IB031 Úvod do strojového učení

Tomáš Brázdil

Course Info

Resources:

- Lectures & tutorials (the **main** source)
- Many books, few perfect for introductory level
  One relatively good, especially the first part:
  A. Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media; 3rd edition, 2022
- (Almost) infinitely many online courses, tutorials, materials, etc.

Evaluation

The evaluation is composed of three parts:

- Mid-term exam: Written exam from the material of the first half of the semester.
- End-term exam: The "big" one containing everything from the semester (with possibly more stress in the second half).
- Projects: During tutorials, you will work on larger projects (in pairs).

Each part contributes the following number of points:

- Mid-term exam: 25
- End-term exam: 50
- Project: 25

To pass, you need to obtain at least 60 points.

Distinguishing Properties of the Course

- Introductory, prerequisites are held to a minimum
- Formal and precise: Be prepared for a complete and "mathematical" description of presented methods.

Distinguishing Properties of the Course

- Introductory, prerequisites are held to a minimum
- Formal and precise: Be prepared for a complete and "mathematical" description of presented methods.

I assume that you have basic knowledge of

- Elementary understanding of mathematical notation (operations on sets, logic, etc.)
- Linear algebra: Vectors in $\mathbb{R}^n$, operations on vectors (including the dot product). Geometric interpretation!
- Calculus: Functions of multiple real variables, partial derivatives, basic differential calculus.
- Probability: Notion of probability distribution, random variables/vectors, expectation.

What Is Machine Learning?

Machine learning is the science (and art) of programming computers so they can learn from data.

What Is Machine Learning?

Machine learning is the science (and art) of programming computers so they can learn from data.

Here is a slightly more general definition:

Arthur Samuel, 1959

Machine learning is the field of study that allows computers to learn without being explicitly programmed.

What Is Machine Learning?

Machine learning is the science (and art) of programming computers so they can learn from data.

Here is a slightly more general definition:

Arthur Samuel, 1959

Machine learning is the field of study that allows computers to learn without being explicitly programmed.

And a more engineering-oriented one:

Tom Mitchell, 1997

A computer program is said to learn from experience E concerning some task T and some performance measure P if its performance on T, as measured by P, improves with experience E.

Example

In the context of spam filtering:

- The task T is to flag spam in new emails.
- The experience E is represented by a set of emails labeled either spam or ham by hand (the training data).
- The performance measure P could be the accuracy, which is the ratio of the number of correctly classified emails and all emails.

  There are many more performance measures; we will study the basic ones later.

Example

In the context of spam filtering:

- The task T is to flag spam in new emails.
- The experience E is represented by a set of emails labeled either spam or ham by hand (the training data).
- The performance measure P could be the accuracy, which is the ratio of the number of correctly classified emails and all emails.

  There are many more performance measures; we will study the basic ones later.

In the context of housing price prediction:

- The task T is to predict prices of new houses based on their basic parameters (size, number of bathrooms, etc.)
- The experience E is represented by information about existing houses.
- The performance measure P could be, e.g., an absolute difference between the predicted and real price.

Examples (cont.)

In the context of game playing:

- The task T is to play chess.
- The experience E is represented by a series of self-plays where the computer plays against itself.
- The performance measure P is winning/losing the game.

  Here, the trick is to spread the delayed and limited feedback about the result of the game throughout the individual decisions in the game.

Examples (cont.)

In the context of game playing:

- The task T is to play chess.
- The experience E is represented by a series of self-plays where the computer plays against itself.
- The performance measure P is winning/losing the game.
  Here, the trick is to spread the delayed and limited feedback about the result of the game throughout the individual decisions in the game.

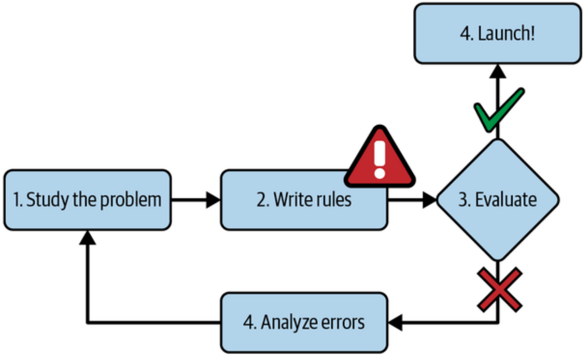
In the context of customer behavior:

- The task T is to group customers with similar shopping habits in an e-shop.
- The experience E consists of lists of items individual customers bought in the shop.
- The performance measure P?
  Measure how "nicely" the customers are grouped.
  (whether people with similar habits, as seen by humans, fall into the same group).

Comparison of Programming and Learning

How to code the spam filter?

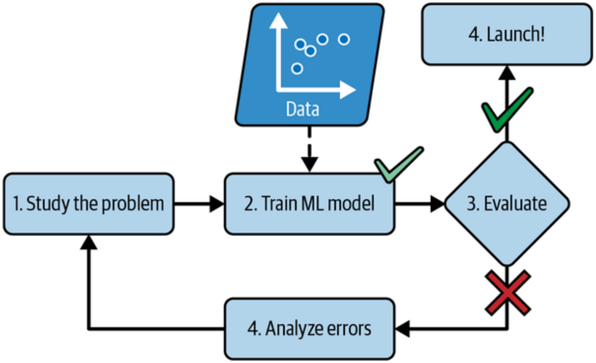
- Examine what spam mails typically contain: Specific words ("Viagra"), sender's address, etc.
- Write down a rule-based system that detects specific features.
- Test the program on new emails and (most probably) go back to look for more spam features.



Comparison of Programming and Learning

The machine learning way:

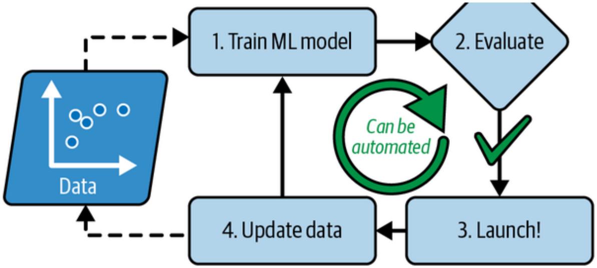
- Study the problem and collect lots of emails, labeling them spam or ham.
- Train a machine learning model that reads an email and decides whether it's spam or ham.
- Test the model and (most probably) go back to collect more data and adjust the model.



ML Solutions are Adaptive

Spam filter: Authors of spam might and will adapt to your spam filter (possibly change the wording to pass through).

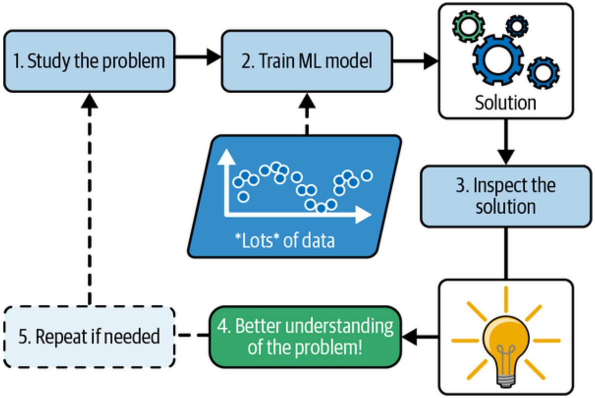
ML systems can be adjusted to new situations by retraining on new data (unless the data becomes ugly).



ML for Human Understanding

Spam filter: A trained system can be inspected for notorious spam features.

Some models allow direct inspection, such as decision trees or linear/logistic regression models.



Usage of Machine Learning

Machine learning suits various applications, especially where traditional methods fall short. Here are some areas where it excels:

- Solving complex problems where fine-tuning and rule-based solutions are inadequate.
- Tackling complex issues that resist traditional problem-solving approaches.
- Adapting to fluctuating environments through retraining on new data.
- Gaining insights from large and complex datasets.

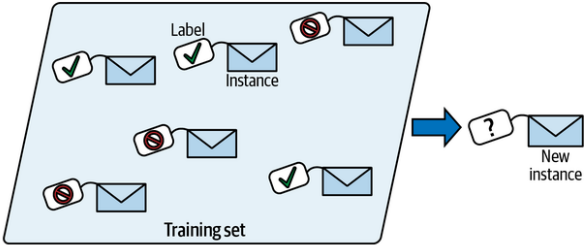
In summary, machine learning offers innovative solutions and adaptability for today's complex and ever-changing problems, (sometimes) providing insights beyond the reach of traditional approaches.

Types of Learning

There are main categories based on information available during the training:

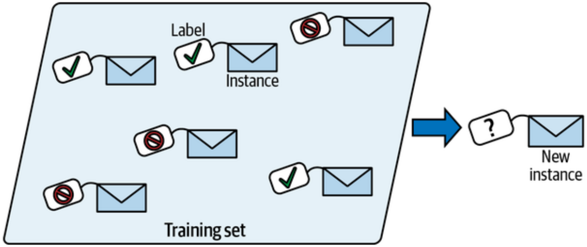
- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Self-supervised learning
- Reinforcement learning

Supervised Learning



Labels are available for all input data.

Supervised Learning

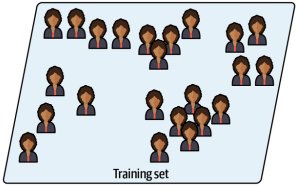


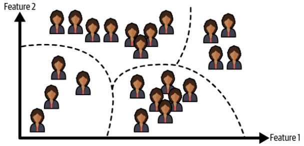
Labels are available for all input data.

Typical supervised learning tasks are

- *Classification* where the aim is to classify inputs into (typically few) classes
  (e.g., the spam filter where the classes are spam/ham)
- *Regression* where a numerical value is output for a given input
  (e.g., housing prices)

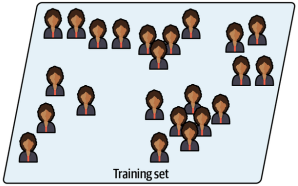
Unsupervised Learning

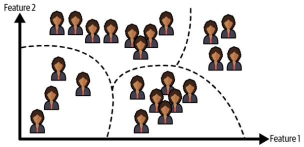




No labels are available for input data.

Unsupervised Learning



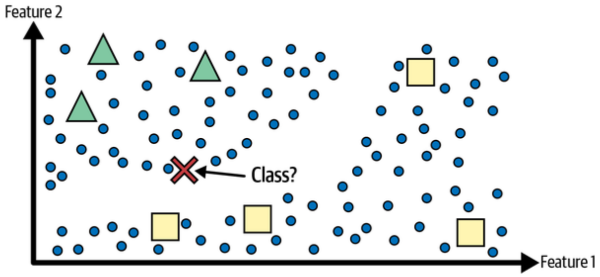


No labels are available for input data.

Typical unsupervised learning tasks are

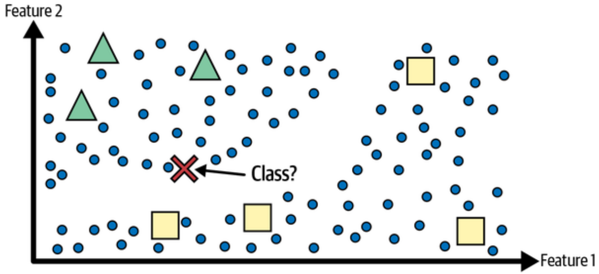
- *Clustering* where inputs are grouped according to their features
  (e.g., clients of a bank grouped according to their age, wealth, etc.)
- *Association* where interesting relations and rules are discovered among the features of inputs
  (e.g., market basket mining where associations between various types of goods are being learned from the behavior of customers)
- *Dimensionality reduction* reduce high-dimensional data to few dimensions (e.g., images to few image features)

Semi-Supervised Learning



Labels for some data.

Semi-Supervised Learning

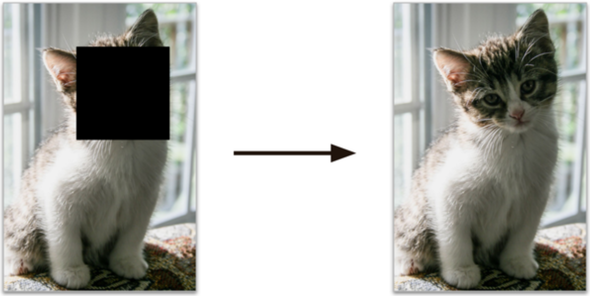


Labels for some data.

For example, Medical data, where elaborate diagnosis is available only for some patients.

Combines supervised and unsupervised learning: e.g., clusters all data and labels the unlabeled inputs with the most common labels in their clusters.

Self-Supervised Learning

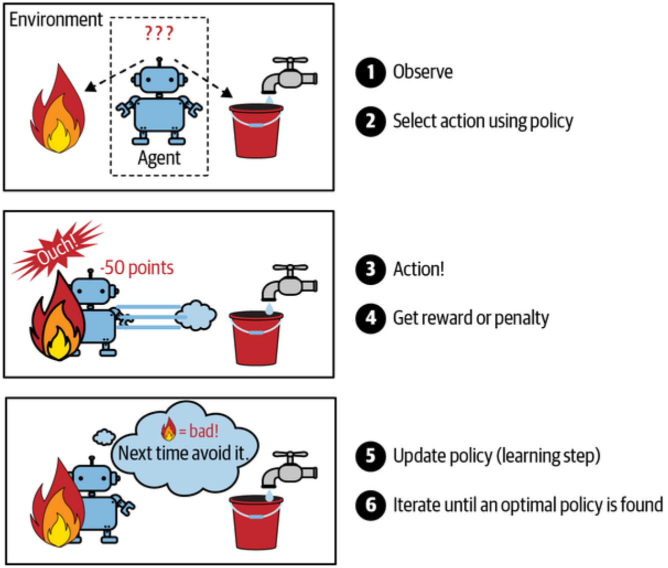


Generate labels from (unlabeled) inputs.

The goal is to learn typical features of the data.

It can be later modified to generate images, classify, etc.

Reinforcement Learning


Environment
???
Agent
1 Observe
2 Select action using policy
Ouch!
-50 points
3 Action!
4 Get reward or penalty
= bad!
Next time avoid it.
5 Update policy (learning step)
6 Iterate until an optimal policy is found

Learn from performing *actions* and getting feedback from *environment*.

ML Applications Highlights

- ChatGPT (and similar generative models)
  - The basis forms a generative language model, i.e., a text-generating model trained on texts in a self-supervised way
  - Currently extended to multimodal versions (text, image, sound)

ML Applications Highlights

- ChatGPT (and similar generative models)
  - The basis forms a generative language model, i.e., a text-generating model trained on texts in a self-supervised way
  - Currently extended to multimodal versions (text, image, sound)
- Machine translation, image captioning
  - Google translate, etc.
  - Typically (semi)-supervised learning,

ML Applications Highlights

- ChatGPT (and similar generative models)
  - The basis forms a generative language model, i.e., a text-generating model trained on texts in a self-supervised way
  - Currently extended to multimodal versions (text, image, sound)
- Machine translation, image captioning
  - Google translate, etc.
  - Typically (semi)-supervised learning,
- Various image recognition and processing tasks
  - In medicine where it is slowly making its way into hospitals as assistance tools
  - Automotive, advertising, quality control etc., etc., etc.

ML Applications Highlights

- ChatGPT (and similar generative models)
  - The basis forms a generative language model, i.e., a text-generating model trained on texts in a self-supervised way
  - Currently extended to multimodal versions (text, image, sound)
- Machine translation, image captioning
  - Google translate, etc.
  - Typically (semi)-supervised learning,
- Various image recognition and processing tasks
  - In medicine where it is slowly making its way into hospitals as assistance tools
  - Automotive, advertising, quality control etc., etc., etc.
- Science
  - Chemistry & biology: E.g., prediction of features of chemical compounds (Alpha-fold)

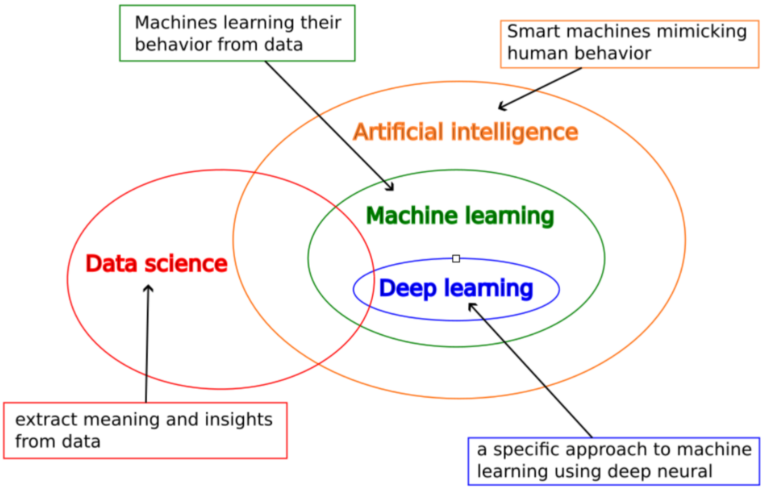
ML Applications Highlights

- ChatGPT (and similar generative models)
  - The basis forms a generative language model, i.e., a text-generating model trained on texts in a self-supervised way
  - Currently extended to multimodal versions (text, image, sound)
- Machine translation, image captioning
  - Google translate, etc.
  - Typically (semi)-supervised learning,
- Various image recognition and processing tasks
  - In medicine where it is slowly making its way into hospitals as assistance tools
  - Automotive, advertising, quality control etc., etc., etc.
- Science
  - Chemistry & biology: E.g., prediction of features of chemical compounds (Alpha-fold)
- Various "table" data processing in finance, management, etc.
  - Often straightforward methods (linear/logistic regression)
  - Essential but not fancy

ML Applications Highlights

- ChatGPT (and similar generative models)
  - The basis forms a generative language model, i.e., a text-generating model trained on texts in a self-supervised way
  - Currently extended to multimodal versions (text, image, sound)
- Machine translation, image captioning
  - Google translate, etc.
  - Typically (semi)-supervised learning,
- Various image recognition and processing tasks
  - In medicine where it is slowly making its way into hospitals as assistance tools
  - Automotive, advertising, quality control etc., etc., etc.
- Science
  - Chemistry & biology: E.g., prediction of features of chemical compounds (Alpha-fold)
- Various "table" data processing in finance, management, etc.
  - Often straightforward methods (linear/logistic regression)
  - Essential but not fancy
- Game playing: More fancy than useful, learning models beating humans in several difficult games.

ML in Context


Machines learning their behavior from data
Smart machines mimicking human behavior
Artificial intelligence
Machine learning
Data science
Deep learning
extract meaning and insights from data
a specific approach to machine learning using deep neural

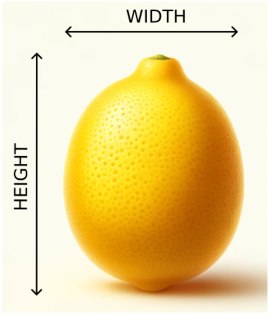
Supervised Learning

Example - Fruit Recognition

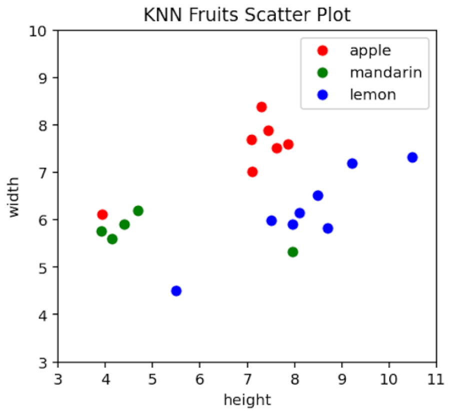
The goal: Create an automatic system for fruit recognition, concretely apple, lemon, and mandarin.

