# CLASSIFICATION I

## Machine Learning for Autonomous Robots

Manuel Meder
Robotics Group, University of Bremen

November 15, 2022 – Bremen, Deutschland

University of Bremen

German Research Center for Artificial Intelligence

# Introduction

# The Reason

*What is classification about?*


(a) Head


(b) Tail

*Why do we need a classifier?*

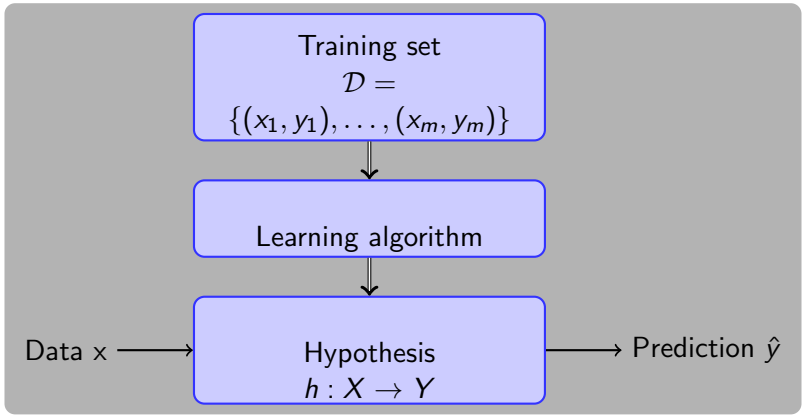
Actually, humans are pretty good at classification and pattern recognition, but in order to let machines do this job ...

# Classifying the Classifiers

Classification Methods:

- supervised
- generative vs. discriminative
  *generative*: Gaussian distribution, Naive Bayes
  specifies a way to generate the feature vector for a possible class *discriminative*: SVM, Neural networks, Logistic regression
  directly map feature vectors to outputs
- simple methods: nearest neighbour
- modern methods: naive Bayes, logistic regression, decision tree
- advanced methods: random forest, support vector machine (SVM), neural networks

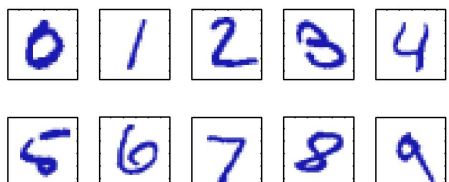
# Supervised Learning




- If $Y$ is a discrete domain: classification
- If $Y$ is a continuous domain: regression

# How would you start?

What would you do, e.g. to make a machine understand the following pictures?!



Keep it simple, 2 min to think about it:

- Write down what your approach is and what resources do you require

# The Basic Algorithm

## Basic

1. compare each unlabeled instance with all instances in the training set
2. find the instance from the training set that matches the unlabeled one best, i.e. find the nearest neighbour
3. label the unlabeled instance with the class of the nearest neighbour

# The k-Nearest Neighbour Algorithm

## Majority out of k

1. use the initial steps of the basic approach, but find the k nearest neighbours
2. use the majority class of the k nearest neighbours to label the instance in question

What does **nearest** mean, or better: How to determine the **nearest** neighbours, given an input vector?!

# Distance Measures

## Given:

Two input vectors with numeric values:

$$\mathbf{a} = (a_1, a_2, ..., a_n)$$

$$\mathbf{b} = (b_1, b_2, ..., b_n)$$

## Normalized Euclidean distance

$$d_{euclidean}(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{n} \frac{(a_i - b_i)^2}{(\sigma_i)^2}}, \text{ where}$$

$\sigma_i$ standard deviation of $i^{th}$ feature across the training set

# Distance Measures - importance of scale

**Features can vary in scale**

Changing the scale of one dimension affect the set of nearest neighbours

Countermeasure is standardization of scale, e.g. through

- normalization
- express feature values as multiples of the standard deviation

## Mahalanobis distance
(distance between points in a multi-dimensional space)

$$d_{Mahalanobis}(\mathbf{a}, \mathbf{b}) = \sqrt{(\mathbf{a} - \mathbf{b})^T S^{-1} (\mathbf{a} - \mathbf{b})} \text{ , where}$$

- S covariance matrix, e.g. uncorrelated features
  then $S = (\sigma_0^2\ \sigma_1^2\ ...\ \sigma_n^2) I_n$
- $I_n$ n-dimensional identity matrix

