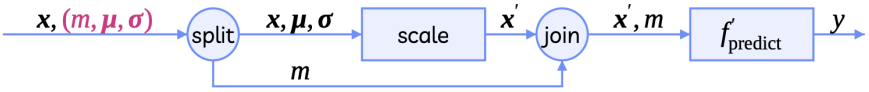


Figure 2.26: Prediction with standardization

Another formulation for the tolerance is the following:

- $\max_{\beta_0,\ldots,\beta_p,\varepsilon^{(1)},\ldots,\varepsilon^{(i)}} m - c\sum_{i=1}^{n} \varepsilon^{(i)}$
- subject to the previous constraints

Regarding c, it is a weighting factor that balances the trade-off between the margin and the tolerance. It is a flexibility parameter, and it is usually set by the user.

- $c = 0 \rightarrow$ points that are inside the margin cost zero, you can put the line wherever you want, the model is always the same (**high bias**)
- $c = \infty \rightarrow$ points that are inside the margin cost infinite, the model is the best wrt the learning data but it is not generalizable (**high variance**)

Warning: ***Differences between SMCs***

There is one learning parameter c, it is a flexibility parameter. **Differences**:

- c extreme values:
  - $c = +\infty$ (1st) and $c = 0$ (2nd) $\rightarrow$ high bias
  - $c = 0$ (1st) and $c = +\infty$ (2nd) $\rightarrow$ high variance
- learnability:
  - with the 2nd, a model can **always** be learned
  - with the 1st, there is a $c \leq 0$ such that a c value lower than that leads to an impossibility to learn a model from D

Definition: *Kernel*

We can formuate alternatilvely the $f'_{predict}$ in this way:

$$f'_{predict}(x) = \beta_0 + \sum_{i=1}^{n} \alpha^{(i)} K(x, x^{(i)})$$

where $K(x, x^{(i)})$ is the kernel function, that is a function that computes the similarity between two points.

The idea behind this function is to:

- transform the data in a higher-dimensional space ($X = R^p \rightarrow X = R^q$) and then
- compute the inner product in the destination space, i.e., $k(x, x^{(i)}) = \phi(x^T)\phi(x^{(i)})$

hoping that an hyperplane in the destination space is a non-linear hyperplane in the original space.

When using a kernel function, the learing technique is called `Support Vector Machines`.

Example: *Common Kernel Functions*

- **Linear Kernel:** $K(x, x^{(i)}) = x^T x^{(i)}$, the most efficient and computationally cheapest
- **Polynomial Kernel:** $K(x, x^{(i)}) = (x^T x^{(i)} + 1)^d$, where d is the degree of the polynomial
- **Radial Basis Function (RBF) Kernel:** $K(x, x^{(i)}) = exp(-\gamma ||x - x^{(i)}||^2)$, where γ is a flexibility parameter, this is the most widely used

Intuition of RBF: it maps an x to the space where coordinates are the distances to relevant observations of the learning data. In practice, the decision boundary can smoothly follor any path (with risk of overfitting).

X_j	Y	RF	SVM
Numerical	Binary classification	✅	✅
Categorical	Binary classification	✅	❌
Numerical + Categorical	Binary classification	✅	❌
Numerical	Multiclass classification	✅	❌
Categorical	Multiclass classification	✅	❌
Numerical + Categorical	Multiclass classification	✅	❌
Numerical	Regression	✅	❌
Categorical	Regression	✅	❌
Numerical + Categorical	Regression	✅	❌

Figure 2.27: Applicability?

How can improve the SVM applicability to other types of data?