Inputs: Measures of *height* and *width* of each fruit.

Suppose we have a dataset of dimensions of several fruits labeled with the correct class.



Data



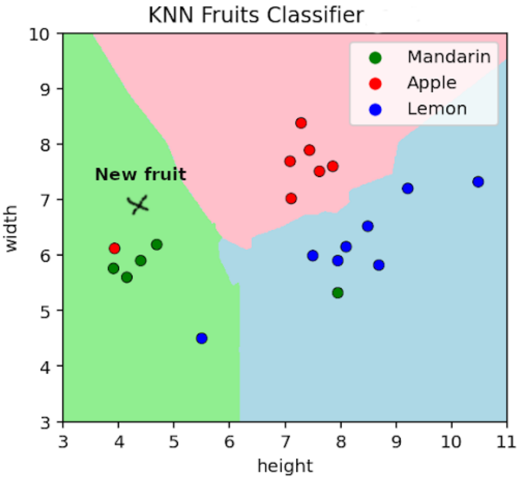
Use similarity to solve the problem.

	height	width	fruit
0	3.91	5.76	Mandarin
1	7.09	7.69	Apple
2	10.48	7.32	Lemon
3	9.21	7.20	Lemon
4	7.95	5.90	Lemon
5	7.62	7.51	Apple
6	7.95	5.32	Mandarin
7	4.69	6.19	Mandarin
8	7.50	5.99	Lemon
9	7.11	7.02	Apple
10	4.15	5.60	Mandarin
11	7.29	8.38	Apple
12	8.49	6.52	Lemon
13	7.44	7.89	Apple
14	7.86	7.60	Apple
15	3.93	6.12	Apple
16	4.40	5.90	Mandarin
17	5.50	4.50	Lemon
18	8.10	6.15	Lemon
19	8.69	5.82	Lemon

KNN Classification

Given a new fruit.
What is it?

Find five closest examples



Where is the machine learning?

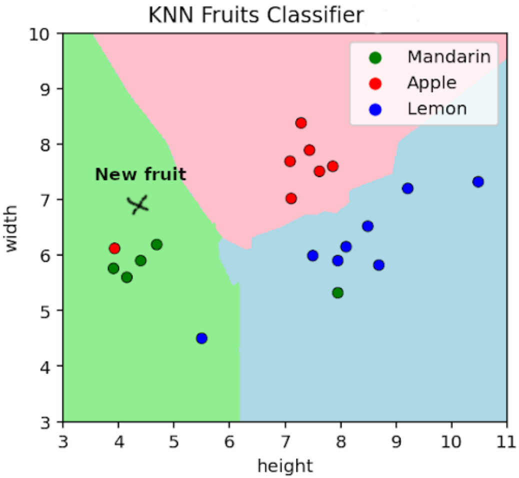
KNN Classification

Given a new fruit.
What is it?

Find five closest examples

Among the five closest:

- $M = 4$ mandarins
- $A = 1$ apples
- $L = 0$ lemons



Where is the machine learning?

KNN Classification

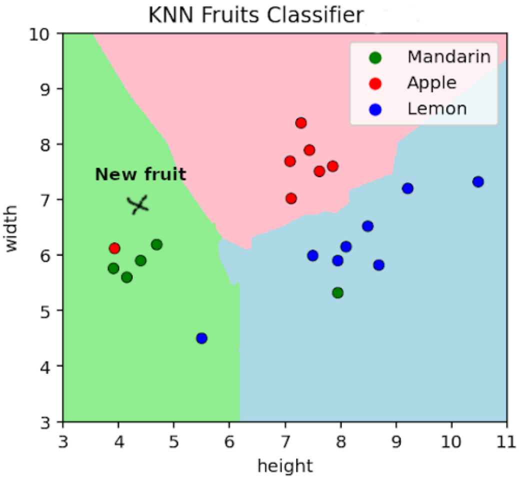
Given a new fruit.
What is it?

Find five closest examples

Among the five closest:

- $M = 4$ mandarins
- $A = 1$ apples
- $L = 0$ lemons

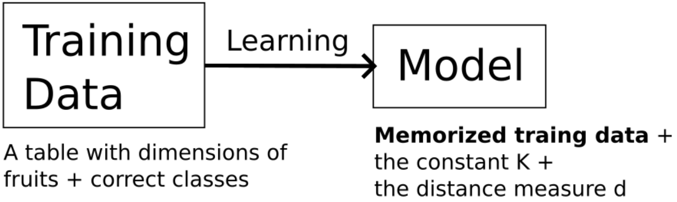
It is a **mandarin**!



Where is the machine learning?

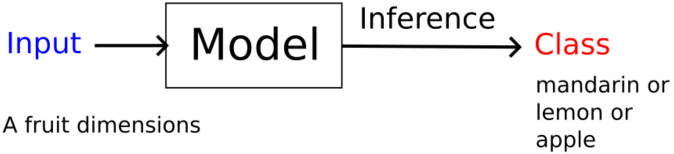
Learning in Fruit Classification with KNN

Learning:



Training Data
Learning
Model
A table with dimensions of fruits + correct classes
Memorized traing data +
the constant K +
the distance measure d

Inference:



Fruit Classification Algorithm

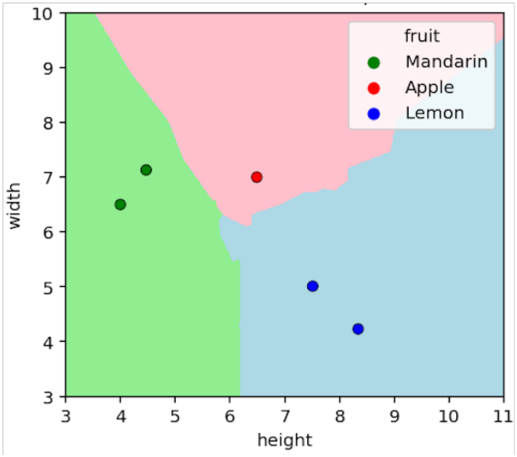
```
Input: A fruit $F$ with dimensions height, width
Output: mandarin, lemon, apple
1: Find $K$ examples $\{E_1, \ldots, E_K\}$ in the dataset whose
   dimensions are closest to the dimensions of the fruit $F$
2: Count the number of examples of each class in $\{E_1, \ldots, E_K\}$
           $M$ mandarins in $\{E_1, \ldots, E_K\}$
           $L$ lemons in $\{E_1, \ldots, E_K\}$
           $A$ apples in $\{E_1, \ldots, E_K\}$
3: if $M \geq L$ and $M \geq A$ then return mandarin
4: else if $L \geq A$ then return lemon
5: else return apple
6: end if
```

Does it work?

Testing the Model for Fruit Classification

Consider a test set of new instances ($K = 5$, d is Euclidean):

height	**width**	**fruit**
4.0	6.5	Mandarin
4.47	7.13	Mandarin
6.49	7.0	Apple
7.51	5.01	Lemon
8.34	4.23	Lemon



Perfect classification of new data! Just deploy and sell!!

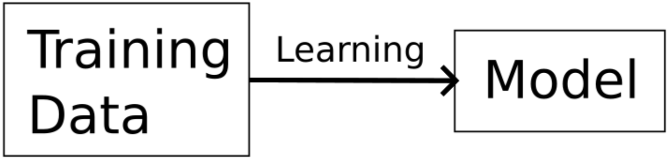
K Nearest Neighbors

…on ideal data

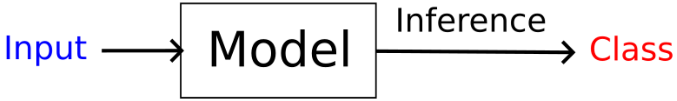
Learning and Inference

Two crucial components of machine learning are the following:

Learning:
Creating model



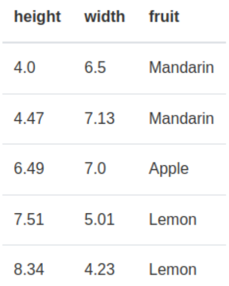
Inference:
Using model



Training Data

Assume table training data, i.e., of the form

$$\begin{array}{cccc|c} x_{11} & x_{12} & \cdots & x_{1n} & c_1 \\ x_{21} & x_{22} & \cdots & x_{2n} & c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{p1} & x_{p2} & \cdots & x_{pn} & c_p \end{array}$$



height	width	fruit
4.0	6.5	Mandarin
4.47	7.13	Mandarin
6.49	7.0	Apple
7.51	5.01	Lemon
8.34	4.23	Lemon

Formally, we define **training dataset**

$$\mathcal{T} = \{(\vec{x}_k, c_k) \mid k = 1, \ldots, p\}$$

Here each $\vec{x}_k \in \mathbb{R}^n$ is an input vector and $c_k \in C$ is the correct class.

$$\begin{aligned} \mathcal{T} = \{&(4.0, 6.5), M), \\ &(4.47, 7.13), M), \\ &(6.49, 7.0), A), \\ &\ldots\} \end{aligned}$$

KNN: Learning

Consider the training set:

$$\mathcal{T} = \{(\vec{x}_k, c_k) \mid k = 1, \ldots, p\}$$

and *memorize it* exactly as it is.

Store in a table.

Possibly use a clever representation allowing fast computation of nearest neighbors such as KDTrees (out of the scope of this lecture).

Also,

- determine the number of neighbors $K \in \mathbb{N}$,
- and the distance measure d.

Inference in KNN

Assume a KNN "trained" by memorizing $\mathcal{T} = \{(\vec{x}_k, c_k) \in \mathbb{R}^n \times C \mid k = 1, \ldots, p\}$, a constant $K \in \mathbb{N}$ and a distance measure d.

For d, consider Euclidean distance, but different norms may also be used to define different distance measures.

Inference in KNN

Assume a KNN "trained" by memorizing $\mathcal{T} = \{(\vec{x}_k, c_k) \in \mathbb{R}^n \times C \mid k = 1, \ldots, p\}$, a constant $K \in \mathbb{N}$ and a distance measure d.

For d, consider Euclidean distance, but different norms may also be used to define different distance measures.

Input: A vector $\vec{z} = (z_1, \ldots, z_n) \in \mathbb{R}^n$
Output: A class from C

```
1: Find $K$ indices of examples $X = \{i_1, \ldots, i_K\} \subseteq \{1, \ldots, p\}$ with minimum distance to $\vec{z}$, i.e., satisfying
      $\max\{d(\vec{z}, \vec{x}_\ell) \mid \ell \in X\} \leq \min\{d(\vec{z}, \vec{x}_\ell) \mid \ell \in \{1, \ldots, p\} \setminus X\}$
2: For every $c \in C$ count the number $\#c$ of elements $\ell$ in $X$ such that $c_\ell = c$
3: Return some
      $c_{max} \in \arg\max_{c \in C} \#c$
   A class $c_{max} \in C$ which maximizes $\#c$.
```

The resulting model

What exactly constitutes the model? The *model* consists of

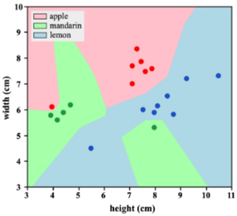
- The *trained parameters*: In this case the memorized training data.
- The *hyperparameters* set “from the outside”: In this case, the number of neighbors K and the distance measure d.

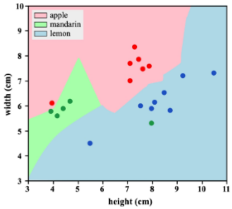
The resulting model

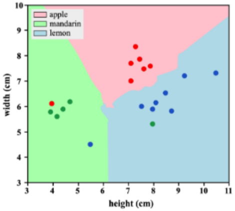
What exactly constitutes the model? The *model* consists of

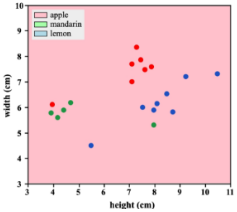
- The *trained parameters*: In this case the memorized training data.
- The *hyperparameters* set “from the outside”: In this case, the number of neighbors K and the distance measure d.

Note that different settings of K lead to different classifiers (for the same d):









In Practice

…to get an efficient solution:

In Practice

…to get an efficient solution:

- Deal with issues in the data
  - Data almost always comes in weird formats, with inconsistencies, missing values, wrong values, etc.
  - Data rarely have the ideal form for a given learning model.

  We need to ingest, validate, and preprocess the data.

In Practice

. . . to get an efficient solution:

- Deal with issues in the data
  - Data almost always comes in weird formats, with inconsistencies, missing values, wrong values, etc.
  - Data rarely have the ideal form for a given learning model.

  We need to ingest, validate, and preprocess the data.
- Deal with issues in the model
  - In KNN, the training memorizes the example, but at least the K can be tuned.

  We need to tune the model.

In Practice

…to get an efficient solution:

- Deal with issues in the data
  - Data almost always comes in weird formats, with inconsistencies, missing values, wrong values, etc.
  - Data rarely have the ideal form for a given learning model.

  We need to ingest, validate, and preprocess the data.
- Deal with issues in the model
  - In KNN, the training memorizes the example, but at least the K can be tuned.

  We need to tune the model.
- Deal with the wrong model by testing and validation in as realistic conditions as possible.

In Practice

…to get an efficient solution:

- Deal with issues in the data
  - Data almost always comes in weird formats, with inconsistencies, missing values, wrong values, etc.
  - Data rarely have the ideal form for a given learning model.

  We need to ingest, validate, and preprocess the data.
- Deal with issues in the model
  - In KNN, the training memorizes the example, but at least the K can be tuned.

  We need to tune the model.
- Deal with the wrong model by testing and validation in as realistic conditions as possible.
- Deal with deployment - real-world application issues involving, e.g., implementation in embedded devices with limited resources.

Models Considered in This Course

Throughout this course, we will meet the following models:

- KNN (already did)
- Decision trees
- (Naive) Bayes classifier
- Clustering: K-means and hierarchical
- Linear and logistic regression
- Support Vector Machines (SVM)
- Kernel linear models
- Neural networks (light intro to feed-forward networks)
- Ensemble methods + random forests
- (maybe some reinforcement learning)

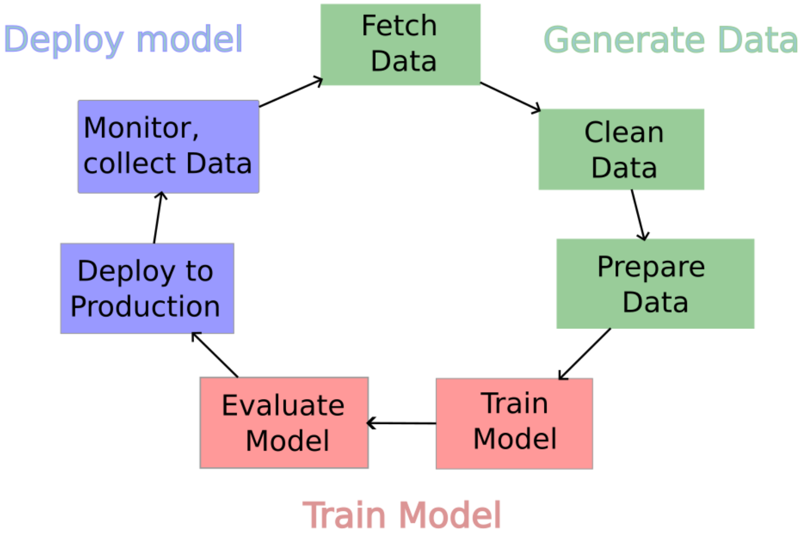
Models Considered in This Course

Throughout this course, we will meet the following models:

- KNN (already did)
- Decision trees
- (Naive) Bayes classifier
- Clustering: K-means and hierarchical
- Linear and logistic regression
- Support Vector Machines (SVM)
- Kernel linear models
- Neural networks (light intro to feed-forward networks)
- Ensemble methods + random forests
- (maybe some reinforcement learning)

. . . but first, let us see the whole machine learning pipeline.

Machine Learning Pipeline



Fetch Data

Always start with

Fetch Data

Always start with

- The problem formulation & understanding.
  For example, diagnosis of diabetes from medical records. What info is (possibly) sufficient for such a diagnosis?

Fetch Data

Always start with

- The problem formulation & understanding.

  For example, diagnosis of diabetes from medical records. What info is (possibly) sufficient for such a diagnosis?

- Find data sources.

  In our example, the sources are hospitals. It would be best to persuade them to give you the data and sign a contract.

Fetch Data

Always start with

- The problem formulation & understanding.
  For example, diagnosis of diabetes from medical records. What info is (possibly) sufficient for such a diagnosis?
- Find data sources.
  In our example, the sources are hospitals. It would be best to persuade them to give you the data and sign a contract.
- Collect the data.
  In our example, the data is possibly small (just tables with results of tests). But for other diagnoses, you may include huge amounts of data from MRI, CT, etc. Then, the collection itself might be a serious technical problem.

Fetch Data

Always start with

- The problem formulation & understanding.
  For example, diagnosis of diabetes from medical records. What info is (possibly) sufficient for such a diagnosis?
- Find data sources.
  In our example, the sources are hospitals. It would be best to persuade them to give you the data and sign a contract.
- Collect the data.
  In our example, the data is possibly small (just tables with results of tests). But for other diagnoses, you may include huge amounts of data from MRI, CT, etc. Then, the collection itself might be a serious technical problem.
- Integrate data from various sources.
  A serious diagnostic system must be trained/tested on data from many hospitals. You must blend the data from various sources (different formats, etc.).

Fetch Data

For simple “toy” machine learning projects, you may fetch prepared datasets from various databases on the internet.

Fetch Data

For simple “toy” machine learning projects, you may fetch prepared datasets from various databases on the internet.

The data should be stored in an identified location and versioned.
You will probably keep adding data and training models on the ever-changing datasets. You have to be able to keep track of the changes and map training data to particular models.

Tools such as ML Flow or Weights & Biases might be helpful.

Fetch Data

For simple "toy" machine learning projects, you may fetch prepared datasets from various databases on the internet.

The data should be stored in an identified location and versioned.
You will probably keep adding data and training models on the ever-changing datasets. You have to be able to keep track of the changes and map training data to particular models.

Tools such as ML Flow or Weights & Biases might be helpful.

Data Separation
At this point, you should randomize the ordering of the data and select a test set to be used in model evaluation!

The test data are supposed to simulate the actual conditions, i.e., they should be "unseen".

Fetch Data

For simple “toy” machine learning projects, you may fetch prepared datasets from various databases on the internet.

