Introduction to supervised learning

Nistor Grozavu

LIPN - CNRS UMR 7030

2020/2021

What is supervised Learning ?

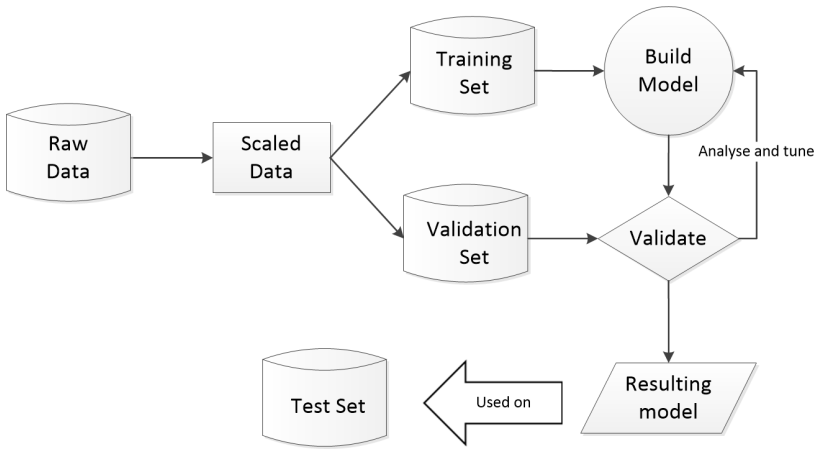
Supervised learning is a Machine Learning Task the aim of which is to learn a model from **labeled training data** in order to classify similar unlabeled data. Potential applications of supervised learning include:

- Regression (Cf. previous Lecture)
- Prediction
- Classification
  - Speech and hand-writing processing
  - Pattern Recognition
- Rule Mining

Supervised Learning workflow

In supervised learning, we distinguish 2 types of data:

- The training data, subdivided into 2 subsets:
  - The **Training set**, on which the label are known and that will be used to build the model.
  - The **Validation set**, on which the labels are also known and that can be used to rate the model and improve it if necessary. (noy used by all algorithms)
- The test data, or **Test set**, on which the model will be used to find unknown labels.



Supervised Learning data sets examples

Supervised data are described by several attributes and a target class (or label), which is known in the training and validation set, but unknown in the test set.

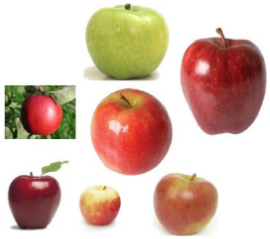
Size	Weight	Shoe size	Sex
176	72	43	M
159	61	37	F
180	66	39	F
185	85	44	M
177	70	41	F
155	88	38	M
210	110	45	M

Table: Training data

Size	Weight	Shoe size	Sex
205	85	47	?
172	60	40	?
164	57	38	?
169	52	36	?
183	78	42	?
175	65	44	?
191	77	41	?

Table: Test data

Supervised Learning Example



Labeled images of apples

Labeled images of peaches

Mixture of fruits images to classify

The notion classifier

Besides for regression applications, most supervised algorithms are known as **classifiers**.

Classifiers

- A classifier learns a model in the form of a function, a set of logic rules, the parameters of a probabilistic model, the parameters of a neural network, a set of prototypes, etc.
- The classifier will use the model it learned to label new and previously unknown data.

- When the target attribute is an integer number, we usually refer to it as a **class**.
- When the target attribute is categorical, we talk about **labels**.
- When the target attribute is a real number, the process involved is a regression.

Formalism

- Let us denote $\boldsymbol{X} = \{x_1, \cdots, x_N\}, x_i \in \mathcal{X}$ be the matrix the N observed examples of the training set.
  - The x_i are vectors containing the attributes of each object.
- Let $Y = \{y_1, \cdots, y_N\}, y_i \in [1..K]$ be the vector containing the labels/classes associated to the observed examples.
- We note $\mathcal{L} = \{(x_i, y_i), i \in [1..N]\}$ the training set.
- A classifier induced from the traning set $\mathcal{L}$ will be denoted $\psi(\cdot, \mathcal{L})$. It is a function that for any vector x_i from $\mathcal{X}$ associates a class:

$$\psi(\cdot, \mathcal{L}) : \mathcal{X} \rightarrow [1..K]$$

- Applying ψ on a new object x from the test set will therefore return a class prediction:

$$\hat{y} = \psi(x, \mathcal{L})$$

Types of models and classifiers

How are patterns and models expressed ? There are 2 extremes:

- **Black box representation**: The model, structure or function is impossible to grasp for a human unfamiliar with the generating algorithm.
  - Examples: Deep learning algorithms
- **White box representation**: The model and its construction process are easy to understand and reveal which kind of structure to expect.
  - Examples: Decision trees, K-Nearest neighbors

The different types of models come from various fields such as AI, statistics and research in databases.

The 5 steps of supervised learning

1. Decide on a training set that will be representative of the real-world use of your classifier.
2. Determine your input features and the representations that you want to use in your model.
3. Decide on the structure of your learning function, and choose a supervised algorithm compatible with this model.
4. Run the algorithm on your training set. If your algorithm allows it, do cross-validation checking and adjust your model.
5. Evaluate the accuracy of your algorithm and apply it on a separate test set.

The bias-variance trade-off

Supervised learning faces several challenges.

The bias-variance trade-off in supervised learning

The bias-variance trade-off is a problem in which a supervised algorithm has to achieve simultaneously two seemlingly incompatible goals:

- Building a model that gives good results on the validation set.
- Building a model that can generalize beyond de training set.

- The **bias** is the error from erroneous assumptions in the learning algorithm or model that usually results in the algorithms missing important links between the input variables and the output, thus leading to **underfitting**.
- The **variance** is the error caused by a too high sensitivity of the model toward small variations in the training set. This results in **overfitting** and the model being unable to generalize to data outside of the training set.

Function complexity and amount of training data

The second issue is the available amount of training data when compared to the relative complexity of the real model to be learned:

- If the model is simple, a learning algorithm with a high bias and a low variance should be able to learn it from a small amount of data.
- If the model is complex, it will only be learnable from a very large amount of training data and using a learning algorithm with a low bias and a high variance.

Remark

Good learning algorithms should be able to adjust their bias-variance trade-off based on the amount of available data and the apparent complexity of the model to be learned.

Picking the right input variables

The problem with too many input variables

Even if the real learning model depends only on a very small number of variables, the algorithm may never figure it out if it is flooded with a very high number of input variables.

- The result may end up being a very complex and overfitting model.
- Models with too many variables cannot easily be understood and interpreted.

- A good understanding of your data and of the problem that you want to modelize will help remove irrelevant features.
- Scaling your data may have a huge influence on the results.
- It is important to check for correlation between the attributes and to remove redundant variables.

Similarity and distance

- Very much like in clustering, the distance function is a key element for most supervised classifiers.
- Creating custom distance functions is sometimes required.

Euclidian distance	$\|\|a-b\|\|_2 = \sqrt{\sum_i (a_i - b_i)^2}$
Squared Euclidian distance	$\|\|a-b\|\|_2^2 = \sum_i (a_i - b_i)^2$
Manhattan distance	$\|\|a-b\|\|_1 = \sum_i \|a_i - b_i\|$
Maximum distance	$\|\|a-b\|\|_\infty = \max_i \|a_i - b_i\|$
Mahalanobis distance	$\sqrt{(a-b)^\top S^{-1}(a-b)}$ where S is the covariance matrix
Hamming distance	$Hamming(a,b) = \sum_i (1 - \delta_{a_i, b_i})$

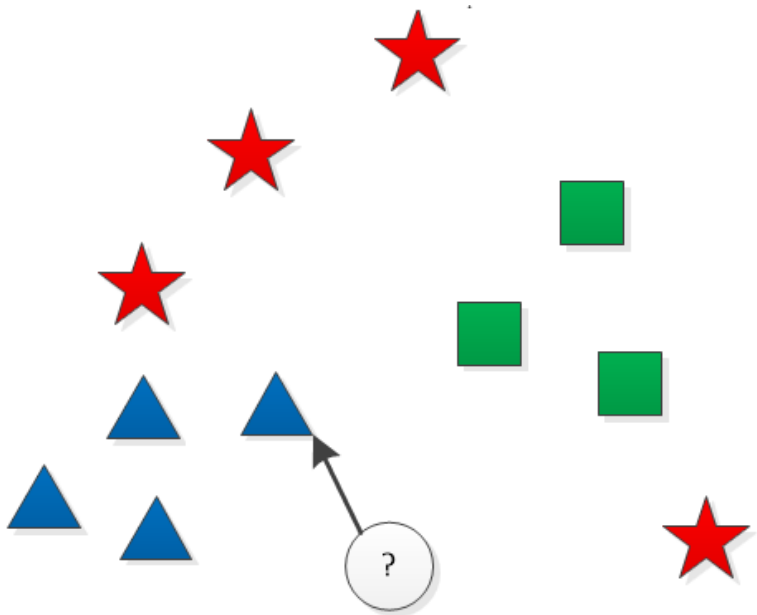
Table: Examples of common distances

1-Nearest Neighbor

The simplest and laziest classifier consists in using the training set itself as a model without building or computing anything.

1-NN Classifier

- "Learning" process: Remember all the observed examples.
- Classification process: When a new data arrives, find the most similar registered example (distance-wise) and assign it to the same class.

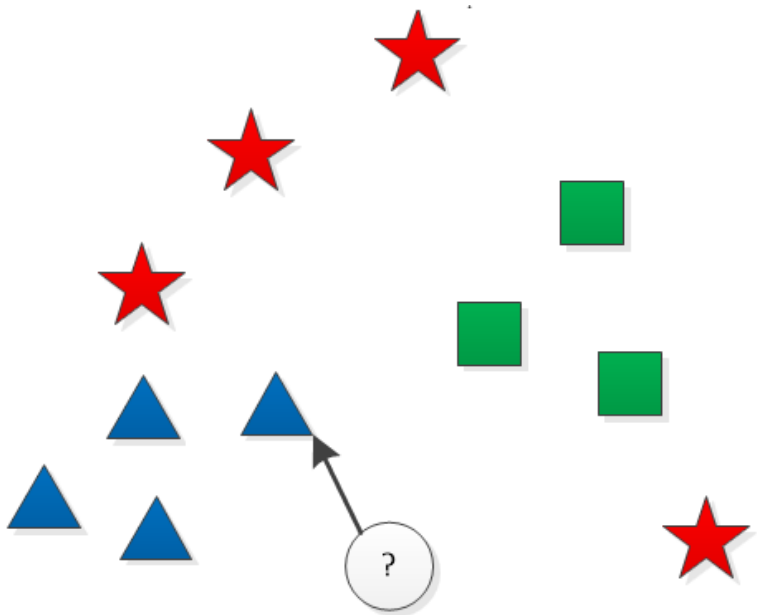


1-Nearest Neighbor

The simplest and laziest classifier consists in using the training set itself as a model without building or computing anything.

1-NN Classifier

- "Learning" process: Remember all the observed examples.
- Classification process: When a new data arrives, find the most similar registered example (distance-wise) and assign it to the same class.



1-Nearest Neighbor

The 1-Nearest Neighbor classifier is sensitive to noise and prone to overfitting.

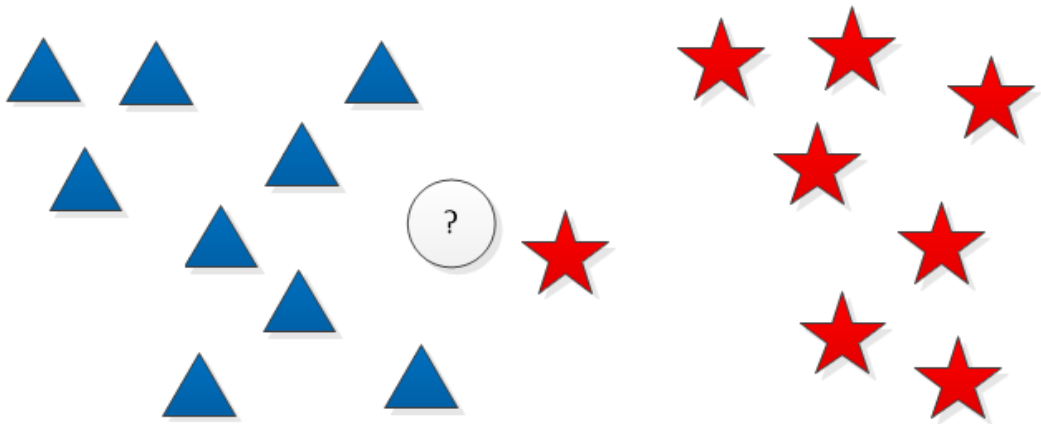


Figure: The 1-NN algorithm would assign this data to the red class. On the other hand, a majority vote would assign it to the blue class.

