UNIVERSITÀ DEGLI STUDI DI TRENTO

Department of Information Engineering and Computer Science

Master Degree in
Computer Science

Machine Learning

Professor
Andrea Passerini

Assistant
Luca Erculani

Anno accademico 2019/2020

Contents

1 Introduction

A first definition of machine learning by T. Mitchell is the following one:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

That is, machine learning builds a model that allows the computer to learn from experience, but does not solve the problem directly (as it's not always possible).
For a well-posed learning problem there a few key components:

- **Task**: For example recognising characters;
- **Performance measure**: to evaluate the learned system, that is the way the success of the predictors is measured (for example the number of misclassified characters).
- **Training experience**: to train the learning system, that is what is given to the system as experience E so that the algorithm can improve. The easiest way to do so is to give a training set made of couples.

1.1 Designing a machine learning system

The following steps should be followed when designing a machine learning system:

- Formalize the learning task;
- Collect data;
- Extract features;
- Choose class of learning models;
- Train model;
- Evaluate model.

1.1.1 Formalize the learning task

As a first step, it's important to define the task that the learning system should address, for example recognizing handwritten characters.
Often the main task can be divided in smaller tasks such that the problem gets easier to be solved. Considering the example before, it could be split in:

- Segment the image into words and then each word into characters.
- Identify the language of the writing.

- Classify each character into the language alphabet.

At last to define the task, one should also choose an appropriate performance number for evaluating such system, for example the number of misclassified characters could be a good performance measure.

1.1.2 Collect data

Since the machine learning system needs to learn from something, we need a so called training set, that is a set of example written in a *machine readable format*.
For such reason, often the data require some manual intervention, for example in labelling, in order to have a so called supervised learning, that is the training set is formed of a couple containing the initial data and also the correct result that the system should get. This, though, could be quite expansive to be achieved so a much cheaper learning mode is being used, the so called semi-supervised learning, where supervised data is mixed to data which does not have the expected result nor it's labeled.

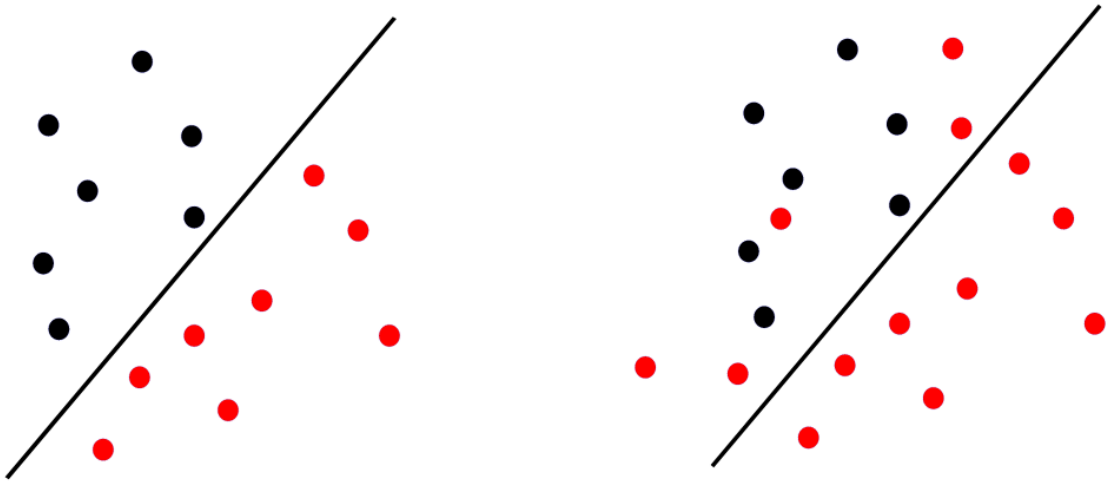
1.1.3 Extract features

Not all the data that is given is really useful and instead sometime is harmful. For this reason a relevant set of feature should be extracted from all the data, that is fundamental information for the input in order for the algorithm to give the correct solution. To do this the best way possible, some prior knowledge is usually necessary.
Mind that:

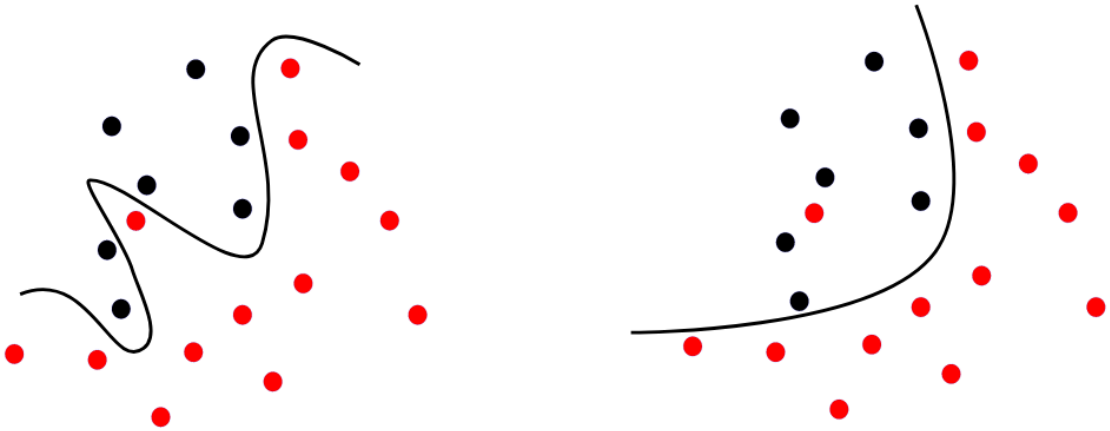
- Too few features could miss some relevant information preventing the system from learning in a correct way.
- Too many feature could heavy the system too much and the training phase could take a lot of time.
- Including noisy features complicates the training phase as the algorithm needs to learn to avoid these.