Note: Euclidean distance is the special case where the $S = I$

# Further Distance Measures

## Manhattan distance
(distance along axes at right angles)

$$d_{Manhattan} = \sum_i |a_i - b_i|$$

## Chebyshev distance
(maximum distance of any feature dimension)

$$d_{Chebyshev} = \max_i |a_i - b_i|$$

## Levenshtein distance*
(distance between two strings)

*Basically, how much do we have to edit one string, to get the other one*

*(original work: A. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," Soviet Physics Doklady, vol. 10, no. 8, pp. 707-710, 1966.
see `http://www.levenshtein.net/` or `http://en.wikipedia.org/wiki/Levenshtein_distance` for details)

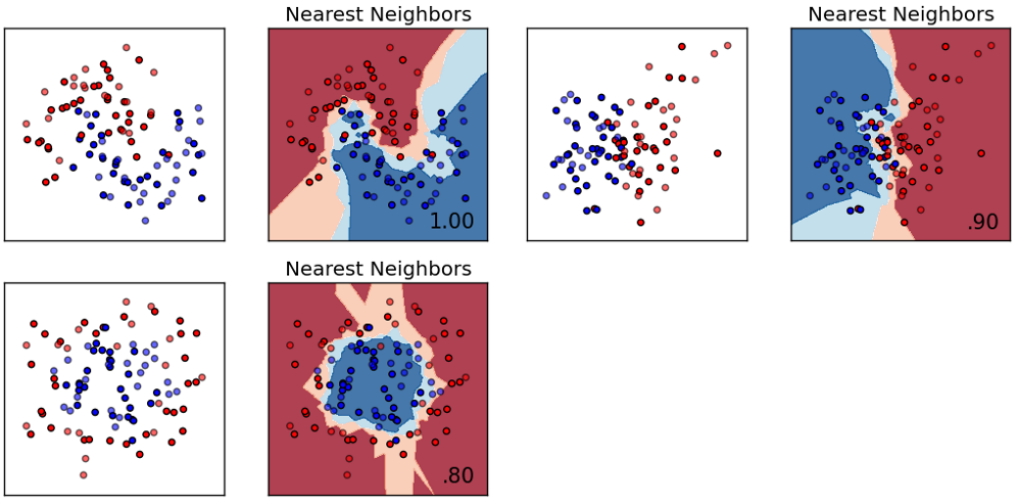
# Evaluation

The nearest neighbour method is simple, easy to use and can achieve good results.
Everything seems perfect! What problem can you find?

*Hints:*

- Did this method really learn a classifier? If yes, what is it?!
- This method works, but is it efficient?!

# Classifier Comparison with Scikit-learn




(https://scikit-learn.org/0.15/_images/plot_classifier_comparison_0011.png, adapted)

# Evaluation

**Pro:**

- online training is possible
- shows which neighbors had influence on prediction
- simple approach which can work for complex problems

**Con:**

- requires all training data, no model learned
- finding correct scaling factors might be expensive
- prediction time increases with sample size, i.e. bad time complexity

# Bayes Based Classification

# Introducing (or Recalling) Bayes theorem

**Bayes' theorem**

$$P(y_i|\mathbf{x}) = \frac{P(\mathbf{x}|y_i)P(y_i)}{P(\mathbf{x})}, \text{ where}$$

| | |
|---|---|
| $P(y_i)$ | a priori probability for class $y_i$ |
| $P(\mathbf{x}\|y_i)$ | likelihood of class $y_i$ with respect to (continuous) data |
| $P(\mathbf{x})$ | probability density function of $\mathbf{x}$ $= \sum_{i=1}^{2} P(\mathbf{x}\|y_i)P(y_i)$ (for two classes $y_1$ and $y_2$) |

Note:
Probability of data D given hypothesis H
Likelihood of hypothesis H given data D

# Classification Task using Bayes

- If we know the so called posterior probability $P(y|\mathbf{x})$, then we can take the class with the highest probability.
- But measuring $P(y|\mathbf{x})$ directly is hard. So what to do?!
  - Acquire "enough" training data and use them to count the fraction of the specific class.

# Naive Bayes

Using Bayes for a classification of a boolean class ($y \in \{-1, 1\}$) and a boolean feature vector of size n, we would require the estimation of

$$2^{n+1} - 1 \text{ parameters}$$

(note $P(e) + P(\neg e) = 1$, $2^n - 1$ parameters for $P(\mathbf{x}|y)$ for $y = -1$ or $y = 1$ and one for $P(y = 1)$; see also Mitchell)

Hence, we make a (most often) wrong, but useful **assumption** to reduce the number of required parameters:

## Conditional independence

*Features are conditionally independent of each other given the class*

With full dependence: $P(x_1, x_2, \ldots, x_n|y) = P(x_1, x_2, \ldots, x_{n-1}|x_n, y)P(x_n|y) = \ldots$

With conditional independence:

