6b. Data mining - classification

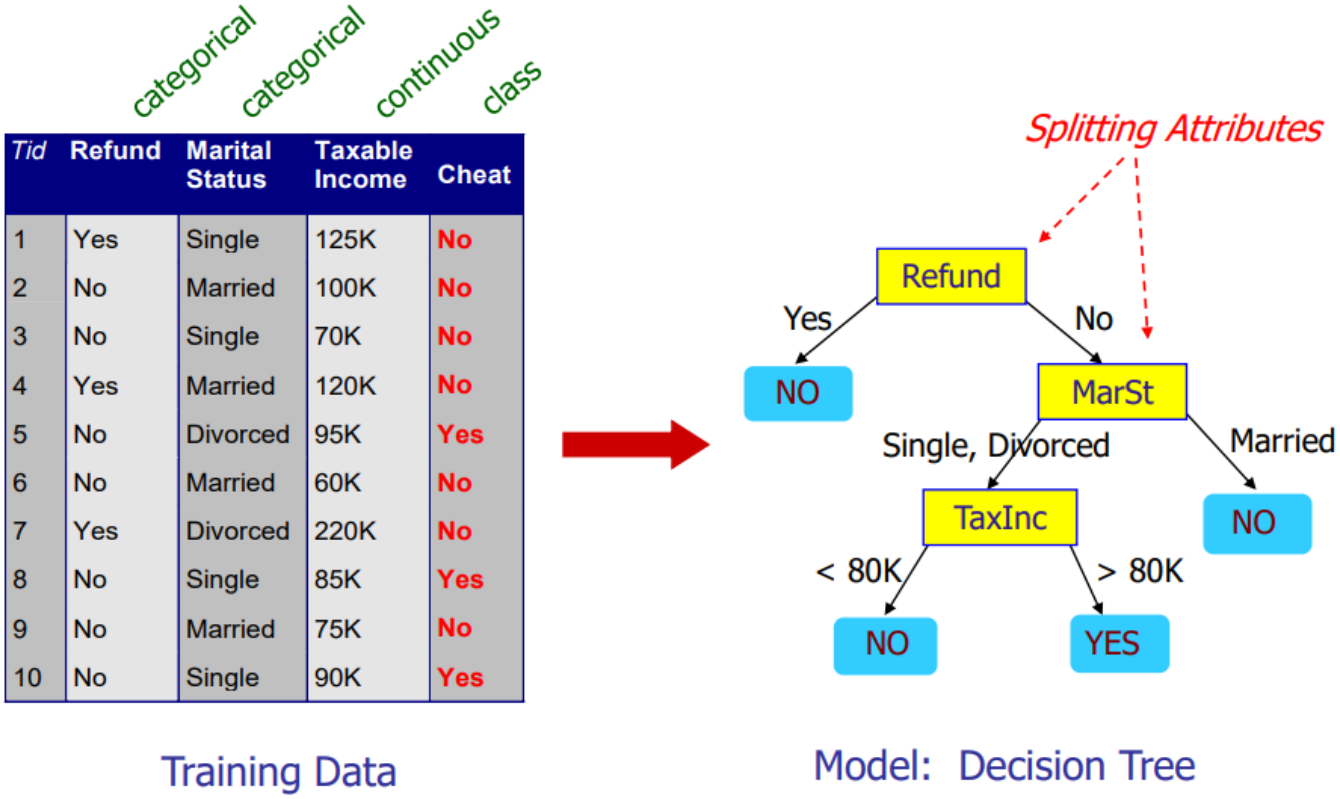
Definitions

Given a collection of class labels a collection of data objects labelled with a class label, the objective is to **find** a **descriptive profile** of **each class** which will allow us later on to the assignment of unlabelled objects to the appropriate class.

- **Training set:** collection of *labeled* data objects used to learn the classification model
- **Test set:** collection of labeled data objects used to validate the classification model
  There are several classification techniques, explained below.

Decision trees

The following is an example of a training dataset and a decision tree.



There are many algorithms to build a decision tree

Hunt's Algorithm

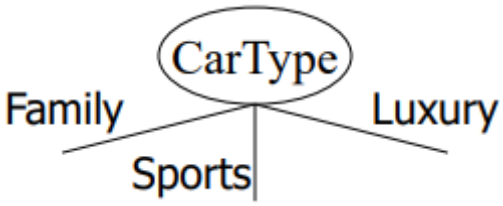
Given D as the entire training dataset and D_t the dataset that reach a node(containing a splitting attribute) t, we have:

- if D_t contains records that belong to more than one class:

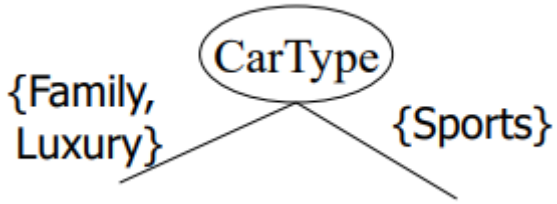
    - select the *best* attribute A on which to split D_t and label node t as A
- if D_t contains recordss that belong to the same class y_t, then t is a leaf node labeled as y_t
- if D_t is an empty dataset then t is a leaf node labeled as the default class y_d

Splitting

Multi-way split: Use as many partitions as distinct values



Binary split: Divides values into subsets



For a ***continuous*** attribute, we can use either a **discretization-based** approach to form an ordinal categorical attribute or a **binary decision approach** where we consider all possible values and then find the best cut.