1.1.4 Model class

Many model classes are available and choosing the best one is important. For example a binary classification, that is a simple linear model that separates data in to two classes.



We call features all the points, that is the example given, and classes the colors, that is the supposed result of the algorithm. It's easy to notice that a binary classifier works as a separator that is a feature can have a class or another. Not always is possible to define a linear function that separates the features, so in some cases a more complex model can be used.



It's possible to notice that in this case the red feature in the middle of all the black ones introduce a lot fo noise. Mind that we still are in a training phase and we do not mind for some false positive o negatives, we want our system to be good in real life not in the training set. If now a complex model is designed, such as the one on the left, the system doesn't learn how to recognize noise and to get rid of it. This problem is called overfitting and should be avoided.
Quindi non vogliamo che il nostro sistema semplicemente memorizzi i dati del training, ma che riesca a generalizzarli.

1.1.5 Learning settings

We have already talked about possible types of learning, such as supervised or semi-supervised learning, but they are not all that there is.

- **Supervised learning**: the learner is given a series of pair where one element is input and one is the correct output. What the algorithm should do is to learn a function f that maps each input to output. Formally given a set of inputs $\mathcal{X}$ and outputs $\mathcal{Y}$, the learner is provided with couples $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ in order to create a *model* $f : \mathcal{X} \mapsto \mathcal{Y}$. Since each example need to have a correct output, usually a domain expert is involved in *labelling* the couples.

- **Unsupervised learning**: the learner is provided with a set of input examples $\mathcal{X}$ but without any labelling information, that is no output. In this case the learner models training examples, for instance grouping examples into clusters according to their similarity.

- **Semi-supervised learning**: this stays in the middle of the formers. As in supervised learning, the learner is provided with a set of input/output pairs: $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. The difference is that the set of inputs is usually much larger than the outputs one providing additional unlabelled examples. As in supervised learning the goal is to find a model f that maps input into outputs: $f : \mathcal{X} \mapsto \mathcal{Y}$. The unlabelled data is exploited in order to improve performance, for example by forcing the model to produce similar outputs for similar inputs.

- **Reinforcement learning**: in this type of learning, the actor is not provided anymore with inputs and outputs, but is given a set of possible states $\mathcal{S}$ and a set of possible actions $\mathcal{A}$ that allows it to move to the next state. When the learner performs an action a from a state s, it is given a reward $r(s, a)$. The task is to learn a policy allowing to choose for each state s the action a maximizing the overall reward. One problem is that the reward may not always be immediate but instead might be delayed. If this is the case, the learner needs to find the right trade-off between exploitation and exploration. An example of delayed reward is chess where the learner knows its reward only at the end.

1.1.5.1 Choice of learning algorithm

The choice of the algorithm to use is based on the knowledge we have of the data:

- Full knowledge of probability distribution of data: Bayesian decision theory.
- Form of probabilities known, parameters unknown: parameter estimation from training data.
- Form of probabilities unknown, training examples available: do not model input data (generative methods), learn a function predicting the desired output given the input.
- Form of probabilities unknown, training examples unavailable (only inputs): unsupervised methods, cluster examples by similarity.

1.1.5.2 Learning tasks

For what concerns the supervised learning, possible tasks are:

- *Binary*: it's the easiest task, or the element belongs to a class, or to the other.
- *Multiclass*: an element can belong to one of the set of the class.
- *Multilabel*: given n possible classes, the example can be assigned to a subset of $m \leq n$ classes.
- *Regression*: in this case a real value is predicted, for example the biodegradation rate of a molecular compound.
- *Ordinal regression* (ranking): a set of examples is ordered according to their relative importance with respect to the task, for example ordering email according to their urgency.

For what concerns unsupervised learning:

- *Clustering*: data is divided into groups in order to have homogeneous elements in each group and so that each group is enough different from the others.
- *Dimensionality reduction*: in this case there are many features and the aim is to reduce their dimensionality maintaining as mach information as possible.
- *Novelty detection*: given a certain distribution of data, the aim of this task is to find all the element that are novel, that is don't respect the distribution. This approach is used for example in network traffic analysis in order to detect anomalous traffic that indicates a possible attack.

1.1.6 Train Model

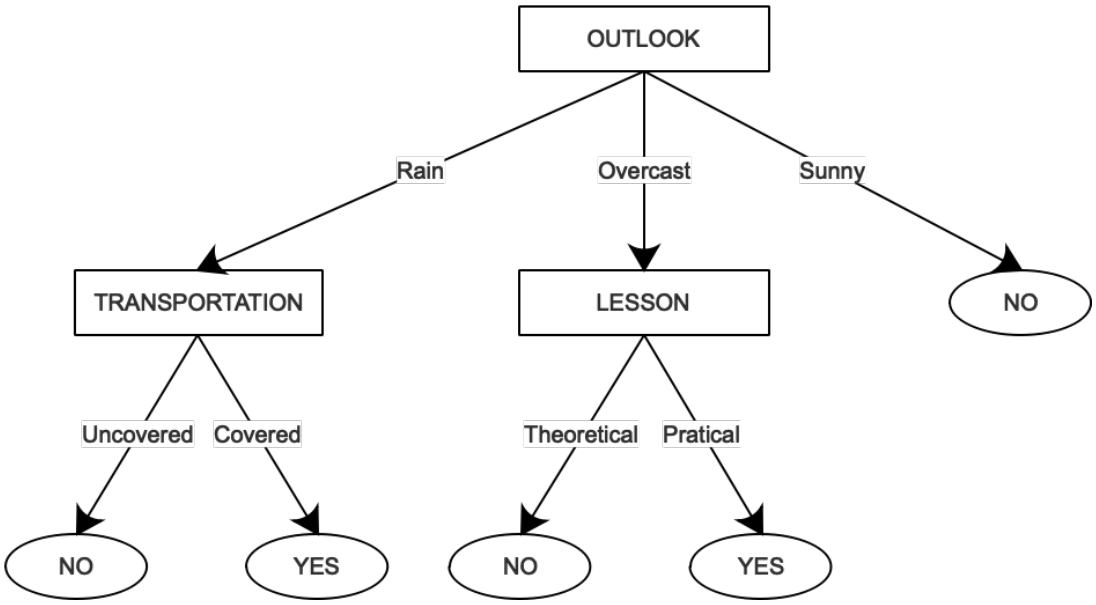
Training a model implies searching through the space of possible models (aka hypotheses) given the chosen model class.
Such search typically aims at fitting the available training examples well according to the chosen performance measure.
However, the learned model should performe well on unseen data, **generalization**, and not simply memorize training examples, **overfitting**.
Different techniques can be used to improve generalization, usually by trading off model complexity with training set fitting.