K-Nearest Neighbors

The K-Nearest Neighbors algorithm (KNN) considers the K closest observed data from the training set to decide on a class for an unlabeled data. K is a parameter chosen by the user.

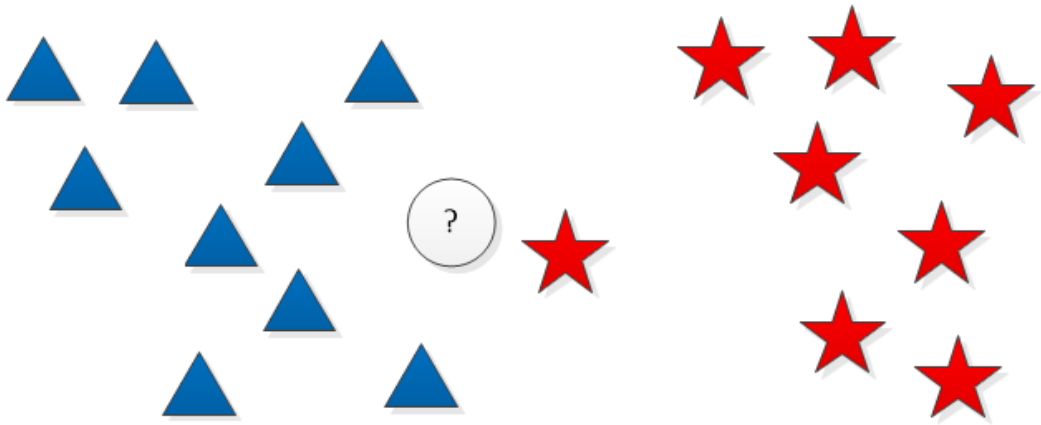
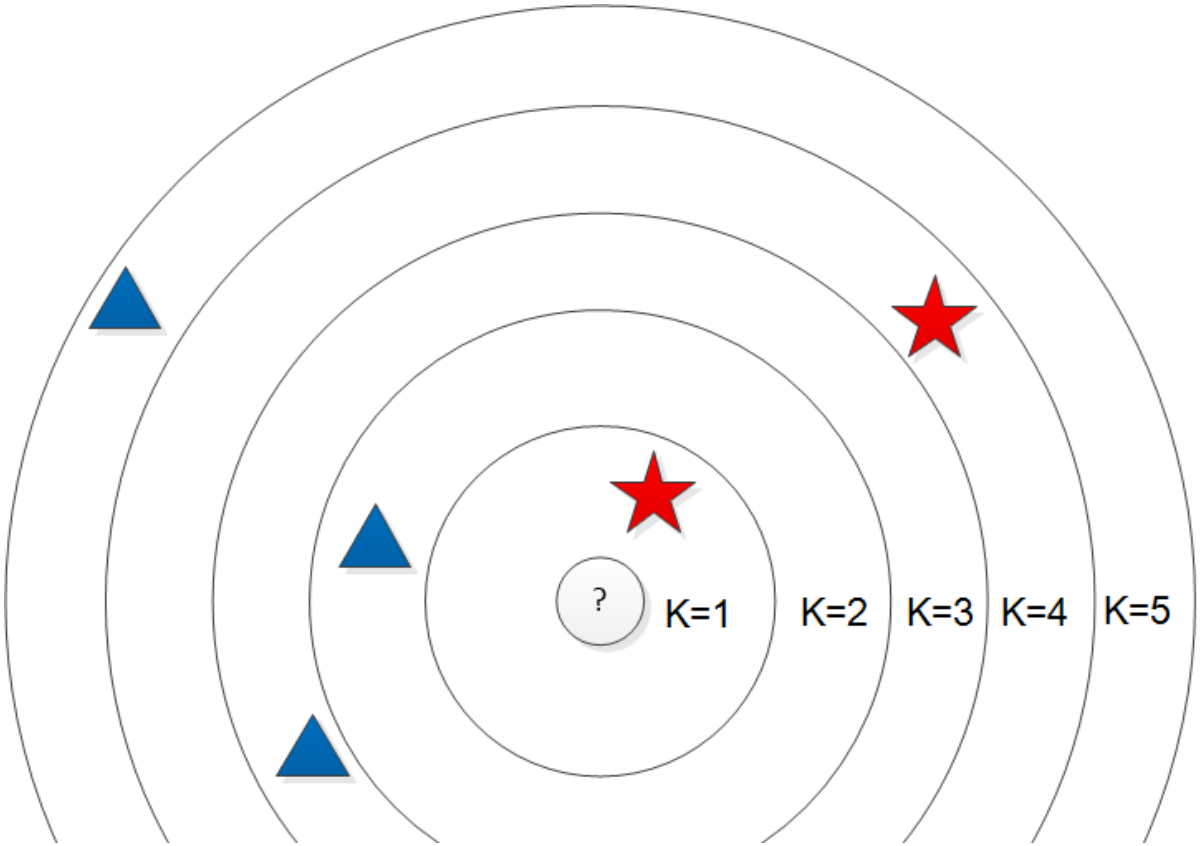


Figure: For K>1, the KNN algorithm would assign this unlabeled data to the blue class.

K-Nearest Neighbors

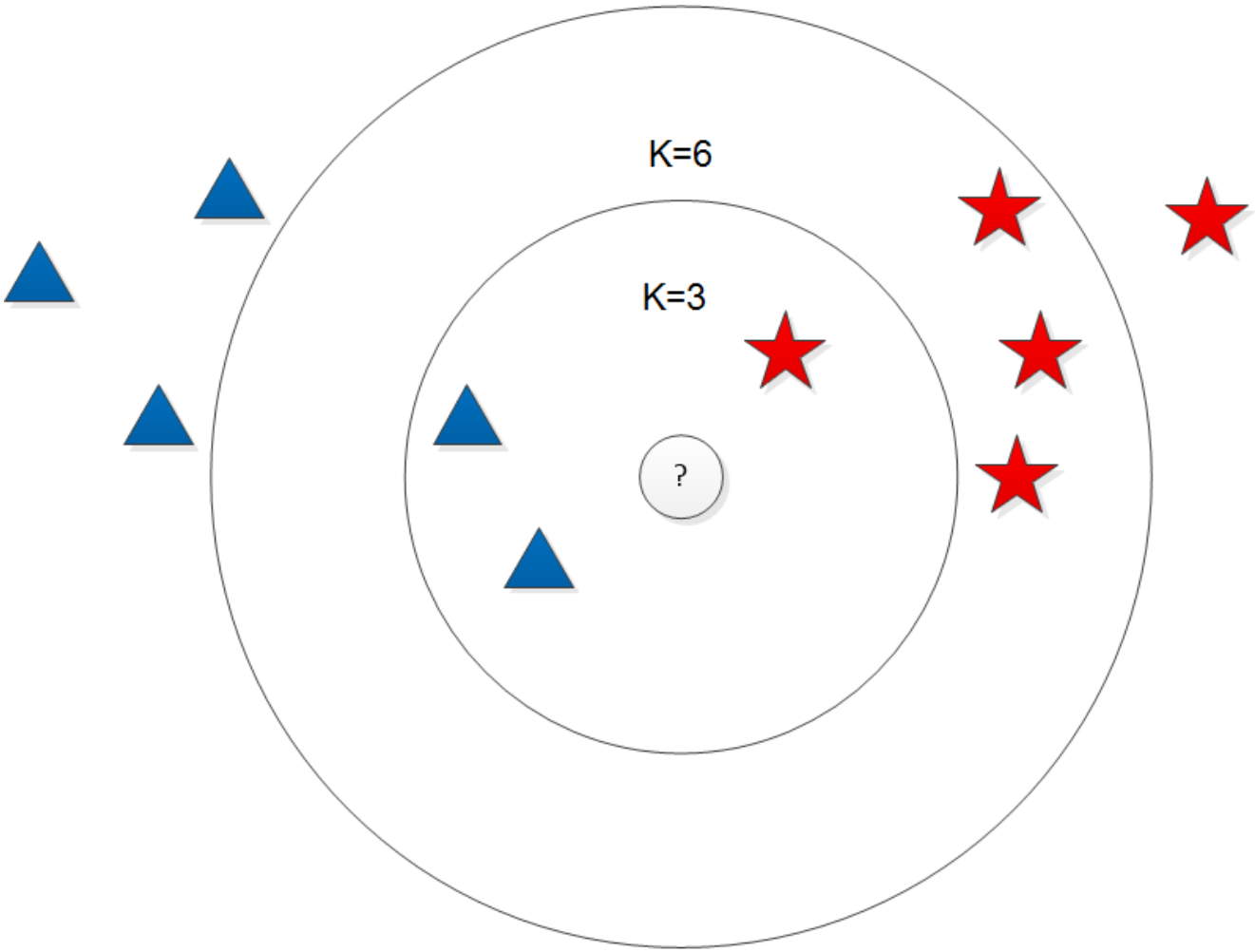
When there are only two classes, K is usually an odd number to avoid two classes having an equal number of votes.



K-Nearest Neighbors: Weaknesses

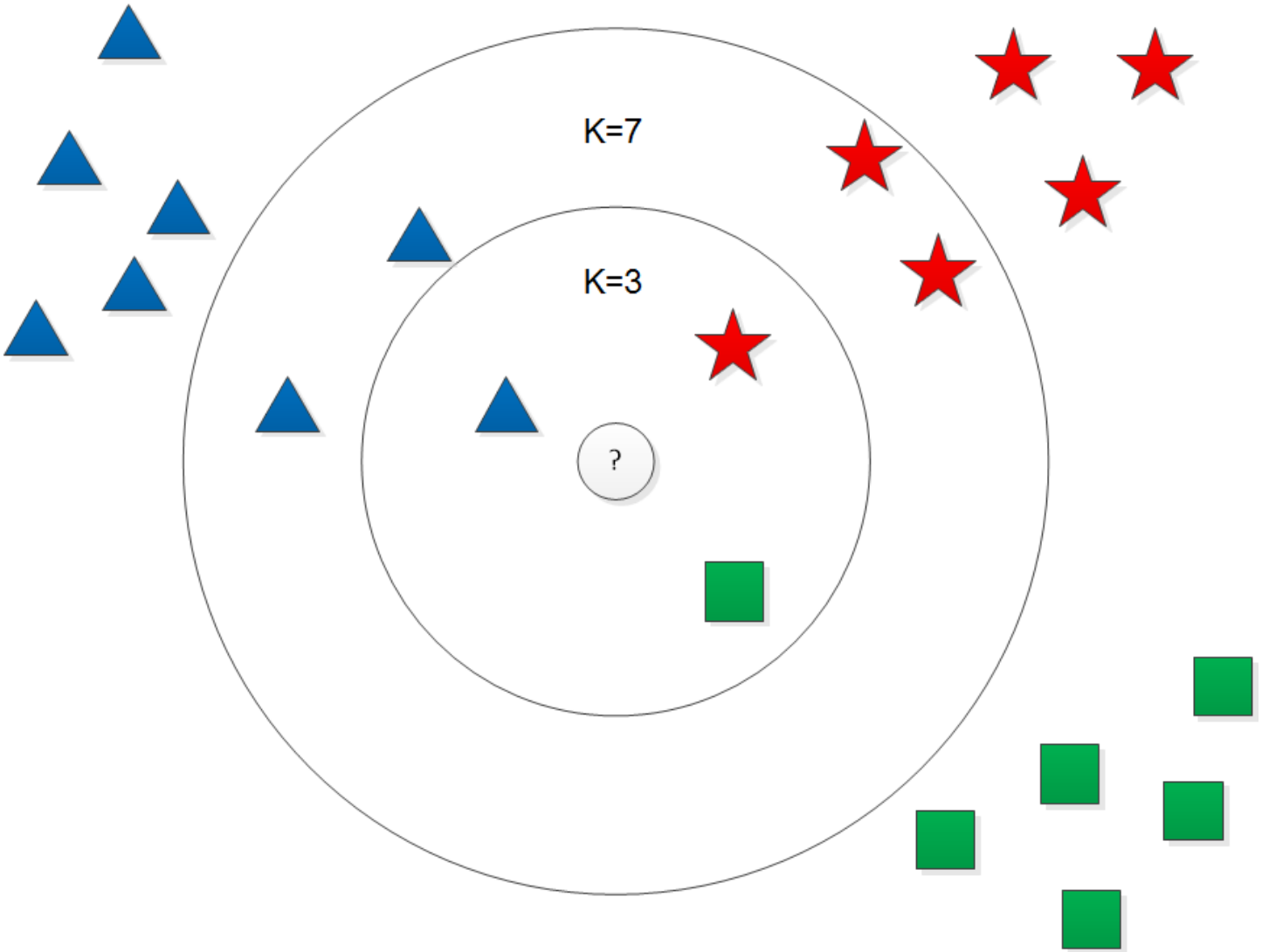
K is a critical parameter that can render the algorithm quickly unstable:

- Depending on the K, the class changes completely.