(1) $P(x_1, x_2, \ldots, x_n|y) = P(x_1|y)P(x_2|y)\ldots P(x_n|y)$

(2) $P(y, x_1, x_2, \ldots, x_n) = P(x_1, x_2, \ldots, x_n|y)P(y) = P(y)\prod_i P(x_i|y)$

**It helps**

*Now, only $2n + 1$ parameters need to be estimated!*

## Finding the Posterior Probability

Searching for:

$$P(y|x_1, x_2, \ldots x_n) = \frac{P(y)P(x_1, x_2, \ldots x_n|y)}{P(x_1, x_2, \ldots x_n)}$$

Since $P(x_1, x_2, \ldots x_n)$ does not depend on class y and feature values are known, it can be replaced by a constant $\alpha$. This leads to:

$$P(y|x_1, x_2, \ldots x_n) = \frac{1}{\alpha}P(y, x_1, x_2, \ldots x_n) = \frac{1}{\alpha}P(y)\prod_i P(x_i|y)$$

# Using Naive Bayes for Classification

In order to decide on a class $y_1$ or $y_2$ for existing data,we can apply the following reasoning:

| | |
|---|---|
| If $P(y_1\|\mathbf{x}) > P(y_2\|\mathbf{x})$ | label data with class $y_1$ |
| If $P(y_1\|\mathbf{x}) < P(y_2\|\mathbf{x})$ | label data with class $y_2$ |
| Else | random label |

e.g. spam filters use a log ratio, so that:

$$\ln \frac{P(spam|data)}{P(\neg spam|data)} = \ln \frac{P(spam)}{P(\neg spam)} + \sum_i \ln \frac{P(word_i|spam)}{P(word_i|\neg spam)}$$

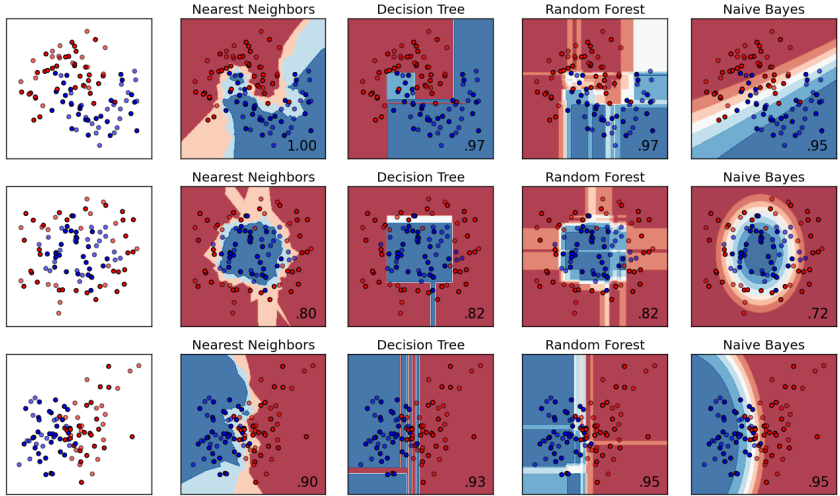
so that if its spam:

$$\ln \frac{P(spam|data)}{P(\neg spam|data)} > 0$$

# Discrete or Continuous Feature

- discrete features
  It is rather simple to extract the probabilities from discrete inputs using counts
- continuous features
  Different methods exists, i.e. parametric approach (**Gaussian**), kernel method or discretization

# Classifier Comparison with Scikit-learn




(https://scikit-learn.org/0.15/_images/plot_classifier_comparison_0011.png)

## Evaluation

**Pro:**

- though based on a (most often) wrong assumption, often used in practice (e.g., spam detection)
- feature distributions can be handled independently, so easier to deal with dimensionality
- correct class needs "only" to be more probable than any other allowing to cover an imperfect modeling (e.g. the independence assumption)
- popular because time for training and testing complexity has linear dependence to number of samples, i.e. optimal time complexity

**Con:**

- can not deal with problems where features are dependent

# Logistic Regression (Motivation via Bayes Theorem)

Logistic regression allows classification for a binary output, but uses a parametric form to approximate the probability $P(y|x)$. Bayes theorem can be rewritten as a representation of the logistic function. Here for two classes $y_1 = -1$ and $y_2 = 1$:

$$P(y_1|\mathbf{x}) = \frac{P(\mathbf{x}|y_1)P(y_1)}{P(\mathbf{x})} = \frac{P(\mathbf{x}|y_1)P(y_1)}{P(\mathbf{x}|y_1)P(y_1)+P(\mathbf{x}|y_2)P(y_2)} = \frac{1}{1+\frac{P(\mathbf{x}|y_2)P(y_2)}{P(\mathbf{x}|y_1)P(y_1)}}$$