Impurity

Attributes with homogeneous class distribution are preferred to create a **split**, thus we need a measure of the node impurity that roughly tells us the class distribution

Gini Index

- Gini Index for a given node t

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

$p(j \mid t)$ is the relative frequency of class j at node t

  - Maximum (1 - $1/n_c$) when records are equally distributed among all classes, implying higher impurity degree
  - Minimum (0.0) when all records belong to one class, implying lower impurity degree

C1	0
C2	6
Gini=0.000	

C1	1
C2	5
Gini=0.278	

C1	2
C2	4
Gini=0.444	

C1	3
C2	3
Gini=0.500	

Entropy(or INFO)

- Entropy at a given node t

$$Entropy(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)$$

$p(j \mid t)$ is the relative frequency of class j at node t

  - Maximum (log n_c) when records are equally distributed among all classes, implying higher impurity degree
  - Minimum (0.0) when all records belong to one class, implying lower impurity degree

- Entropy based computations are similar to GINI index computations

Stopping criteria

Stop expanding a node when all the records belong to the same class or when all the records have similar values.

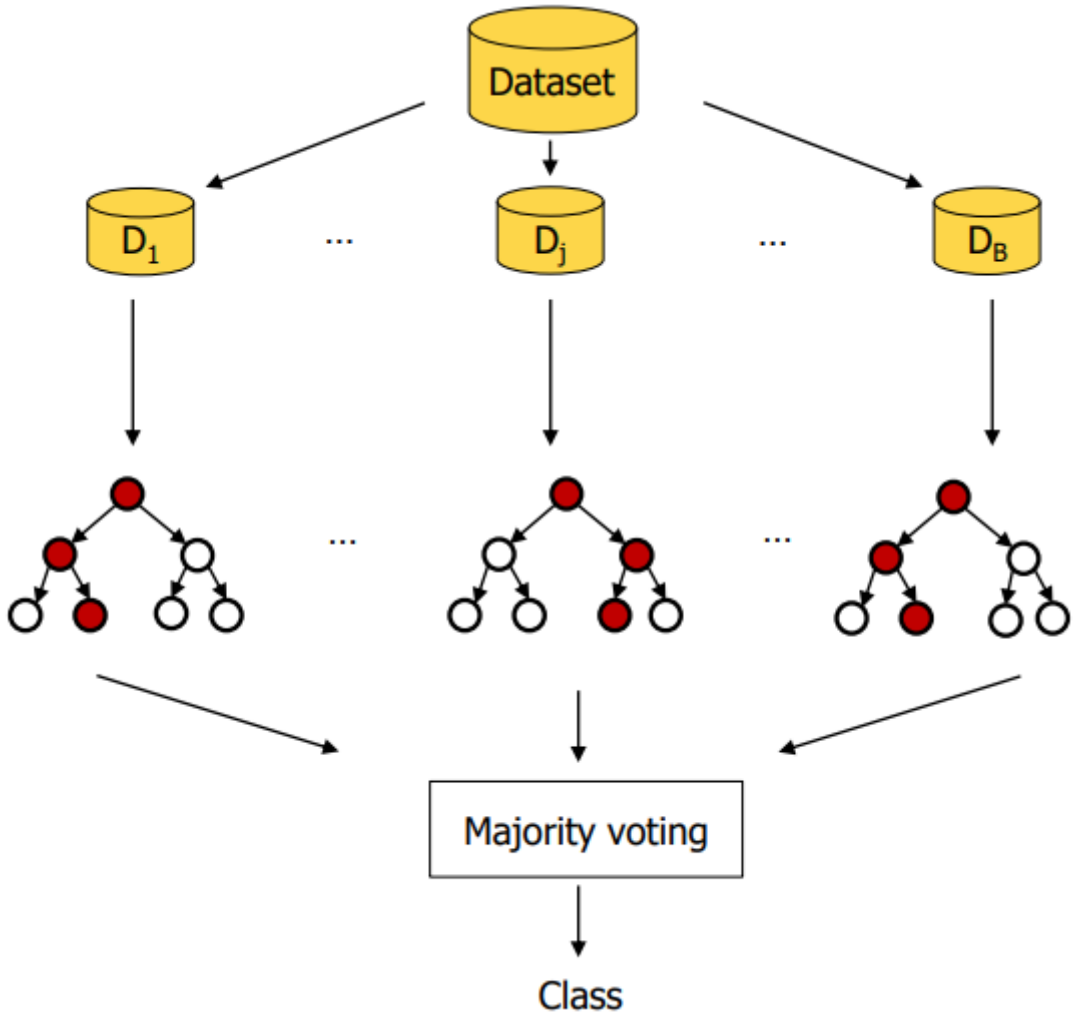
We can also use a **pre/post-pruning** approach in case of **overfitting**(when a model is trained on too specific datasets)

Evaluation of decision trees

- Accuracy
  - For simple datasets, comparable to other classification techniques
- Interpretability
  - Model is interpretable for small trees
  - Single predictions are interpretable
- Incrementality
  - Not incremental
- Efficiency
  - Fast model building
  - Very fast classification
- Scalability
  - Scalable both in training set size and attribute number
- Robustness
  - Difficult management of missing data

Random forest

Basically it's a set of decision trees where the outcome is decided on a majority-voting basis. Trees are trained on a *random sample with replacement* dataset



A random forest

Algorithm

Given a training set D of n instances with p features.