The data should be stored in an identified location and versioned.
You will probably keep adding data and training models on the ever-changing datasets. You have to be able to keep track of the changes and map training data to particular models.

Tools such as ML Flow or Weights & Biases might be helpful.

Data Separation
At this point, you should randomize the ordering of the data and select a test set to be used in model evaluation!

The test data are supposed to simulate the actual conditions, i.e., they should be “unseen”.

Data Exploration
Compute basic statistics to identify missing values, outliers, etc.

Clean Data

The cleaning usually comprises the following steps:

- Fix or remove incorrect or corrupted values.
- Identify outliers and decide what to do with them.

  Outliers may harm some training methods and are not "representative". However, sometimes, they naturally belong to the dataset, and expert insight is needed.
- Fix formatting.

  For example, the Date may be expressed in many ways, and a simple Yes/No answer.
- Resolve missing values (by either removing the whole examples or imputing)

  Many methods have been developed for missing values imputation. It is a susceptible issue because new values may strongly bias the model.
- Remove duplicates.

The above steps often affect the training and need expertise in the application domain.

Later in this course, we will discuss techniques for data cleaning.

ID	Age	Income	Gender	Customer_Satisfaction
1	38	46641.356413713	nan	Unsatisfied
2	42	49129.0615585107	female	Neutral
3	18	119965.049731014	Male	nan
4	18	66828.0762224329	nan	very unsatisfied
5	58	57422.2721106762	female	very unsatisfied
6	28	59502.8174855665	Other	Satisfied
7	18	42659.6675768587	Other	Neutral
8	18	54019.1173206374	Other	Satisfied
9	40	25429.1604541137	female	Unsatisfied
10	21	15595.5862129548	Other	Satisfied
11	18	58094.2328460069	Other	very unsatisfied
12	18	39097.3278583155	female	Very Satisfied
13	30		Other	Satisfied
14	50	30617.3914472273	Female	Very Satisfied
15	18		nan	Neutral
16	34	39902.4430953214	male	nan
17	49	68381.6997683133	Female	Very Satisfied
18	33	44796.0962271524	Other	Very Satisfied
19	47	39218.9560738814	Female	very unsatisfied
20		14544.9226784447	Other	Satisfied

Prepare Data

Unlike cleaning, which is application-dependent, data preparation/transformation is model-dependent. This usually subsumes:

- **Scaling**: Settings values of inputs to a similar range.

  Some models, especially those utilizing distance, are sensitive to large differences between input sizes.

- **Encoding**: Encode non-numeric data using real-valued vectors.

  Many models, especially those based on geometry, work only with numeric data. Non-numeric data such as Yes/No, Short/Medium/Long must be encoded appropriately.

- **Binning or Discretization** Convert continuous features into discrete bins to capture patterns in ranges.

Comment: Sometimes **Normalization**, that is changing the distribution of inputs to resemble the normal distribution, is mentioned. However, this step is typically not essential for machine learning itself. However, it is important to use statistical inference to test the significance of learned parameters.

Prepare Data

- **Feature selection** Throw out input features that are too "similar" to other features.

  For example, if the temperature is measured both in Celsius and in Kelvin, keep one of them. The relationship can, of course, be a more complex (non-linear) correlation.

Prepare Data

- **Feature selection** Throw out input features that are too "similar" to other features.

  For example, if the temperature is measured both in Celsius and in Kelvin, keep one of them. The relationship can, of course, be a more complex (non-linear) correlation.

- **Dimensionality reduction** Transforming data from $\mathbb{R}^n$ to $\mathbb{R}^m$ where $m << n$.

  Growing dimension means growing difficulty of training for all models. Some models cease to work for high-dimensional data. The reduction typically searches for a few important characteristic features of inputs.

Prepare Data

- **Feature selection** Throw out input features that are too "similar" to other features.

  For example, if the temperature is measured both in Celsius and in Kelvin, keep one of them. The relationship can, of course, be a more complex (non-linear) correlation.

- **Dimensionality reduction** Transforming data from $\mathbb{R}^n$ to $\mathbb{R}^m$ where $m << n$.

  Growing dimension means growing difficulty of training for all models. Some models cease to work for high-dimensional data. The reduction typically searches for a few important characteristic features of inputs.

- **Feature aggregation** Introducing new features using operations on the original ones.

  We will see kernel transformations later in this course, allowing simple models to solve complex problems.

Train Model

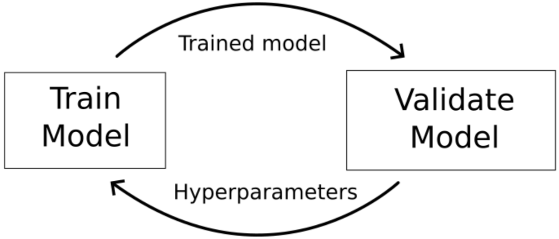
Now the dataset has been cleaned; we may train a model.

Train Model

Now the dataset has been cleaned; we may train a model.

Before training, we should split the dataset into

- *training* dataset on which the model will learn
- *validation* dataset on which we fine-tune hyperparameters



The resulting model is obtained after several iterations of the above process.

Evaluate Model

Here, we use the test set that we separated during data fetching.

In some cases, a brand new test set can be generated.
patients are examined regularly, creating new records continuously.

In some cases, it is tough to obtain new data.
For example, new expensive and difficult measurements are needed to obtain new data.

Evaluate Model

Here, we use the test set that we separated during data fetching.

In some cases, a brand new test set can be generated.
patients are examined regularly, creating new records continuously.

In some cases, it is tough to obtain new data.
For example, new expensive and difficult measurements are needed to obtain new data.

Critical issue: Make sure that you are truly testing exactly the whole inference process.

Often, just a model is tested, and the testing and production inference engines are separated. This leads to truly nasty errors in the production!

We will discuss various generic metrics helpful in measuring the quality of the resulting model.

Deploy to Production

Deployment of machine learning models is a complex question, application dependent.

The recently emerging area of MLOps is concerned with the engineering side of the model deployment.

Deploy to Production

Deployment of machine learning models is a complex question, application dependent.

The recently emerging area of MLOps is concerned with the engineering side of the model deployment.

From the technical point of view, the typical issues solved by ML Ops teams are

- how to extract/process data in real-time
- how much storage is required
- how to store/collect model (and data) artifacts/predictions
- how to set up APIs, tools, and software environments
- What the period of predictions (instantaneous or batch predictions) should be
- how to set up hardware requirements (or cloud requirements for on-cloud environments) by the computational resources required
- how to set up a pipeline for continuous training and parameter tuning

Deploy to Production

From the user's point of view:

- How to get a sensible and valuable user output?
  - AI researchers will be satisfied with tons of running text in terminals.
  - "Normal" people need a graphical interface with understandable output.
  - Experts working in other domains typically demand speed and clarity at the extreme.

Deploy to Production

From the user's point of view:

- How to get a sensible and valuable user output?
  - AI researchers will be satisfied with tons of running text in terminals.
  - "Normal" people need a graphical interface with understandable output.
  - Experts working in other domains typically demand speed and clarity at the extreme.
- How do you persuade users that the AI is working for them?
  - Especially if safety is at stake, you need to have outstanding arguments and explanations ready for end-users
  - In many areas, the devices need to be certified (medicine, automotive) for ML-based systems.

This complex subject will be only touched on in this course.

Monitor, collect Data

Deployed machine learning models must be constantly monitored.

Because of the influx of new data, ML models work in highly dynamic environments.

For example, an image-processing medical diagnostic model suddenly misdiagnosed a patient because a nurse marked the sample with a marker pen.

Every customer has a different infrastructure and may produce data slightly differently.

Data for retraining and improvement should be stored.

Also, many areas allow the *active learning* where users provide feedback for (continuous) retraining of the models.

Decision Trees

Decision Trees

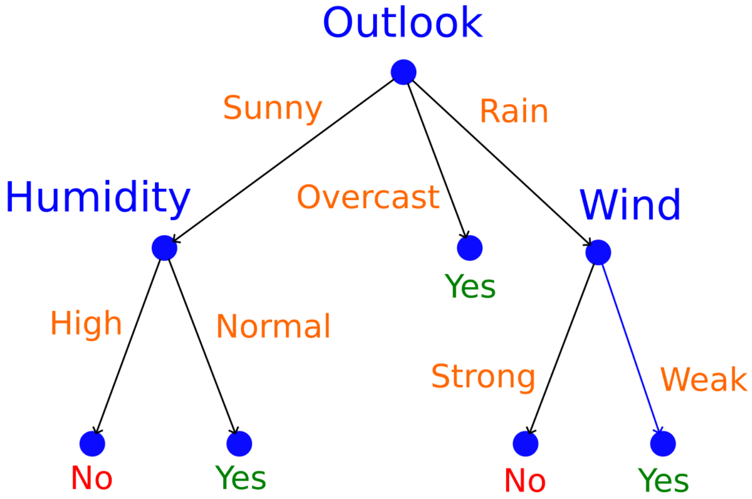
- One of the widely used methods for machine learning.
- Intuitively simple, directly explainable.
- Basis for random forests (a powerful model).

Decision Trees

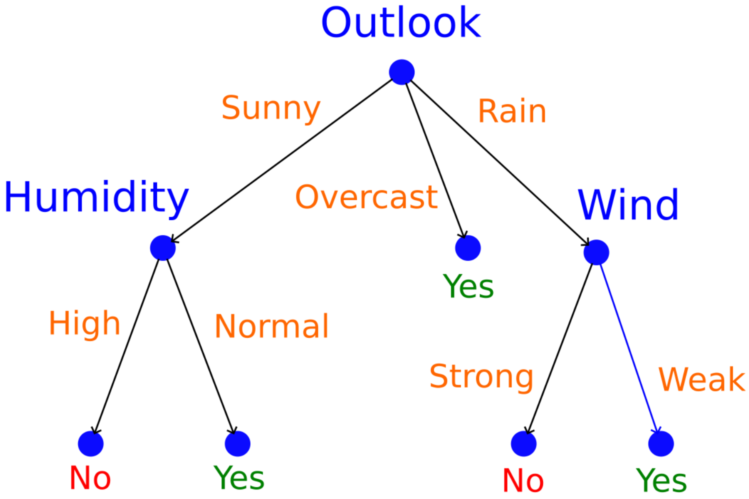
- One of the widely used methods for machine learning.
- Intuitively simple, directly explainable.
- Basis for random forests (a powerful model).

- We will consider the ID3 algorithm.
  Quinlan, 1979
- Various adjustments that appear in C4.5, CART, etc.

Consider the weather forecast for tennis playing. How would you decide whether to play today?



Consider the weather forecast for tennis playing. How would you decide whether to play today?



How do we obtain such a tree based on experience/data?

Learning Decision Trees

Consider data represented as follows:

- A finite set of *attributes* $\mathcal{A} = \{A_1, \ldots, A_n\}$.
- Each attribute $A \in \mathcal{A}$ has its *set of values* $V(A)$.

We start with trees on discrete datasets, that is, assume $V(A)$ finite for all $A \in \mathcal{A}$.

Learning Decision Trees

Consider data represented as follows:

- A finite set of *attributes* $\mathcal{A} = \{A_1, \ldots, A_n\}$.
- Each attribute $A \in \mathcal{A}$ has its *set of values* $V(A)$.

We start with trees on discrete datasets, that is, assume $V(A)$ finite for all $A \in \mathcal{A}$.

Objects to be classified are described by vectors of values of all attributes:

$$\vec{x} = (x_1, \ldots, x_n) \in V(A_1) \times \cdots \times V(A_n)$$

Given $\vec{x}$ and an attribute A_k we denote by $A_k(\vec{x})$ the value x_k of the attribute A_k in $\vec{x}$.

Learning Decision Trees

Consider data represented as follows:

- A finite set of *attributes* $\mathcal{A} = \{A_1, \ldots, A_n\}$.
- Each attribute $A \in \mathcal{A}$ has its *set of values* $V(A)$.

We start with trees on discrete datasets, that is, assume $V(A)$ finite for all $A \in \mathcal{A}$.

Objects to be classified are described by vectors of values of all attributes:

$$\vec{x} = (x_1, \ldots, x_n) \in V(A_1) \times \cdots \times V(A_n)$$

Given $\vec{x}$ and an attribute A_k we denote by $A_k(\vec{x})$ the value x_k of the attribute A_k in $\vec{x}$.

Consider a set C of *classes*.

We consider a multiclass classification in general, i.e., C is an arbitrary finite set.

Example

The tennis problem:

- The attributes are:

$$A_1 = \mathit{Outlook}, A_2 = \mathit{Temperature}, A_3 = \mathit{Humidity}, A_4 = \mathit{Wind}$$

Example

The tennis problem:

- The attributes are:

  $$A_1 = \mathit{Outlook}, A_2 = \mathit{Temperature}, A_3 = \mathit{Humidity}, A_4 = \mathit{Wind}$$

- The sets of values of the attributes:
  - $V(A_1) = \{\mathit{Sunny}, \mathit{Overcast}, \mathit{Rain}\}$
  - $V(A_2) = \{\mathit{Hot}, \mathit{Mild}, \mathit{Cool}\}$
  - $V(A_3) = \{\mathit{High}, \mathit{Normal}\}$
  - $V(A_4) = \{\mathit{Strong}, \mathit{Weak}\}$

Example

The tennis problem:

- The attributes are:

$$A_1 = Outlook, A_2 = Temperature, A_3 = Humidity, A_4 = Wind$$

- The sets of values of the attributes:
  - $V(A_1) = \{Sunny, Overcast, Rain\}$
  - $V(A_2) = \{Hot, Mild, Cool\}$
  - $V(A_3) = \{High, Normal\}$
  - $V(A_4) = \{Strong, Weak\}$
- Consider

$$\vec{x} = (Overcast, Hot, Normal, Weak) \in V(A_1) \times V(A_2) \times V(A_3) \times V(A_4)$$

Example

The tennis problem:

- The attributes are:

$$A_1 = \mathit{Outlook}, A_2 = \mathit{Temperature}, A_3 = \mathit{Humidity}, A_4 = \mathit{Wind}$$

- The sets of values of the attributes:
  - $V(A_1) = \{\mathit{Sunny}, \mathit{Overcast}, \mathit{Rain}\}$
  - $V(A_2) = \{\mathit{Hot}, \mathit{Mild}, \mathit{Cool}\}$
  - $V(A_3) = \{\mathit{High}, \mathit{Normal}\}$
  - $V(A_4) = \{\mathit{Strong}, \mathit{Weak}\}$
- Consider

$$\vec{x} = (\mathit{Overcast}, \mathit{Hot}, \mathit{Normal}, \mathit{Weak}) \in V(A_1) \times V(A_2) \times V(A_3) \times V(A_4)$$

Then

$$A_3(\vec{x}) = \mathit{Humidity}(\vec{x}) = \mathit{Normal}$$

$$A_4(\vec{x}) = \mathit{Wind}(\vec{x}) = \mathit{Weak}$$

Example

The tennis problem:

- The attributes are:

  $$A_1 = \mathit{Outlook}, A_2 = \mathit{Temperature}, A_3 = \mathit{Humidity}, A_4 = \mathit{Wind}$$

- The sets of values of the attributes:
  - $V(A_1) = \{\mathit{Sunny}, \mathit{Overcast}, \mathit{Rain}\}$
  - $V(A_2) = \{\mathit{Hot}, \mathit{Mild}, \mathit{Cool}\}$
  - $V(A_3) = \{\mathit{High}, \mathit{Normal}\}$
  - $V(A_4) = \{\mathit{Strong}, \mathit{Weak}\}$
- Consider

  $$\vec{x} = (\mathit{Overcast}, \mathit{Hot}, \mathit{Normal}, \mathit{Weak}) \in V(A_1) \times V(A_2) \times V(A_3) \times V(A_4)$$

  Then

  $$A_3(\vec{x}) = \mathit{Humidity}(\vec{x}) = \mathit{Normal}$$

  $$A_4(\vec{x}) = \mathit{Wind}(\vec{x}) = \mathit{Weak}$$

- $C = \{\mathit{Yes}, \mathit{No}\}$

Decision Trees

Consider (directed, rooted) *trees* $\mathcal{T} = (T, E)$ where T is a set of nodes and $E \subseteq T \times T$ is a set of directed edges.

Decision Trees

Consider (directed, rooted) *trees* $\mathcal{T} = (T, E)$ where T is a set of nodes and $E \subseteq T \times T$ is a set of directed edges.

Denote by $T_{leaf} \subseteq T$ the set of all *leaves* of the tree and by T_{int} the set $T \setminus T_{leaf}$ of *internal nodes*.

Decision Trees

Consider (directed, rooted) *trees* $\mathcal{T} = (T, E)$ where T is a set of nodes and $E \subseteq T \times T$ is a set of directed edges.

Denote by $T_{leaf} \subseteq T$ the set of all *leaves* of the tree and by T_{int} the set $T \setminus T_{leaf}$ of *internal nodes*.

A *decision tree* is

- a tree $\mathcal{T} = (T, E)$ where
- each leaf $\tau \in T_{leaf}$ is assigned a class $C(\tau) \in C$,
- each internal node $\tau \in T_{int}$ is assigned an attribute $A(\tau) \in \mathcal{A}$,
- and there is a bijection between edges from τ and values of the attribute $A(\tau)$. Given an edge $(\tau, \tau') \in E$ we write $V(\tau, \tau')$ to denote the value of the attribute $A(\tau)$ assigned to the edge.

Decision Trees

Consider (directed, rooted) *trees* $\mathcal{T} = (T, E)$ where T is a set of nodes and $E \subseteq T \times T$ is a set of directed edges.

Denote by $T_{leaf} \subseteq T$ the set of all *leaves* of the tree and by T_{int} the set $T \smallsetminus T_{leaf}$ of *internal nodes*.

A *decision tree* is

- a tree $\mathcal{T} = (T, E)$ where
- each leaf $\tau \in T_{leaf}$ is assigned a class $C(\tau) \in C$,
- each internal node $\tau \in T_{int}$ is assigned an attribute $A(\tau) \in \mathcal{A}$,
- and there is a bijection between edges from τ and values of the attribute $A(\tau)$. Given an edge $(\tau, \tau') \in E$ we write $V(\tau, \tau')$ to denote the value of the attribute $A(\tau)$ assigned to the edge.

Inference: Given an input $\vec{x}$, we traverse the tree from the root to a leaf, always choosing edges labeled with values of attributes from $\vec{x}$. The output is the class labeling the leaf.