$$= \frac{1}{1+\exp(-a)} = \sigma(a), \text{ where}$$

$$a = \ln \frac{P(\mathbf{x}|y_1)P(y_1)}{P(\mathbf{x}|y_2)P(y_2)} = \ln \frac{P(y_1|x)}{P(y_2|x)}$$

$$\sigma(a) = \frac{1}{1+\exp(-a)}$$

# Logistic Regression

For classification take a simple functional form for a, such as $\mathbf{w}^T\mathbf{x}$, thus:

$$P(y_1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}), \text{ where}$$

$\sigma(a)$ logistic function

$w$ model with $n = |x|$ parameters

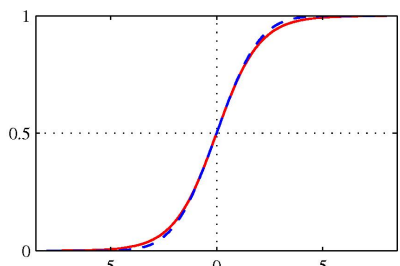



Figure: logistic sigmoid function in red

# Logistic Regression

**The model is assumed to be**

$$a = \ln\left(\frac{\sigma}{1-\sigma}\right) = \mathbf{w}^T\mathbf{x} = w_0 + w_1x_1 + w_2x_2 + \cdots + w_kx_k, \text{ where}$$

$w_i$ regression coefficient, which have to be estimated from data

Logistic uses the above model and transforms it into probabilities.

**Training the classifier using maximum likelihood**

$$\max_{\mathbf{w}} \sum_{i=1}^{m} \ln P(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = \max_{\mathbf{w}} \sum_{i=1}^{m} -\ln(1 + e^{y^{(i)}w^Tx^{(i)}})$$

Could be solved by gradient descent or second order methods.

# Evaluation

**Pro:**

- Can be seen as single-layer neural network used with sigmoidal
- Does not make any assumption about the distributions of the explanatory variables
- Weights indicate which features are important
- Output can be interpreted as probability

# Evaluation

**Con:**

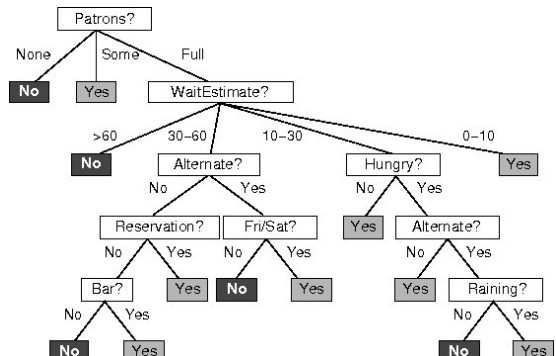
- For an n-dimensional feature space, this model requires adaption of n parameters
- Requires large training sets the more explanatory variables (feature dimension) to not overfit
- Slower convergence than Naive Bayes, but outperforms Naive Bayes, when many training examples are available
- Not online feasable
- Works with real-valued output, dummy variables for categorical features
- Estimation problems, when the explanatory variables are highly correlated

# Decision Trees

# Decision Trees

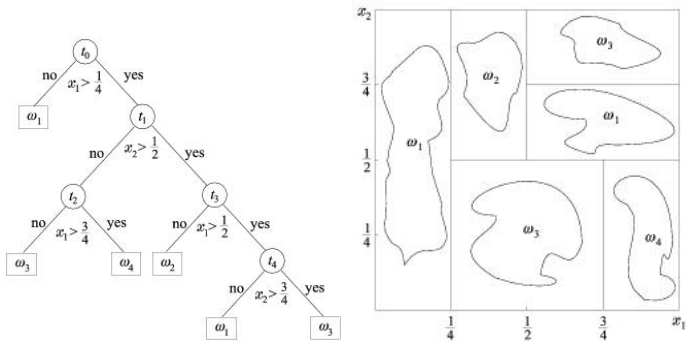
- Inductive, learning from example
- Non linear classifier
- Multistage decision system
- Symbolic (not probabilistic)

Restaurant Visiting Decision Tree



# Basic Understanding

- It can deal with discrete or continuous input values.
- The feature space is split into unique regions in a sequential manner.
- A leaf represents the final class label



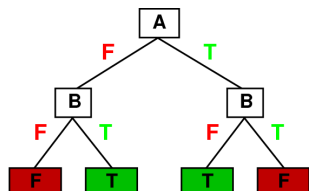

(a) Decision tree (b) Feature space

http://aispace.org/dTree/ http://www.aispace.org/bayes/index.shtml

# Expressiveness

Decision trees can express any Boolean function of the input attributes.
E.g., for Boolean functions, truth table row → path to leaf:

| A | B | A xor B |
|---|---|---|
| F | F | F |
| F | T | T |
| T | F | T |
| T | T | F |




Trivially, there is a consistent decision tree for any training set
→ one path to leaf for each example (unless $f$ nondeterministic in $x$)
Is it good!?