K-Nearest Neighbors: Weaknesses

With more than 2 classes, things can quickly become complicated ...



K-Nearest Neighbors: Weaknesses

- Because the distance between instances is based on all the attributes, less relevant attributes and even the irrelevant ones are used in the classification of a new instance.
- Because the algorithm delays all processing until a new classification/prediction is required, significant processing is needed to make the prediction.

Weighted Nearest Neighbors

The **Weighted Nearest Neighbors** solves 2 of the previous problems by adding a weight w_k to each neighbor.

Examples:

$$w_k = \begin{cases} \frac{1}{k} & \text{if } k \leq K \\ 0 & \text{if } k > K \end{cases}$$

$$w_k = \begin{cases} \frac{1}{dist} & \text{if } k \leq K \\ 0 & \text{if } k > K \end{cases}$$

Remark

The real Weighted Nearest Neighbors classifier uses a much more complex weight system that satisfies $\sum_{n=1}^{N} w_{ni} = 1$.

K-Nearest Neighbors: Summary

Pros

- Very simple and intuitive
- Low Complexity
- Great results with well-behaved classes

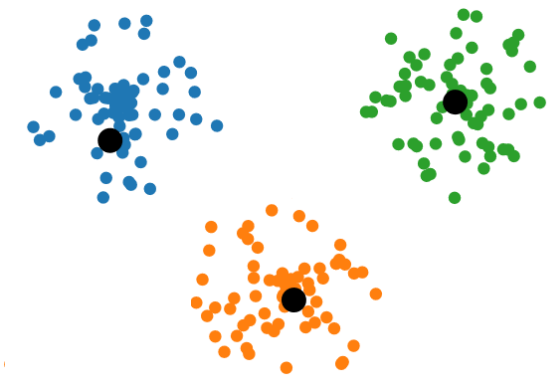
Cons

- No model: No way to properly describe each class. No possibility to re-use the knowledge
- Does not scale well because it requires to store all the training set
- Critical choice of the parameter K
- Ill-adapted for categorical data

Learning without remembering all the data

The main issues of the KNN algorithm is that all data have to be kept in memory and that it is unstable when classes that are not well separated:

- It's a problem with both large and complex datasets.



Idea

Instead of using all the data, we could use a prototype representing each class (like in the mean-shift and K-Means algorithm).

- Can be learned incrementally.
- Helps building a model.

Issues

- Works only with spherical classes
- Doesn't work with classes that aren't well separated

Learning without remembering all the data

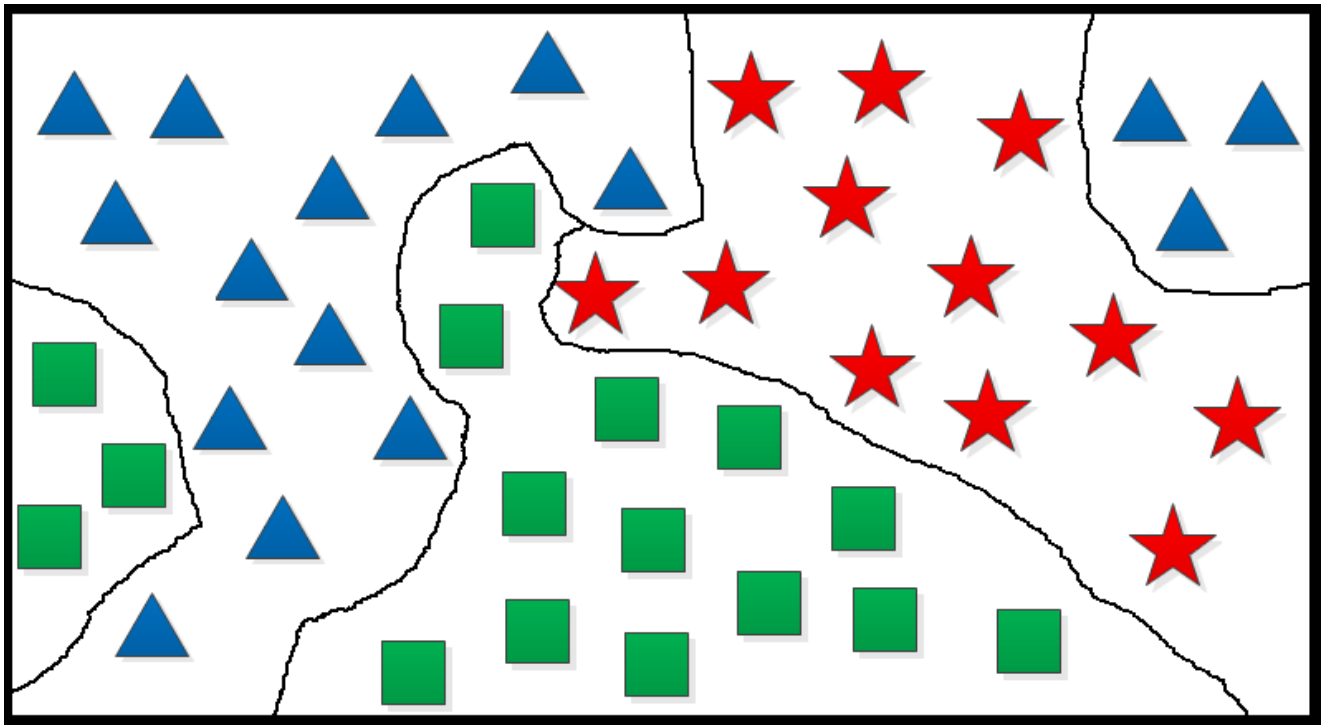


Figure: A single prototype per class will never work here ...

Learning without remembering all the data

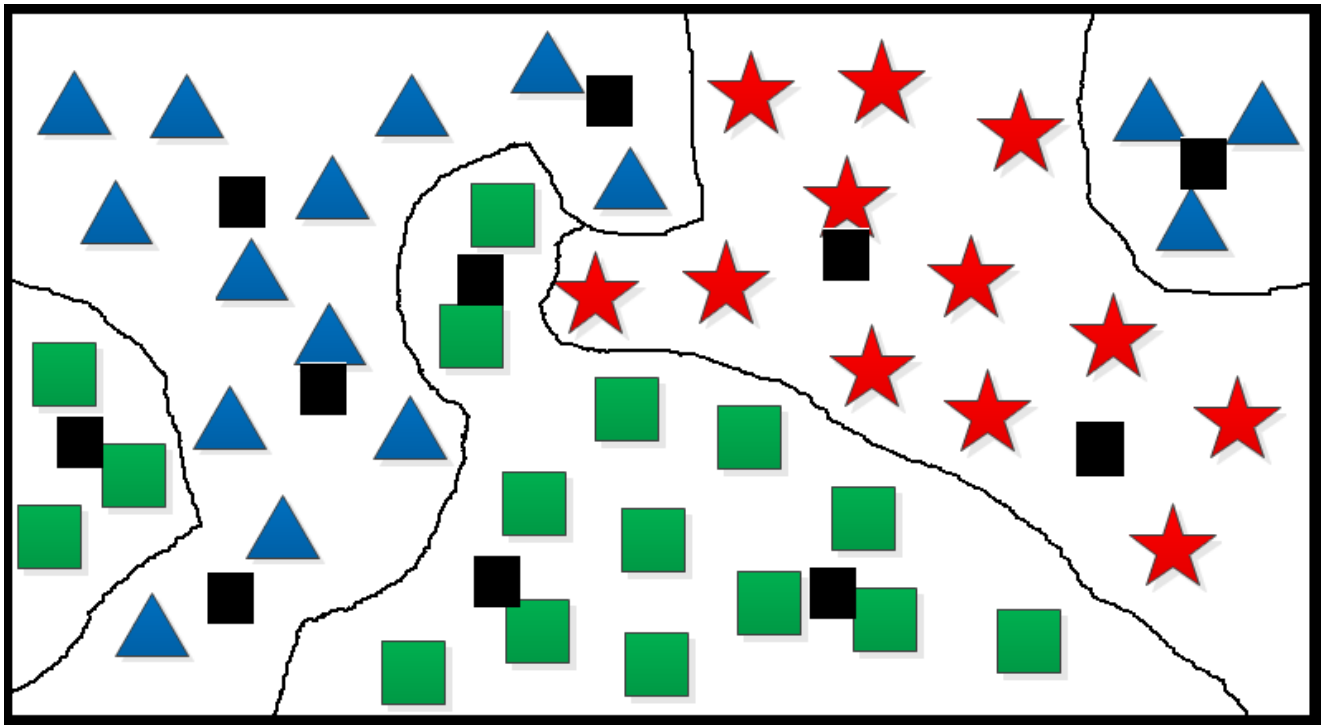


Figure: However, several prototypes per class could work !

Learning Vector Quantization algorithm

The LVQ algorithm (Kohonen) is a primitive neural network classifier that represents the classes from the training set using several prototypes per class.

- It is closely related to both the K-Means and the KNN algorithm.
- It is an early ancestor to the Self-Organizing Maps (Cf. Lecture 7)

Remark

In many neural networks algorithms, prototypes learned from an iterative process are called neurons due to their evolutive behavior and the fact that thay do not represent a cluster or class on their own.

Learning Vector Quantization algorithm

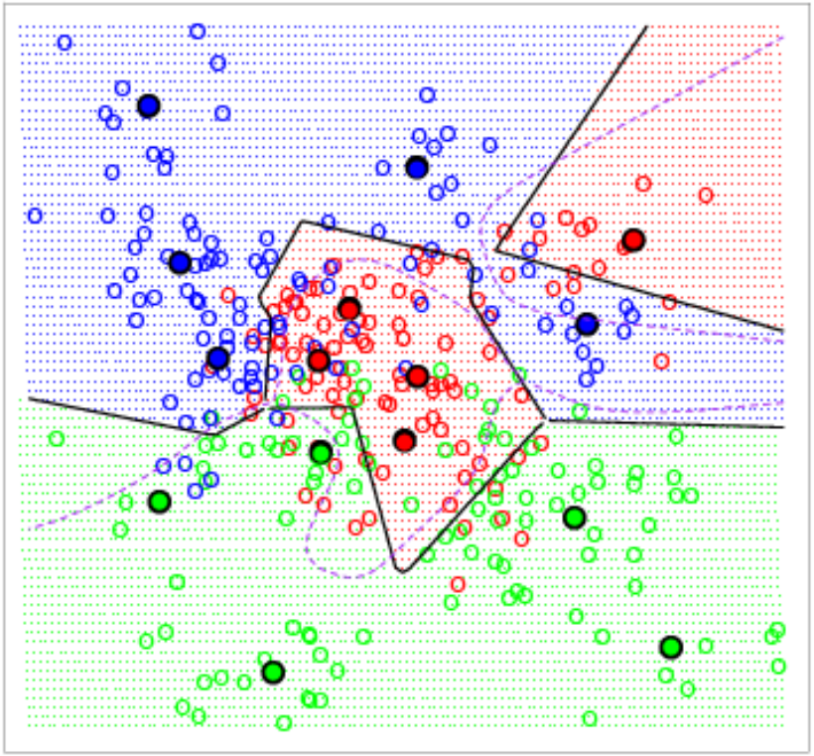


Figure: Example of a LVQ algorithm using 5 prototypes per class (3 classes) – Elements of statistical learning ©Hastie et al. 2001

Learning Vector Quantization algorithm

1. Initialization:
   - Set up the initial M prototypes $Z = \{z_1, \cdots, z_M\}$. It can be done randomly or using an initialization with the K-Means algorithm. Then use a majority vote to assign each of prototype to a class $C(z_m)$.
   - Choose a learning rate $\epsilon \in [0, 1]$.
2. Go through the training set and update Z for each observation:
   - For each observation x_i, find the nearest prototype z_m.
   - If $C(x_i) = C(z_m)$, move z_m towards x_i: $z_m \leftarrow z_m + \epsilon(x_i - z_m)$.
   - If $C(x_i) \neq C(z_m)$, move z_m away from x_i: $z_m \leftarrow z_m - \epsilon(x_i - z_m)$
3. Repeat step 2 until convergence.
   - Optional: Reduce ϵ after each step 2 to enhance convergence.