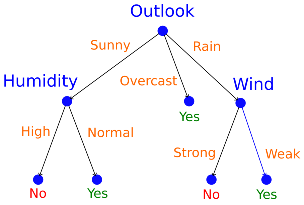
Example

$$T = \{O, H, W, z_1, z_2, z_3, z_4, z_5\}$$

$$T_{leaf} = \{z_1, z_2, z_3, z_4, z_5\},\ T_{int} = \{O, H, W\}$$

$$E = \big\{(O, H), (O, W), (H, z_1), (H, z_2), (O, z_3), (W, z_4), (W, z_5)\big\}$$

$$C(z_1) = C(z_3) = \mathit{No},\ C(z_2) = C(z_4) = \mathit{Yes}$$



$$A(O) = \mathit{Outlook},\ A(H) = \mathit{Humidity},\ A(W) = \mathit{Wind}$$

$$V(O, H) = \mathit{Sunny},\ V(O, z_3) = \mathit{Overcast},\ V(O, W) = \mathit{Rain}$$
$$V(H, z_1) = \mathit{High},\ V(H, z_2) = \mathit{Normal}$$
$$V(W, z_4) = \mathit{Strong},\ V(W, z_5) = \mathit{Weak}$$

Inference: For (*Rain*, *Hot*, *High*, *Strong*) we reach z_4, yielding *No*.

Training Dataset

Consider a *training dataset*

$$\mathcal{D} = \{(\vec{x}_k, c_k) \mid k = 1, \ldots, p\}$$

Here $\vec{x}_k \in V(A_1) \times \cdots \times V(A_k)$ and $c_k \in C$ for every k.

Technically $\mathcal{D}$ can be a multiset containing several occurrences of the same vector.

Index	**Outlook**	**Temperature**	**Humidity**	**Wind**	**PlayTennis**
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

$$
\begin{aligned}
\mathcal{D} = \big\{&((\mathit{Sunny}, \mathit{Hot}, \mathit{High}, \mathit{Weak}), \mathit{No}),\\
&((\mathit{Sunny}, \mathit{Hot}, \mathit{High}, \mathit{Strong}), \mathit{No})\\
&\ldots\\
&((\mathit{Rain}, \mathit{Mild}, \mathit{High}, \mathit{Strong}), \mathit{No})\big\}
\end{aligned}
$$

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

  $$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

  We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

  $$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

  We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.
- Finally,
  - create a root node τ of a decision tree,

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

  $$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

  We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.
- Finally,
  - create a root node τ of a decision tree,
  - assign the attribute A to τ,

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

  $$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

  We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.
- Finally,
  - create a root node τ of a decision tree,
  - assign the attribute A to τ,
  - for every $v \in V(A)$, recursively construct a decision tree with a root τ_v using $\mathcal{D}_v$,

Learning Decision Trees

The learning algorithm ID3 works as follows:

- Start with the whole training dataset $\mathcal{D}$.
- If there is just a single class in $\mathcal{D}$, create a single node decision tree that returns the class.
- Otherwise, identify an attribute $A \in \mathcal{A}$ which *best classifies* the examples in $\mathcal{D}$. For every $v \in V(A)$ we obtain

  $$\mathcal{D}_v = \{\vec{x} \mid \vec{x} \in \mathcal{D}, A(\vec{x}) = v\}$$

  We aim to have each $\mathcal{D}_v$ as pure as possible, that is, ideally, to contain examples of just a single class.
- Finally,
  - create a root node τ of a decision tree,
  - assign the attribute A to τ,
  - for every $v \in V(A)$, recursively construct a decision tree with a root τ_v using $\mathcal{D}_v$,
  - for every $v \in V(A)$ introduce an edge (τ, τ_v) assigned v.

```
1:  function ID3(dataset $\mathcal{D}$, attribute set $\mathcal{A}$)
2:      Create a root node $\tau$ for the tree
3:      if $\mathcal{D} = \emptyset$ then
4:          Return the single node $\tau$ assigned with a default class.
5:      else if all examples in $\mathcal{D}$ are of the same class $c$ then
6:          Return the single-node tree, where $\tau$ is assigned $c$
7:      else if set of attributes $\mathcal{A}$ is empty then
8:          Return the single-node tree where $\tau$ is assigned
            the most common class in $\mathcal{D}$
9:      else
10:         Choose attribute $A \in \mathcal{A}$ best classifying examples in $\mathcal{D}$.
11:         Set the decision attribute for $\tau$ to $A$
12:         for each value $v \in D(A)$ do
13:             Compute a decision tree ID3($\mathcal{D}_v, \mathcal{A} \setminus \{A\}$) with root $\tau_v$,
14:             add a new edge $(\tau, \tau_v)$ assigned $v$.
15:         end for
16:     end if
17: return $\tau$
18: end function
```

Best Classifying Attribute

We aim to choose an attribute that best informs us about the class.

As a result, we would possibly use as few attributes as possible and obtain a small tree containing only class-relevant decisions.

How to choose an attribute that best classifies examples in $\mathcal{D}$?

There are several measures used in practice.

The most common are

- *information gain*
- *Gini impurity decrease*

Information Gain

The information gain is based on the notion of entropy.

Information Gain

The information gain is based on the notion of entropy.

We need some notation:

- Given a training dataset $\mathcal{D}$ and a class $c \in C$ we denote by p_c the proportion of examples with class c in $\mathcal{D}$.

Information Gain

The information gain is based on the notion of entropy.

We need some notation:

- Given a training dataset $\mathcal{D}$ and a class $c \in C$ we denote by p_c the proportion of examples with class c in $\mathcal{D}$.
- We define the *entropy* of $\mathcal{D}$ by

$$Entropy(\mathcal{D}) = \sum_{c \in C} -p_c \log_2 p_c$$

Information Gain

The information gain is based on the notion of entropy.

We need some notation:

- Given a training dataset $\mathcal{D}$ and a class $c \in C$ we denote by p_c the proportion of examples with class c in $\mathcal{D}$.
- We define the *entropy* of $\mathcal{D}$ by

$$Entropy(\mathcal{D}) = \sum_{c \in C} -p_c \log_2 p_c$$

- The *information gain* of an attribute A is then defined by

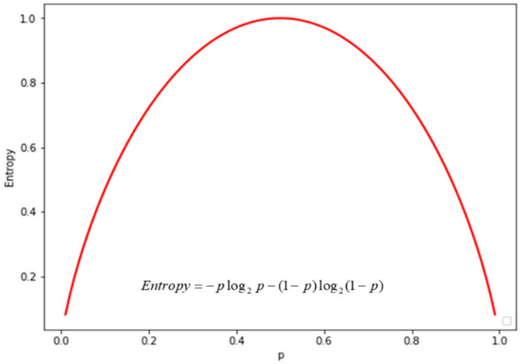
$$Gain(\mathcal{D}, A) = Entropy(\mathcal{D}) - \sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Entropy(\mathcal{D}_v)$$

In every step of the ID3 algorithm, we choose an attribute *maximizing* the information gain for the current dataset $\mathcal{D}$.

Information Gain

The intuition behind information gain:

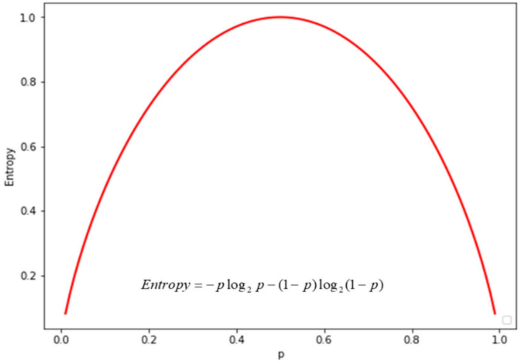
- Consider $C = \{0, 1\}$ and p the proportion of examples of class 1. p measures the "uncertainty" of the class:



Information Gain

The intuition behind information gain:

- Consider $C = \{0, 1\}$ and p the proportion of examples of class 1. p measures the "uncertainty" of the class:

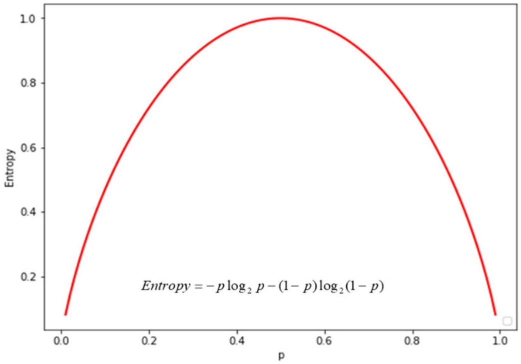


- $\sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Entropy(\mathcal{D}_v)$ is weighted uncertainty of classes in each $\mathcal{D}_v$ (weighted by the relative size of $\mathcal{D}_v$).

Information Gain

The intuition behind information gain:

- Consider $C = \{0, 1\}$ and p the proportion of examples of class 1. p measures the "uncertainty" of the class:



- $\sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Entropy(\mathcal{D}_v)$ is weighted uncertainty of classes in each $\mathcal{D}_v$ (weighted by the relative size of $\mathcal{D}_v$).
- $Gain(\mathcal{D}, A)$ measures reduction in uncertainty of classes by splitting $\mathcal{D}$ according to A.

Gini Impurity

- We define *Gini impurity* of $\mathcal{D}$ by

$$Gini(\mathcal{D}) = 1 - \sum_{c \in C} p_c^2$$

Gini Impurity

- We define *Gini impurity* of $\mathcal{D}$ by

$$Gini(\mathcal{D}) = 1 - \sum_{c \in C} p_c^2$$

- The *impurity decrease* of an attribute A is then defined similarly to the gain in the entropy case

$$ImpDec(\mathcal{D}, A) = Gini(\mathcal{D}) - \sum_{v \in V(A)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} Gini(\mathcal{D}_v)$$

In every step of the ID3 algorithm, we choose an attribute *maximizing* the impurity decrease for the current dataset $\mathcal{D}$.

Gini Impurity

What is the intuition behind $Gini(\mathcal{D})$?

Gini Impurity

What is the intuition behind $Gini(\mathcal{D})$?

Assume we randomly independently choose objects from $\mathcal{D}$.

$1 - \sum_{c \in C} p_c^2$ is the probability of choosing two objects of different classes in two consecutive independent trials.

Indeed, p_c is the probability of choosing an object of class c, p_c^2 the probability of choosing objects of the class c twice, and $\sum_{c \in C} p_c^2$ the probability of choosing two objects of the same class.

In what follows (and at the exam), we will work only with the Gini impurity as it is easier to compute.

Example

Consider our tennis example (see the table).

- Consider the whole dataset $\mathcal{D}$.
  - $p_{Yes} = 9/14$
  - $p_{No} = 5/14$
  - $Gini(\mathcal{D}) = 1 - (9/14)^2 - (5/14)^2 = 0.45918$

Example

Consider our tennis example (see the table).

- Consider the whole dataset $\mathcal{D}$.
  - $p_{Yes} = 9/14$
  - $p_{No} = 5/14$
  - $Gini(\mathcal{D}) = 1 - (9/14)^2 - (5/14)^2 = 0.45918$
- For $A = Outlook$ we get
  - $Gini(\mathcal{D}_{Sunny}) = 1 - (2/5)^2 - (3/5)^2 = 0.48$
  - $Gini(\mathcal{D}_{Overcast}) = 1 - 1^2 - 0^2 = 0$
  - $Gini(\mathcal{D}_{Rain}) = 1 - (3/5)^2 - (2/5)^2 = 0.48$

Example

Consider our tennis example (see the table).

- Consider the whole dataset $\mathcal{D}$.
  - $p_{Yes} = 9/14$
  - $p_{No} = 5/14$
  - $Gini(\mathcal{D}) = 1 - (9/14)^2 - (5/14)^2 = 0.45918$
- For $A = Outlook$ we get
  - $Gini(\mathcal{D}_{Sunny}) = 1 - (2/5)^2 - (3/5)^2 = 0.48$
  - $Gini(\mathcal{D}_{Overcast}) = 1 - 1^2 - 0^2 = 0$
  - $Gini(\mathcal{D}_{Rain}) = 1 - (3/5)^2 - (2/5)^2 = 0.48$

  Thus

$$ImpDec(\mathcal{D}, Outlook) = \\ 0.459 - (5/14) \cdot 0.48 - (4/14) \cdot 0 - (5/14) \cdot 0.48 \\ = 0.117$$

- $ImpDec(\mathcal{D}, Temperature) = 0.018$
- $ImpDec(\mathcal{D}, Humidity) = 0.091$
- $ImpDec(\mathcal{D}, Wind) = 0.030$

So the largest information gain is given by the *Outlook*.

Example

Going further on, consider $\mathcal{D} = \mathcal{D}_{Sunny}$. We get

- $ImpDec(\mathcal{D}, Temperature) = 0.279$
- $ImpDec(\mathcal{D}, Humidity) = 0.48$
- $ImpDec(\mathcal{D}, Wind) = 0.013$

The best choice attribude after *Sunny* in *Outlook* is *Humidity*.

Example

Going further on, consider $\mathcal{D} = \mathcal{D}_{Sunny}$. We get

- $ImpDec(\mathcal{D}, Temperature) = 0.279$
- $ImpDec(\mathcal{D}, Humidity) = 0.48$
- $ImpDec(\mathcal{D}, Wind) = 0.013$

The best choice attribude after *Sunny* in *Outlook* is *Humidity*.

Now consider $\mathcal{D} = \mathcal{D}_{Rain}$.

- $ImpDec(\mathcal{D}, Temperature) = 0.013$
- $ImpDec(\mathcal{D}, Humidity) = 0.013$
- $ImpDec(\mathcal{D}, Wind) = 0.48$

The best choice attribude after *Rain* in *Outlook* is *Wind*.

Continuous-Valued Attributes

What if values of an attribute A come from a continuous variable?

A is a numerical attribute that can take any value in an interval, such as temperature, size, time, etc.

Continuous-Valued Attributes

What if values of an attribute A come from a continuous variable?

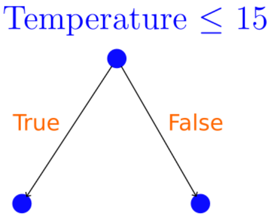
A is a numerical attribute that can take any value in an interval, such as temperature, size, time, etc.

Consider an internal node $\tau \in T_{int}$ assigned such a continuous attribute A. Then

- τ is assigned a threshold value called a *cut point* $H \in \mathbb{R}$,
- there are two edges $e_{\text{true}}, e_{\text{false}}$ from τ,
- e_{true} labeled with True and e_{false} labeled with False.

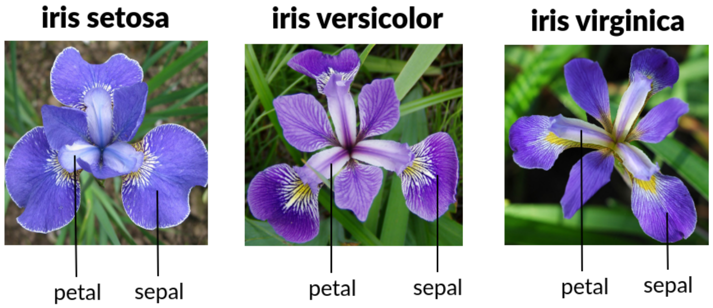
During inference, when considering an example $\vec{x}$ in the node τ,

- evaluate $A(\vec{x}) \leq H$,
- if $A(\vec{x}) \leq H$, then follow e_{true},
- else follow e_{false}.



In training, the cut point is chosen from the attribute values in the training set using information gain/impurity decrease similar to discrete attributes.

Iris Example



Attributes
Sepal.Length, Sepal.Width, Petal.Length, Petal.Width

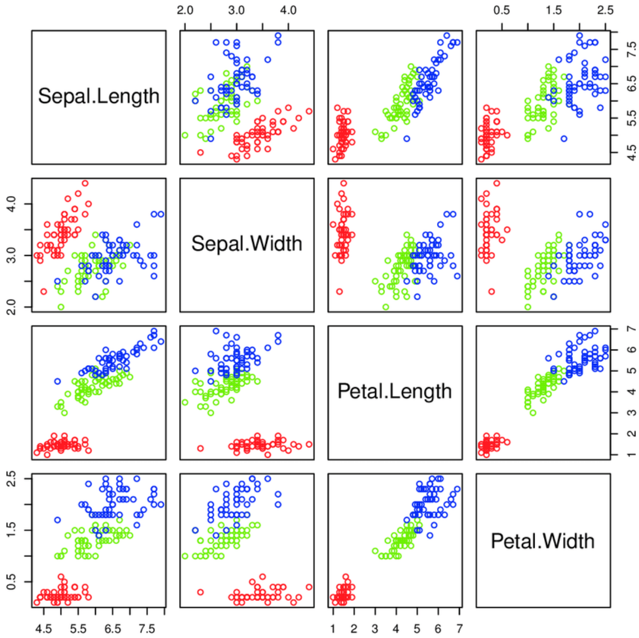
Classes (Variety)
Setosa, Versicolor, Virginica

Iris Example

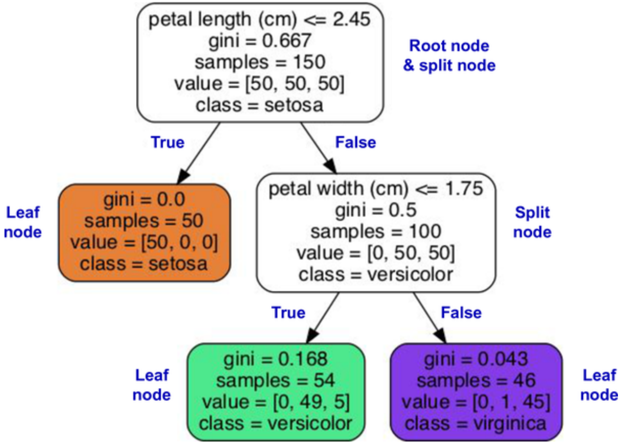
The dataset (150 examples):

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Variety
5.5	3.5	1.3	0.2	Setosa
6.8	2.8	4.8	1.4	Versicolor
6.7	3.1	4.7	1.5	Versicolor
6.9	3.1	5.1	2.3	Virginica
7.3	2.9	6.3	1.8	Virginica
5.4	3.7	1.5	0.2	Setosa
4.6	3.4	1.4	0.3	Setosa
6.2	2.8	4.8	1.8	Virginica
5.4	3.0	4.5	1.5	Versicolor
4.7	3.2	1.6	0.2	Setosa
6.7	3.3	5.7	2.1	Virginica
5.0	3.4	1.5	0.2	Setosa
5.0	3.0	1.6	0.2	Setosa
4.4	2.9	1.4	0.2	Setosa
6.0	3.4	4.5	1.6	Versicolor
5.1	3.5	1.4	0.2	Setosa
6.6	3.0	4.4	1.4	Versicolor
5.9	3.2	4.8	1.8	Versicolor
5.6	2.8	4.9	2.0	Virginica
...				

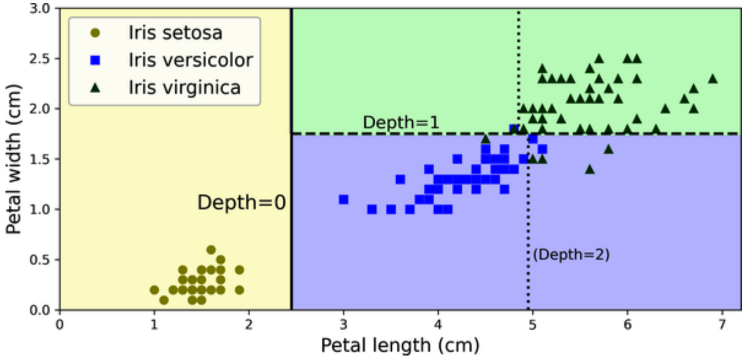
Iris Example



Iris Example - Decision Tree



Iris Example - Decision Tree Boudaries



If the leaves are split further, the Depth = 2 boundary would be added.

Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

One example is *mean decrease impurity* defined as follows:

Consider a decision tree T trained on a dataset $\mathcal{D}$ using the ID3.

Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

One example is *mean decrease impurity* defined as follows:

Consider a decision tree T trained on a dataset $\mathcal{D}$ using the ID3.

For every node τ of $\mathcal{T}$, denote by $\mathcal{D}[\tau]$ the subset of $\mathcal{D}$ which was used in the ID3 procedure when the node τ was created (line 2).

Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

One example is *mean decrease impurity* defined as follows:

Consider a decision tree T trained on a dataset $\mathcal{D}$ using the ID3.

For every node τ of $\mathcal{T}$, denote by $\mathcal{D}[\tau]$ the subset of $\mathcal{D}$ which was used in the ID3 procedure when the node τ was created (line 2).

Consider an attribute A and denote by $T[A] \subseteq T_{int}$ the set of all nodes of $\mathcal{T}$ assigned the attribute A by ID3 (line 11).

Attribute Importance Computation

How important are attributes for the trained tree $\mathcal{T}$? Depends on

- how close they are to the root of $\mathcal{T}$,
- how large information gain/decrease in impurity they give.

There are several formulae for computing the importance.

One example is *mean decrease impurity* defined as follows:

Consider a decision tree T trained on a dataset $\mathcal{D}$ using the ID3.

For every node τ of $\mathcal{T}$, denote by $\mathcal{D}[\tau]$ the subset of $\mathcal{D}$ which was used in the ID3 procedure when the node τ was created (line 2).

Consider an attribute A and denote by $T[A] \subseteq T_{int}$ the set of all nodes of $\mathcal{T}$ assigned the attribute A by ID3 (line 11).

Then define the importance as the average decrease in Gini impurity (i.e., average *ImpDec*) in the nodes of $T[A]$:

$$\mathit{GiniImportance}(A) = \sum_{\tau \in T[A]} \frac{|\mathcal{D}[\tau]|}{|\mathcal{D}|} \mathit{ImpDec}(\mathcal{D}[\tau], A)$$

Decision Trees

Practical Issues

Practical Issues

- Data preprocessing
- Model tunning (overfitting and underfitting)
- Sensitivity to changes in data/hyperparameters
- Learning representation problems (the XOR)

Data Preprocessing

Little preprocessing is needed for decision trees.

Data Preprocessing

Little preprocessing is needed for decision trees.

Of course, ideally, clean up the data, but

- Missing values are not such a big issue
  (considering a concrete example, exclude the attributes with missing values)

Data Preprocessing

Little preprocessing is needed for decision trees.

Of course, ideally, clean up the data, but

- Missing values are not such a big issue
(considering a concrete example, exclude the attributes with missing values)
- Outliers are not such a big issue either; the division of nodes is done based on relative counts, not concrete values.

Data Preprocessing

Little preprocessing is needed for decision trees.

Of course, ideally, clean up the data, but

- Missing values are not such a big issue (considering a concrete example, exclude the attributes with missing values)
- Outliers are not such a big issue either; the division of nodes is done based on relative counts, not concrete values.
- Decision trees can cope with continuous and categorical values directly (i.e., no encoding necessary)

Data Preprocessing

Little preprocessing is needed for decision trees.

Of course, ideally, clean up the data, but

- Missing values are not such a big issue (considering a concrete example, exclude the attributes with missing values)
- Outliers are not such a big issue either; the division of nodes is done based on relative counts, not concrete values.
- Decision trees can cope with continuous and categorical values directly (i.e., no encoding necessary)

Imbalanced classes might cause problems because of small information gain/impurity decrease in splitting.

Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- 10^6 examples have the class 1,
- 100 examples have the class 0.

Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- 10^6 examples have the class 1,
- 100 examples have the class 0.

Then

- $p_1 = 10^6/(10^6 + 100) \approx 1$ and $p_0 = 100/10^6 \approx 0$,
- thus the Gini impurity $1 - p_1^2 - p_0^2 \approx 0$.

Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- 10^6 examples have the class 1,
- 100 examples have the class 0.

Then

- $p_1 = 10^6/(10^6 + 100) \approx 1$ and $p_0 = 100/10^6 \approx 0$,
- thus the Gini impurity $1 - p_1^2 - p_0^2 \approx 0$.

Consider an attribute A with $V(A) = \{a, b\}$.

Splitting $\mathcal{D}$ according to A gives to sets $\mathcal{D}_a$ and $\mathcal{D}_b$.

Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- 10^6 examples have the class 1,
- 100 examples have the class 0.

Then

- $p_1 = 10^6/(10^6 + 100) \approx 1$ and $p_0 = 100/10^6 \approx 0$,
- thus the Gini impurity $1 - p_1^2 - p_0^2 \approx 0$.

Consider an attribute A with $V(A) = \{a, b\}$.

Splitting $\mathcal{D}$ according to A gives to sets $\mathcal{D}_a$ and $\mathcal{D}_b$.

What is the impurity decrease caused by the attribute?

$$ImpDec(\mathcal{D}, A) = Gini(\mathcal{D}) - \frac{|\mathcal{D}_a|}{|\mathcal{D}|} Gini(\mathcal{D}_a) - \frac{|\mathcal{D}_b|}{|\mathcal{D}|} Gini(\mathcal{D}_b)$$

Imbalanced Classes

Consider a dataset $\mathcal{D}$ where

- there are two classes, $C = \{0, 1\}$,
- 10^6 examples have the class 1,
- 100 examples have the class 0.

Then

- $p_1 = 10^6/(10^6 + 100) \approx 1$ and $p_0 = 100/10^6 \approx 0$,
- thus the Gini impurity $1 - p_1^2 - p_0^2 \approx 0$.

Consider an attribute A with $V(A) = \{a, b\}$.

Splitting $\mathcal{D}$ according to A gives to sets $\mathcal{D}_a$ and $\mathcal{D}_b$.

What is the impurity decrease caused by the attribute?

$$ImpDec(\mathcal{D}, A) = Gini(\mathcal{D}) - \frac{|\mathcal{D}_a|}{|\mathcal{D}|} Gini(\mathcal{D}_a) - \frac{|\mathcal{D}_b|}{|\mathcal{D}|} Gini(\mathcal{D}_b)$$

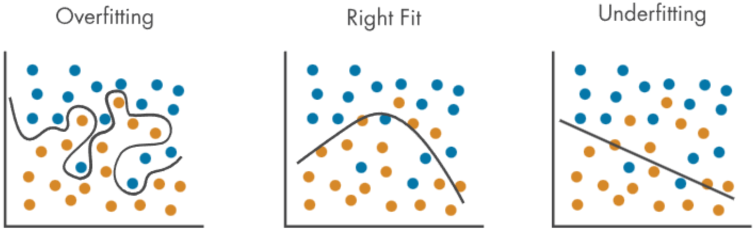
For small $|\mathcal{D}_a|$ (say ≤ 1000) we have small $|\mathcal{D}_a|/|\mathcal{D}|$

For not so small $\mathcal{D}_a$ we have $Gini(\mathcal{D}_a) \approx 0$.

In both cases, the impurity decrease is very small.

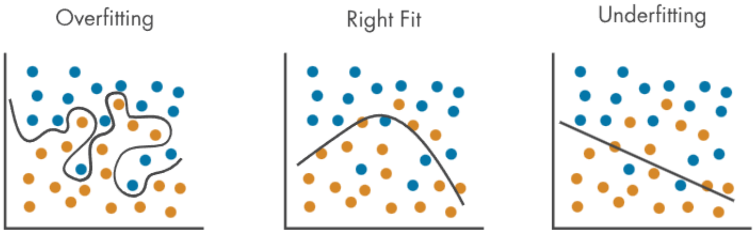
Model Tuning - Over/Under Fitting

The behavior of the model on the training set:



Model Tuning - Over/Under Fitting

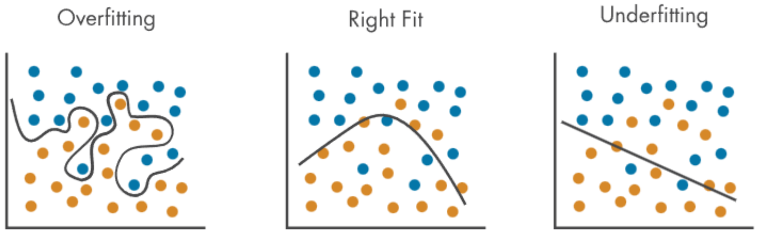
The behavior of the model on the training set:



- The left one is strongly overfitting. It would possibly not work well on new data.

Model Tuning - Over/Under Fitting

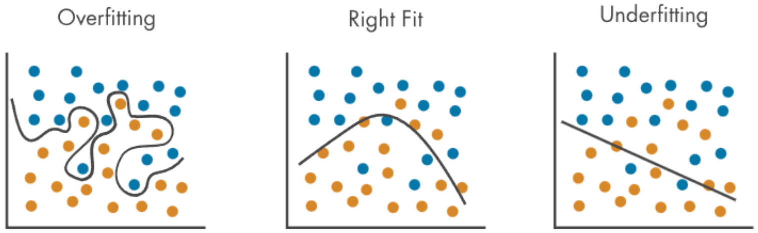
The behavior of the model on the training set:



- The left one is strongly overfitting. It would possibly not work well on new data.
- The right one is strongly underfitting. It would probably give poor classification results.

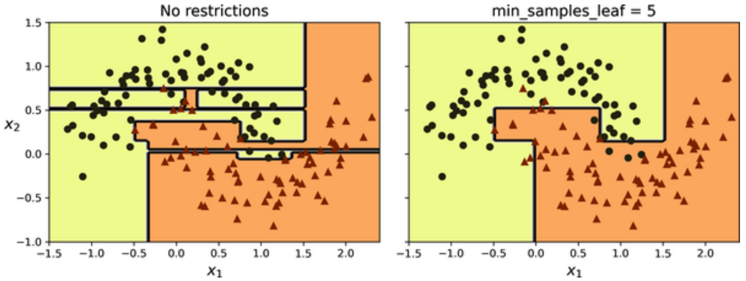
Model Tuning - Over/Under Fitting

The behavior of the model on the training set:



- The left one is strongly overfitting. It would possibly not work well on new data.
- The right one is strongly underfitting. It would probably give poor classification results.
- The middle one seems good (but still needs to be tested on fresh data).

Model Tuning - Overfitting in Decision Trees

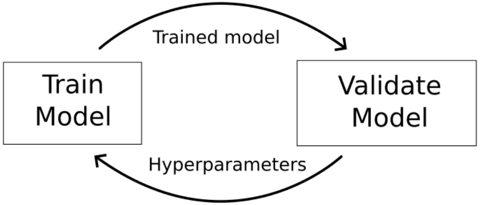


See the overfitting on the left and the “nice” model on the right.

Both overfitting and underfitting are best avoided. But how do we find out?

Model Tuning (In General)

Recall from the first lecture:

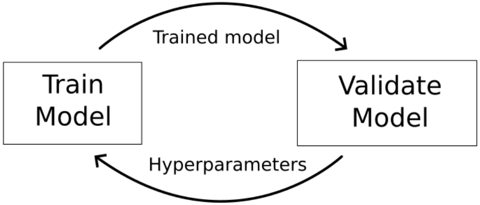


The validation should be done on a **validation set** separated from the training set.

We will discuss more sophisticated techniques later.

Model Tuning (In General)

Recall from the first lecture:



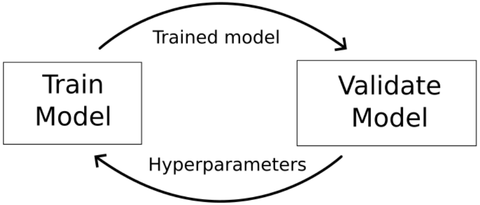
The validation should be done on a **validation set** separated from the training set.

We will discuss more sophisticated techniques later.

What hyperparameters to set? (see the next slide)

Model Tuning (In General)

Recall from the first lecture:



The validation should be done on a **validation set** separated from the training set.

We will discuss more sophisticated techniques later.

What hyperparameters to set? (see the next slide)

What to observe? In the case of decision trees, one should observe the difference between performance measures (e.g., classification accuracy) on the training and validation sets.

The too-large difference implies an improperly fitting model.

How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

- Pre-pruning: Build the tree so it does not overfit by restricting its size.

How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

- Pre-pruning: Build the tree so it does not overfit by restricting its size.
- Post-pruning: Overfit with a large tree and remove subtrees to make it smaller.

How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

- Pre-pruning: Build the tree so it does not overfit by restricting its size.
- Post-pruning: Overfit with a large tree and remove subtrees to make it smaller.
- Ensemble methods: Fit several different trees and let them classify together (e.g., using majority voting).

How to Fit Decision Trees?

There are several approaches available for decision trees.

Generally, the overfitting can be either prevented or resolved.

- Pre-pruning: Build the tree so it does not overfit by restricting its size.
- Post-pruning: Overfit with a large tree and remove subtrees to make it smaller.
- Ensemble methods: Fit several different trees and let them classify together (e.g., using majority voting).

The post-pruning approach has been more successful in practice than the pre-pruning because it is usually hard to say when to stop growing the tree.

We shall meet this controversy also in deep learning, where recent history shows a similar phenomenon.

The ensemble methods will be covered later when we discuss random forests.

Pre-Prunning - Hyperparamaters

Hyperparameters controlling the size of the tree:

- Maximum depth - do not grow the tree beyond the max depth

  The deeper the tree, the more complex models you can create $\Rightarrow$ overfitting. Low depth may restrict expressivity.

Pre-Prunning - Hyperparamaters

Hyperparameters controlling the size of the tree:

- Maximum depth - do not grow the tree beyond the max depth
  The deeper the tree, the more complex models you can create $\Rightarrow$ overfitting. Low depth may restrict expressivity.
- Minimum number of examples to split a node - if $\mathcal{D}[\tau]$ is small, τ becomes a leaf (labeled with the majority class)
  Higher values prevent a model from learning relations specific only for a few examples. Too high values may result in underfitting.

Pre-Prunning - Hyperparamaters

Hyperparameters controlling the size of the tree:

- Maximum depth - do not grow the tree beyond the max depth
  The deeper the tree, the more complex models you can create $\Rightarrow$ overfitting. Low depth may restrict expressivity.
- Minimum number of examples to split a node - if $\mathcal{D}[\tau]$ is small, τ becomes a leaf (labeled with the majority class)
  Higher values prevent a model from learning relations specific only for a few examples. Too high values may result in underfitting.
- Minimum number of examples required to be in a leaf
  Similar to the previous one. A higher number means we cannot have very specific branches concerned with particular combinations of values.

Pre-Prunning - Hyperparamaters

Hyperparameters controlling the size of the tree:

- Maximum depth - do not grow the tree beyond the max depth
  The deeper the tree, the more complex models you can create $\Rightarrow$ overfitting. Low depth may restrict expressivity.
- Minimum number of examples to split a node - if $\mathcal{D}[\tau]$ is small, τ becomes a leaf (labeled with the majority class)
  Higher values prevent a model from learning relations specific only for a few examples. Too high values may result in underfitting.
- Minimum number of examples required to be in a leaf
  Similar to the previous one. A higher number means we cannot have very specific branches concerned with particular combinations of values.
- Minimum information gain/impurity decrease
  A small impurity decrease means that the split does not contribute too much to the classification (their proportions after a split are similar to proportions before a split). However, keep in mind that it is *weighted average impurity* after the split.

Post-Pruning - Reduced Error Pruning

Train a large tree and then remove nodes that make classification worse on the validation set.

Post-Pruning - Reduced Error Pruning

Train a large tree and then remove nodes that make classification worse on the validation set.

Given a decision tree $\mathcal{T}$ and its internal node $\tau \in T_{int}$, we denote by $\mathcal{T}_{-\tau}$ the tree obtained from $\mathcal{T}$ by removing the subtree rooted in τ, i.e., τ is a leaf of $\mathcal{T}_{-\tau}$.

Post-Pruning - Reduced Error Pruning

Train a large tree and then remove nodes that make classification worse on the validation set.

Given a decision tree $\mathcal{T}$ and its internal node $\tau \in T_{int}$, we denote by $\mathcal{T}_{-\tau}$ the tree obtained from $\mathcal{T}$ by removing the subtree rooted in τ, i.e., τ is a leaf of $\mathcal{T}_{-\tau}$.

```
 1: Train $\mathcal{T}$ to maximum fit on the *training dataset.*
 2: **while** true **do**
 3:     $Err[\mathcal{T}] \leftarrow$ the error of $\mathcal{T}$ on the validation set.
 4:     **for** $\tau \in T_{int}$ **do**
 5:         $Err[\mathcal{T}_{-\tau}] \leftarrow$ the error of $\mathcal{T}_{-\tau}$ on the validation set.
 6:     **end for**
 7:     **if** $Err[\mathcal{T}] \leq \min\{Err[\mathcal{T}_{-\tau}] \mid \tau \in T_{int}\}]$ **then return** $\mathcal{T}$
 8:     **else**
 9:         $\mathcal{T} \leftarrow argmin\{Err[\mathcal{T}_{-\tau}] \mid \tau \in T_{int}\}$
10:     **end if**
11: **end while**
```

The error $Err[\mathcal{T}]$ can be any measure of the “badness” of the decision tree $\mathcal{T}$. For example, $1 - Accuracy$.

Other Pruning Methods

There are more pruning methods.

- Rule Post-Pruning:
  - Transform the tree into a set of rules.
    Rules correspond to paths in the tree; they have a form of implication: Specific values of attributes imply a class.
  - Remove the attribute conditions from the premises of the implications.

  This gives a more refined pruning: Instead of removing the whole subtree, each path of the subtree can be pruned individually.

Other Pruning Methods

There are more pruning methods.

- Rule Post-Pruning:
  - Transform the tree into a set of rules.
    Rules correspond to paths in the tree; they have a form of implication: Specific values of attributes imply a class.
  - Remove the attribute conditions from the premises of the implications.

  This gives a more refined pruning: Instead of removing the whole subtree, each path of the subtree can be pruned individually.
- Using cost complexity measure: Evaluate trees not only based on the classification error but also based on their size.

Other Pruning Methods

There are more pruning methods.

- Rule Post-Pruning:
  - Transform the tree into a set of rules.
    Rules correspond to paths in the tree; they have a form of implication: Specific values of attributes imply a class.
  - Remove the attribute conditions from the premises of the implications.

  This gives a more refined pruning: Instead of removing the whole subtree, each path of the subtree can be pruned individually.
- Using cost complexity measure: Evaluate trees not only based on the classification error but also based on their size.