- For $b = 1, \ldots, B$
  - Sample randomly with remplacement n' training examples to generate D_b
  - Train a classification tree on D_b
    - During the tree construction, for each candidate split
      - $m << p$ random features are selected($m \approx \sqrt{p}$)
      - best split is computed among these features
- Class is assigned by majority voting among the B predictions

Evaluation of random forest

- Accuracy
  - Higher than decision trees
- Interpretability
  - Model and prediction are not interpretable
    - A prediction may be given by hundreds of trees
  - Provide global feature importance
    - an estimate of which features are important in the classification
- Incrementality
  - Not incremental
- Efficiency
  - Fast model building
  - Very fast classification
- Scalability
  - Scalable both in training set size and attribute number
- Robustness
  - Robust to noise and outliers

K-nearest neighbour

It's an instance-based classifier, as it stores training records to predict class label of unseen cases

Nearest Neighbor classifier

In order to work it needs:

- set of training records
- distance metric to compute distance
- value of k, the number of neighbor to retrieve
  To classify an unknown record:

1. Compute distance to other training records
2. Identify k nearest records
3. Use class labels of nearest neighbor to determine the class label of unknown record(e.g. by taking majority vote)

Choosing K

Choosing the value of k is important since:

- if k is too **small**, the model becomes sensitive to noise points
- if k is too **big**, the neighborhood may include points from other classes

Scaling issues

Attribute domain should be normalized.
There's also the problem with **high dimensional data** due to the **curse of dimensionality**

Evaluation of KNN

- Accuracy
  - Comparable to other classification techniques for simple datasets
- Interpretability
  - Model is not interpretable
  - Single predictions can be ”described” by neighbors
- Incrementality
  - Incremental
  - Training set *must* be available
- Efficiency
  - (Almost) no model building
  - Slower classification, requires computing distances
- Scalability
  - Weakly scalable in training set size
  - Curse of dimensionality for increasing attribute number
- Robustness
  - Depends on distance computation

Bayesian classification

The model is based on the probability that a new object belongs to a certain class and uses the **Bayes** theorem

Let C and X be random variables

$$P(C,X) = P(C|X)\,P(X)$$

$$P(C,X) = P(X|C)\,P(C)$$

Hence

$$P(C|X)\,P(X) = P(X|C)\,P(C)$$

and also

$$P(C|X) = P(X|C)\,P(C) / P(X)$$

*P(C,X) means "Probability that C and X occurs together". P(C|X) means "Probability that C happens given X".

Intro

Let's declare **C** any class label and **X** the record to be classified.
We then compute $P(C|X)$(aka probability that X belong to C) for all classes.
Assign then X to the class with the highest $P(C|X)$.
By applying the theorem we have: $P(C|X) = P(X|C) \times P(C)/P(X)$
but:

- $P(X)$ costant for all C
- $P(C) = N_c/N$

Computing P(X|C)

However how to estimate $P(X|C)$? We can use a naive hypothesis, by decomposing the x record into the set of its attributes $< x_1, x_2, .., x_k >$ and then we compute $P(x_1|C) \times P(x_2|C) \times \cdots \times P(x_k|C)$ by assuming these attributes are independent.
To compute $P(x_k|C)$ for discrete attributes $\rightarrow |x_{kC}|/N$
For continous attributes,use probability distribution

Example

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

outlook	
P(sunny|p) = 2/9	**P(sunny|n) = 3/5**
P(overcast|p) = 4/9	**P(overcast|n) = 0**
P(rain|p) = 3/9	**P(rain|n) = 2/5**
temperature	
P(hot|p) = 2/9	**P(hot|n) = 2/5**
P(mild|p) = 4/9	**P(mild|n) = 2/5**
P(cool|p) = 3/9	**P(cool|n) = 1/5**
humidity	
P(high|p) = 3/9	**P(high|n) = 4/5**
P(normal|p) = 6/9	**P(normal|n) = 2/5**
windy	
P(true|p) = 3/9	**P(true|n) = 3/5**
P(false|p) = 6/9	**P(false|n) = 2/5**

P(p) = 9/14
P(n) = 5/14

Data to be labeled

X = <rain, hot, high, false>

For class p

$P(X|p)\cdot P(p) =$
$= P(rain|p)\cdot P(hot|p)\cdot P(high|p)\cdot P(false|p)\cdot P(p)$
$= 3/9\cdot 2/9\cdot 3/9\cdot 6/9\cdot 9/14 = 0.010582$

For class n

$P(X|n)\cdot P(n) =$
$= P(rain|n)\cdot P(hot|n)\cdot P(high|n)\cdot P(false|n)\cdot P(n)$
$= 2/5\cdot 2/5\cdot 4/5\cdot 2/5\cdot 5/14 = 0.018286$

P(rain|P) is 3/9 where 9 are the number of total records with class P and 3 is the number of P records with rain as outlook