# Expressiveness

- A simple decision tree probably won't generalize to new examples
- E.g. for a set of all boolean function on $n$ attributes:
  $2^n$ (functions, i.e. number of leaves)
  It takes at least(!) $2^n$ bits to define such function.

  E.g. there are $2^{2^n} = 18{,}446{,}744{,}073{,}709{,}551{,}616$ distinct decision trees for 6 boolean variables.

# Building a Small Decision Tree

**Goal**

- Small decision tree
- Find common feature of instances
- Include generalization to allow classification of unseen data

**Problem**

- Finding the smallest decision tree is an NP-hard problem.

**Solution approach**

- Use heuristics

# General Principle: Information Gain and Impurity of Nodes

Consider a node in the decision tree and the associated examples:

- Looking at the root node all classes are mixed up, the node is "impure" requires more tests for classification
- Leaves that are pure (enough) can be used for classification

## Building a Good Decision Tree

1. Select root node
2. **If** node is pure (enough) or no more attributes to test
3. **Then** stop
4. **Else** construct subset of nodes using the rest of attributes

   Choose the attribute which is associated with the purest node
5. Recurse on each subnode

## Measuring Impurity

Different impurity functions $\phi$ exists, whereas the best known are:

### (A) Entropy H

$\phi = \sum_{i=1}^{K} -p_i \log_2 p_i$

where

- $\phi$ $\phi(\mathcal{D})$ - impurity of dataset $\mathcal{D}$
- $K$ number of distinct classes
- $p_i$ $P(\mathcal{D}, i)$ - probability of class $i$ in $\mathcal{D}$ (counting)

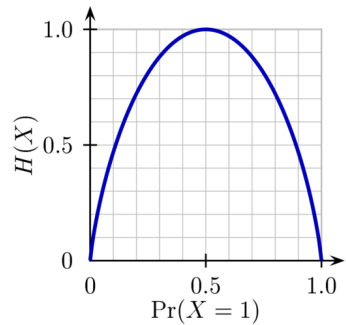



Figure: **Binary entropy function** $H_2(p, 1-p) \equiv p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$ with $0 \log \frac{1}{0} := 0$

# Measuring Impurity

Different impurity functions $\phi$ exists, whereas the best known are:

**(A) Entropy H**

$$\phi = \sum_{i=1}^{K} -p_i \log_2 p_i$$

where

- $\phi$ $\phi(\mathcal{D})$ - impurity of dataset $\mathcal{D}$
- K number of distinct classes
- $p_i$ $P(\mathcal{D}, i)$ - probability of class $i$ in $\mathcal{D}$ (counting)

Given:
$P(y_1) = 0.4$,
$P(y_2) = 0.3$, and
$P(y_3) = 0.3$

Result:
H(p) =
$-0.4 \log(0.4)$
$-0.3 \log(0.3)$
$-0.3 \log(0.3)$
$= 1.571$

## (B) Gini index G

$$\phi = \sum_{i=1}^{K} p_i(1 - p_i) = 1 - \sum_{i=1}^{K} p_i^2, \text{ where}$$

$G = 0$ meaning completely equal classes, $G = 1$ meaning completely unequal classes, e.g. 1 for a node with all classes

**Meaning:** How often is a randomly chosen element from the set incorrectly labelled
*('Chosen with probability $p_i$ and probability of assigning the wrong label $1 - p_i$')*
Example: Given that Prob (Bus)= 0.4, Prob (Car)= 0.3 and
Prob (Train)= 0.3, we can now compute Gini index as Gini Index= $1 - (0.4^2 + 0.3^2 + 0.3^2) = 0.660$

# Using the Information Gain with the Impurity measure

Using information gain to compute the splitting criterion, e.g. such as C4.5 - a well known algorithm to learn a decision-tree classifier:

$$\mathit{information\_gain}(\mathcal{D}, a) = \phi(\mathcal{D}) - \sum_{i=1}^{v_a} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \phi(\mathcal{D}_i)$$

where

- $D$ Set of samples for which the best splitting criterion is searched for
- $a$ attribute under investigation
- $v_a$ number of distinct values for a categorical attribute (or sections for a continuous attribute)
- $\mathcal{D}_i$ Subset of samples with the same attribute value (or in the same section)

# Decision Tree Overfitting/Pruning

## Overfitting

Applying greedy heuristics such as entropy has a drawback. The resulting decision trees overfit to training data and might not generalize well.