1.1.7 Evaluate Model

Once a fitting model is found, it needs to be evaluated on its ability to generalize to unseen examples.
For this reason there is the need to have a set of data that is different from the training one. Usually from a unique dataset, two are derived, one for training and one for evaluation.

2 Decision tree learning

From data is sometime possible to build a tree of features that allows to find an item's target value: given an example, with respect to the values the example assumes for each feature, it's possible to build a path which will lead to a leaf, that is a label.



A tree is a disjunction of conjunctions of constraints over attribute values, that is it express a disjunction normal form. Each path from the root to a leaf is a conjunction of hte constraints specified in the nodes along it, e.g., if it's cloudy and the lesson is theoretical, then a student is not going to go to the lesson.

$$\text{OUTLOOK} = \text{Overcast} \wedge \text{LESSON} = \text{Theoretical}$$

One of the strongest advantages of this model is its explainability: it's easy to know what the model does and what the outcome may be.
Such representation works best with binary and multiclass classification takes, even though some extensions to regression also exists, for example exploiting means properties. Another field in which it works fine is with missing attribute, that is for example when some feature don't have any value.

2.1 Algorithm

The learning algorithm in this case is simply a greedy approach proposes by Quinlan: for each node, starting from the root:

1. Choose the best attribute to be evaluated;
2. Add a child for each attribute value;
3. Split node training set into children according to value of chosen attribute;
4. Stop splitting a node if it contains examples from a single class, or there are no more attribute to test.

The idea is to start with a whole training set and then choose the feature so that the set will become as parted as possible. We stop when the example in the training set are all of the same class or there are no more features to be tested.

2.1.1 Choosing the best attribute

The best strategy to choose the best attribute on which to split, is to use sets' **homogeneity**, which is based on the concept of **entropy**.

Definition 2.1: Entropy

Entropy is a measure of the amount of information contained in a collection of instances S which can take a number c of possible values:

$$H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

Where p_i is the fraction of S taking value i.

Entropy is the way in which the increment of homogeneity is measured since it's based on the information contained in a set: if all the values inside the set are the same, then there is no information, otherwise if two values are one the opposite of the other, then there is maximum entropy since the probability of taking one or the other becomes 5%.
Given the entropy then it's possible to compute the **information** fain:

Definition 2.2: Information Gain

The information gain IG is the expected reduction in entropy obtained by partitioning a set S according to the value of a certain attribute A:

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

The greater the difference (IG), the better it is.

2.1.2 Overfitting

With decision tree it's possible to end up with overfitting for example when from each set of n elements, only one element is selected at the time. In this way the leaves would just contain one example creating a complex tree and leading to overfitting.
So the goal is not to obtain pure leaves, but instead to have also leaves that can contains example not strictly from that class.
There are two ways to avoid this possibility in decision trees:

- **Pre-pruning**: decide whether to stop splitting a node even if it contains training example with different labels.
- **Post-pruning**: learn a full tree, risking overfitting, and then in case prune to remove subtrees. Usually this strategy is the one used.

Let's focus on the post-pruning strategy: for this we shall introduce a validation set, that is a set against which to test the trained model. Let's consider a full tree, then:

1. For each node in the tree: evaluate the performance on the validation set when removing the subtree rooted at it;
2. If all node removals worsen performance, then stop;
3. Choose the node whose removal has the best performance improvement;
4. Replace the subtree rooted at it with a leaf;
5. Assign to the leaf the majority label of all examples in the subtree;
6. Return to 1.

When no more improvements are obtained through pruning, then the pruning is finished.
Obviously validation, training and test set must all have the same statistics.

2.2 Problem with decision tree

2.2.1 Discretization

A first problem with decision trees are continuous-valued attributes. One easy solution is to discretize the variable in order to be used in the internal nodes tests.
Discretization implies the usage of thresholds, which can be set in order to maximize the attribute quality criterion, e.g. information gain.
The procedure to achieve this is:

- Examples are sorted according to their continuous attribute values;
- For each pair of successive examples having different labels, a *candidate threshold* is places ad the average of the two attribute values.
- For each candidate threshold, the information gain is achieved splitting examples according to how it is computed;
- The threshold producing the higher information gain is used to discretize the attribute.

2.2.1.1 Example

Let's suppose that we want to accept a work based on the age via the following table:

age	class
23	y
27	y
34	n
38	n
35	y
32	n

As a first thing, ordering data can be a great deal:

age	class
23	y
27	y
32	n
34	n
35	y
38	n

Then a threshold can be chosen, for example in the point where the class changes, even if some other examples are present afterwards, so for example age=32 can be a threshold.

2.2.2 Alternative attributes test measures

Information gain has a problem: if for example the set contained people with an id, it would place each person into a leaf, because this would make the information gain as big as possible. Information gain works fine, but in some cases it divides too much.