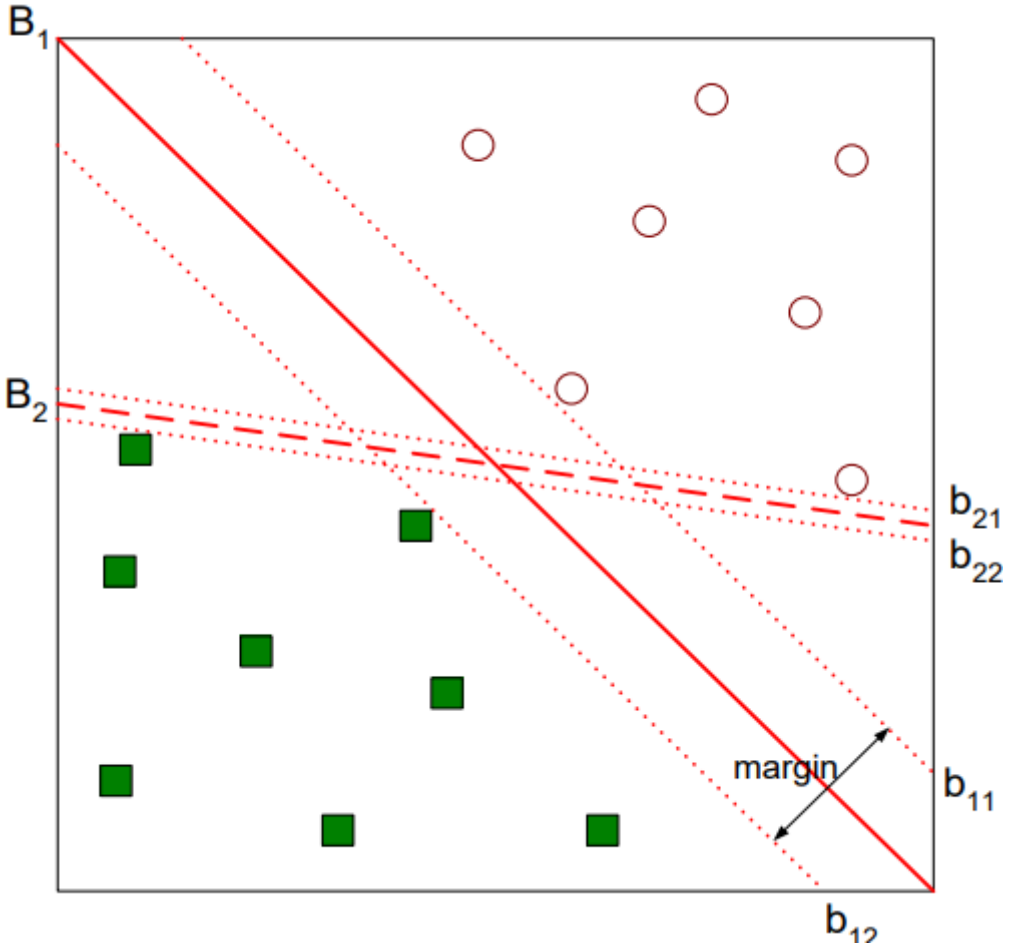
Evaluation of Naive Bayes Classifiers

- Accuracy
  - Similar or lower than decision trees
    - Naïve hypothesis simplifies model
- Interpretability
  - Model and prediction are not interpretable
    - The weights of contributions in a single prediction may be used to explain
- Incrementality
  - Fully incremental
  - Does *not* require availability of training data
- Efficiency
  - Fast model building
  - Very fast classification
- Scalability
  - Scalable both in training set size and attribute number
- Robustness
  - Affected by attribute correlation

Support Vector Machine

Approach used to usually address a binary classification problem.

We can start by finding a linear hyperplane(a decision boundary) that splits data.



Hyperplane that splits data in green squares and empty circles.
The idea is to find the hyperplane that maximises the margin. In this case B1 is better than B2.

If the decision boundary is not linear, we first transform the data into higher dimension space in order to have a linear decision boundary.