## Solution

Stop growing a branch during building the decision tree (pre-pruning) or cut branches off (post-pruning) according to some heuristics:

- Identify irrelevant attributes:
  - The resulting subsets have same proportions as the original set.
  - Information gain is close to zero, i.e., below a threshold
  - Apply a statistical significance test.

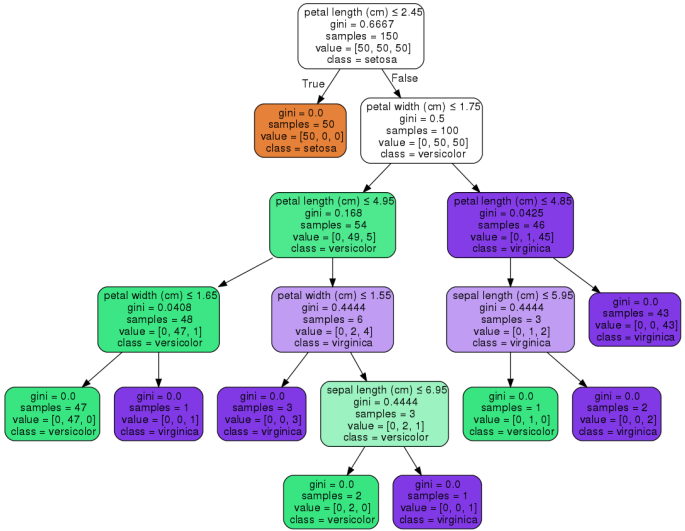
## Reduced Error pruning

Each decision node is a candidate. Use a test set and compute the expected misclassification from bottom up and compare it to a leaf that sets the class label to the most common class label. Prune where the error is highest and repeat the process until there is no error reduction possible.

# Decision Tree: Continuous Features

- *Brute force (default): For each occurring value of each attribute construct a decision threshold and choose the one with the best impurity gain.*
- Discretization: Discretize the continuous variables and treat them as categorical.
- Separation: Separate the relevant attribute values into groups and test either test for the group or build a threshold.
  The grouping can be made with equal distant intervals, by sorting attribute values and choosing equal sized groups or mapping a probability distribution and take quantiles or similar.

# Decision Tree: Iris Dataset




https://scikit-learn.org/stable/_images/iris.png

**Pro:**

- Classification without much computation, though training can be expensive
- Can handle continuous and categorical features and variables, though less appropriate to estimate a continuous variable
- Uses a white box model, i.e., provides an explanation with each classification (might be very long) - understandable for humans
- Performs well in many situations

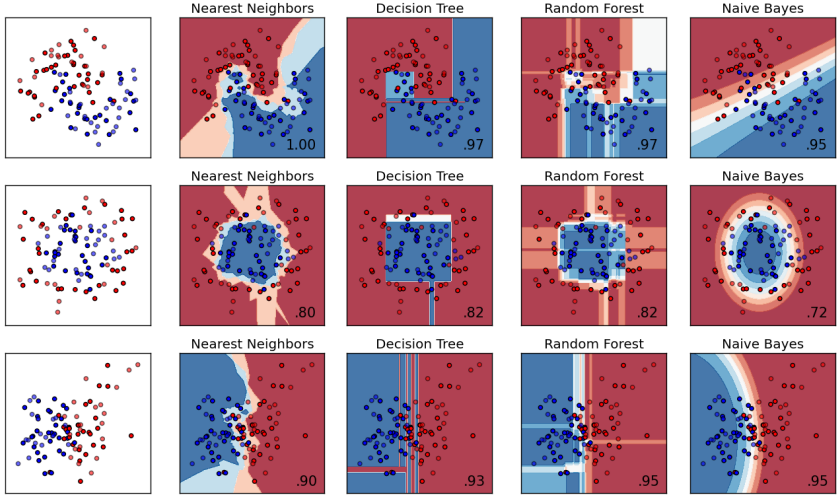
**Con:**

- Tend to overfit on training data
- Not online feasable, new data might change splitting attribute
- Large trees with large data sets
- But relies on heuristics for generating compact trees
- "Rectangular" decision boxes, though features's distribution might not correspond to this

# Random Forest

- Main idea: create a large set of decision trees in a randomized fashion to obtain more robust decision
- First step: choose random set of features during each attribute splitting in a tree (This can give more importance to correlated features.)
- Second step: create random training sets by selecting m samples with replacement – also called bagging
- Third step: repeat both repetitions and combine trees via majority vote
- Concept of random projections and bagging also used in other context
- Widely used powerful algorithm

# Conclusion: Classifier Comparison with Scikit-learn




(https://scikit-learn.org/0.15/_images/plot_classifier_comparison_0011.png, adapted)

# Optimality and Risk

Classification methods are not perfect, but wrong classification can have *different effects*. However, we want to make sure to make the best decision given the existing information.