In order to avoid the tree from becoming too much spread, it's possible to measure the entropy with respect to the attribute value instead of the class value:

$$H_A(S) = -\sum_{v \in Values(A)} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$$

Bigger the entropy, bigger is the division the attribute is making of the tree.
Now instead of using the information gain, we could downweight it with respect to the new entropy and compute the **gain ratio**:

$$IGR(S, A) = \frac{IG(S, A)}{H_A(S)}$$

2.2.3 Missing values

At the beginning we said that decision trees could deal also with missing values. Let's suppose that at a certain node n we'd like to split on attribute A, but the example x of class $c(x)$ is missing its value for attribute A. Two solutions are possible:

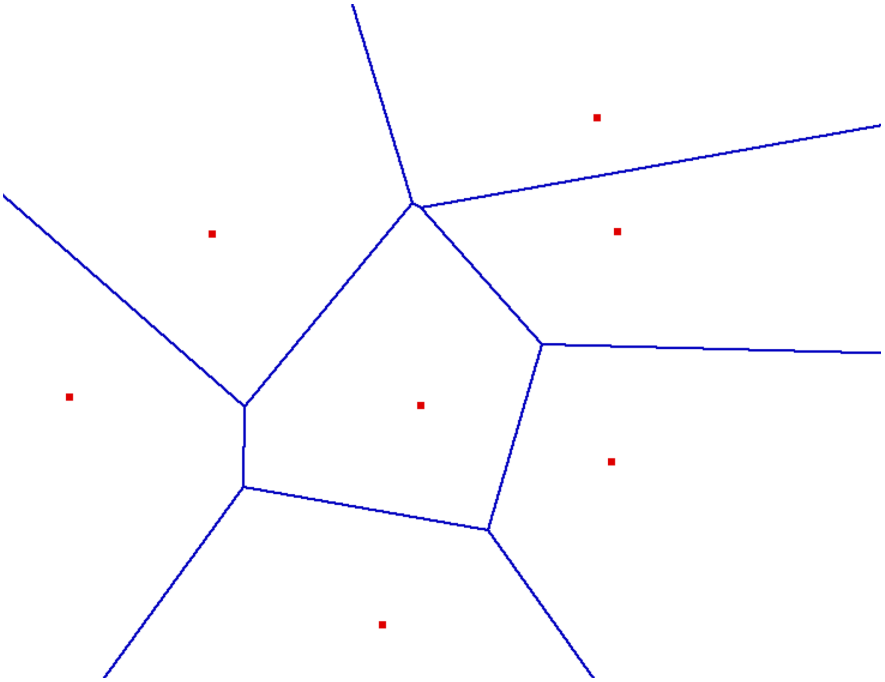
- Simple: assign to x the most common attribute values among examples in n, or the most common of examples in n with class $c(x)$. This works in every condition, but it's not very performing.
- Complex: propagate x to each of the children of n with a fractional value equal to the proportion of example with the corresponding attribute value. This implies that at test time, for each candidate class, all fractions of the test example which reached a leaf with that class are summed, and the example is assigned the class with the highest overall value.

2.3 Random Forests

The decision trees are easy to be used, but not really great in terms of accuracy. An extension of the decision trees is random forests.

Given a set of N examples, sample N examples with replacement, that is the same example can be selected multiple times. Then train a decision tree on the sample, selecting at each node m features at random among which to choose the best one. Repeat this procedure multiple times to generate a forest with multiple trees.
As for the testing, each example is tested against each tree in the forest, and the majority class among the predictions is returned.

3 k-Nearest Neighbour Learning



Definition 3.1: Metrics

Given a set $\mathcal{X}$, a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a *metric* for $\mathcal{X}$ if for any $x, y, z \in \mathcal{X}$ the following properties are satisfied:

1. **Reflixivity**: $d(x, y) = 0 \iff x = y$;
2. **Symmetry**: $d(x, y) = d(y, x)$;
3. **Triangle inequality**: $d(x, y) + d(y, z) \geq d(x, z)$

A frequently used, and well-known, function d is the Euclidean distance defined in $\mathbb{R}^n$:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3.1 The algorithm

3.1.1 Classification problem

```
for all test examples x do
    for all training examples (x_i, y_i) do
        compute distance d(x, x_i)
    end for
    select the k-nearest neighbours of x
    return class of x as majority class among neighbours:
        argmax_y Σ_{i=1}^{k} δ(y, y_i)
end for
```

Where

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

3.1.2 Regression problem

```
for all test examples x do
    for all training examples (x_i, y_i) do
        compute distance d(x, x_i)
    end for
    select the k-nearest neighbours of x
    return the average output value among neighbours:
        (1/k) Σ_{i=1}^{k} y_i
end for
```

3.2 Characteristics

- **Instance-based learning**: the model used for prediction is calibrated for the test example to be processed;
- **Lazy learning**: computation is mostly deferred to the classification phase;
- **Local learner**: assumes prediction should be mainly influenced by nearby instances;
- **Uniform feature weighting**: all features are uniformly weighted in computing distances.