Remark

The learning rate ϵ is a critical parameter that can change drastically the outcome of the classification.

Learning Vector Quantization: Classification

- Once the prototypes have been learned, the LVQ classifier behaves like the 1-NN algorithm using the prototypes instead of the training data.
- The class of each presented unlabeled data is determined based on the class of the closest prototype.

Remark

Using LVQ, the prototype can be trained (updated) in real time while being use on unlabeled data. This algorithm is therefore great for online learning.

LVQ: Summary

Pros

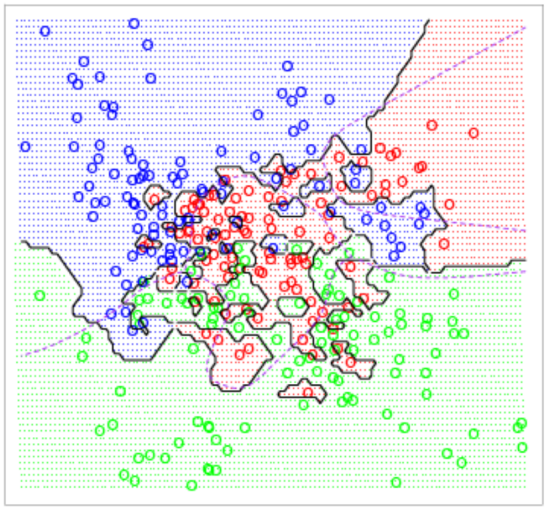
- Low Complexity
- Low memory consumption
- Can deal with online and incremental data
- Can build a good model
- Is still easy and intuitive

Cons

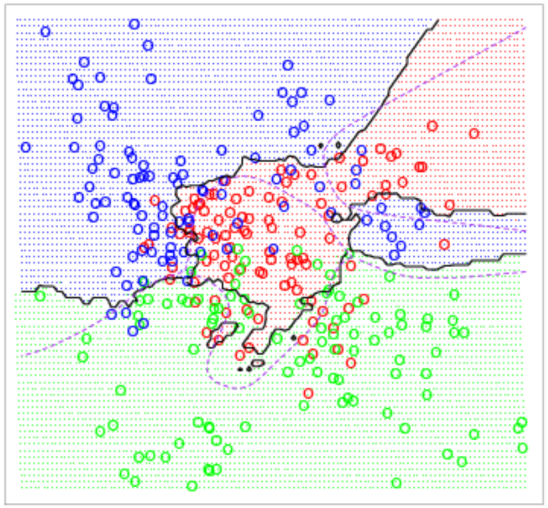
- Is often less accurate than KNN.
- Critical choice of the learning rate parameter ϵ
- Ill-adapted for categorical data

LVQ vs KNN

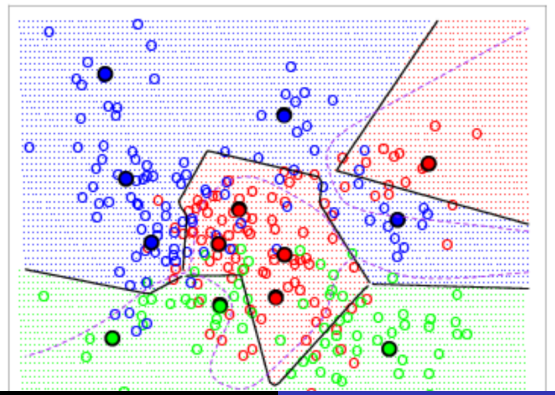
In the bias variance trade-off, 1-NN tends toward bias while LVQ tends toward variance. KNN is somewhere in between depending on the value of K.



(a) 1-NN



(b) 15-NN



LVQ vs KNN

- Use KNN when: You have a relatively small data set, you don't need to build a model, you don't need to generalize from your training set.
- Use LVQ when: You have a large data set, you need to build a model, you are dealing with a semi-supervised problem, you need to learn data incrementaly or on-line, you can afford a slightly lower accuracy or want a higher variance.

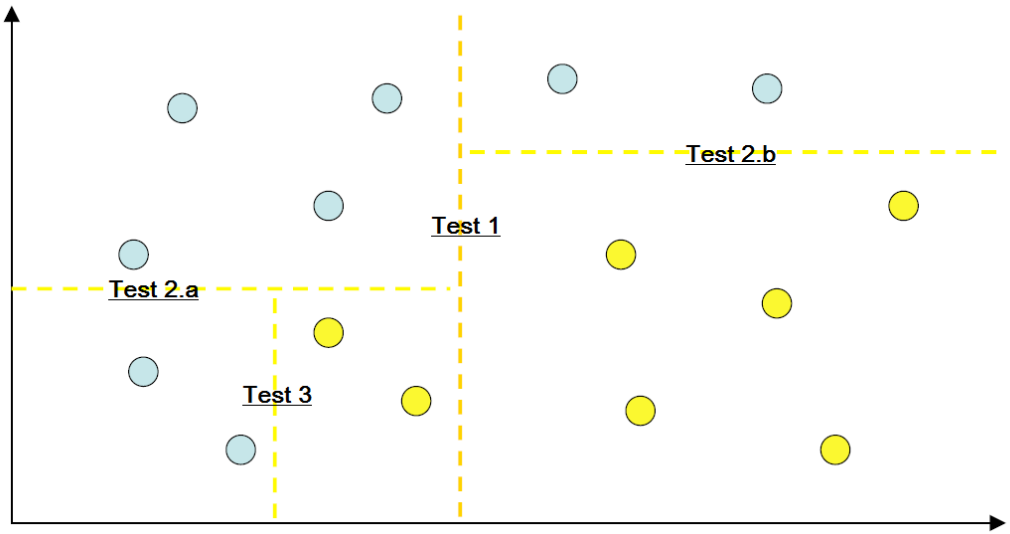
Remark

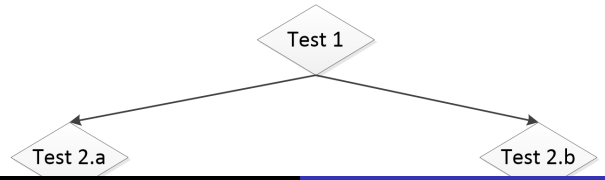
For simple problems, both will work just fine.

Decision trees

Decisions trees are very common classifiers that mine rules from the training set:

- They are mostly applied to categorical data, but not only.
- They decompose the feature space according to the most discriminating variable at each stage.
- There are usually more than one possible tree per data set.





Decision trees

A decision tree is a tree of a function-discrete representation. It can be used as a decision support tool that uses a tree-like graph or model of decisions and their possible consequences. Learning

decision trees are among the most commonly used classification methods. The main algorithms are ID3, ID4, C4.5 and C5.0

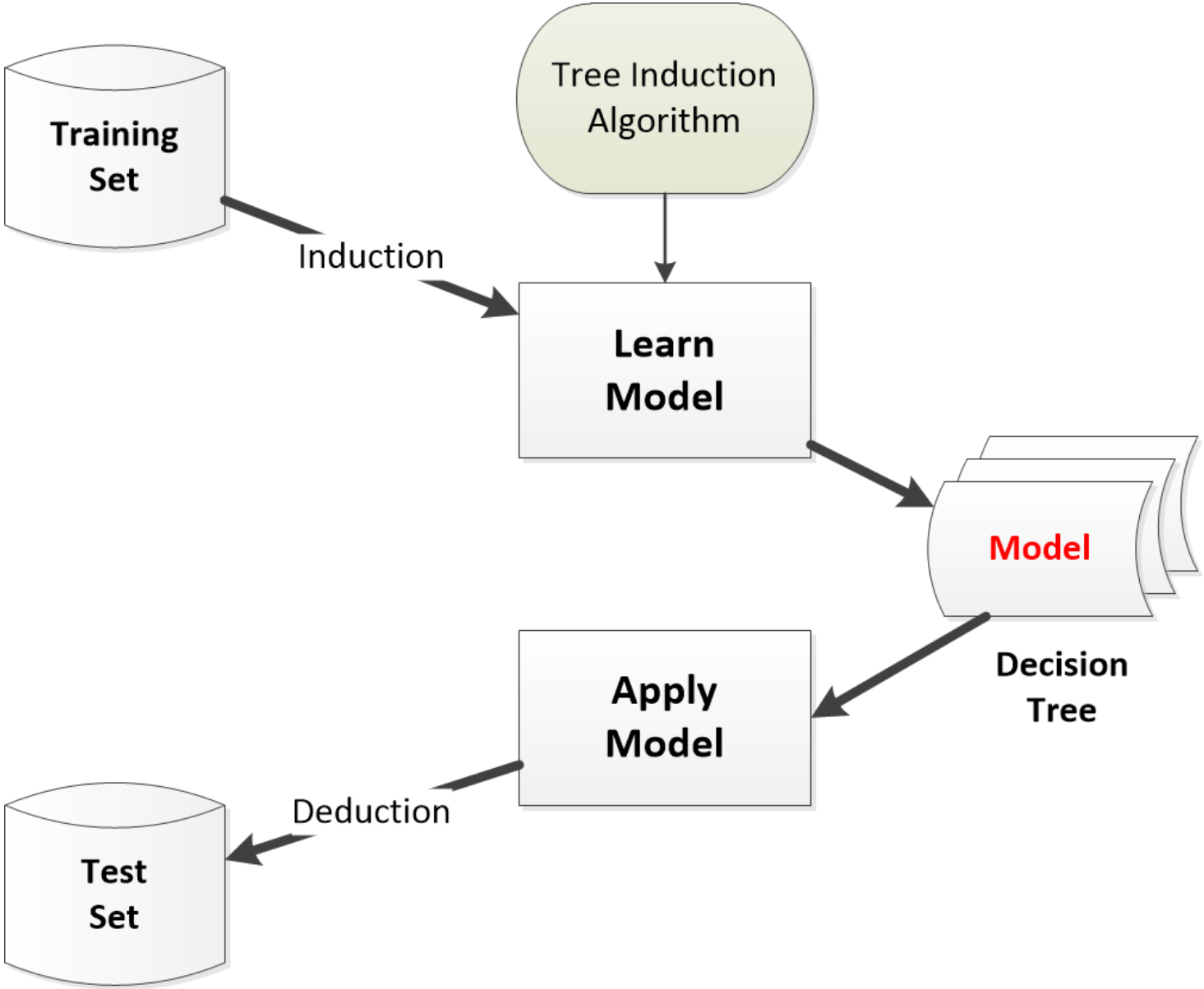
Properties

- **Expressiveness:** It can represent disjunctions of conjunctions
- **Readability:** It can be translated as a set of decision rules

Notae

- Disjunction : A or B
- Conjunction : A and B

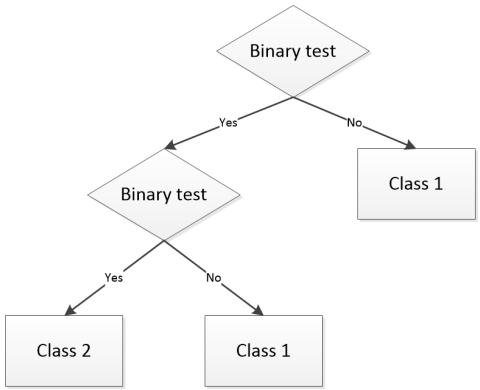
Decision tree: Classification Process



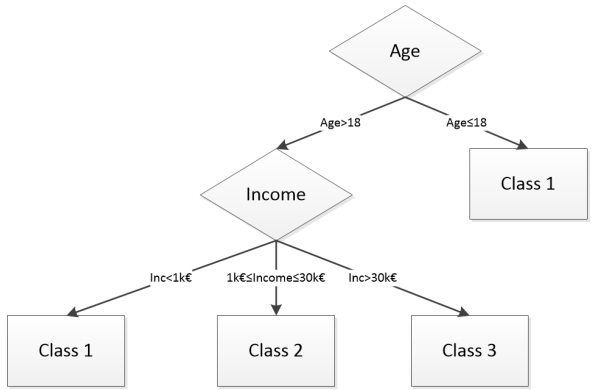
Types of trees

Decisions trees can be categorized according to three criteria:

- The type of data: Numerical, Categorical, Mixed
- The type of nodes: Binary leaves, multiple leaves
- The overall shape of the tree

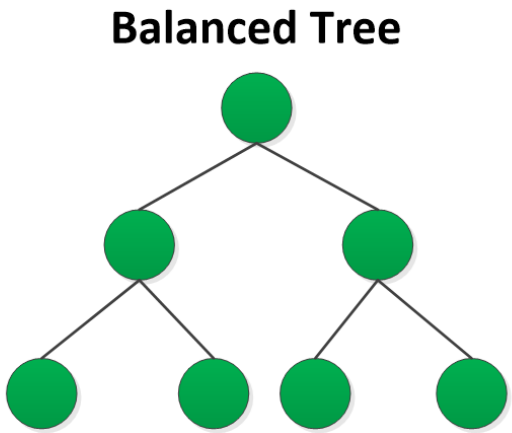


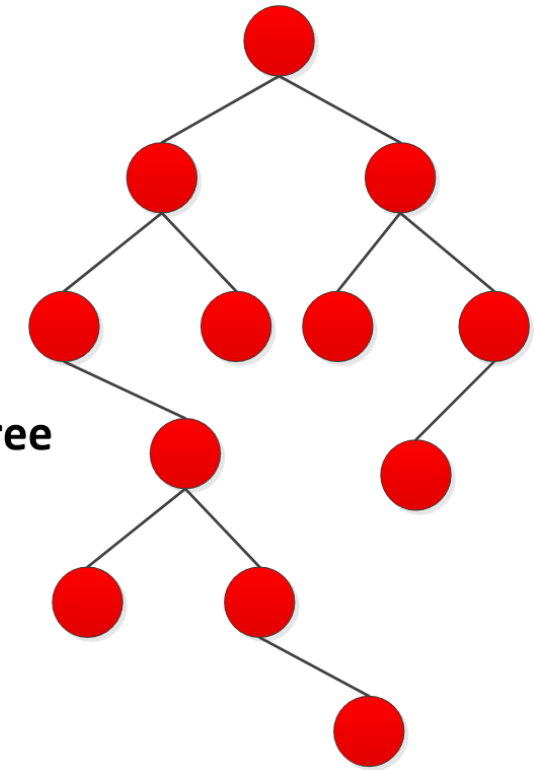
(c) Example of a binary tree

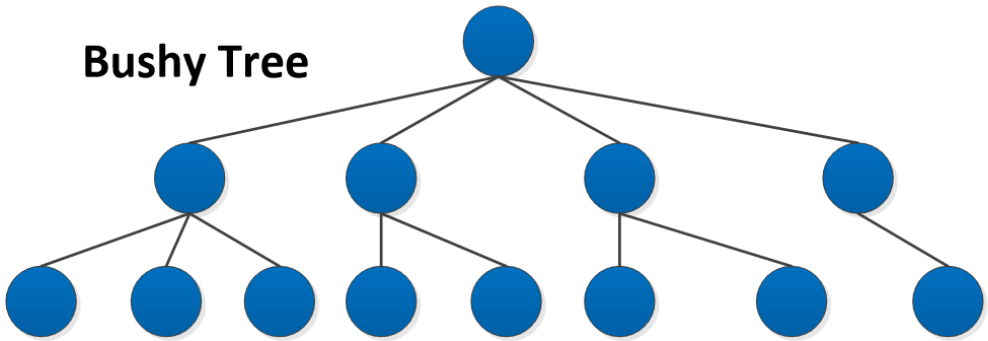


(d) Example of a numerical non-binary tree

Types of trees







Types of trees

- Deep trees are usually very biased, can't generalize much outside of their training set and are difficult to interpret.

Setting the right Depth for your tree

Most Decision trees algorithms will allow you to choose the maximum depth of your tree.

- How Deep is too deep will depend on the complexity of the problem
- Deeper trees tend towards overfitting, while less deep trees will tend towards underfitting.
- The best option is to start from a deep tree and to prune it in a way that minimizes the error on the training set.

- While balanced-trees are usually the most preferable option, bushy trees should not be frowned upon in problems with a lot of classes, or when they can help reducing the depth of the tree.

Link between decision trees and 1-NN classifiers

Remarks

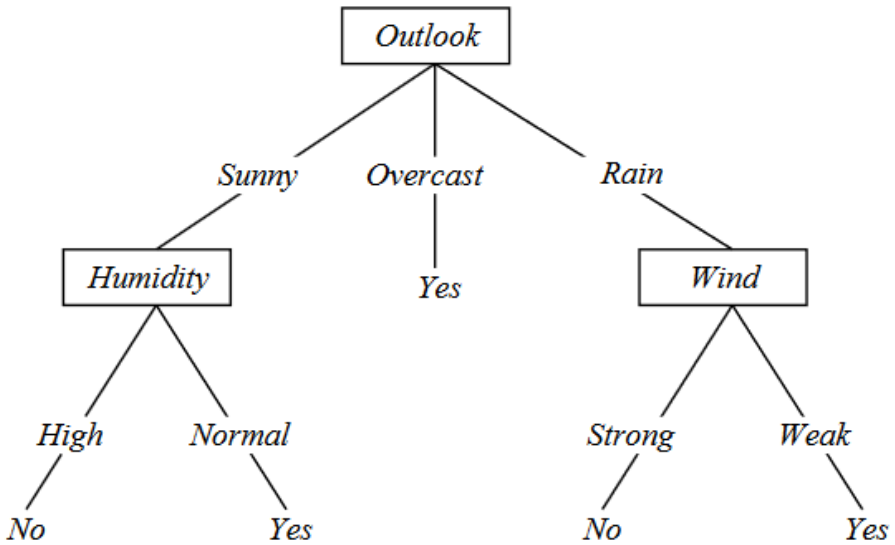
- For most decision trees, it is possible to build an equivalent 1-NN classifier.
- Each leave of a decision tree is equivalent to a data in the learning set of a 1-NN classifier.

This process is much easier with discrete variables.

The Weather Problem

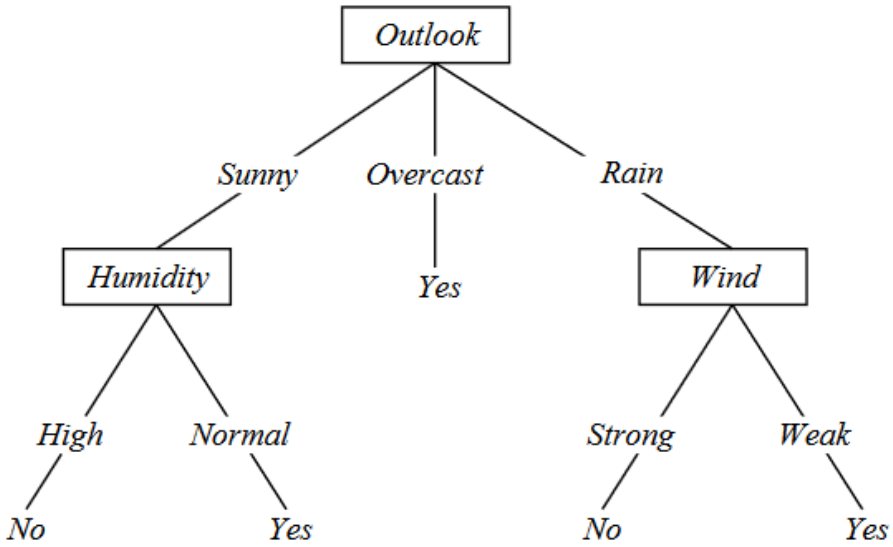
Outlook	Temperature	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	Normal	Weak	Yes
...	...	...	...	...

The goal of this problem is to determine which weather conditions are the most favorable to let your kids go play outside.



From the resulting tree, we can see that temperature is not among the most relevant parameters here.

The Weather Problem



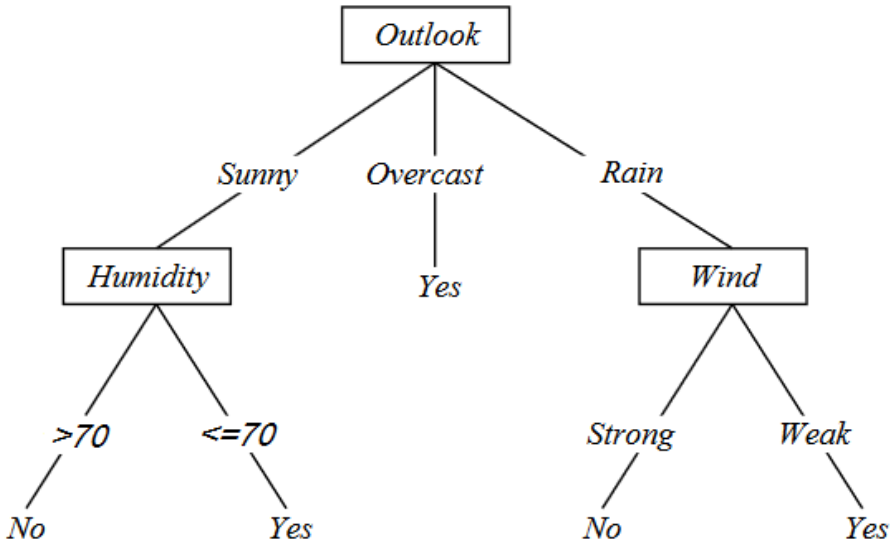
From this tree we can extract the following rules:

- If (Outlook=Sunny) AND (Humidity=High) THEN Play=No
- If (Outlook=Sunny) AND (Humidity=Normal) THEN Play=Yes
- If (Outlook=Overcast) THEN Yes
- If (Outlook=Rain) AND (Wind=Strong) THEN Play=No
- If (Outlook=Rain) AND (Wind=Weak) THEN Play=Yes

The Weather Problem

Outlook	Temperature	Humidity	Wind	Play
Sunny	29	85	Weak	No
Sunny	27	90	Strong	No
Overcast	28	86	Weak	Yes
Rain	24	80	Weak	Yes
...	...	...	...	...

The same problem can be processed with mixed attributes. A similar tree can be found.

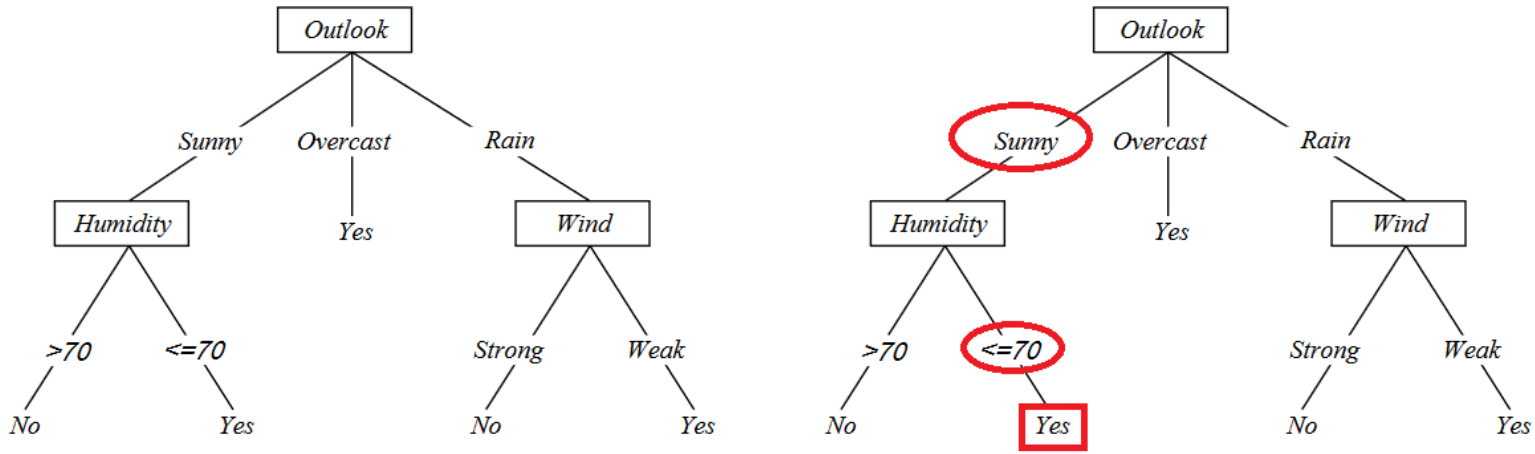


Remark

It usually takes more time to compute a decision tree with numerical values, because of the time required to find the optimal cut value.