Evaluation of SVC

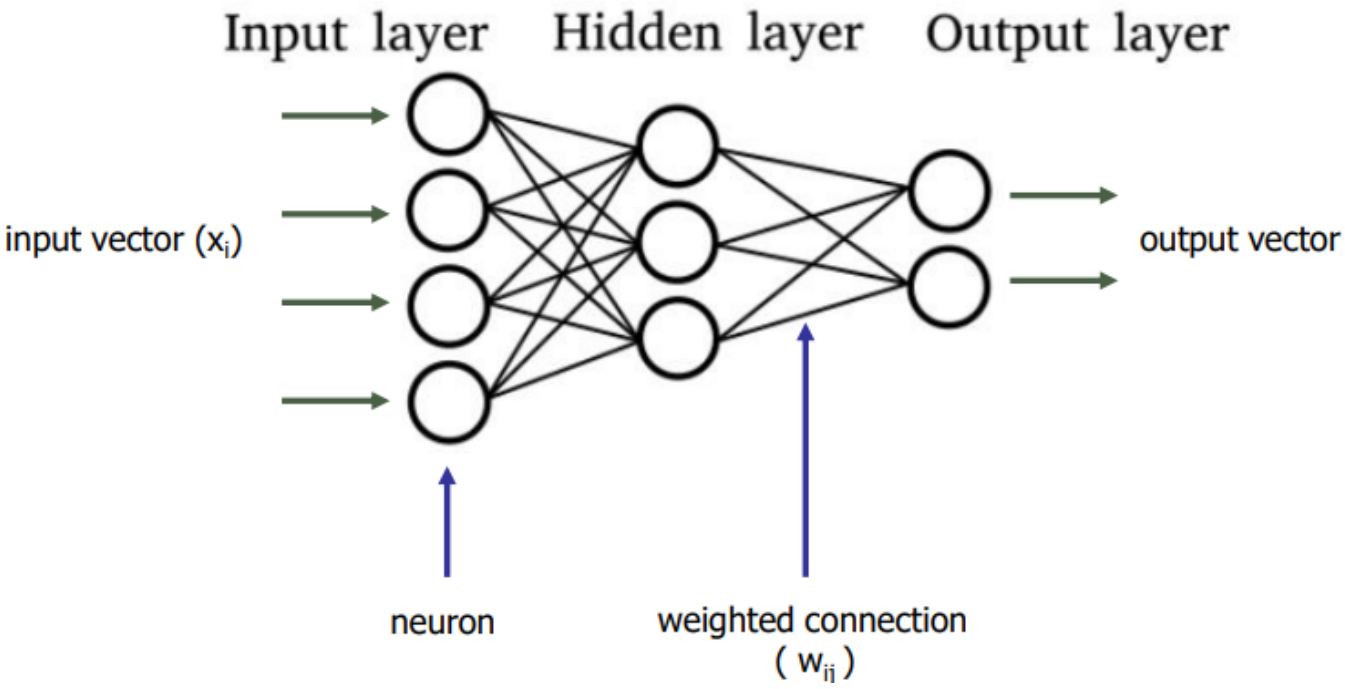
- Accuracy
  - Among best performers
- Interpretability
  - Model and prediction are not interpretable
    - Black box model
- Incrementality
  - Not incremental
- Efficiency
  - Model building requires significant parameter tuning
  - Very fast classification
- Scalability
  - Medium scalable both in training set size and attribute number
- Robustness
  - Robust to noise and outliers

Artificial Neural Networks

Inspired to the structure of the human brain.
Neurons are the elaboration units, **synopsis** are the connection.
Several kind of neural networks, based on the task.

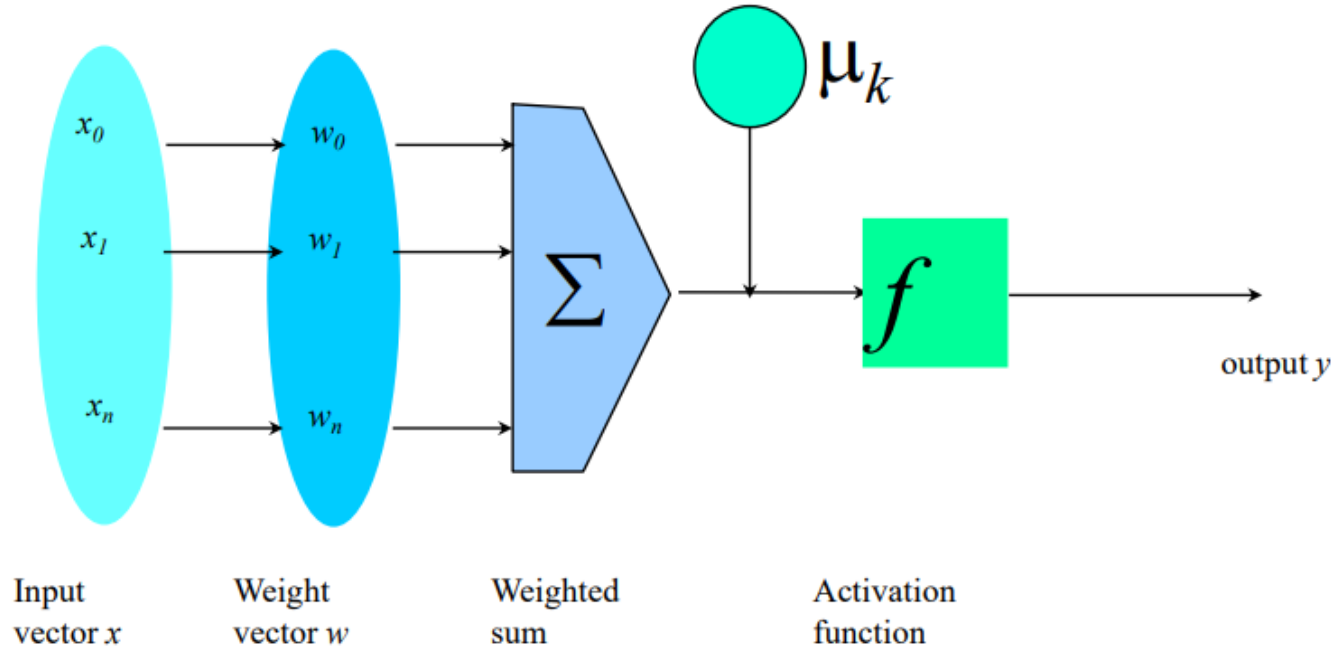
Feed Forward Neural Network

First proposed model, mainly used to work with numerical data.



Structure of a FFNN

Inputs are the attributes of an object. In case of a continuos attribute, often a normalization is applied before feeding it into the NN.
There is an **output** node for each **class label**: the result is a probability said object belong to a certain class.



Structure of a FFNN from a logical point of view
Activation function: Similutes biological stimula(?) to the input and provies **non-linearity** to

the computation.
μ_k is also called the correction function.