Definition: *Encoding*

The idea is to encode the data in a way that the SVM can learn a model from it. This is done by:

- **One-Hot Encoding:** for categorical variables, each category is transformed in a binary variable, increasing the number of predictors
- **Ordinal Encoding:** for ordinal variables, each category is transformed in a numerical variable

One-vs-One

Technique to extend the SVM to multiclass classification. The idea is to learn a binary classifier for each pair of classes, and then to use a voting system to classify the new data. $\rightarrow M^{\frac{k(k-1)}{2}}$

One-vs-All

Technique to extend the SVM to multiclass classification. The idea is to learn a binary classifier for each class, and then to use a voting system to classify the new data. $\rightarrow M^k$

Warning: *Missing Values*

- drop the relative observations
- drop the relative predictors
- impute the missing values

2.5 Naive Bayes

This learning techniques utilizes the **Bayes' Theorem** to classify the data. It is a probabilistic approach, and it is based on the assumption that the predictors are independent. Here there are the concepts of **prior probability** and **posterior probability**.

$$Pr(A|B) = \frac{Pr(A)Pr(B|A)}{Pr(B)}$$

where:

- $Pr(A|B)$ is the posterior probability
- $Pr(A)$ is the prior probability
- $Pr(B|A)$ is the likelihood
- $Pr(B)$ is the evidence

Assumptions:

- $\mathrm{X} = X_1 \times \cdots \times X_p$
- $\mathrm{Y} = \{y_1, \ldots, y_k\}$

Definition: *Naive Bayes*

- **Learning:** the model is a set of conditional probabilities, one for each predictor and each category
- **Predict:** for each category, compute the product of the conditional probabilities and the prior probability, then choose the category with the highest value

Hence, $p(y_m|x_{1,l_1}, \ldots, x_{p,l_p}) = p(y_m)\frac{p(x_{1,l_1}, \ldots, x_{p,l_p}|y_m)}{p(x_{1,l_1}, \ldots, x_{p,l_p})}$

In the **Naive** approach, where the predictors are independent, it becomes:

$$p(y_m|x_{1,l_1}, \ldots, x_{p,l_p}) = \frac{p(y_m)}{p(x_{1,l_1}, \ldots, x_{p,l_p})} p(x1, l_1|y_m) \ldots p(x_{p,l_p}|y_m)$$

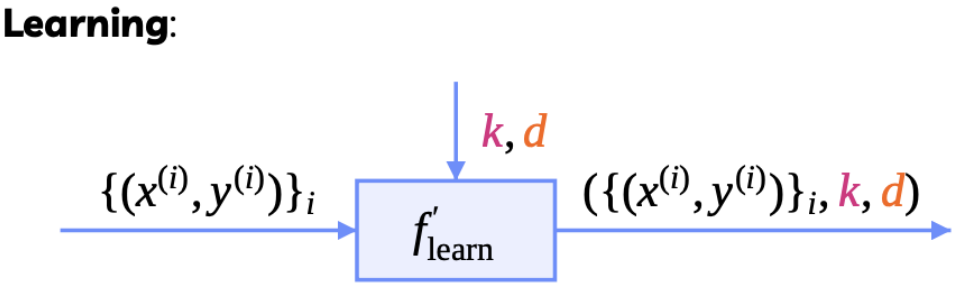
that can be found in the dataset.

Figure 2.28: Naive Bayes

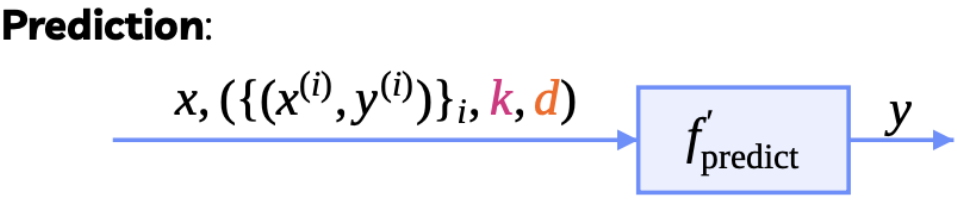
The model p is some data structure holding $k + \sum_{j=1}^{p} kh_j$ numbers.

2.6 k-Nearest Neighbors (KNN)

This learning technique is based on the idea that similar data points are close to each other in the space. The model is the dataset itself, and the learning technique is just a function that computes the distance between the new data and the dataset.



f'_{learn} does nothing, $f'_{predict}$ computes the distance between the new data and the dataset, then selects the k-nearest neighbors and returns the majority class. k and d are parameters!