4 Linear Algebra

Definition 4.1: Vector Space

A set $\mathcal{X}$ is called a vector space over $\mathbb{R}$ if addition and scalar multiplication are defined and, for all $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{X}$ and $\lambda, \mu \in \mathbb{R}$, satisfy the following properties:

- Addition:
  - *Associative*: $\boldsymbol{x} + (\boldsymbol{y} + \boldsymbol{z}) = (\boldsymbol{x} + \boldsymbol{y}) + \boldsymbol{z}$
  - *Commutative*: $\boldsymbol{x} + \boldsymbol{y} = \boldsymbol{y} + \boldsymbol{x}$
  - *Identity element*: $\exists 0 \in \mathcal{X} : \boldsymbol{x} + 0 = \boldsymbol{x}$
  - *Inverse element*: $\forall \boldsymbol{x} \in \mathcal{X} \exists \boldsymbol{x'} \in \mathcal{X} : \boldsymbol{x} + \boldsymbol{x'} = 0$
- Scalar multiplication:
  - *Distributive over elements*: $\lambda(\boldsymbol{x} + \boldsymbol{y} = \lambda\boldsymbol{x} + \lambda\boldsymbol{y}$
  - *Distributive over scalars*: $(\lambda + \mu)\vec{x} = \lambda\boldsymbol{x} + \mu\boldsymbol{x}$
  - *Associative over scalars*: $\lambda(\mu\boldsymbol{x}) = (\lambda\mu)\boldsymbol{x}$
  - *Identity element*: $\exists 1 \in \mathbb{R} : 1\boldsymbol{x} = \boldsymbol{x}$

Definition 4.2: Subspace

A subspace is any non-empty subset of $\mathcal{X}$ being itself a vector space.

Definition 4.3: Linear Combination

Given n scalars $\lambda_i \in \mathbb{R}$ and n vectors $\boldsymbol{x}_i \in \mathcal{X}$, their linear combination is:

$$\sum_{i=1}^{n} \lambda_i \boldsymbol{x}_i$$

Definition 4.4: Span

The span of vectors $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ is defined as the set of their linear combinations:

$$\left\{ \sum_{i=1}^{n} \lambda_i \boldsymbol{x}_i, \lambda_i \in \mathbb{R} \right\}$$

Definition 4.5: Linear independency

A set of vectors $\boldsymbol{x}_i$ is linearly independent if none of them can be written as a linear combination of the others.

Definition 4.6: Basis

A set of vectors $\boldsymbol{x}_i$ is a basis for $\mathcal{X}$ if any element in $\mathcal{X}$ can be uniquely written as a linear combination of vectors $\boldsymbol{x}_i$.

A necessary condition for this to be true is that vectors $\boldsymbol{x}_i$ are linearly independent. All bases of $\mathcal{X}$ have the same number of elements, called the *dimension* of the vector space.

Definition 4.7: Linear Map

Given two vector spaces $\mathcal{X}, \mathcal{Z}$, a function $f : \mathcal{X} \to \mathcal{Z}$ is a linear map if all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}, \lambda \in \mathbb{R}$:

- $f(\boldsymbol{x} + \boldsymbol{y}) = f(\boldsymbol{x}) + f(\boldsymbol{y})$
- $f(\lambda \boldsymbol{x}) = \lambda f(\boldsymbol{x})$

A linear map between two finite-dimensional spaces $\mathcal{X}, \mathcal{Z}$, $\mathcal{X} \underset{f}{\to} \mathcal{Z}$ of dimensions n, m can always be written as a matrix of basis transformation:

$$M \in \mathbb{R}^{m \times n} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

Let $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ and $\{\boldsymbol{z}_1, ..., \boldsymbol{z}_m\}$ be some bases for $\mathcal{X}$ and $\mathcal{Z}$ respectively. For any $\boldsymbol{x} \in \mathcal{X}$ we have that it can be written as the linear combination of vectors in the basis, hence:

$$f(\boldsymbol{x}) = f\left(\sum_{i=1}^{n} \lambda_i \boldsymbol{x}_i \right) = \sum_{i=1}^{n} \lambda_i f(\boldsymbol{x}_i)$$

Since there is a function f that maps values from $\mathcal{X}$ to $\mathcal{Z}$, then:

$$f(\boldsymbol{x}_i) = \sum_{j=1}^{m} a_{ji} \boldsymbol{z}_i$$

Which leads to:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_i a_{ji} \boldsymbol{z}_j = \sum_{j=1}^{m} \left(\sum_{i=1}^{n} \lambda_i a_{ji} \right) \boldsymbol{z}_j = \sum_{j=1}^{m} \mu_j \boldsymbol{z}_j$$

In short:

$$M\lambda = \mu$$

Definition 4.8: Matrix Properties Transpose

Given a matrix M, the transposed matrix M^T is the matrix obtained by exchanging rows and columns. Tra transposed of the product of two matrixes M, N, is the product of the two transposed:

$$(MN)^T = N^T M^T$$

Definition 4.9: Matrix Properties Trace

The trace of a matrix M is the sum of the diagonal elements of the matrix:

$$tr(M) = \sum_{i=1}^{n} M_{ij}$$

Definition 4.10: Matrix Properties ăInverse

The inverse matrix is a matrix that multiplies with the original matrix gives the identity:

$$MM^{-1} = I$$

Definition 4.11

Matrix Properties Rank The rank of a $n \times m$ matrix is the dimension of the space spanned by its columns.

The following properties for matrix derivation hold:

$$\frac{\partial M\boldsymbol{x}}{\partial \boldsymbol{x}} = M$$

$$\frac{\partial \boldsymbol{y}^T M\boldsymbol{x}}{\partial \boldsymbol{x}} = M^T\boldsymbol{y}$$

$$\frac{\partial \boldsymbol{x}^T M\boldsymbol{x}}{\partial \boldsymbol{x}} = \left(M^T + M\right)\boldsymbol{x}$$

$$\frac{\partial \boldsymbol{x}^T M\boldsymbol{x}}{\partial \boldsymbol{x}} = 2M\boldsymbol{x} \quad \text{if } M \text{ is symmetric}$$

$$\frac{\partial \boldsymbol{x}^T \boldsymbol{x}}{\partial \boldsymbol{x}} = 2\boldsymbol{x}$$

Definition 4.12: Norm

A function $\|\cdot\| : \mathcal{X} \to \mathbb{R}_0^+$ is a norm if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}, \lambda \in \mathbb{R}$:

- $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$
- $\|\lambda\boldsymbol{x}\| = |\lambda| \, \|\boldsymbol{x}\|$
- $\|\boldsymbol{x}\| > 0$ if $\boldsymbol{x} \neq 0$