Activation Functions

There can be several **activation functions:**

- **Sigmoid/Tanh:** Often used in the **hidden/output** layer to produce values in a range of $[0, 1]$ and $[-1, 1]$ respectively
- **Binary step:** output 1 when input is non-zero. Not really used
- **ReLU-Rectified Linear Unit:** Used in deep networks(e.g. CNNs) as it avoids *vanishing gradient*(il gradiente si fa sempre più piccolo fino a sparire)
- **Softmax:** applied **only** to the output layer and works by considering *all* neurons in the layer. After the softmax, the output vector can be interpreted as a discrete distribution of probability

Building a FFNN

For each node, a **set of weights** and **offset values** are defined.
Then an iterative approach is used.
Initially, random values for weights and offset are assigned.
Then one instance in the training set is processed:

- For each neuron, **compute result** by taking into account weights,activation function and offset
- **Forward** propagation up to the output layer
- **Evaluation of the error** with the expected result
- **Backpropagation** of the error by updating weights and offset
  The process ends when:
- a certain % accuracy is achieved-
- max number of epochs(iteration) is reached
- a certain % of error is achieved

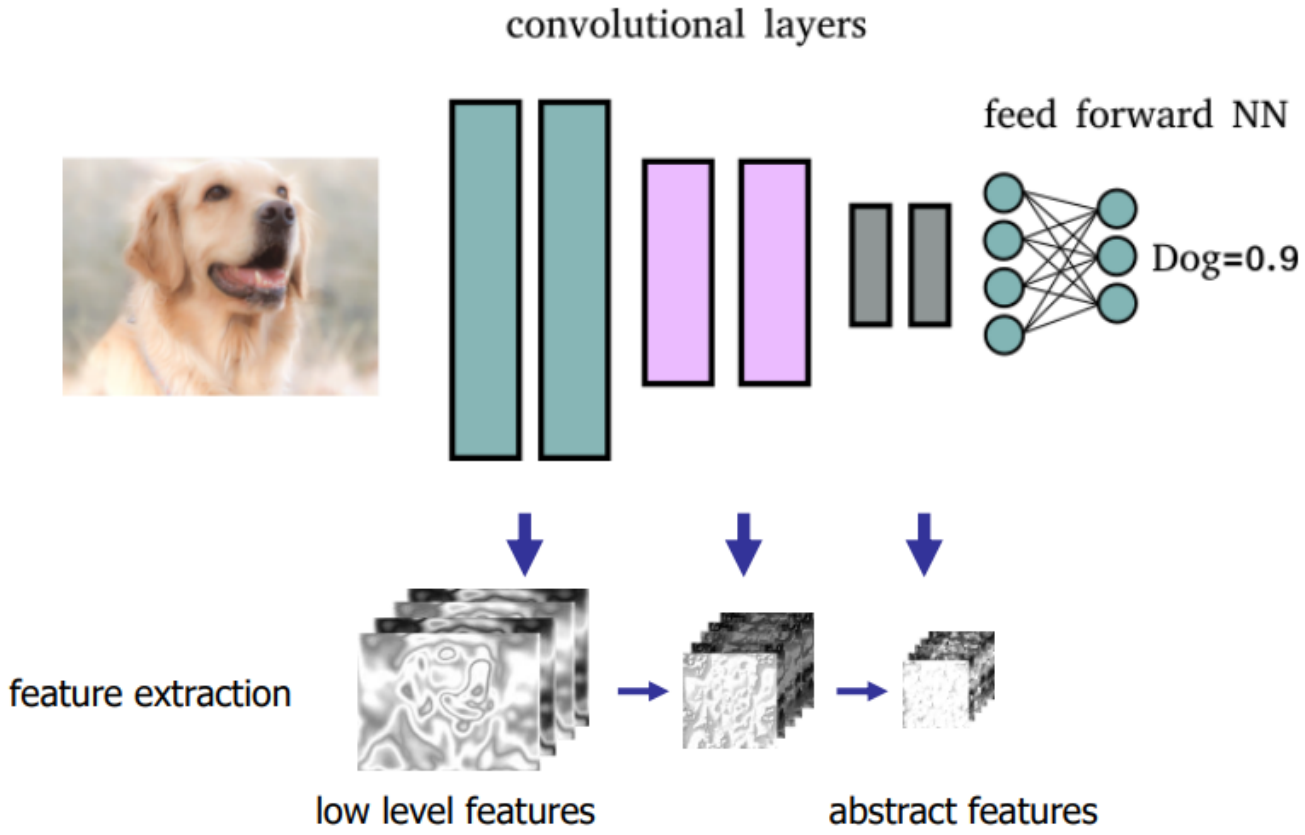
Evaluation of FFNN

- Accuracy
  - Among best performers
- Interpretability
  - Model and prediction are not interpretable
    - Black box model
- Incrementality
  - Not incremental

- Efficiency
  - Model building requires *very complex* parameter tuning
    - It requires significant time
  - Very fast classification
- Scalability
  - Medium scalable both in training set size and attribute number
- Robustness
  - Robust to noise and outliers
  - Requires large training set
    - Otherwise unstable when tuning parameters