For regression, return $\frac{1}{k} \sum_{i \in I} y^{(i)}$

With the proper distance function, the KNN can be used for any type of data.

Example: *Common distance functions*

- **Euclidean Distance:** $d(x, x^{(i)}) = \sqrt{\sum_{j=1}^{p} (x_j - x_j^{(i)})^2}$
- **Manhattan Distance:** $d(x, x^{(i)}) = \sum_{j=1}^{p} |x_j - x_j^{(i)}|$
- **Minkowski Distance:** $d(x, x^{(i)}) = (\sum_{j=1}^{p} |x_j - x_j^{(i)}|^q)^{\frac{1}{q}}$

k is a flexibility parameter, and it is usually set by the user. It is a trade-off between bias and variance:

- $k = 1 \rightarrow$high variance, low bias
- $k = n \rightarrow$high bias, low variance

3 Unsupervised Learning

Definition: *Unsupervised Learning*

Unsupervised Learning is a type of machine learning that looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision.

3.1 Clustering

We assume that the system that generated the data followed some scheme (**pattern**), we do not know it and we want to discover it from a dataset.

- the example is the dataset $P^*(X)$
- what we learn from the dataset is not, in general, usable in another dataset $\rightarrow$ **find** patterns is fairer than **learn** patterns

We first look for a **grouped** data, but we do not know the groups. This is the **clustering** problem.

Definition: *Clustering*

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups.

Given a dataset $D \in P^*(X)$, find a partitioning $\{D_1, \ldots, D_k\}$ such that each D_i is a cluster.

- close toghether?
- how close?
- where does k come from?

This is an optimization problem, where for any k,d, there exists at least one optimal solution. In principle one can try all the partitions and measure the distance, but in practice k is unknown and trying all the partitions is unfeasible.

$$\text{argmax}_{D_1,\ldots,D_k}(\sum\sum d(x,x')) - (\sum\sum d(x,x'))$$
$$\text{subject to } \bigcup D_i = D \text{ and } D_i \cap D_j = \emptyset$$

- **maximize** the distance between any two x, x' when they belong to different clusters
- minimize the distance between any two x, x' when they belong to the same cluster
- clusters ave to form a partition

Can't be also optimized k? No, it is pointless.

- $k = 1 \rightarrow$no clustering
- $k = n \rightarrow$each point is a cluster

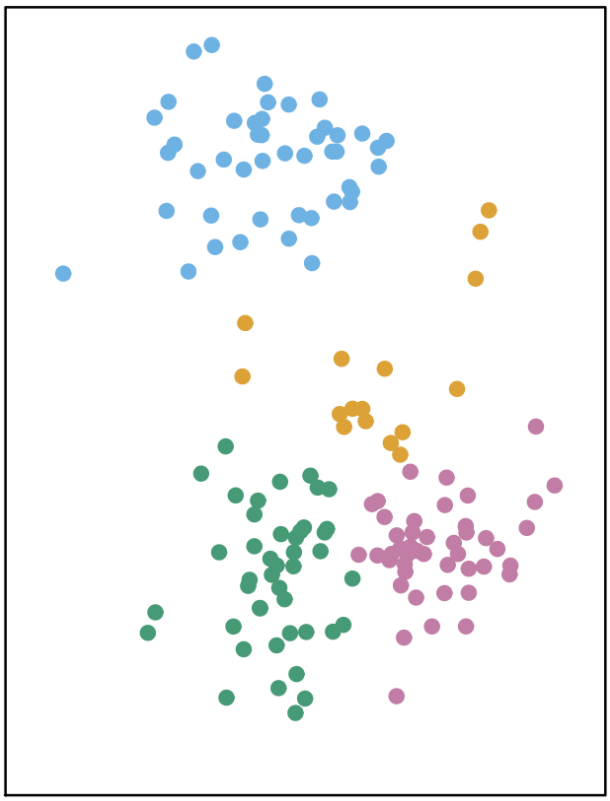


Figure 3.1: Clustering

How to access clustering in practice?