A norm defines a metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$:

$$d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|$$

Definition 4.13: Bilinear Form

A function $Q : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a bilinear form if for all $\cdot, \boldsymbol{y}.\boldsymbol{z} \in \mathcal{X}, \lambda, \mu \in \mathbb{R}$:

- $Q(\lambda \boldsymbol{x} + \mu \boldsymbol{y}, \boldsymbol{z}) = \lambda Q(\boldsymbol{x}, \boldsymbol{z}) + \mu Q(\boldsymbol{y}, \boldsymbol{z})$
- $Q(\boldsymbol{x}, \lambda \boldsymbol{y} + \mu \boldsymbol{z}) = \lambda Q(\boldsymbol{x}, \boldsymbol{y}) + \mu Q(\boldsymbol{x}, \boldsymbol{z})$

Definition 4.14: Bilinear Form Symmetry

A bilinear form is said to be *symmetric* if $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$:

$$Q(\boldsymbol{x}, \boldsymbol{y}) = Q(\boldsymbol{y}, \boldsymbol{x})$$

Definition 4.15: Dot Product

A dot product $< \cdot, \cdot >: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric bilinear form which is positive semi-definite:

$$\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0 \forall \boldsymbol{x} \in \mathcal{X}$$

A positive definite dot product defines a corresponding norm via:

$$\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$$

One main property of the dot product is:

$$\cos\theta = \frac{\langle \boldsymbol{x}, \boldsymbol{z} \rangle}{\|\boldsymbol{x}\| \, \|\boldsymbol{z}\|}$$

From which follow that if two vectors are **orthogonal**, that is $\theta = \frac{\pi}{2} + k\pi, k \in \mathbb{N}$, then, $\langle \boldsymbol{x}, \boldsymbol{z} \rangle = 0$. Moreover a set of vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is said to be **orthonormal** if, for all vectors $\boldsymbol{x}_i, \boldsymbol{x}_j$ in the set:

$$\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 4.16: Eigenvalue and Eigenvector

Given an $n \times n$ matrix M, the real value λ and (non-zero) vector $\boldsymbol{x}$ are an eigenvalue and corresponding eigenvector of M if:

$$M\boldsymbol{x} = \lambda \boldsymbol{x}$$

Definition 4.17: Singular Matrix

A matrix is singular if it has a zero eigenvalue:

$$m\boldsymbol{x} = 0\boldsymbol{x} = 0$$

A singular matrix has linearly dependent columns:

$$\begin{bmatrix} M_1 & \dots & M_{n-1} & M_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{n_1} \\ x_n \end{bmatrix} = 0$$

$$M_1 x_1 + \dots + M_{n-1} x_{n-1} + M_n x_n = 0$$

$$M_n = M_1 \frac{-x_1}{x_n} + \dots + M_{n-1} \frac{-x_{n-1}}{x_n}$$

The determinant $|M|$ of a $n \times n$ matrix M is the product of its eigenvalues.
Since a matrix is invertible if its determinant is not zero, then it must not be singular.
We said that the eigenvalue and eigenvector of a matrix are a value and a vector such that:

$$A\boldsymbol{x} = \lambda \boldsymbol{x}$$

Trying to solve for λ we obtain:

$$\frac{\boldsymbol{x}^T A \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}} = \lambda \frac{\boldsymbol{x}^T \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$

Definition 4.18: Raleigh quotient

$$\lambda = \frac{\boldsymbol{x}^T A \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$

5 Probability

5.1 Discrete

Definition 5.1: Probability Mass Function

Given a discrete random variable X taking values in $\mathcal{X} = \{v_1, \dots, v_m\}$, its probability mass function $P : \mathcal{X} \to [0, 1]$ is defined as:

$$P(v_i) = \Pr[X = v_i]$$

And satisfies the following conditions:

- $P(x) \geq 0$
- $\sum_{x \in \mathcal{X}} P(x) = 1$

So a variable is said to be discrete if it can assume m possible values $\{v_1, \dots, v_m\}$ which are mutual exclusive and if all probability of the values to happen are non-negative and the sum of probabilities of the events must be 1.

Definition 5.2: Expected Value

The expected value of a random variable x, also known as *mean* or *average*, is:

$$\mathrm{E}[x] = \mu = \sum_{x \in \mathcal{X}} xP(x) = \sum_{i=1}^{m} v_i P(v_i)$$

The expected value is linear:

$$\mathrm{E}[\lambda x + \lambda' y] = \lambda \mathrm{E}[x] + \lambda' \mathrm{E}[y] \tag{5.1}$$

Definition 5.3: Variance

The variance of a random variable is the moment of inertia of its probability mass function:

$$\mathrm{Var}[x] = \sigma^2 = \mathrm{E}\left[(x - \mu)^2\right] = \sum_{x \in \mathcal{X}} (x - \mu)^2 P(x)$$

Another important value is the standard deviation σ which indicates the typical amount of deviation from the mean.
The followings are properties of mean and variance:

- **Second moment**: it's similar to the expected value:

$$\mathrm{E}\left[x^2\right] = \sum_{x \in \mathcal{X}} x^2 P(x)$$

- It's possible to write the variance in term via the mean:

$$\mathrm{Var}[x] = \mathrm{E}\left[x^2\right] - \mathrm{E}\left[x\right]^2$$

- The variance is *not* always linear, for example when the variable is multiplied by a scalar:

$$\mathrm{Var}[\lambda x] = \lambda^2 \mathrm{Var}[x] \tag{5.2}$$

- The variance of the sum of two variables corresponds to the sum of the variance *only* if the two variables are not *correlated*:

$$\mathrm{Var}[x + y] = \mathrm{Var}[x] + \mathrm{Var}[y] \tag{5.3}$$

5.2 Discrete Probability Distribution

5.2.1 Bernoulli

This probabilistic distribution is used to model a binary event, that is an event that can only result in success or failure.
The probability p of an event x is the probability of success ($x = 1$), while the probability of failure ($x = 0$) is $1 - p$. The probability mass function is:

$$P(x, p) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0 \end{cases}$$

A scenarios that can be modelled using this distribution is for example the toss of a coin where head is success and tail is failure (or viceversa).