Convolutional Neural Networks

Allows automatic extraction of features

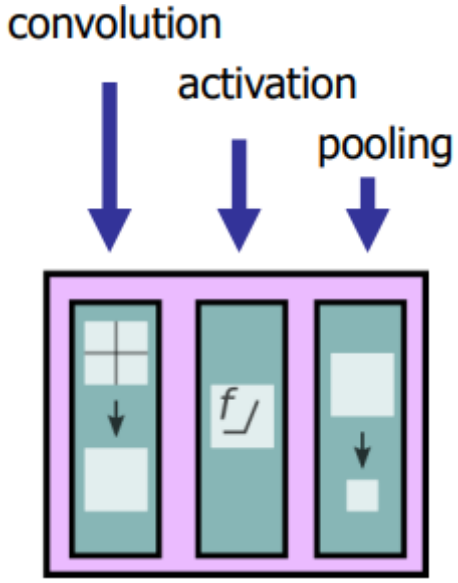


Example of how a CNN works

Convolutional Layer

A typical convolutional layer:

- **convolutional stage:** feature extraction by means of lots of filters
- **sliding filters activation**: apply an activation function to the input tensor
- **pooling:** tensor downsampling



A singole convolutional layer

Convolution

A **sliding filter**,a filter that moves over a tensor(n-dimensional vector), produces the values of the output tensor.
This filter contains the trainable weights of the neural network

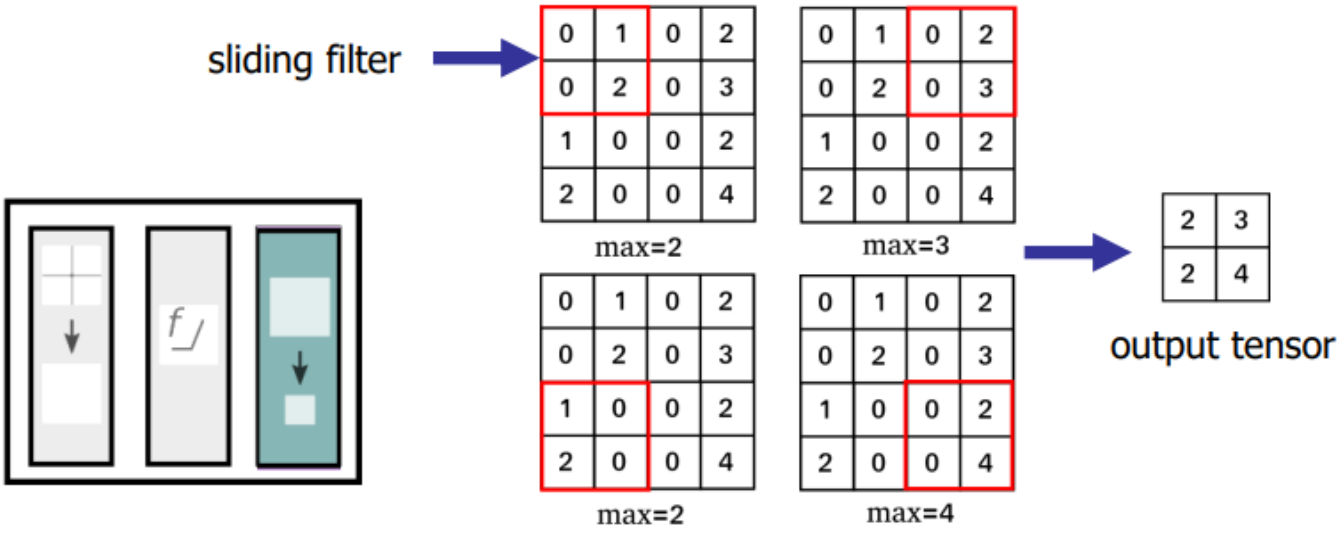
Activation

ReLU is tipically used for **CNN**.
An activation function is used on each function

Pooling

Performs a tensor **downsampling**.
A **sliding filter** moves over the tensor in a disjoint way. Each time, the filter replace the values in the tensor with a summary statistic.

Maxpool: the most common, it computes the maximum value as statistic



Recurrent Neural Network

Mostly used in time series, speech recognition and text processing.
It process **sequential data.**
Differently from the FFNN, they are able to keep a state which evolves during time.