- inspect D manually
- insert the clustering inside the larger information processyng system and measure some other indexes
- measure some performance indexes

Example: ***Performance Index: Silhouette Index***

It considers, form each observation, the average distance to the observations in the same cluster and the min distance to the observations in other clusters.

- $a(i) = \frac{1}{|D_i|} \sum_{x \in D_i} d(x, D_i)$
- $b(i) = \min_{j \neq i} \frac{1}{|D_j|} \sum_{x \in D_j} d(x, D_i)$
- $s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$
- $S = \frac{1}{n} \sum s(i)$

The index is between -1 and 1, where 1 is the best clustering. The larger the index, the better the clustering.

3.1.1 Hierarchical Clustering

Definition: *Hierarchical Clustering*

It is an iterative method in which:

- at each j-th iteration, there exist a partition $D_1, \ldots, D_{k_j}$
- at most two clusters differ between partitions at subsequent iterations
- you do not set k

It is of two types:

- **Agglomerative**: start with each point as a cluster and merge the closest clusters
- **Divisive**: start with all the points in a cluster and split the farthest points

Cluster distance can be computed in different ways:

- **Single Linkage**: the distance between two clusters is the distance between the closest points in the two clusters
- **Complete Linkage**: the distance between two clusters is the distance between the farthest points in the two clusters
- **Average Linkage**: the distance between two clusters is the average distance between all the points in the two clusters
- **Centroid Linkage**: the distance between two clusters is the distance between the centroids of the two clusters