  Typically introduce regularization into the error functions: Given a decision tree $\mathcal{T}$

  $$Err_\alpha(\mathcal{T}) = Err(\mathcal{T}) + \alpha|\mathcal{T}|$$

  The original paper by Breiman et al. (1984) defined $Err(\mathcal{T})$ to be the misclassification rate on the training dataset, and $|\mathcal{T}|$ is the number of nodes of the tree $\mathcal{T}$.

Sensitivity to Small Changes and Randomness

- Decision trees are sensitive to small changes in data and hyperparameters.

  Several attributes may provide (almost) identical information gain but divide the training dataset very differently.
- Some implementations choose attributes partially in random (sci-kit-learn). You may get completely different trees.

Sensitivity to Small Changes and Randomness

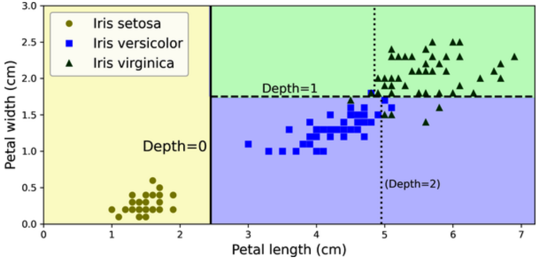
- Decision trees are sensitive to small changes in data and hyperparameters.

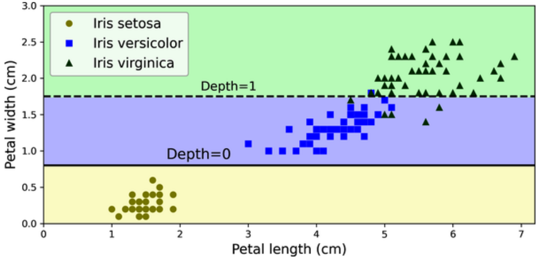
  Several attributes may provide (almost) identical information gain but divide the training dataset very differently.
- Some implementations choose attributes partially in random (sci-kit-learn). You may get completely different trees.

A solution is to train an ensemble of many decision trees and then use majority voting for classification.

This is the fundamental idea behind random forests (see later lectures).

Iris - Illustration



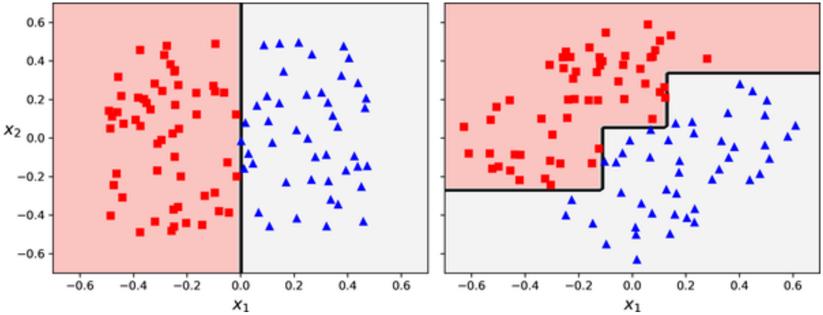


Decision trees trained on the Iris dataset.

Iris Setosa is perfectly separated by many choices for the first split.

Axis Sensitivity

The decision makes divisions along particular axes:

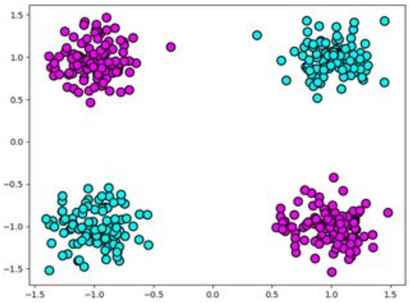


That is, rotated data may result in a completely different model.

That is why decision trees are often preceded by the *principal component analysis (PCA)* transformation, which aligns data along the axes of maximum data variance.

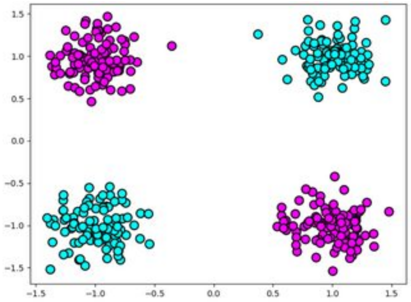
XOR Training Problem

Consider the following training dataset:

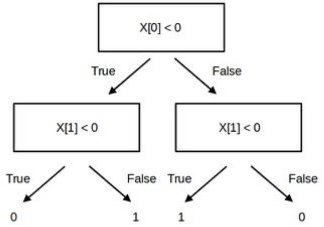


XOR Training Problem

Consider the following training dataset:

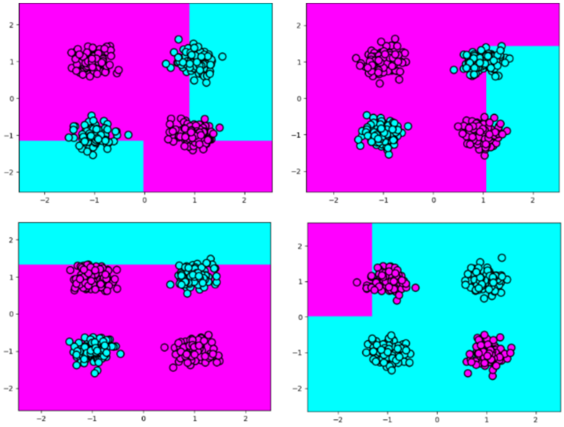


An ideal decision tree would look like this:



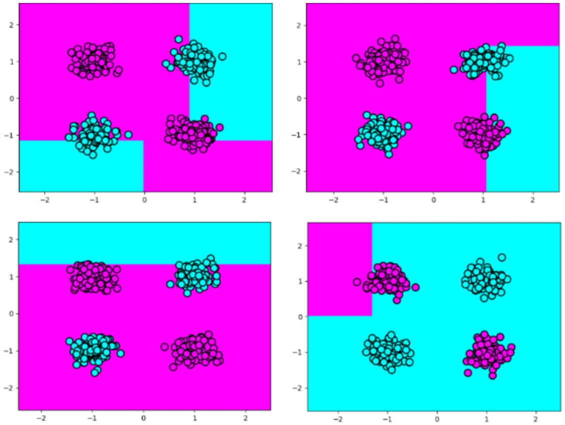
Attempts at Training on XOR

Max depth = 2:



Attempts at Training on XOR

Max depth = 2:

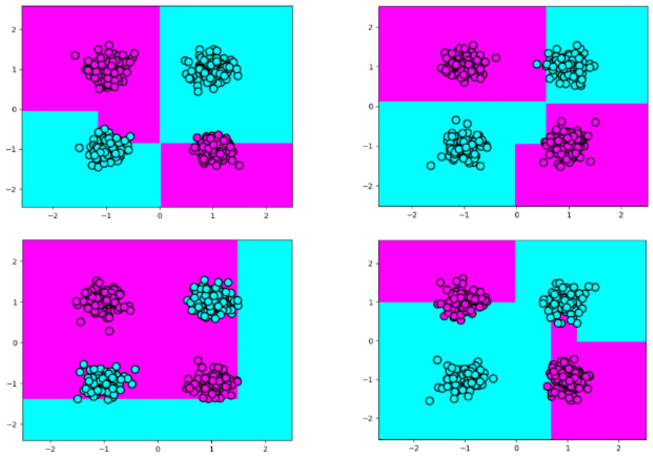


The problem: Both information gain and decrease in impurity consider only the relationship of a *single* attribute and the class.

However, there is no relationship between a single attribute and the class; both attributes need to be considered together!

More Attempts at Training on XOR

Max depth $= 3$:



It's better but still fails occasionally.

Advantages of Decision Trees

- Simple to understand and interpret; trees can be visualized.
- Uses a white box model, where conditions are easily explained by boolean logic.

Advantages of Decision Trees

- Simple to understand and interpret; trees can be visualized.
- Uses a white box model, where conditions are easily explained by boolean logic.
- Can approximate an arbitrary (reasonable) boundary and capture complex non-linear relationships between attributes.
- Capable of handling multi-class problems.

Advantages of Decision Trees

- Simple to understand and interpret; trees can be visualized.
- Uses a white box model, where conditions are easily explained by boolean logic.
- Can approximate an arbitrary (reasonable) boundary and capture complex non-linear relationships between attributes.
- Capable of handling multi-class problems.
- Little data preparation, unlike other techniques requiring normalization, dummy variables, or missing value removal.
- Handles numerical and categorical data.
- Not sensitive to outliers since the splitting is based on the proportion of examples within the split ranges and not on absolute values.

Advantages of Decision Trees

- Simple to understand and interpret; trees can be visualized.
- Uses a white box model, where conditions are easily explained by boolean logic.

- Can approximate an arbitrary (reasonable) boundary and capture complex non-linear relationships between attributes.
- Capable of handling multi-class problems.

- Little data preparation, unlike other techniques requiring normalization, dummy variables, or missing value removal.
- Handles numerical and categorical data.
- Not sensitive to outliers since the splitting is based on the proportion of examples within the split ranges and not on absolute values.

- The cost of using a well-balanced tree is logarithmic in the number of data points used to train it.

Disadvantages of Decision Trees

- Overfitting: Trees can be over-complex and not generalize well, needing pruning or limits on tree depth.
- Instability: Small data variations can result in very different trees. This is mitigated in ensemble methods.
- Non-smooth predictions: Decision trees make piecewise constant approximations, which are not suitable for extrapolation.

Disadvantages of Decision Trees

- Overfitting: Trees can be over-complex and not generalize well, needing pruning or limits on tree depth.
- Instability: Small data variations can result in very different trees. This is mitigated in ensemble methods.
- Non-smooth predictions: Decision trees make piecewise constant approximations, which are not suitable for extrapolation.
- Difficulty expressing certain concepts, such as XOR, parity, or multiplexer problems (see the next slide).
- Bias in trees: Decision trees can create biased trees if some classes dominate. Balancing the dataset is recommended.

Disadvantages of Decision Trees

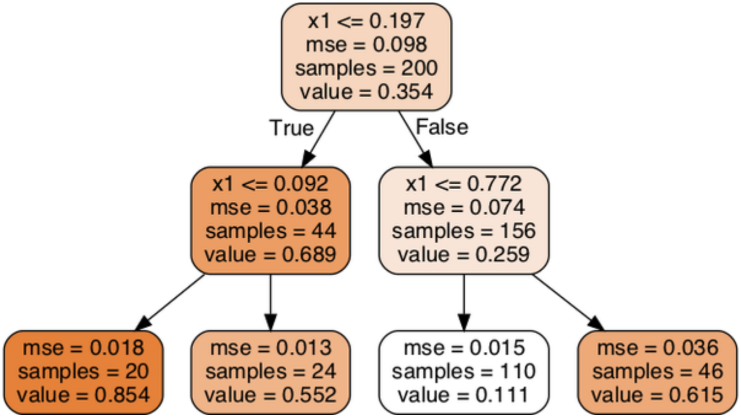
- Overfitting: Trees can be over-complex and not generalize well, needing pruning or limits on tree depth.
- Instability: Small data variations can result in very different trees. This is mitigated in ensemble methods.
- Non-smooth predictions: Decision trees make piecewise constant approximations, which are not suitable for extrapolation.
- Difficulty expressing certain concepts, such as XOR, parity, or multiplexer problems (see the next slide).
- Bias in trees: Decision trees can create biased trees if some classes dominate. Balancing the dataset is recommended.
- Learning optimal trees is NP-complete: Heuristic algorithms like greedy algorithms are used, which do not guarantee globally optimal trees. Ensemble methods can help.

History of Decision Trees

- Hunt and colleagues use exhaustive search decision-tree methods (CLS) to model human concept learning in the 1960's.
- In the late 70's, Quinlan developed ID3 with the information gain heuristic to learn expert systems from examples.
- Simultaneously, Breiman, Friedman, and colleagues develop CART (Classification and Regression Trees), similar to ID3.
- In the 1980s, various improvements were introduced to handle noise, continuous features, missing features, and improved splitting criteria. Various expert-system development tools results.
- Quinlan's updated decision-tree package (C4.5) released in 1993.

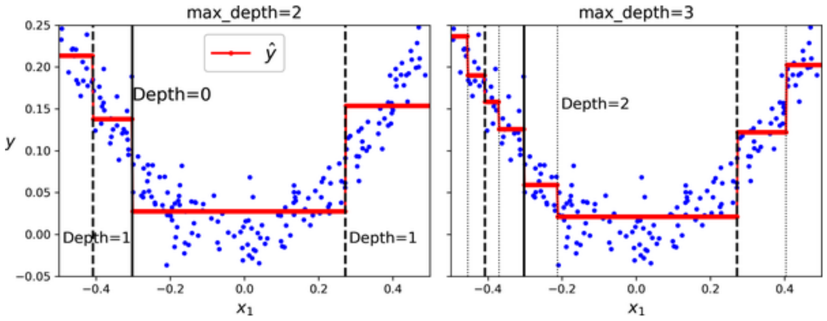
Comment on Regression Trees

Decision trees can also be used to approximate functions. Assign a function value to the leaves instead of classes.



Here, “mse” is the mean-squared-error. We get to this notion later in connection with linear models and neural networks.

Comment on Regression Trees



Intuitively, for every subinterval of x_1, the value (the red line) is at the average y over the subinterval.

How are the subintervals being set? We will answer this question later once we master the mean-squared error.

Probabilistic Classification

Probabilistic Classification – Idea

Imagine that

- I look out of a window and see a bird,
- it is black, approx. 25 cm long and has a rather yellow beak.

Probabilistic Classification – Idea

Imagine that

- I look out of a window and see a bird,
- it is black, approx. 25 cm long and has a rather yellow beak.

My daughter asks: What kind of bird is this?

Probabilistic Classification – Idea

Imagine that

- I look out of a window and see a bird,
- it is black, approx. 25 cm long and has a rather yellow beak.

My daughter asks: What kind of bird is this?

My usual answer: This is *probably* a kind of blackbird (kos černý in Czech).

Probabilistic Classification – Idea

Imagine that

- I look out of a window and see a bird,
- it is black, approx. 25 cm long and has a rather yellow beak.

My daughter asks: What kind of bird is this?

My usual answer: This is *probably* a kind of blackbird (kos černý in Czech).

Here *probably* means that out of my extensive catalog of four kinds of birds that I can recognize, "blackbird" gets the highest degree of belief based on *features* of this particular bird.

Frequentists might say that the largest proportion of birds with similar features I have ever seen were blackbirds.

Probabilistic Classification – Idea

Imagine that

- I look out of a window and see a bird,
- it is black, approx. 25 cm long and has a rather yellow beak.

My daughter asks: What kind of bird is this?

My usual answer: This is *probably* a kind of blackbird (kos černý in Czech).

Here *probably* means that out of my extensive catalog of four kinds of birds that I can recognize, "blackbird" gets the highest degree of belief based on *features* of this particular bird.

Frequentists might say that the largest proportion of birds with similar features I have ever seen were blackbirds.

The degree of belief (Bayesian), or the relative frequency (frequentists), is the *probability*.

Basic Discrete Probability Theory

- A finite or countably infinite set Ω of *possible outcomes*, Ω is called *sample space*.

  Experiment: Roll one dice once. Sample space: $\Omega = \{1, \ldots, 6\}$

Basic Discrete Probability Theory

- A finite or countably infinite set Ω of *possible outcomes*, Ω is called *sample space*.

  Experiment: Roll one dice once. Sample space: $\Omega = \{1, \dots, 6\}$
- Each element ω of Ω is assigned a "probability" value $f(\omega)$, here f must satisfy
  - $f(\omega) \in [0, 1]$ for all $\omega \in \Omega$,
  - $\sum_{\omega \in \Omega} f(\omega) = 1$.

  If the dice is fair, then $f(\omega) = \frac{1}{6}$ for all $\omega \in \{1, \dots, 6\}$.

Basic Discrete Probability Theory

- A finite or countably infinite set Ω of *possible outcomes*, Ω is called *sample space*.

  Experiment: Roll one dice once. Sample space: $\Omega = \{1, \dots, 6\}$
- Each element ω of Ω is assigned a "probability" value $f(\omega)$, here f must satisfy
  - $f(\omega) \in [0, 1]$ for all $\omega \in \Omega$,
  - $\sum_{\omega \in \Omega} f(\omega) = 1$.

  If the dice is fair, then $f(\omega) = \frac{1}{6}$ for all $\omega \in \{1, \dots, 6\}$.
- An *event* is any subset E of Ω.
- The *probability* of a given event $E \subseteq \Omega$ is defined as

$$P(E) = \sum_{\omega \in E} f(\omega)$$

  Let E be the event that an odd number is rolled, i.e., $E = \{1, 3, 5\}$. Then $P(E) = \frac{1}{2}$.

Basic Discrete Probability Theory

- A finite or countably infinite set Ω of *possible outcomes*, Ω is called *sample space*.

  Experiment: Roll one dice once. Sample space: $\Omega = \{1, \ldots, 6\}$
- Each element ω of Ω is assigned a "probability" value $f(\omega)$, here f must satisfy
  - $f(\omega) \in [0, 1]$ for all $\omega \in \Omega$,
  - $\sum_{\omega \in \Omega} f(\omega) = 1$.

  If the dice is fair, then $f(\omega) = \frac{1}{6}$ for all $\omega \in \{1, \ldots, 6\}$.
- An *event* is any subset E of Ω.
- The *probability* of a given event $E \subseteq \Omega$ is defined as

$$P(E) = \sum_{\omega \in E} f(\omega)$$

  Let E be the event that an odd number is rolled, i.e., $E = \{1, 3, 5\}$. Then $P(E) = \frac{1}{2}$.
- Basic laws: $P(\Omega) = 1$, $P(\emptyset) = 0$, given disjoint sets A, B we have $P(A \cup B) = P(A) + P(B)$, $P(\Omega \setminus A) = 1 - P(A)$.

Conditional Probability and Independence

- $P(A \mid B)$ is the probability of A given B (assume $P(B) > 0$) defined by

  $$P(A \mid B) = P(A \cap B)/P(B)$$

  (We assume that B is all and only information known.)

  A fair dice: what is the probability that 3 is rolled assuming that an odd number is rolled? ... and assuming that an even number is rolled?

Conditional Probability and Independence

- $P(A \mid B)$ is the probability of A given B (assume $P(B) > 0$) defined by

  $$P(A \mid B) = P(A \cap B)/P(B)$$

  (We assume that B is all and only information known.)

  A fair dice: what is the probability that 3 is rolled assuming that an odd number is rolled? ... and assuming that an even number is rolled?

- A and B are independent if $P(A \cap B) = P(A) \cdot P(B)$.

  It is easy to show that if $P(B) > 0$, then

  $$A, B \text{ are independent iff } P(A \mid B) = P(A).$$

Random Variables and Random Vectors

- A *random variable* X is a function $X : \Omega \to \mathbb{R}$.
  A dice: $X : \{1, \dots, 6\} \to \{0, 1\}$ such that $X(n) = n \mod 2$.
- A *random vector* is a function $X : \Omega \to \mathbb{R}^d$.