The Weather Problem: Unlabeled Example

Outlook	Temperature	Humidity	Wind	Play
Sunny	25	55	Weak	?

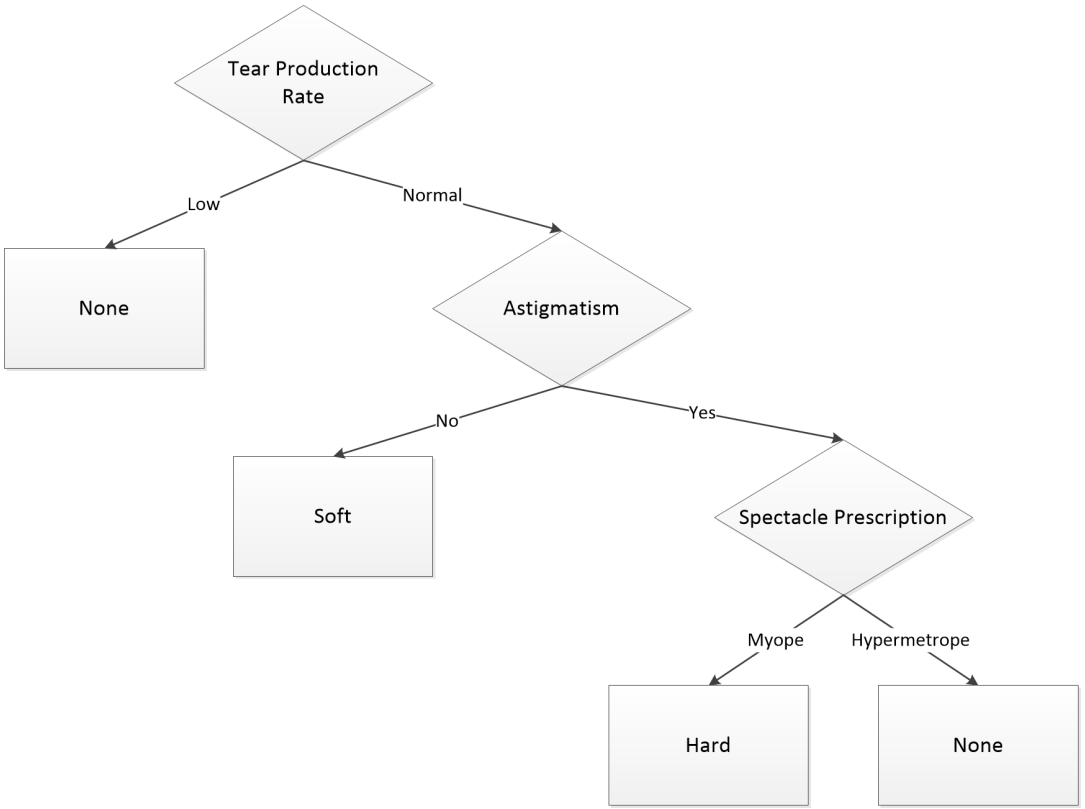


This example would be labeled **Yes** by the decision tree.

The Contact Lenses Data

Age	Prescription	Astigmatism	Tear production	Recommended Lenses
Young	Myope	No	Low	None
Young	Myope	No	Normal	Soft
Young	Myope	Yes	Low	None
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	No	Low	None
Young	Hypermetrope	No	Normal	Soft
Young	Hypermetrope	Yes	Low	None
Young	Hypermetrope	Yes	Normal	Hard
Pre-presbyopic	Myope	No	Low	None
Pre-presbyopic	Myope	No	Normal	Soft
Pre-presbyopic	Myope	Yes	Low	None
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	No	Low	None
Pre-presbyopic	Hypermetrope	No	Normal	Soft
Pre-presbyopic	Hypermetrope	Yes	Low	None
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	No	Low	None
Presbyopic	Myope	No	Normal	None
Presbyopic	Myope	Yes	Low	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	No	Low	None
Presbyopic	Hypermetrope	No	Normal	Soft
Presbyopic	Hypermetrope	Yes	Low	None
Presbyopic	Hypermetrope	Yes	Normal	None

The Contact Lenses Data



Summary

Pros

- Intuitive, easy to understand and to use
- Build comprehensive models
- The most commonly classifier for decision making
- Can learn in a single sweep

Cons

- The process to build the tree is complex
- There are always several possible trees
- Choosing the depth of the tree is a complex decision
- Does not work well with datasets that have too many attributes.

Naive Bayes Classifiers: Introduction

Naive Bayes classifiers are a family of simple **probabilistic** *classifiers based on* **Bayes Theorem** *and a strong independence hypothesis between the features.*

Intuition

To find out the probability of the previously unseen instance belonging to each class, simply pick the "most probable" class.

- These probability are assessed using Bayes theorem applied on the training data.

Naive Bayes Classifiers: Introduction

Bayes Theorem

$$p(c_j|x) = \frac{p(x|c_j)p(c_j)}{p(x)}$$

- $p(c_j|x)$ The probability of instance x belonging to class c_j.
  - We want to compute this probability.
- $p(x|c_j)$ The probability of generating instance x knowing class c_j.
  - Knowing the distribution function of class c_j and the features of x, what is the probability of x ?
- $p(c_j)$ The occurrence probability of class c_j.
  - How frequent is class c_j in the training set ?
- $p(x)$ The occurrence probability of instance x.
  - This can usually be ignored because it is independent from c_j and the same for all instances.

Naive Bayes Classifiers: Simple example

One year ago, on my way back from Holland, I was arrested by a police officer names "Claude". I was a bit high and can't remember whether Officer Claude was a male or a female ...

Using a bayesian classifier, and a police data base with names and sex, we can try to guess whether it is more likely that officer Claude was a male or a female. We have two classes: c_1=male and c_1=female.

Note: In French, "Claude" can be a male or a female name

Claude Gensac

Claude François

What is the probability of being named "Claude" given that you're a male

What's the probability of being a m

$$p(male|Claude) = \frac{p(Claude|male)p(male)}{p(Claude)}$$

What's the probability of being named "Claude" ?

Naive Bayes Classifiers: Simple example

Name	Sex
Claude	Male
Laura	Female
Claude	Female
Claude	Female
Arthur	Male
Karima	Female
Rose	Female
Sergio	Male

Table: Training data (List of Police officers in Lille)

$$p(male|Claude) = \frac{p(Claude|male)p(male)}{p(Claude)}$$
$$= \frac{1/3 \times 3/8}{3/8} = \frac{0.125}{3/8}$$
$$p(female|Claude) = \frac{p(Claude|female)p(female)}{p(Claude)}$$
$$= \frac{2/5 \times 5/8}{3/8} = \frac{0.250}{3/8}$$

Since $0.125 < 0.250$, we can conclude that most likely Officer Claude was a female !

Naive Bayes with several features

- In the previous example there was only one features: the name. What happens when there are more ?
- To make the problem simpler, naive Bayes classifiers assume that the attributes have independent distributions (which is not always true).

Independence Hypothesis

- Let us note $\boldsymbol{x} = \{x_1, ..., x_d\}$ a data with d features.
- Under the hypothesis that all attributes are independent, we can write:

$$p(\boldsymbol{x}|c_j) = p(x_1|c_j) \times p(x_2|c_j) \times \cdots \times p(x_d|c_j)$$

Therefore, we have:

$$p(c_j|\boldsymbol{x}) \propto p(c_j) \prod_{i=1}^{d} p(x_i|c_j)$$

Naive Bayes with several features

- Note that Naive Bayes **is not sensitive** to irrelevant features.

Suppose that we are trying to classify persons gender based on several features, including eye color (which is irrelevant):
$p(Jessica|c_j) = p(eye = brown|c_j) \times p(wears\_dress = yes|c_j) \times \cdots$

$p(Jessica|female) = 9000/10000 \times 7500/10000 \times \cdots$
$p(Jessica|male) = 9001/10000 \times 3/10000 \times \cdots$

$p(eye = brown|female)$ and $p(eye = brown|male)$ should be almost identical and won't affect the outcome much. Wearing a dress however ...

Remark

This assumes that the estimates of the probabilities are good enough. Therefore the training set must be as big and as unbiased as possible.

Naive Bayes: Properties

Pros

- The only things to store are the probabilities: The training data need not be kept in memory and a single scan of the data is necessary to acquire the probabilities.
- The model is quite simple to understand.
- One of the fastest prediction model.

Cons

- Naive Bayes assumes that the features are fully independent. It is usually not true and can lead to more or less bias when several of them are too correlated.
- Naive Bayes tend to be biased toward the training data and can't generalize easily (e.g. It is impossible to classify a new instance with a single -or more- attribute values the occurrence of which is 0 in the training set).

Mosquito identification

In this example we consider 3 species of mosquitoes:

- *Culex Pipiens*, the common house mosquito
- *Anopheles Stephensi*, a common mosquito from the middle East
- *Aedes Aegypti*, the yellow fever mosquito (may also carry Dengue fever, Zika, or Chikungunya)

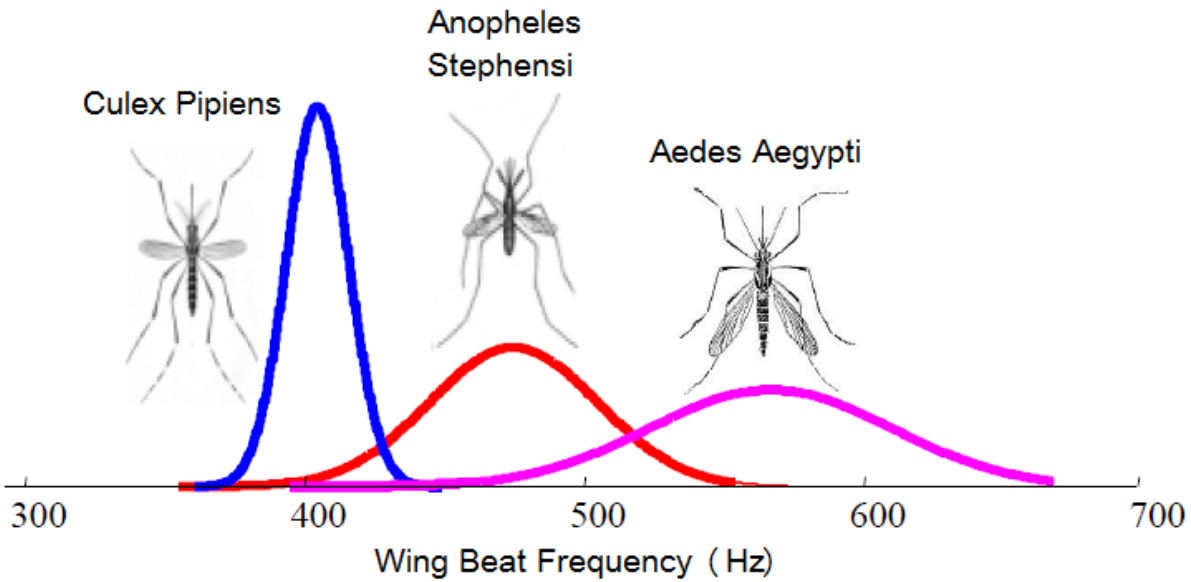
(g) Culex Pipiens

(h) Anopheles Stephensi

(i) Aedes Aegypti

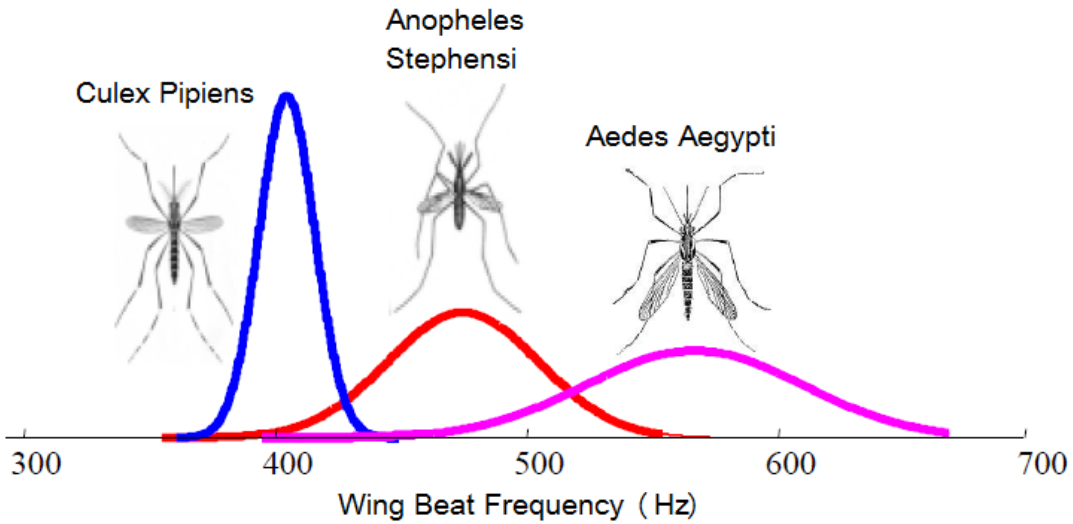
Mosquito identification

Mosquitoes have distinct wing beat frequencies.