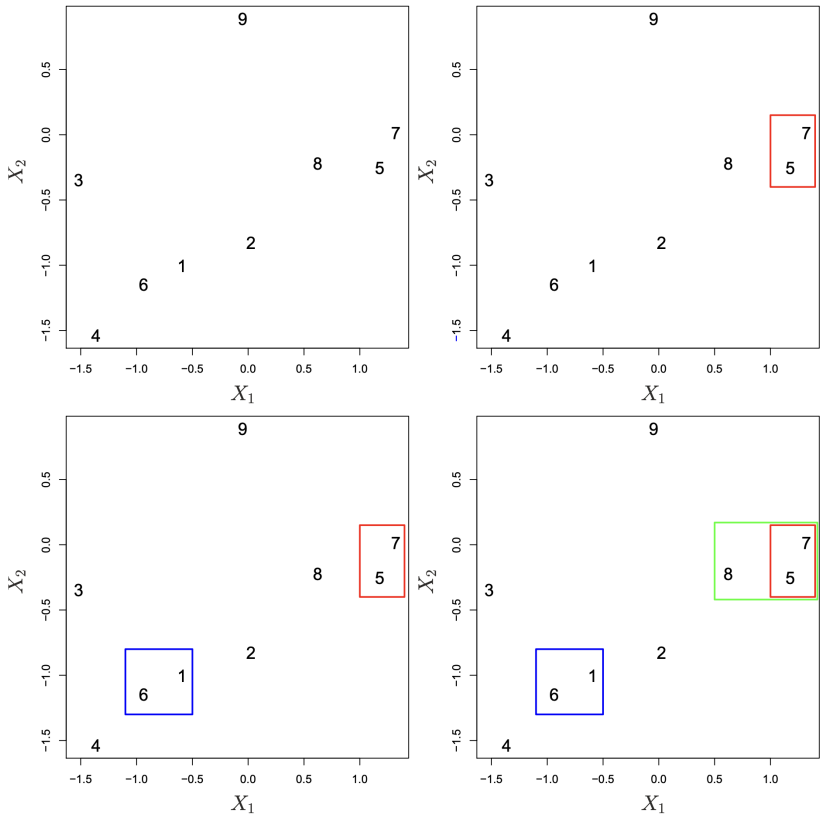


Figure 3.2: Hierarchical Clustering in R

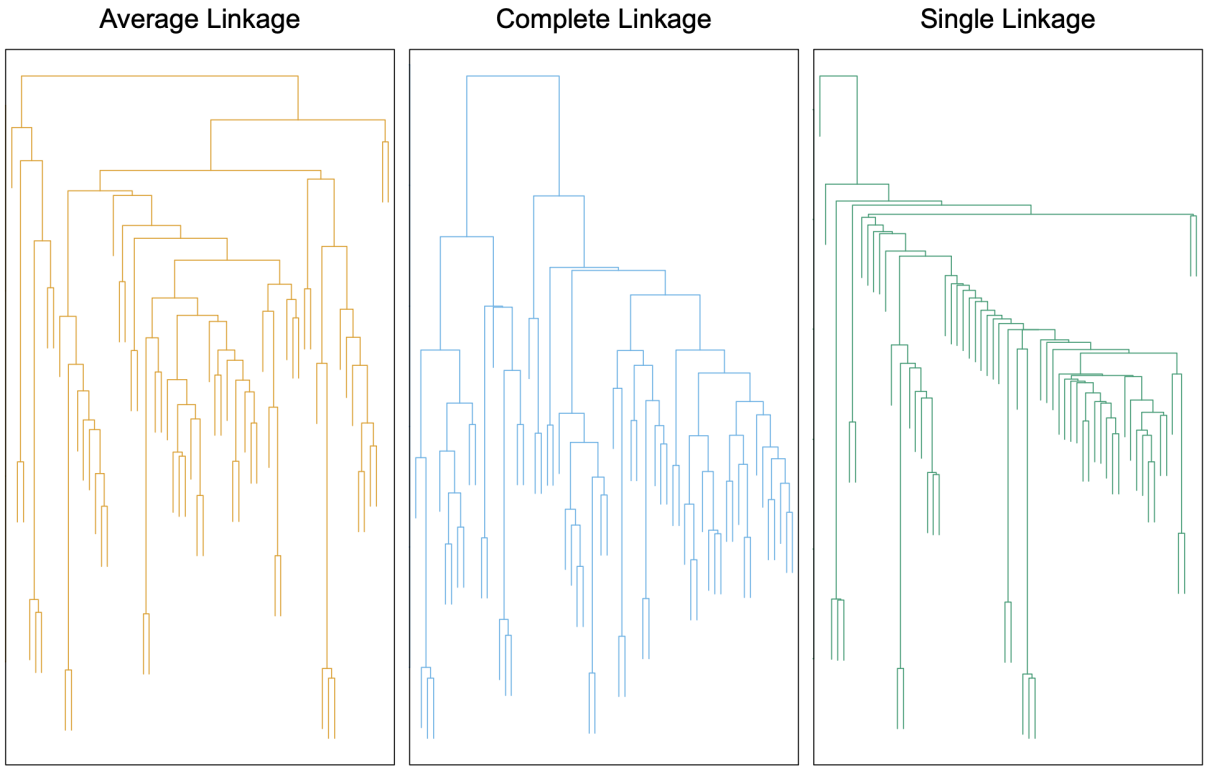


Figure 3.3: Distances in Hierarchical Clustering

3.1.2 k-means

Definition: ***k-means***

K-means clustering is a simple and elegant approach for partitioning a data set into K distinct, non-overlapping clusters. To perform K-means clustering, we must first specify the desired number of clusters K; then the K-means algorithm will assign each observation to exactly one of the K clusters. It is an iterative method in which:

- you set k
- you set the initial centroids
- you assign each point to the closest centroid
- you update the centroids
- you repeat until convergence

A **centroid** is the average of all the points in a cluster.

This technique is not deterministic, due to the random initialization of the centroids. It is also sensitive to the initial centroids, so it is better to run it multiple times and choose the best result.