Random Variables and Random Vectors

- A *random variable* X is a function $X : \Omega \to \mathbb{R}$.
  A dice: $X : \{1, \ldots, 6\} \to \{0, 1\}$ such that $X(n) = n \mod 2$.
- A *random vector* is a function $X : \Omega \to \mathbb{R}^d$.
  We use $X = (X_1, \ldots, X_d)$ where X_i is a random variable returning the i-th component of X.
- Consider random variables X_1, X_2 and Y. The variables X_1, X_2 are *conditionally independent given Y* if for all x_1, x_2 and y we have that

$$P(X_1 = x_1, X_2 = x_2 \mid Y = y) = P(X_1 = x_1 \mid Y = y) \cdot P(X_2 = x_2 \mid Y = y)$$

Random Vectors – Example

Let Ω be a space of colored geometric shapes that are divided into two categories (**1** and **0**).

Assume a random vector $X = (X_{color}, X_{shape}, X_{cat})$ where

- $X_{color} : \Omega \to \{red, blue\}$,
- $X_{shape} : \Omega \to \{circle, square\}$,
- $X_{cat} : \Omega \to \{\mathbf{1}, \mathbf{0}\}$.

The following tables give probability distribution of values:

category **1**:

	circle	square
red	0.2	0.02
blue	0.02	0.01

category **0**:

	circle	square
red	0.05	0.3
blue	0.2	0.2

Random Vectors – Example

Example:

$P(red, circle, \mathbf{1}) = P(X_{color} = red, X_{shape} = circle, X_{cat} = \mathbf{1}) = 0.2$

Random Vectors – Example

Example:
$P(red, circle, \mathbf{1}) = P(X_{color} = red, X_{shape} = circle, X_{cat} = \mathbf{1}) = 0.2$

"Summing over" all possible values of some variable(s) gives the distribution of the rest:

$$\begin{aligned} P(red, circle) &= P(X_{color} = red, X_{shape} = circle) \\ &= P(red, circle, \mathbf{1}) + P(red, circle, \mathbf{0}) \\ &= 0.2 + 0.05 = 0.25 \end{aligned}$$

Random Vectors – Example

Example:
$P(red, circle, \mathbf{1}) = P(X_{color} = red, X_{shape} = circle, X_{cat} = \mathbf{1}) = 0.2$

"Summing over" all possible values of some variable(s) gives the distribution of the rest:

$$\begin{aligned} P(red, circle) &= P(X_{color} = red, X_{shape} = circle) \\ &= P(red, circle, \mathbf{1}) + P(red, circle, \mathbf{0}) \\ &= 0.2 + 0.05 = 0.25 \end{aligned}$$

$$P(red) = 0.2 + 0.02 + 0.05 + 0.3 = 0.57$$

Random Vectors – Example

Example:
$P(red, circle, \mathbf{1}) = P(X_{color} = red, X_{shape} = circle, X_{cat} = \mathbf{1}) = 0.2$

"Summing over" all possible values of some variable(s) gives the distribution of the rest:

$$\begin{aligned} P(red, circle) &= P(X_{color} = red, X_{shape} = circle) \\ &= P(red, circle, \mathbf{1}) + P(red, circle, \mathbf{0}) \\ &= 0.2 + 0.05 = 0.25 \end{aligned}$$

$$P(red) = 0.2 + 0.02 + 0.05 + 0.3 = 0.57$$

Thus also, all conditional probabilities can be computed:

$$P(\mathbf{1} \mid red, circle) = \frac{P(red, circle, \mathbf{1})}{P(red, circle)} = \frac{0.2}{0.25} = 0.8$$

Bayesian Classification

Let Ω be a sample space (a universum) of all objects that can be classified. We assume a probability P on Ω.

Bayesian Classification

Let Ω be a sample space (a universum) of all objects that can be classified. We assume a probability P on Ω.

We consider the problem of **binary classification**:

- Let Y be the random variable for the category which takes values in $\{\mathbf{0}, \mathbf{1}\}$.

Bayesian Classification

Let Ω be a sample space (a universum) of all objects that can be classified. We assume a probability P on Ω.

We consider the problem of **binary classification**:

- Let Y be the random variable for the category which takes values in $\{\mathbf{0}, \mathbf{1}\}$.
- Let X be the random vector describing n features of a given instance, i.e., $X = (X_1, \ldots, X_n)$
  - Denote by $\vec{x} \in \mathbb{R}^n$ values of X,
  - and by $x_i \in \mathbb{R}$ values of X_i.

Bayesian Classification

Let Ω be a sample space (a universum) of all objects that can be classified. We assume a probability P on Ω.

We consider the problem of **binary classification**:

- Let Y be the random variable for the category which takes values in $\{\mathbf{0}, \mathbf{1}\}$.
- Let X be the random vector describing n features of a given instance, i.e., $X = (X_1, \dots, X_n)$
  - Denote by $\vec{x} \in \mathbb{R}^n$ values of X,
  - and by $x_i \in \mathbb{R}$ values of X_i.

Bayes classifier: Given a vector of feature values $\vec{x}$,

$$C^{Bayes}(\vec{x}) := \begin{cases} \mathbf{1} & \text{if } P(Y = \mathbf{1} \mid X = \vec{x}) \geq P(Y = \mathbf{0} \mid X = \vec{x}) \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Intuitively, C^{Bayes} assigns to $\vec{x}$ the most probable category it might be in.

Bayesian Classification – Example

Imagine a conveyor belt with apples and apricots.

A machine is supposed to correctly distinguish apples from apricots based on their weight and diameter.

Bayesian Classification – Example

Imagine a conveyor belt with apples and apricots.

A machine is supposed to correctly distinguish apples from apricots based on their weight and diameter.

That is,

- $Y \in \{\mathbf{1}, \mathbf{0}\}$
  (here our interpretation is $\mathbf{1} =$ apple, $\mathbf{0} =$ appricot)
- $X = (X_{weight}, X_{diam})$

Bayesian Classification – Example

Imagine a conveyor belt with apples and apricots.

A machine is supposed to correctly distinguish apples from apricots based on their weight and diameter.

That is,

- $Y \in \{\mathbf{1}, \mathbf{0}\}$
  (here our interpretation is $\mathbf{1}$ = apple, $\mathbf{0}$ = appricot)
- $X = (X_{weight}, X_{diam})$

We are given a fruit of a diameter of $5cm$ that weighs $40g$.

Bayesian Classification – Example

Imagine a conveyor belt with apples and apricots.

A machine is supposed to correctly distinguish apples from apricots based on their weight and diameter.

That is,

- $Y \in \{\mathbf{1}, \mathbf{0}\}$
  (here our interpretation is $\mathbf{1}$ = apple, $\mathbf{0}$ = appricot)
- $X = (X_{weight}, X_{diam})$

We are given a fruit of a diameter of $5cm$ that weighs $40g$.

The Bayes classifier compares $P(Y = \mathbf{1} \mid X = (40g, 5cm))$ with $P(Y = \mathbf{0} \mid X = (40g, 5cm))$ and selects the more probable category given the features.

Crucial question: Is such a classifier good?

There are other classifiers, e.g., one which compares the weight divided by 10 with the diameter and decides based on the answer, or maybe a classifier that sums the weight and the diameter and compares the result with a constant, etc.

Bayes Classifier

Let C be an arbitrary *classifier*, that is a function that to every feature vector $\vec{x} \in \mathbb{R}^n$ assigns a class from $\{\mathbf{0}, \mathbf{1}\}$.

Bayes Classifier

Let C be an arbitrary *classifier*, that is a function that to every feature vector $\vec{x} \in \mathbb{R}^n$ assigns a class from $\{\mathbf{0}, \mathbf{1}\}$.

Define the error of the classifier C by

$$E_C = P(Y \neq C)$$

(Here we slightly abuse notation and apply C to samples, technically we apply the composition $C \circ X$ of C and X which first determines the features using X and then classifies according to C).

Bayes Classifier

Let C be an arbitrary *classifier*, that is a function that to every feature vector $\vec{x} \in \mathbb{R}^n$ assigns a class from $\{\mathbf{0}, \mathbf{1}\}$.

Define the error of the classifier C by

$$E_C = P(Y \neq C)$$

(Here we slightly abuse notation and apply C to samples, technically we apply the composition $C \circ X$ of C and X which first determines the features using X and then classifies according to C).

The Bayes classifier C^{Bayes} minimizes E_C, that is

$$E_{C^{Bayes}} = \min_{C \text{ is a classifier}} E_C$$

Practical Use of Bayes Classifier

The crucial problem: The probability P is not known!
In particular, where to get $P(Y = \mathbf{1} \mid X = \vec{x})$?
Note that $P(Y = \mathbf{0} \mid X = \vec{x}) = 1 - P(Y = \mathbf{1} \mid X = \vec{x})$

Practical Use of Bayes Classifier

The crucial problem: The probability P is not known!
In particular, where to get $P(Y = \mathbf{1} \mid X = \vec{x})$?
Note that $P(Y = \mathbf{0} \mid X = \vec{x}) = 1 - P(Y = \mathbf{1} \mid X = \vec{x})$

Given no other assumptions, this requires a table showing
the probability of the category $\mathbf{1}$ for each possible feature vector $\vec{x}$.

Where do you get these probabilities?

Practical Use of Bayes Classifier

The crucial problem: The probability P is not known!
In particular, where to get $P(Y = \mathbf{1} \mid X = \vec{x})$?
Note that $P(Y = \mathbf{0} \mid X = \vec{x}) = 1 - P(Y = \mathbf{1} \mid X = \vec{x})$

Given no other assumptions, this requires a table showing the probability of the category **1** for each possible feature vector $\vec{x}$.

Where do you get these probabilities?

In some cases, the probabilities might come from the knowledge of the solved problem (e.g., applications in physics might be supported by a theory giving the probabilities).

In most cases, however, P is estimated from sampled data by

$$\bar{P}(Y = \mathbf{1} \mid X = \vec{x}) = \frac{\text{number of samples with } Y = \mathbf{1} \text{ and } X = \vec{x}}{\text{number of samples with } X = \vec{x}}$$

(We use $\bar{P}$ to denote an estimate of P from data.)

Estimating P

Consider a problem with $X = (X_1, X_2, X_3)$ where each X_i returns either 0 or 1. What might the data look like?

Estimating P

Consider a problem with $X = (X_1, X_2, X_3)$ where each X_i returns either 0 or 1. What might the data look like?

Part of the data table:

Y	X_1	X_2	X_3
1	1	0	1
1	0	1	1
0	1	0	1
0	0	0	1
1	0	0	0
0	1	1	1
...			

All data with $X_1 = 1$, $X_2 = 0$, $X_3 = 1$:

Y	X_1	X_2	X_3
1	1	0	1
1	1	0	1
0	1	0	1
0	1	0	1
1	1	0	1
1	1	0	1

Estimate: $\bar{P}(\mathbf{1} \mid 1, 0, 1) = 2/3$

Estimating P

Consider a problem with $X = (X_1, X_2, X_3)$ where each X_i returns either 0 or 1. What might the data look like?

Part of the data table:

Y	X_1	X_2	X_3
1	1	0	1
1	0	1	1
0	1	0	1
0	0	0	1
1	0	0	0
0	1	1	1
...			

All data with $X_1 = 1$, $X_2 = 0$, $X_3 = 1$:

Y	X_1	X_2	X_3
1	1	0	1
1	1	0	1
0	1	0	1
0	1	0	1
1	1	0	1
1	1	0	1

Estimate: $\bar{P}(\mathbf{1} \mid 1, 0, 1) = 2/3$

The probability table and the necessary data are typically too large!

Concretely, if all $X_1, \ldots, X_n$ are binary, there are 2^n probabilities $P(Y = \mathbf{1} \mid X = \vec{x})$, one for each possible $\vec{x} \in \{0, 1\}^n$.

Let's Look at It the Other Way Round

[Bayes,1764]

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

Let's Look at It the Other Way Round

[Bayes,1764]

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

Proof.

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{P(A \cap B)}{P(A)} \cdot P(A)}{P(B)} = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

□

Bayesian Classification

Determine the category for $\vec{x}$ by computing

$$P(Y = y \mid X = \vec{x}) = \frac{P(Y = y) \cdot P(X = \vec{x} \mid Y = y)}{P(X = \vec{x})}$$

for both $y \in \{\mathbf{0}, \mathbf{1}\}$ and deciding whether or not the following holds:

$$P(Y = \mathbf{1} \mid X = \vec{x}) \geq P(Y = \mathbf{0} \mid X = \vec{x})$$

Bayesian Classification

Determine the category for $\vec{x}$ by computing

$$P(Y = y \mid X = \vec{x}) = \frac{P(Y = y) \cdot P(X = \vec{x} \mid Y = y)}{P(X = \vec{x})}$$

for both $y \in \{\mathbf{0}, \mathbf{1}\}$ and deciding whether or not the following holds:

$$P(Y = \mathbf{1} \mid X = \vec{x}) \geq P(Y = \mathbf{0} \mid X = \vec{x})$$

So, to make the classifier, we need to compute the following:

- The prior $P(Y = \mathbf{1})$ (then $P(Y = \mathbf{0}) = 1 - P(Y = \mathbf{1})$)
- The conditionals $P(X = \vec{x} \mid Y = y)$ for $y \in \{\mathbf{0}, \mathbf{1}\}$ and for every $\vec{x}$

Estimating the Prior and Conditionals

- $P(Y = \mathbf{1})$ can be easily estimated from data by

$$\bar{P}(Y = \mathbf{1}) = \frac{\text{number of samples with } Y = \mathbf{1}}{\text{number of all samples}}$$

Estimating the Prior and Conditionals

- $P(Y = \mathbf{1})$ can be easily estimated from data by

$$\bar{P}(Y = \mathbf{1}) = \frac{\text{number of samples with } Y = \mathbf{1}}{\text{number of all samples}}$$

- If the dimension of features is small, $P(X = \vec{x} \mid Y = y)$ can be estimated from data similarly as $P(Y = \mathbf{1} \mid X = \vec{x})$ by

$$\bar{P}(X = \vec{x} \mid Y = y) = \frac{\text{number of samples with } Y = y \text{ and } X = \vec{x}}{\text{number of samples with } Y = y}$$

Unfortunately, for higher dimensional data too many samples are needed to estimate all $P(X = \vec{x} \mid Y = y)$ (there are too many $\vec{x}$'s).

So where is the advantage of using the Bayes thm.??

Estimating the Prior and Conditionals

- $P(Y = \mathbf{1})$ can be easily estimated from data by

$$\bar{P}(Y = \mathbf{1}) = \frac{\text{number of samples with } Y = \mathbf{1}}{\text{number of all samples}}$$

- If the dimension of features is small, $P(X = \vec{x} \mid Y = y)$ can be estimated from data similarly as $P(Y = \mathbf{1} \mid X = \vec{x})$ by

$$\bar{P}(X = \vec{x} \mid Y = y) = \frac{\text{number of samples with } Y = y \text{ and } X = \vec{x}}{\text{number of samples with } Y = y}$$

  Unfortunately, for higher dimensional data too many samples are needed to estimate all $P(X = \vec{x} \mid Y = y)$ (there are too many $\vec{x}$'s).

  So where is the advantage of using the Bayes thm.??

We introduce *independence assumptions* about the features!

Naive Bayes

- We assume that features are (conditionally) independent *given the category*. That is for all $\vec{x} = (x_1, \ldots, x_n)$ and $y \in \{\mathbf{0}, \mathbf{1}\}$ we **assume**:

$$\begin{aligned} P(X = x \mid Y = y) &= P(X_1 = x_1, \cdots, X_n = x_n \mid Y) \\ &= \prod_{i=1}^{n} P(X_i = x_i \mid Y = y) \end{aligned}$$

Naive Bayes

- We assume that features are (conditionally) independent *given the category*. That is for all $\vec{x} = (x_1, \ldots, x_n)$ and $y \in \{\mathbf{0}, \mathbf{1}\}$ we **assume**:

$$\begin{aligned} P(X = x \mid Y = y) &= P(X_1 = x_1, \cdots, X_n = x_n \mid Y) \\ &= \prod_{i=1}^{n} P(X_i = x_i \mid Y = y) \end{aligned}$$

- Therefore, we only need to specify $P(X_i = x_i \mid Y = y)$ for each possible pair of a feature-value x_i and $y \in \{\mathbf{0}, \mathbf{1}\}$.

Naive Bayes

- We assume that features are (conditionally) independent *given the category*. That is for all $\vec{x} = (x_1, \ldots, x_n)$ and $y \in \{\mathbf{0}, \mathbf{1}\}$ we **assume**:

$$\begin{aligned} P(X = x \mid Y = y) &= P(X_1 = x_1, \cdots, X_n = x_n \mid Y) \\ &= \prod_{i=1}^{n} P(X_i = x_i \mid Y = y) \end{aligned}$$

- Therefore, we only need to specify $P(X_i = x_i \mid Y = y)$ for each possible pair of a feature-value x_i and $y \in \{\mathbf{0}, \mathbf{1}\}$.

  Note that if all X_i are binary (values in $\{0, 1\}$), this requires specifying only $2n$ parameters:

$$P(X_i = 1 \mid Y = \mathbf{1}) \text{ and } P(X_i = 1 \mid Y = \mathbf{0}) \text{ for each } X_i$$

  as $P(X_i = 0 \mid Y = y) = 1 - P(X_i = 1 \mid Y = y)$ for $y \in \{\mathbf{0}, \mathbf{1}\}$.

Compared to specifying 2^n parameters without any independence assumption.

Estimating the marginal probabilities

Estimate the probabilities $P(X_i = x_i \mid Y = y)$ by

$$\bar{P}(X_i = x_i \mid Y = y) = \frac{\text{number of samples with } X_i = x_i \text{ and } Y = y}{\text{number of samples with } Y = y}$$

Estimating the marginal probabilities

Estimate the probabilities $P(X_i = x_i \mid Y = y)$ by

$$\bar{P}(X_i = x_i \mid Y = y) = \frac{\text{number of samples with } X_i = x_i \text{ and } Y = y}{\text{number of samples with } Y = y}$$

Example: Consider a problem with $X = (X_1, X_2, X_3)$ where each X_i returns either 0 or 1. The data is

Y	X_1	X_2	X_3
1	1	0	1
1	0	1	1
0	1	0	1
0	0	0	1
1	0	0	0
0	1	1	1

$\bar{P}(X_1 = 1 \mid Y = \mathbf{1}) = 1/3$ $\bar{P}(X_1 = 1 \mid Y = \mathbf{0}) = 2/3$
$\bar{P}(X_2 = 1 \mid Y = \mathbf{1}) = 1/3$ $\bar{P}(X_2 = 1 \mid Y = \mathbf{0}) = 1/3$
$\bar{P}(X_3 = 1 \mid Y = \mathbf{1}) = 2/3$ $\bar{P}(X_3 = 1 \mid Y = \mathbf{0}) = 1$

Naive Bayes – Example

Consider classification of geometric shapes:

$X_1 \in \{small, medium, large\}$

$X_2 \in \{red, blue, green\}$

$X_3 \in \{square, triangle, circle\}$

Naive Bayes – Example

Consider classification of geometric shapes:

$X_1 \in \{small, medium, large\}$

$X_2 \in \{red, blue, green\}$

$X_3 \in \{square, triangle, circle\}$

Assume that we have already estimated the following probabilities:

	$Y = \mathbf{1}$	$Y = \mathbf{0}$
$\bar{P}(Y)$	0.5	0.5
$\bar{P}(small \mid Y)$	0.4	0.4
$\bar{P}(medium \mid Y)$	0.1	0.2
$\bar{P}(large \mid Y)$	0.5	0.4
$\bar{P}(red \mid Y)$	0.9	0.3
$\bar{P}(blue \mid Y)$	0.05	0.3
$\bar{P}(green \mid Y)$	0.05	0.4
$\bar{P}(square \mid Y)$	0.05	0.4
$\bar{P}(triangle \mid Y)$	0.05	0.3
$\bar{P}(circle \mid Y)$	0.9	0.3

Does $(medium, red, circle)$ belong to the category $\mathbf{1}$?