- Culex Pipiens: $\mathcal{N}(\mu = 390, \sigma = 14)$
- Anopheles Stephensis: $\mathcal{N}(\mu = 475, \sigma = 30)$
- Aedes Aegypti: $\mathcal{N}(\mu = 567, \sigma = 43)$

Mosquito identification



- Culex Pipiens: $\mathcal{N}(\mu = 390, \sigma = 14)$
- Anopheles Stephensis: $\mathcal{N}(\mu = 475, \sigma = 30)$
- Aedes Aegypti: $\mathcal{N}(\mu = 567, \sigma = 43)$

Suppose I see a mosquito with a wing frequency of 500Hz, which one is it?

Mosquito identification

Suppose I see a mosquito with a wing frequency of 500Hz, which one is it?

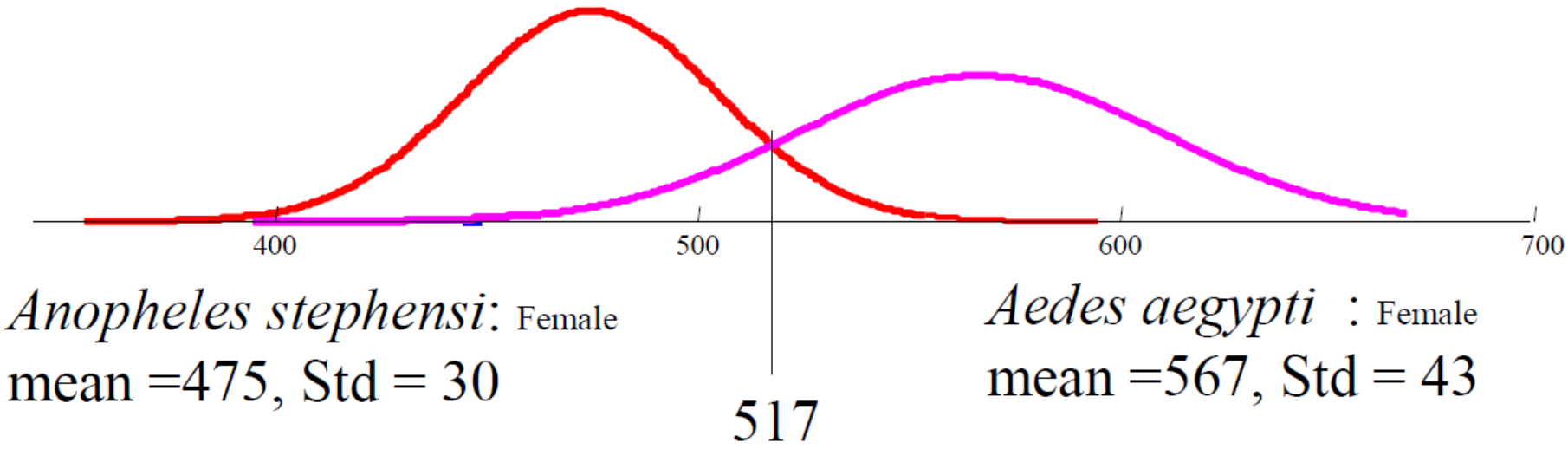
$$p(Culex|wingbeat = 500) = \frac{\exp^{-\frac{(500-390)^2}{2\times 14^2}}}{14\sqrt{2\pi}} \approx 0$$

$$p(Anopheles|wingbeat = 500) = \frac{\exp^{-\frac{(500-475)^2}{2\times 30^2}}}{30\sqrt{2\pi}} = 0.0094$$

$$p(Aedes|wingbeat = 500) = \frac{\exp^{-\frac{(500-576)^2}{2\times 43^2}}}{43\sqrt{2\pi}} = 0.0047$$

Most likely it is an Anopheles.

Mosquito identification: Getting the probability



These does not look like probabilities:
$p(Anopheles|wingbeat = 500) = 0.0094$

$p(Aedes|wingbeat = 500) = 0.0047$

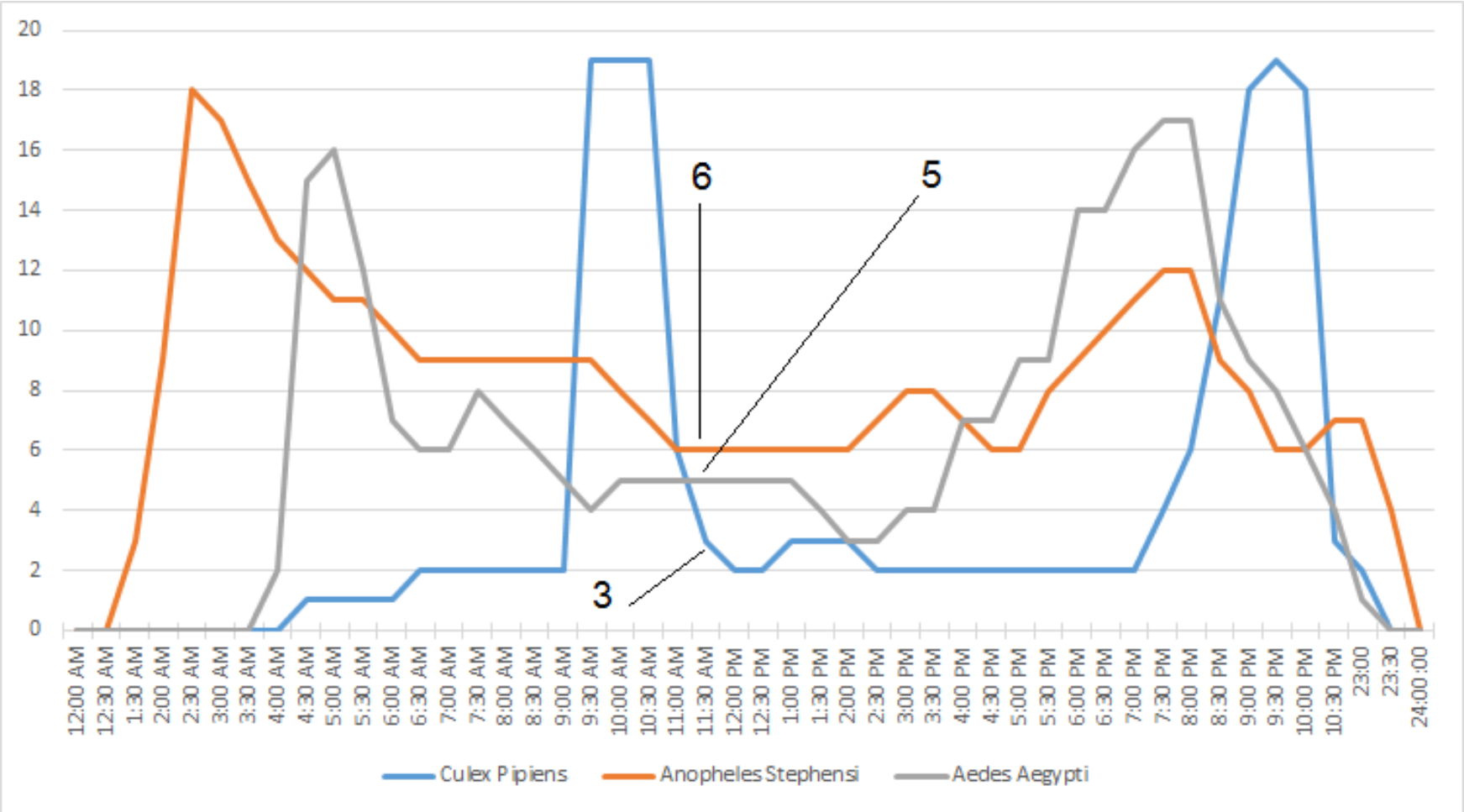
Getting the probabilities

$P(Anopheles|wbf = 500) = \frac{0.0094}{0+0.0094+0.0047} = 0.77$

$P(Aedes|wbf = 500) = \frac{0.0047}{0+0.0094+0.0047} = 0.23$

Mosquito identification: More features

We now have additional informations in the form of a chart representing how many mosquitoes are active depending on the time of the day.



Suppose I am savagely attacked by a mosquito with a wingbeat frequency of 420Hz at 11:30am. Which one is the most likely culprit ?

Mosquito identification: More features

$$P(Culex|420Hz, 11{:}30\text{am}) = \frac{2.87\times10^{-3}}{2.87\times10^{-3}+2.48\times10^{-3}+2.7\times10^{-5}} \times \frac{3}{3+5+6} = 0.1144$$

$$P(Anopheles|420Hz, 11{:}30\text{am}) = \frac{2.48\times10^{-3}}{2.87\times10^{-3}+2.48\times10^{-3}+2.7\times10^{-5}} \times \frac{6}{3+5+6} = 0.1976$$

$$P(Aedes|420Hz, 11{:}30\text{am}) = \frac{2.7\times10^{-5}}{2.87\times10^{-3}+2.48\times10^{-3}+0} \times \frac{5}{3+5+6} = 0.0018$$

Mosquito identification: More features

$$P(Culex|420Hz, 11{:}30\text{am}) = \frac{2.87\times10^{-3}}{2.87\times10^{-3}+2.48\times10^{-3}+2.7\times10^{-5}} \times \frac{3}{3+5+6} = 0.1144$$

$$P(Anopheles|420Hz, 11{:}30\text{am}) = \frac{2.48\times10^{-3}}{2.87\times10^{-3}+2.48\times10^{-3}+2.7\times10^{-5}} \times \frac{6}{3+5+6} = 0.1976$$

$$P(Aedes|420Hz, 11{:}30\text{am}) = \frac{2.7\times10^{-5}}{2.87\times10^{-3}+2.48\times10^{-3}+0} \times \frac{5}{3+5+6} = 0.0018$$

Resulting Probabilities

$P(Culex|420Hz, 11{:}30\text{am}) = 36.5\%$
$P(Anopheles|420Hz, 11{:}30\text{am}) = 63\%$
$P(Aedes|420Hz, 11{:}30\text{am}) = 0.5\%$

Mosquito identification: More features

$$P(Culex|420Hz, 11{:}30\text{am}) = \frac{2.87\times10^{-3}}{2.87\times10^{-3}+2.48\times10^{-3}+2.7\times10^{-5}} \times \frac{3}{3+5+6} = 0.1144$$

$$P(Anopheles|420Hz, 11{:}30\text{am}) = \frac{2.48\times10^{-3}}{2.87\times10^{-3}+2.48\times10^{-3}+2.7\times10^{-5}} \times \frac{6}{3+5+6} = 0.1976$$

$$P(Aedes|420Hz, 11{:}30\text{am}) = \frac{2.7\times10^{-5}}{2.87\times10^{-3}+2.48\times10^{-3}+0} \times \frac{5}{3+5+6} = 0.0018$$

Resulting Probabilities

$P(Culex|420Hz, 11{:}30\text{am}) = 36.5\%$
$P(Anopheles|420Hz, 11{:}30\text{am}) = 63\%$
$P(Aedes|420Hz, 11{:}30\text{am}) = 0.5\%$

- Anopheles Stephensis is again the most likely culprit.