Theorem 5.1: Estimated Value – Bernoulli

$$\mathrm{E}\left[x\right] = p$$

Proof.

$$\mathrm{E}\left[x\right] = \sum_{x \in X} xP(x)$$
$$\mathrm{E}\left[x\right] = \sum_{x \in \{1,0\}} xP(x)$$
$$\mathrm{E}\left[x\right] = 0 \times (1 - p) + 1 \times p = p$$

□

Theorem 5.2: Variance – Bernoulli

$$\text{Var}[x] = p(1-p)$$

Proof.

$$\begin{aligned}
\text{Var}[x] &= \text{E}\left[(x-\mu)^2\right] = \\
&= \sum_{x \in X} (x - \text{E}[x])^2 P(x) = \\
&= \sum_{x \in X} (x-p)^2 P(x) = \\
&= (0-p)^2(1-p) + (1-p)^2 p = \\
&= p^2 - p^3 + (1 + p^2 - 2p)p = \\
&= p^3 - p^3 + p^2 + p - 2p^2 \\
&\text{Var}[x] = p(1-p)
\end{aligned}$$

□

The Bernoulli probability can be expressed also via an analytic function which is often used in if-then-else cases:

$$p(x,p) = p^x (1-p)^{1-x}$$

If $x = 1$ then only the first term remains, resulting in $p(x,p) = p$, while if $x = 0$, then what remains is: $p(x,p) = 1 - p$

5.2.2 Binomial

Bernoulli distribution can be generalized with more than two events: this distribution models the probability of having a certain number of successes in n independent Bernoulli trials.
The parameter of this distribution are p has the probability of a success, and n as the number of trials.
The probability mass function is:

$$P(x; p, n) = \binom{n}{x} p^x (1-p)^{n-x}$$

Which represents the probability of success for x trials and can be seen as the Bernoulli distribution applied to all trials.
An example of event that can be modelled with this distribution is the toss of a coin for n times and trying to guess the probability of having x heads.
Mean and variation are:

$$\text{E}[x] = np \qquad \text{Var}[x] = np(1-p)$$

That is the same as the Bernoulli distribution but multiplied for the n tests.

5.2.3 Joint Probability

If two random variables are given instead of one, then the model must be different since they may not be independent.

Definition 5.4: Joint Probability

Given a pair of discrete random variables X, Y taking values $\mathcal{X} = \{v_1, \dots, v_m\}, \mathcal{Y} = \{w_1, \dots, w_n\}$, the joint probability mass function is defined as:

$$P(v_i, w_i) = \Pr[X = v_i, Y = w_j]$$

With properties:

- $P(x, y) \geq 0$
- $\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) = 1$

The characteristics of this function are:

- *Expected value*: since there are multiple random variables, then there will be one estimated value for each variable. Mind though that the mean for each variable is not simply the mean for that variable without considering the other variables, but is given by the sum of the value multiplied by the joint probability:

$$\mu_x = \mathrm{E}[x] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} x P(x, y)$$

$$\mu_y = \mathrm{E}[y] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} y P(x, y)$$

- *Variance*: as for the expected value, there will be one variance for each variable, and they will be computed based on the joint probability:

$$\sigma_x^2 = \mathrm{Var}[(x - \mu_x)^2] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} (x - \mu_x)^2 P(x, y)$$

$$\sigma_y^2 = \mathrm{Var}[(y - \mu_y)^2] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} (y - \mu_y)^2 P(x, y)$$

- *Covariance*: this is a statistics of how the random variables change at the change of another variable:

$$\sigma_{xy} = \mathrm{E}[(x - \mu_x)(y - \mu_y)] = \sum_{x \in \S} \sum_{y \in \mathcal{Y}} (x - \mu_x)(y - \mu_y) = P(x, y)$$

- *Correlation coefficient*: it is the normalization of the covariance with respect to the product of the variance:

$$\rho = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

5.2.3.1 Multinomial Distribution – One sample

Models the probability of a certain outcome for an event with $m > 2$ possible outcomes[1].
With a multinomial distribution, there are m parameters $p_1, \dots, p_m$ that indicate the probability

[1] If $m = 2$, then it's simply a Bernoulli distribution.

of the j outcome to happen.
For example this model represents the tossing of a dice one time, where m is the number of faces of the dice and p_i is the probability of face i to exit.
Since only one sample is considered, then the mass function of this distribution becomes the probability of that event to happen:

$$P(x_1, \dots, x_m; p_1 \dots, p_m) = \prod_{i=1}^{m} p_i^{x_i} \tag{5.4}$$

Where if $x_i = 1$, then all other outcomes must be 0: $x_j = 0, j \neq i$. The expected value, the variance are the same as the Bernoulli's while the covariance is:

$$\mathrm{E}[x_i] = p_i \qquad \mathrm{Var}[x_i] = p_i(1 - p_i) \qquad \mathrm{Cov}[x_i, x_j] = -p_i p_j$$

5.2.3.2 Multinomial Distribution – General

As for the Bernoulli distribution, also for the multinomial there exists a general version with more samples.
Given n samples of an event with m possible outcomes, the general version models the probability of a certain distribution of outcomes.
For example the following scenario is modelled by a general multinomial distribution: the toss of a coin with m faces, each with p_i probability, n times.
The mass function of this distribution is:

$$P(x_1, \dots, x_m; p_1 \dots, p_m) = \frac{n!}{\prod_{i=1}^{m} x_i!} \prod_{i=1}^{m} p_i^{x_i}$$