Thus we will have to evaluate:

1. the **optimality** of the classifier
2. how to reduce the **risk of misclassification**, especially if misclassification will be *unevenly penalized*

Examples:

- Spam detection (Miss mail worse than getting spam)
- Flight security control (time vs. security vs. annoyed people)
- Disease diagnosis error (If not ill: panic and further treatment; else: dead due to no treatment)
- Detect a hole or cliff in front of a robot

## Bayes Classification (again)

Rewriting the Bayes decision to:

$$\frac{P(\mathbf{x}|y_1)P(y_1)}{P(\mathbf{x})} \lessgtr \frac{P(\mathbf{x}|y_2)P(y_2)}{P(\mathbf{x})}$$

If we assume a priori:

$$P(y_1) = P(y_2)$$

it allows us to simplify the decision, and finding the maximum error of the conditional probability density functions at $\mathbf{x}$:

$$P(\mathbf{x}|y_1) \lessgtr P(\mathbf{x}|y_2)$$

# Overlapping Likelihood (2 classes) & Decision Boundary

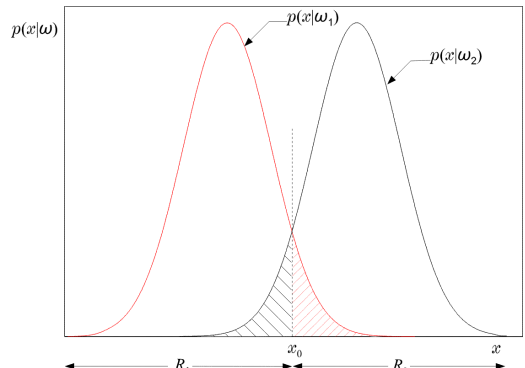



Figure: Overlapping likelihood for data **x** with respect to two equiprobable classes (here: $\omega_1$ and $\omega_2$) and a single feature

What is the classification error here!? The boundaries between the decision regions are called *decision*

# Optimality

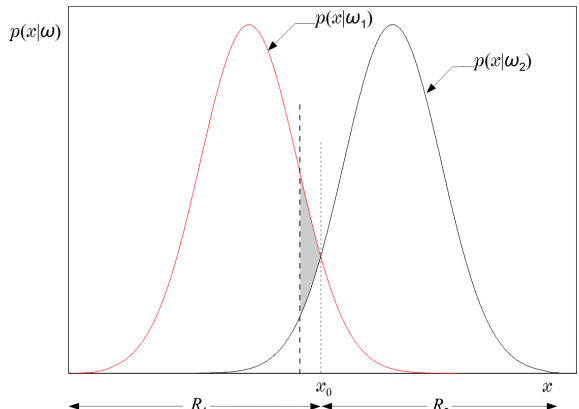



Figure: Increased classification error when threshold/decision boundary is moved

## Misclassification Minimization

For a set of 2 classes, the probability of mistake is defined as:

$$P(\textit{mistake}) = \int_{R_1} P(\mathbf{x}, y_2) d\mathbf{x} + \int_{R_2} P(\mathbf{x}, y_1) d\mathbf{x}$$

Minimize the probability of mistake by choosing the class with smaller integrand (see above formula).
Alternatively, maximize for a set of $M$ classes

$$P(\textit{success}) = \sum_{m=1}^{M} \int_{R_m} P(\mathbf{x}, y_m) d\mathbf{x}$$

by choosing the class for which $P(\mathbf{x}, y_m)$ is largest.

# Risk Minimization

Lets assume there is an unevenly distributed risk associated with a misclassification of two classes. Then the use of only the classification error probability is not sufficient.
Lets define a loss matrix:

$$L = \begin{pmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{pmatrix}, \text{ where}$$

$\lambda_{ij}$ penalty term for classifying a class $y_i$ as $y_j$

*Well, do you already have an idea how to formulate the risk !?*

# Risk Minimization

Risk for classification of class $y_1$:

$$r_1 = \lambda_{11} \int_{R_1} P(\mathbf{x}|y_1) d\mathbf{x} + \lambda_{12} \int_{R_2} P(\mathbf{x}|y_1) d\mathbf{x}$$

or in general for a class $y_k$ out of $M$ classes, the generally associated risk:

$$r_k = \sum_{i=1}^{M} \lambda_{ki} \int_{R_i} P(\mathbf{x}|y_k) d\mathbf{x}, \text{ where}$$