Remarks

At this point you should be wondering where was the training set in this exercise.

- You never saw the training set, you only saw the model: Wing beat frequencies distributions and mosquito activity diagram.
- The training set was used to build the wing beat frequencies laws and the distribution diagram. Once you have them, you don't need the training set anymore.

Important remark

You saw normalization constants pretty much everywhere in the calculi. You don't need them to classify new items. Unless you really want probabilities, you don't have to normalize your results.

Evaluating classifiers

- The evaluation of a classifier is usually done using the **validation set**, the labels of which are known.
- There are several ways to validate classifier results depending on their type and the number of classes.

Accuracy: the simplest evaluation criterion

$$Accuracy = \frac{\text{Number of correctly classified data}}{\text{Total number of data}}$$

- The result is in percentage, 100% being the best.

Evaluating binary classifiers

Binary classifiers (with 2 classes: True and False) have specific validations measures that assess different parameters.
Let us consider the following notations:

- TP: True positive (data classified True and that are really in this class)
- FP: False positive (data classified True but are not)
- TN: True negative (data classified False and are really in this class)
- FN: False negative (data classified False but are actually True)

Remark

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Evaluating binary classifiers

Recall or True Positive Rate (TPR)

$$Recall = \frac{TP}{TP + FN}$$

- It is also called "hit rate" or "sensitivity".
- It is the probability of correctly labeling a Positive case.

Evaluating binary classifiers

Fall-out or False Positive Rate (FPR)

$$FPR = \frac{FP}{TN + FP}$$

specificity (SPC) or True Negative Rate

$$specificity = \frac{TN}{TN + FP} = 1 - FPR$$

The specificity (or TNR) is a statistical measure of how well a binary classifier correctly identifies the negative cases.

Evaluating binary classifiers

Precision or Positive Predictive Value (PPV)

$$precision = \frac{TP}{TP + FP}$$

The precision is the probability that a positive prediction is correct.

F-Measure

$$\text{F-Measure} = \frac{2 \times precision \times recall}{precision + recall}$$

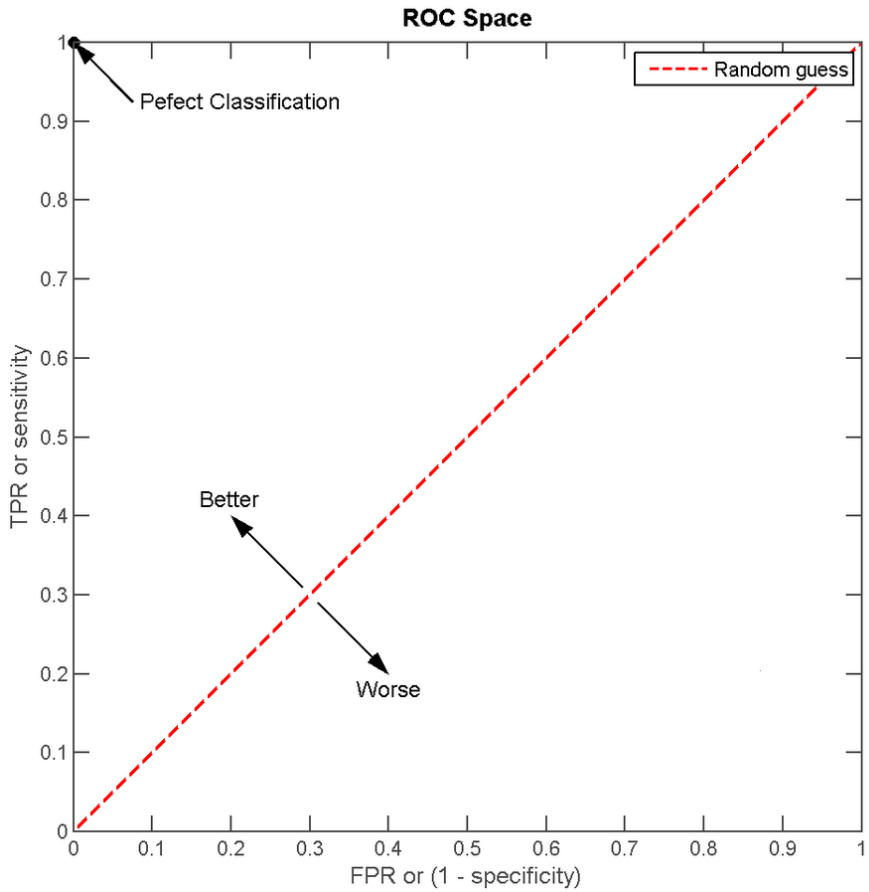
- The F-Measure is the harmonic mean of the precision and the recall.
- It can be used as a single measure to evaluate the performances of a binary classifier.

Evaluating binary classifiers

- TP, FP, TN and FP provide relevant information
- No single measure tells the whole story
- A classifier with 90% accuracy can be useless if 90% of the population does not have cancer and the 10% that do are misclassified.
- If possible, use multiple measures.
- Beware of the obscure terminological confusion in the literature !
  - Depending on the field, specificity is sometimes refers to precision
  - Different name exist for the same thing
  - Always provide the formula when you use terms such as FP, TP, etc.

Evaluating binary classifiers: ROC space

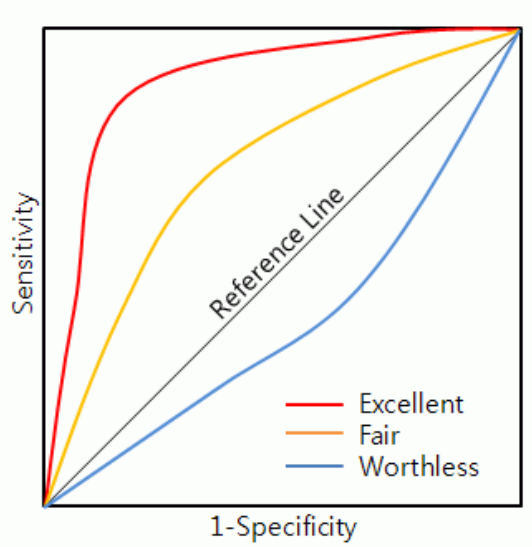
The ROC space (Receiver operating characteristic) is a type of graph based on the fall-out and the sensitivity and that can be used to evaluate a classifier.

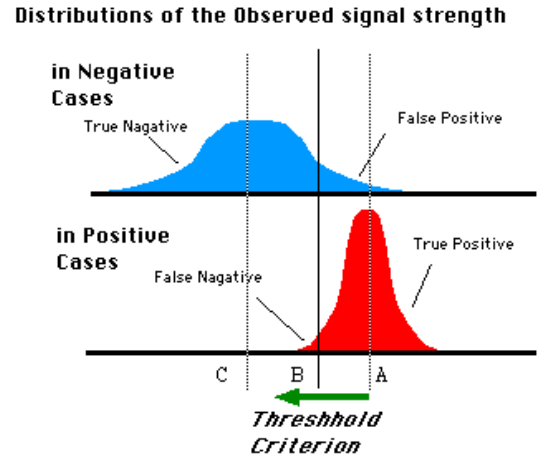


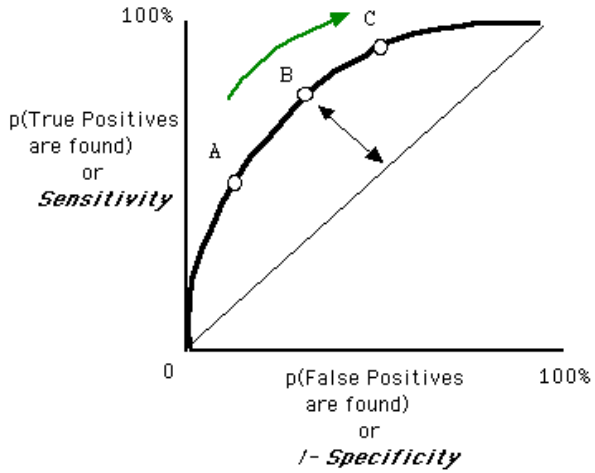
Evaluating binary classifiers: ROC curves

A ROC curve plots uses the ROC space to assess the quality of a classifier. It is plotted using different parameters as reference points to draw the curb.

- Useful to find the right parameters
- Useful to compare binary classifiers







Generalizing to non-binary classifiers

Generalizing to classifiers that have more than 2 classes is often complicated.

- Criteria exist in the literature but they are often quite complex and restricted to specific cases.
- It is possible to do some basic analysis using confusion matrices between the expected classes and the found classes.
- Otherwise, indexes such as the accuracy, or vector comparing measures (e.g. Rand Index and Adjusted Rand Index) are good solutions.

Building a model and complexity issues

The complexity of a model is an important criterion to evaluate a model: When comparing several models/classifiers that show similar performances in term of accuracy (or error), and have similar bias and variance, **the simplest models are usually considered the best**.

Building a model and complexity issues

The complexity of a model is an important criterion to evaluate a model: When comparing several models/classifiers that show similar performances in term of accuracy (or error), and have similar bias and variance, **the simplest models are usually considered the best**.

Examples of complexity measures

- The **Bayesian Information Criterion** (BIC)
- The **Akaike Information Criterion** (AIC)

For both criteria, the lower the better.

Building a model and complexity issues

The complexity of a model is an important criterion to evaluate a model: When comparing several models/classifiers that show similar performances in term of accuracy (or error), and have similar bias and variance, **the simplest models are usually considered the best**.

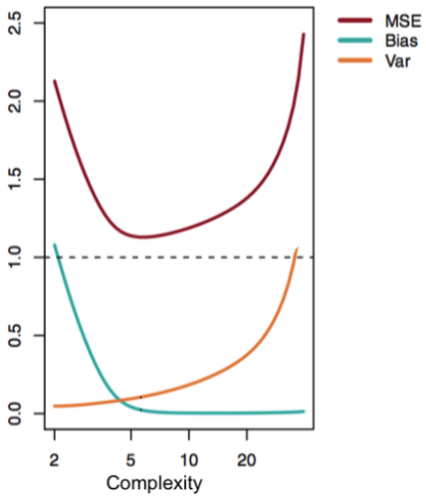
Examples of complexity measures

- The **Bayesian Information Criterion** (BIC)
- The **Akaike Information Criterion** (AIC)

For both criteria, the lower the better.

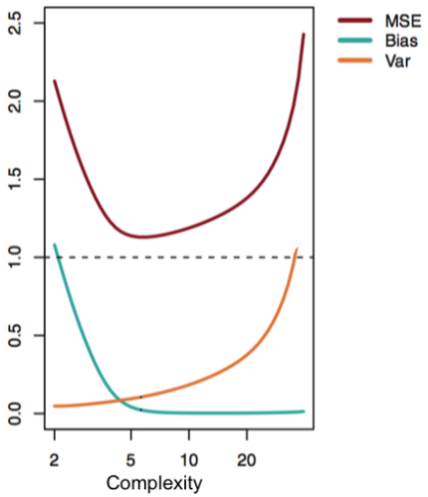
Remark: Models that are too simple tend to have a low accuracy (high error), while models that are too complex tend too overfit.

Bias-variance trade-off and complexity



- The bias tends to decrease with the complexity of the model
- The variance tends to increase with the complexity of the model
- The mean square error (MSE) on validation data first decreases when the model gets more complex, and then increases again when the model gets too complex and overfit.

Bias-variance trade-off and complexity



- The bias tends to decrease with the complexity of the model
- The variance tends to increase with the complexity of the model
- The mean square error (MSE) on validation data first decreases when the model gets more complex, and then increases again when the model gets too complex and overfit.

Deciding between several models relies on finding the one(s) with the best variance-bias trade-off, and the lowest complexity.

Bibliography

- Christopher M. Bishop, Pattern Recognition and Machine Learning (2006)
- R. O. Duda, P. E. Hart, D. Stork, Wiley and Sons, Pattern Classification (2000)
- Tom M. Mitchell, Machine Learning (1997)