Autoencoders

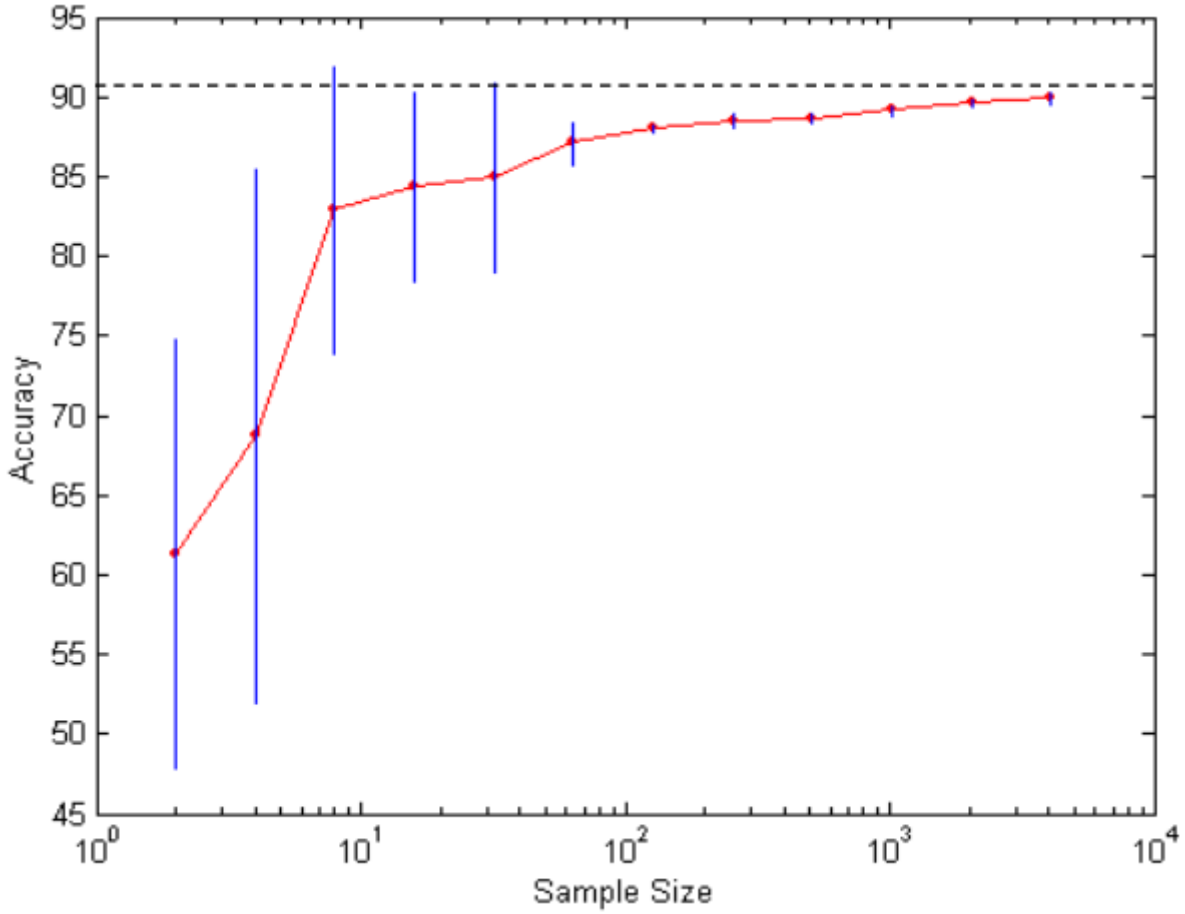
Used to remove noises from images.
Firstly, a compression-step is used to reduce noice, then a decompression step is performed.

Word Embeddings(Word2vec)

Words are represented in a n-dimensional space.
This allows to place words based on their meaning and semantic

Model Evaluation

The accuracy of a model depends mostly on the cardinality of the training/learning set



*Learning curve in relation to sample size

Partitioning Data

Holdout

Uses a **fixed partitioning** approach,that is, 80% of data set is reserved for the **training set**, the other 20% for **test set**.
Most appropriate for large datasets.
This approach can be repeated several times

Cross validation

Firstly, data is partitioned into **k** disjoint subsets(called **folds**).
For **k** times, train on $k-1$ partitions and test on the remaining one.
E.g.: Given $k=3$ with partitions **A,B,C**
Iteration 1: Training on A,B and test on C
Iteration 2: Training on A,C and test on B
Iteration 3: Training on B,C and test on A

In this way each data partition is used for both testing and training.

Leave-one-out

Cross validation with $k = n$ where n is the number of elements.
In this way, each partition contains one element only.
Obviously it's appropriate for **small datasets only**

Metric for evaluating model quality

To compute the accuracy of a model a **confusion matrix** is used

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a	b
	Class=No	c	d

a: TP (true positive)
b: FN (false negative)
c: FP (false positive)
d: TN (true negative)

Confusion Matrix for a binary classification problem. TP and TN correspond to a correct classification

Accuracy

Thus, the **accuracy** is computed as follows:
Accuracy=$\frac{\text{Number of correctly classified objects}}{\text{Number of classified objects}}$
It's not always a reliable measure in case there's an unbalanced distribution of classes.
Let's considering the following scenario

Consider a binary problem

- Cardinality of Class 0 = 9900
- Cardinality of Class 1 = 100

In this case, we have an accuracy of 99%, since most of the elements belong to **Class 0**.

Recall and Precision

We can use other more precise measures by evaluating separately for each class C

- **Recall(r):** $\frac{\text{Number of objects correctly assigned to C}}{\text{Number of objects belonging to C}}$
- **Precision(p):** $\frac{\text{Number of objects correctly assigned to C}}{\text{Number of objects assigned to C}}$

- **F**: $\frac{2rp}{r+p}$ a sort of weighted mean between the two measures above

ROC - Receiver Operating Characteristic

Used for **binary classification problems** to characterize the trade-off between positive hits and false alarms

- **TPR - True positive rate**, on the **y** axis, given as a ratio between **TP** and **TP+FN**(aka the number assigned to the positive class, both correct and wrong)
- **FPR - False positive rate**, on the **x** asis, given as a ratio between **FP** and **FP+TN**(aka the number assigned to the negative class, both correct and wrong)

Building a ROC Curve

Instance	P(+|A)	True Class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

- Use classifier that produces posterior probability for each test instance P(+|A)
- Sort the instances according to P(+|A) in decreasing order
- Apply threshold at each unique value of P(+|A)
- Count the number of TP, FP, TN, FN at each threshold
  - TP rate
    TPR = TP/(TP+FN)
  - FP rate
    FPR = FP/(FP + TN)

The red lines are the thresholds that are applied each time