$\lambda_{ki}$ penalty term for classifying a class $y_k$ as $y_i$
$R_i$ region of feature space assigned to class $y_i$

Note: Here the integral defines the overall probability of a feature vector of class $y_k$ being classified as $y_i$

## Risk Minimization

In order to minimize the average risk, we have to set the regions boundaries to minimize:

$$r = \sum_{k=1}^{M} r_k P(y_k) = \sum_{i=1}^{M} \int_{R_i} (\sum_{k=1}^{M} \lambda_{ki} P(\mathbf{x}|y_k) P(y_k)) d\mathbf{x}$$

Thus for our feature vector $\mathbf{x} \epsilon\ R_i$, we have to select partitions, so that $\mathbf{x} \in R_i$:

$$\text{if } l_i < l_j \text{ , } \forall\ i \neq j$$

$$l_n \equiv \sum_{k=1}^{M} \lambda_{kn} P(\mathbf{x}|y_k) P(y_k)$$

# Appendix

# References: Books


- Artificial Intelligence - A modern Approach, Stuart Russel and Peter Norvig, Prentice Hall, 2003
- Machine Learning and Pattern Recognition, Christopher M. Bishop, Springer, 2006
- Machine Learning, Tom Mitchell, McGraw Hill, 1997
- Pattern Recognition 2nd Edition, Sergios Theodoridis and Konstantinos Koutroumbas, Elsevier, 2003
- Information Theory, Inference, and Learning Algorithms, David J.C. MacKay, Cambridge University Press, 2003
- Classification and Regression Trees, Breiman, Friedman, Olshen, Stone, CRC Press, 1984

# References: Links


- Mengjie Zhang (my former lecturer), http://ecs.victoria.ac.nz/Courses/COMP307_2012T1/LectureSchedule, 2012
- Kardi Teknomo, How to Measure Impurity, http://people.revoledu.com/kardi/tutorial/DecisionTree/how-to-measure-impurity.htm
- Kevin Murphy, Classification, http://people.cs.ubc.ca/~murphyk/Teaching/Stat406-Spring07/reading/genClassifHandout.pdf Logistic Regression, http://www.resample.com/xlminer/help/Lreg/lreg_intro.htm
- Data sets, UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/
- Decision trees, http://www.decisiontrees.net
- Improved use of Continuous Attributes in C4.5, J.R. Quinlan, Journal of Artificial Intelligence Research, 1996

# Building a decision tree

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Figure: Decision tree learning

# Nomenclature

- $n$-dimensional feature vectors $x \in \mathcal{X}$, where $x_i$ is the value of $x$ for the $i$-th feature
- Class $y \in \{-1, 1\}$
- Example set with $m$ elements (unsupervised learning): $\mathcal{D} \in \mathcal{X}^m$, as matrix $X \in \mathbb{R}^{m \times n}$
- Example set with $m$ elements (supervised learning, binary classification): $\mathcal{D} \in (\mathcal{X} \times \{-1, 1\})^m$
- An element $d \in \mathcal{D}$ is called an instance

# Nomenclature

## Training set

$(\mathbf{x}^{(i)}, y^{(i)})$, where

| | |
|---|---|
| i | 1, . . . , m |
| m | size of training set |
| **x** | input vector (features values), $\mathbf{x} \in \mathcal{X} = \mathbb{R}^n$, where $x_n$ is the value of $x$ for the $n$-th feature |
| $y$ | output (discrete here, i.e. class label) |
| n | dimension |

*Btw (By the way):* if the output is not discrete, then we have a regression task here

# The Classification Task

**Given:**

- a training set T with m labeled instances

$$(\mathbf{x}^{(i)}, y^{(i)}); i = 1, \ldots, m$$

- a test set U, with unlabeled instances

$$\mathbf{x}^{(i)}; i > m$$

**Task:**

Find the best label for the instances in the test set, based on the knowledge one can extract from the training set!

# Further Impurity Measures

## (C) Misclassification rate

$$\phi = 1 - \max_i p_i$$

## (D) Twoing rule (Directly rank after binary splitting)

$$\phi = \frac{|\mathcal{D}_L||\mathcal{D}_R|}{n^2} \left( \sum_{i=1}^{k} \left| \frac{L_i}{|\mathcal{D}_L|} - \frac{R_i}{|\mathcal{D}_R|} \right| \right)^2, \text{ where}$$

| | |
|---|---|
| n | number of examples |
| $L_i, R_i$ | number of instances of class i on the left and on the right of a split |
| $\mathcal{D}_R, \mathcal{D}_L$ | two non overlapping subsets (left and right) that result from a split |