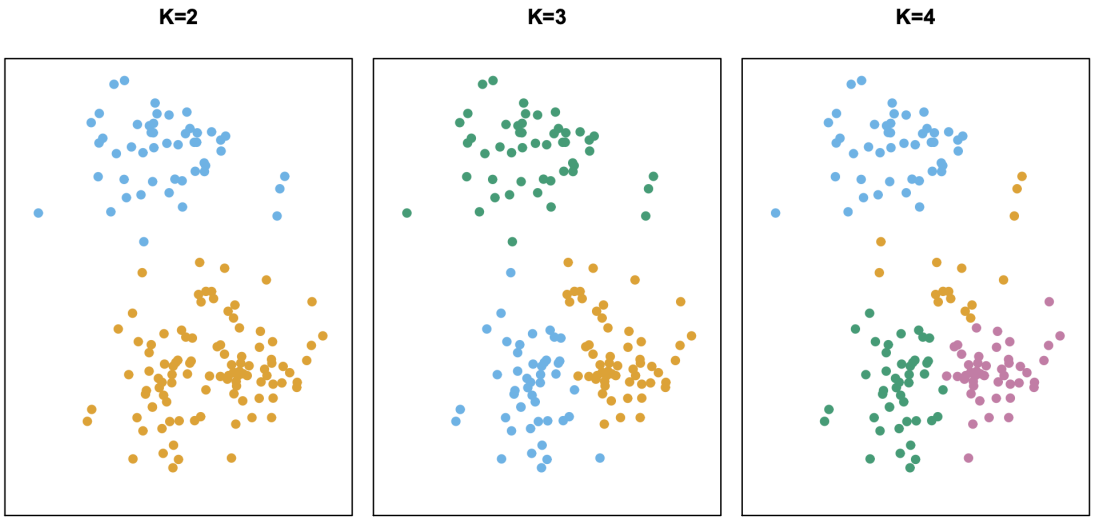


Figure 3.4: k-means

4 Introduction to Natural Language Processing

4.1 Applying Machine Learning to Text

Definition: *Text*

A **text** is a sequence of symbols, which can be characters, words, or sentences. It belongs to an alphabet A:

$$x \in A^*$$

where A^* is the set of all possible strings of symbols from A, in modern times UTF-16 (including emojis).

A dataset X of texts is called a **corpus**. A single text x_i is called a **document**.

4.1.1 Sentiment Analysis

The goal is to gain insights about the sentiments an author was feeling while authoring a document. This is a supervised learning problem, where the input is a document and the output is a sentiment label.

Preprocessing is essential, to obtain multivariate observations from the text.

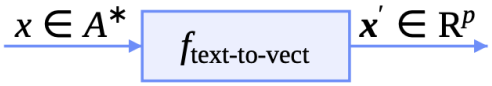


Figure 4.1: Sentiment Analysis

A **Bag of Words (BOW)** is a representation of text that describes the occurrence of words within a document. It involves two steps:

1. Tokenization: splitting the text into words
2. Counting: counting the occurrences of each word

Common preprocessing steps here are:

- Lowercasing
- Removing punctuation
- Removing stop words
- Stemming (reducing words to their root form)

The problem of BOW is that it tends to prefer words that are frequent in the corpus, but not necessarily informative. For this reason we can use **TF-IDF** (Term Frequency-Inverse Document Frequency), which is a measure of how important a word is to a document in a collection or corpus.

It is obtained by multiplying the term frequency (TF) by the inverse document frequency (IDF):

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,D)$$

Where:

- $tf(t,d)$ is the frequency of term t in document d
- $idf(t,D)$ is the logarithm of the ratio of the total number of documents in the corpus and the number of documents where the term t appears
- D is the corpus
- t is the term
- d is the document
- $tfidf(t,d,D)$ is the TF-IDF score of term t in document d

Since with BOW, $p = |W|$, dimensional reduction is often necessary. Common practices are:

- use a very small dictionary
- learn a small dictionary on the learning data
- use tf-idf and get k most important words
- use a dimensionality reduction technique

Also **ordering** is very important:

- ngrams $\rightarrow$ Instead of considering word frequencies (or occurrences, or tf-idf), consider the frequencies of short sequences of up-to words
- part of speech tagging $\rightarrow$ Instead of considering words, consider the frequencies of parts of speech