	$Y = \mathbf{1}$	$Y = \mathbf{0}$
$\bar{P}(Y)$	0.5	0.5
$\bar{P}(medium \mid Y)$	0.1	0.2
$\bar{P}(red \mid Y)$	0.9	0.3
$\bar{P}(circle \mid Y)$	0.9	0.3

Denote $\vec{x} = (medium, red, circle)$.

	$Y = \mathbf{1}$	$Y = \mathbf{0}$
$\bar{P}(Y)$	0.5	0.5
$\bar{P}(medium \mid Y)$	0.1	0.2
$\bar{P}(red \mid Y)$	0.9	0.3
$\bar{P}(circle \mid Y)$	0.9	0.3

Denote $\vec{x} = (medium, red, circle)$.

$$
\begin{aligned}
P(Y = \mathbf{1} \mid X = \vec{x}) = & \\
&= P(\mathbf{1}) \cdot P(medium \mid \mathbf{1}) \cdot P(red \mid \mathbf{1}) \cdot P(circle \mid \mathbf{1}) \,/\, P(X = \vec{x}) \\
&\doteq 0.5 \cdot 0.1 \cdot 0.9 \cdot 0.9 \,/\, P(X = \vec{x}) = 0.0405 / P(X = \vec{x})
\end{aligned}
$$

	$Y = \mathbf{1}$	$Y = \mathbf{0}$
$\bar{P}(Y)$	0.5	0.5
$\bar{P}(medium \mid Y)$	0.1	0.2
$\bar{P}(red \mid Y)$	0.9	0.3
$\bar{P}(circle \mid Y)$	0.9	0.3

Denote $\vec{x} = (medium, red, circle)$.

$$P(Y = \mathbf{1} \mid X = \vec{x}) =$$
$$= P(\mathbf{1}) \cdot P(medium \mid \mathbf{1}) \cdot P(red \mid \mathbf{1}) \cdot P(circle \mid \mathbf{1}) \,/\, P(X = \vec{x})$$
$$\doteq 0.5 \cdot 0.1 \cdot 0.9 \cdot 0.9 \,/\, P(X = \vec{x}) = 0.0405/P(X = \vec{x})$$

$$P(Y = \mathbf{0} \mid X = \vec{x}) =$$
$$= P(\mathbf{0}) \cdot P(medium \mid \mathbf{0}) \cdot P(red \mid \mathbf{0}) \cdot P(circle \mid \mathbf{0}) \,/\, P(X = \vec{x})$$
$$\doteq 0.5 \cdot 0.2 \cdot 0.3 \cdot 0.3 \,/\, P(X = \vec{x}) = 0.009/P(X = \vec{x})$$

(Note that we used the estimates $\bar{P}$ of P to finish the computation above.)

	$Y = \mathbf{1}$	$Y = \mathbf{0}$
$\bar{P}(Y)$	0.5	0.5
$\bar{P}(medium \mid Y)$	0.1	0.2
$\bar{P}(red \mid Y)$	0.9	0.3
$\bar{P}(circle \mid Y)$	0.9	0.3

Denote $\vec{x} = (medium, red, circle)$.

$$\begin{aligned} P(Y = \mathbf{1} \mid X = \vec{x}) = & \\ & = P(\mathbf{1}) \cdot P(medium \mid \mathbf{1}) \cdot P(red \mid \mathbf{1}) \cdot P(circle \mid \mathbf{1}) \,/\, P(X = \vec{x}) \\ & \doteq 0.5 \cdot 0.1 \cdot 0.9 \cdot 0.9 \,/\, P(X = \vec{x}) = 0.0405/P(X = \vec{x}) \end{aligned}$$

$$\begin{aligned} P(Y = \mathbf{0} \mid X = \vec{x}) = & \\ & = P(\mathbf{0}) \cdot P(medium \mid \mathbf{0}) \cdot P(red \mid \mathbf{0}) \cdot P(circle \mid \mathbf{0}) \,/\, P(X = \vec{x}) \\ & \doteq 0.5 \cdot 0.2 \cdot 0.3 \cdot 0.3 \,/\, P(X = \vec{x}) = 0.009/P(X = \vec{x}) \end{aligned}$$

(Note that we used the estimates $\bar{P}$ of P to finish the computation above.)

Apparently,

$$P(Y = \mathbf{1} \mid X = \vec{x}) \doteq 0.0405/P(X = \vec{x}) > 0.009/P(X = \vec{x}) \doteq P(\mathbf{0} \mid X = \vec{x})$$

So we classify $\vec{x}$ to the category $\mathbf{1}$.

Estimating Probabilities in Practice

We already know that $P(X_i = x_i \mid Y = y)$ can be estimated by

$$\bar{P}(X_i = x_i \mid Y = y) = \ell_{y,x_i} \,/\, \ell_y$$

where

- ℓ_{y,x_i} = number of samples with $Y = y$ and $X_i = x_i$
- ℓ_y = number of samples with $Y = y$

Estimating Probabilities in Practice

We already know that $P(X_i = x_i \mid Y = y)$ can be estimated by

$$\bar{P}(X_i = x_i \mid Y = y) = \ell_{y,x_i} \,/\, \ell_y$$

where

- ℓ_{y,x_i} = number of samples with $Y = y$ and $X_i = x_i$
- ℓ_y = number of samples with $Y = y$

Problem: If, by chance, a rare value x_i of a feature X_i never occurs in the training data, we get

$$\bar{P}(X_i = x_i \mid Y = y) = 0 \quad \text{for both } y \in \{\mathbf{0}, \mathbf{1}\}$$

But then $\bar{P}(X = x) = 0$ for x containing the value x_i for X_i, and thus $\bar{P}(Y = y \mid X = x)$ is not well defined.
Moreover, $\bar{P}(Y = y) \cdot \bar{P}(X = x \mid Y = y) = 0$ (for $y \in \{\mathbf{0}, \mathbf{1}\}$) so even this cannot be used for classification.

Probability Estimation Example

Training data:

Size	Color	Shape	Class
small	red	circle	**1**
large	red	circle	**1**
small	red	triangle	**0**
large	blue	circle	**0**

Estimated probabilities:

	$Y = \mathbf{1}$	$Y = \mathbf{0}$
$\bar{P}(Y)$	0.5	0.5
$\bar{P}(small \mid Y)$	0.5	0.5
$\bar{P}(medium \mid Y)$	0	0
$\bar{P}(large \mid Y)$	0.5	0.5
$\bar{P}(red \mid Y)$	1	0.5
$\bar{P}(blue \mid Y)$	0	0.5
$\bar{P}(green \mid Y)$	0	0
$\bar{P}(square \mid Y)$	0	0
$\bar{P}(triangle \mid Y)$	0	0.5
$\bar{P}(circle \mid Y)$	1	0.5

Note that $\bar{P}(medium \mid \mathbf{1}) = P(medium \mid \mathbf{0}) = 0$ and thus also $\bar{P}(medium, red, circle) = 0$.

So what is $\bar{P}(\mathbf{1} \mid medium, red, circle)$?

Smoothing

- To account for estimation from small samples, probability estimates are adjusted or *smoothed*.

Smoothing

- To account for estimation from small samples, probability estimates are adjusted or *smoothed*.
- *Laplace smoothing* adds one to every count of feature values

  $$\tilde{P}(X_i = x_i \mid Y = y) = \frac{\ell_{y,x_i} + 1}{\ell_y + v_i}$$

  where
  - ℓ_y = number of training samples with $Y = y$,
  - ℓ_{y,x_i} = number of training samples with $Y = y$ and $X_i = x_i$,
  - v_i is the number of all distinct values of the variable X_i.

To understand note that

$$\ell_y = \sum_{x_i \text{ is a value of } X_i} \ell_{y,x_i}$$

and thus

$$\bar{P}(X_i = x_i \mid Y = y) = \ell_{y,x_i} \;/\; \sum_{x_i \text{ is a value of } X_i} \ell_{y,x_i}$$

$$\tilde{P}(X_i = x_i \mid Y = y) = (\ell_{y,x_i} + 1) \;/\; \sum_{x_i \text{ is a value of } X_i} (\ell_{y,x_i} + 1)$$

Laplace Smoothing Example

- Assume training set contains 10 samples of category **1**:
    - 4 small
    - 0 medium
    - 6 large

Laplace Smoothing Example

- Assume training set contains 10 samples of category **1**:
  - 4 small
  - 0 medium
  - 6 large
- Estimate parameters as follows
  - $\tilde{P}(\mathit{small} \mid \mathbf{1}) = (4+1)/(10+3) = 0.384$
  - $\tilde{P}(\mathit{medium} \mid \mathbf{1}) = (0+1)/(10+3) = 0.0769$
  - $\tilde{P}(\mathit{large} \mid \mathbf{1}) = (6+1)/(10+3) = 0.538$

Continuous Features

Ω may be (potentially) continuous, X_i may assign a continuum of values in $\mathbb{R}$.

Continuous Features

Ω may be (potentially) continuous, X_i may assign a continuum of values in $\mathbb{R}$.

- The probabilities are computed using *probability density* $p : \mathbb{R} \to \mathbb{R}^+$.

  A random variable $X : \Omega \to \mathbb{R}^+$ has a density $p : \mathbb{R} \to \mathbb{R}^+$ if for every interval $[a, b]$ we have

  $$P(a \leq X \leq b) = \int_a^b p(x)dx$$

  Usually, $P(X_i \mid Y = y)$ is used to denote the *density* of X_i conditioned on $Y = y$.

Continuous Features

Ω may be (potentially) continuous, X_i may assign a continuum of values in $\mathbb{R}$.

- The probabilities are computed using *probability density* $p : \mathbb{R} \to \mathbb{R}^+$.

  A random variable $X : \Omega \to \mathbb{R}^+$ has a density $p : \mathbb{R} \to \mathbb{R}^+$ if for every interval $[a, b]$ we have

  $$P(a \leq X \leq b) = \int_a^b p(x)dx$$

  Usually, $P(X_i \mid Y = y)$ is used to denote the *density* of X_i conditioned on $Y = y$.
- The densities $P(X_i \mid Y = y)$ are usually estimated using Gaussian densities as follows:
  - Estimate the mean μ_{iy} and the standard deviation σ_{iy} based on training data.
  - Then put

  $$\bar{P}(X_i \mid Y = y) = \frac{1}{\sigma_{iy}\sqrt{2\pi}} \exp\left(\frac{-(X_i - \mu_{iy})^2}{2\sigma_{iy}^2}\right)$$

Comments on Naive Bayes

- Tends to work well despite rather a strong assumption of conditional independence of features.

Comments on Naive Bayes

- Tends to work well despite rather a strong assumption of conditional independence of features.
- Experiments show that it is quite competitive with other classification methods.

  Even if the probabilities are not accurately estimated, it often picks the correct maximum probability category.

Comments on Naive Bayes

- Tends to work well despite rather a strong assumption of conditional independence of features.
- Experiments show that it is quite competitive with other classification methods.

  Even if the probabilities are not accurately estimated, it often picks the correct maximum probability category.
- Directly constructs a model from parameter estimates that are calculated from the training data.

Comments on Naive Bayes

- Tends to work well despite rather a strong assumption of conditional independence of features.
- Experiments show that it is quite competitive with other classification methods.

  Even if the probabilities are not accurately estimated, it often picks the correct maximum probability category.
- Directly constructs a model from parameter estimates that are calculated from the training data.
- Typically handles outliers and noise well.

Comments on Naive Bayes

- Tends to work well despite rather a strong assumption of conditional independence of features.
- Experiments show that it is quite competitive with other classification methods.

  Even if the probabilities are not accurately estimated, it often picks the correct maximum probability category.
- Directly constructs a model from parameter estimates that are calculated from the training data.
- Typically handles outliers and noise well.
- Missing values are easy to deal with (simply average overall missing values in feature vectors).

Bayesian Networks (Basic Information)

In the Naive Bayes, we have assumed that *all* features $X_1, \ldots, X_n$ are independent.

Bayesian Networks (Basic Information)

In the Naive Bayes, we have assumed that *all* features $X_1, \ldots, X_n$ are independent.

This is usually not realistic.

E.g. Variables "rain" and "grass wet" are (usually) strongly dependent.

Bayesian Networks (Basic Information)

In the Naive Bayes, we have assumed that *all* features $X_1, \ldots, X_n$ are independent.

This is usually not realistic.

E.g. Variables "rain" and "grass wet" are (usually) strongly dependent.

What if we return some dependencies?

(But now in a well-defined sense.)

Bayesian Networks (Basic Information)

In the Naive Bayes, we have assumed that *all* features $X_1, \ldots, X_n$ are independent.

This is usually not realistic.

E.g. Variables "rain" and "grass wet" are (usually) strongly dependent.

What if we return some dependencies?

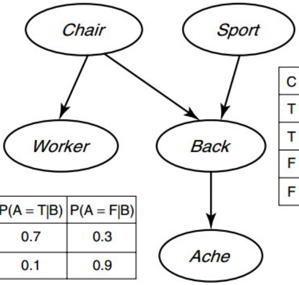
(But now in a well-defined sense.)

Bayesian networks are a graphical model that uses a directed acyclic graph to specify dependencies among variables.

Bayesian Networks – Example

P(C = T)	P(C = F)
0.8	0.2

P(S = T)	P(S = F)
0.02	0.98



C	P(W = T|C)	P(W = F|C)
T	0.9	0.1
F	0.01	0.99

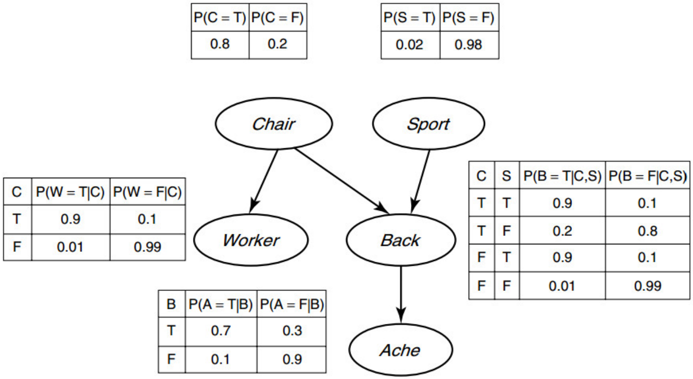
C	S	P(B = T|C,S)	P(B = F|C,S)
T	T	0.9	0.1
T	F	0.2	0.8
F	T	0.9	0.1
F	F	0.01	0.99

B	P(A = T|B)	P(A = F|B)
T	0.7	0.3
F	0.1	0.9

Now, e.g.,

$$P(C, S, W, B, A) = P(C) \cdot P(S) \cdot P(W \mid C) \cdot P(B \mid C, S) \cdot P(A \mid B)$$

Bayesian Networks – Example



P(C = T)	P(C = F)
0.8	0.2

P(S = T)	P(S = F)
0.02	0.98

C	P(W = T|C)	P(W = F|C)
T	0.9	0.1
F	0.01	0.99

C	S	P(B = T|C,S)	P(B = F|C,S)
T	T	0.9	0.1
T	F	0.2	0.8
F	T	0.9	0.1
F	F	0.01	0.99

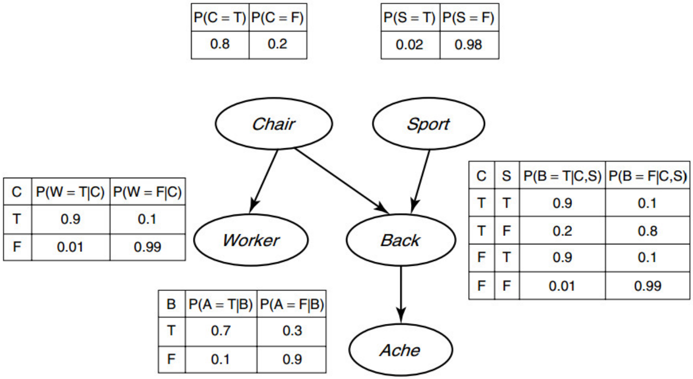
B	P(A = T|B)	P(A = F|B)
T	0.7	0.3
F	0.1	0.9

Now, e.g.,

$$P(C, S, W, B, A) = P(C) \cdot P(S) \cdot P(W \mid C) \cdot P(B \mid C, S) \cdot P(A \mid B)$$

Now, we may, e.g., infer the probability $P(C = T \mid A = T)$ that we sit in the wrong chair, assuming that our back aches.

Bayesian Networks – Example



P(C = T)	P(C = F)
0.8	0.2

P(S = T)	P(S = F)
0.02	0.98

C	P(W = T|C)	P(W = F|C)
T	0.9	0.1
F	0.01	0.99

C	S	P(B = T|C,S)	P(B = F|C,S)
T	T	0.9	0.1
T	F	0.2	0.8
F	T	0.9	0.1
F	F	0.01	0.99

B	P(A = T|B)	P(A = F|B)
T	0.7	0.3
F	0.1	0.9

Now, e.g.,

$$P(C, S, W, B, A) = P(C) \cdot P(S) \cdot P(W \mid C) \cdot P(B \mid C, S) \cdot P(A \mid B)$$

Now, we may, e.g., infer the probability $P(C = T \mid A = T)$ that we sit in the wrong chair, assuming that our back aches.
We have to store only 10 numbers as opposed to $2^5 - 1$ possible probabilities for all vectors of values of C, S, W, B, A.

Bayesian Networks – Learning & Naive Bayes

Many algorithms have been developed for learning:

- the structure of the graph of the network,
- the *conditional probability tables*.

The methods are based on maximum-likelihood estimation, gradient descent, etc.

Automatic procedures are usually combined with expert knowledge.

Bayesian Networks – Learning & Naive Bayes

Many algorithms have been developed for learning:

- the structure of the graph of the network,
- the *conditional probability tables*.

The methods are based on maximum-likelihood estimation, gradient descent, etc.

Automatic procedures are usually combined with expert knowledge.

Can you express the naive Bayes for $Y, X_1, \ldots, X_n$ using a Bayesian network?