As for the Bernoulli distribution and the bionomial distribution, also in this case the statistics of the general distribution are the same as the one for one sample, just multiplied by n. The expected value, variance and covariance are:

$$\mathrm{E}[x_i] = np_i \qquad \mathrm{Var}[x_i] = np_i(1 - p_i) \qquad \mathrm{Cov}[x_i, x_j] = -np_i p_j$$

5.3 Conditional probabilities

In some cases we want to know the probability of an event to happen given that another event has happened. This is called *conditional probability*:

Definition 5.5: Conditional probability

The probability of x once y is observed:

$$P(x|y) = \frac{P(x, y)}{P(y)}$$

This implies the **product rule**:

Definition 5.6: Product rule

$$P(x, y) = P(x|y)P(y) = P(y|x)P(x)$$

This rule is useful because it allows to break down a joint probability into a combination of conditional probabilities.

Definition 5.7: Statistically independence

Two variable X and Y are said to be statistically independent if and only if:

$$P(x, y) = P(x)P(y) \tag{5.5}$$

This implies that if two variables X and Y are statistically independent, then:

$$P(x|y) = P(x) \qquad P(y|x) = P(y)$$

Definition 5.8: Low of Total Probability

The *marginal distribution* of a variable is obtained from a joint distribution summing over all possible values of the other variable (*sum rule*):

$$P(x) = \sum_{y \in \mathcal{Y}} P(x, y) \qquad P(y) = \sum_{x \in \mathcal{X}} P(x, y)$$

This means for example that considering a multinomial variable, it's possible to obtain its probability by removing it from the product and multiplying over the other values.

Definition 5.9: Bayes' Rule

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Bayes' rule comes from the product rule and allows us to invert the dependency between two probabilities. In particular it allows to invert statistical connections between *effect*(x) and *cause*(y):

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

Mind that evidence can be obtained using the sum rule from likelihood and prior:

$$P(x) = \sum_{y} P(x, y) = \sum_{y} P(x|y)P(y)$$

These rules can be applied to more variables:

- **Sum rule**: $P(y) = \sum_{x} \sum_{z} P(x, y, z)$
- **Product rule**: apply the first rule more times, for example $P(x, y, z) = P(x|y, z)P(y, z) = P(x|y, z)P(y|z)P(z) = P(x, y, z) = P(x, y|z)p(z)$
- **Bayes' rule**: $P(y|x, z) = \dfrac{P(x|y, z)P(y|z)}{P(x|z}$

Let's compute the probability of y applying these rules:
For the sum rule (aka law of total), we know that:

$$P(y) = \sum_x \sum_z P(x, y, z)$$

And for the product rule we know that:

$$P(x, y, z) = P(y|x, z)P(x, z)$$

Hence:

$$P(y) = \sum_x \sum_z P(y|x, z)P(x, z)$$

At last applying Bayes' rule to $P(y|x, z)$ we obtain:

$$P(y) = \sum_x \sum_z \frac{P(x|y, z)P(y|z)P(x, z)}{P(x|z)}$$

5.3.1 Example of style

Let's start from the following expression and apply the rules:

$$P(y|x, z)$$

First of all let's apply Bayes' rule:

$$P(y|x, z) = \frac{P(x, z|y)P(y)}{P(x, z)}$$

Consider the discriminator $P(x, z)$, it's possible to apply the product rule:

$$P(y|x, z) = \frac{P(x, z|y)P(y)}{P(x|z)P(z)}$$

Now let's focus on the term $P(x, z|y)$. We can imagine it to come from $P(x, z, y)$ to which was applied the product rule: $P(x, z, y) = P(x, z|y)P(y)$, but it's also true that $P(x, z, y) = P(x|z, y)P(z|y)P(y)$, and hence $P(x, z|y) = P(x|z, y)P(z|y)$. It's possible then to rewrite the before as:

$$P(y|x, z) = \frac{P(x|z, y)P(z|y)P(y)}{P(x|z)P(z)}$$

Now focus on $P(z|y)P(y)$, it's obvious that by reversing the product rule, it equals $P(z, y)$, which can be written also as $P(y|z)P(z)$ since $P(z, y) = p(y, z)$:

$$P(y|x, z) = \frac{P(x|z, y)P(y|z)P(z)}{P(x|z)P(z)}$$

At last it's possible to simplify $P(z)$:

$$P(y|x, z) = \frac{P(x|z, y)P(y|z)}{P(x|z)}$$

5.4 Continuos variable

It's not possible to represent continuous variables with mass function since even if it had a small value, for as small it can be, we are summing it infinite times making it infinite.
Let's consider a continuous variable X and let's consider some intervals: $W = (a < X \leq b)$, $A = (x \leq a)$, $B(X \leq b)$. It's possible to notice that W and A are mutually esclusive, hence it's possible to write:

$$P(B) = P(A) + P(W) \qquad P(W) = P(B) - P(A)$$

Definition 5.10: Cumulative Distribution Function

A function $F(q)$ that models the probability of a continuous variable of having a value less or equal to q is called cumulative distributed function:

$$F(q) = P(X \leq q)$$

Mind that this is a monotonic function non decreasing: if the interval is enlarged, or it stays the same or it increases.
It's possible to get the probability of intervals by computing the difference between the cumulative distribution functions of the extremities:

$$P(a < X \leq) = F(b) - F(a)$$

Definition 5.11: Probability Density Function

The derivative of the cumulative distribution function is called probability density function and it represents the probability in a single point:

$$p(x) = \frac{d}{dx}F(x)$$

The cumulative distribution can also be computed from the density function integrating:

$$F(q) = P(X \leq q) = \int_{-\infty}^{q} p(x)dx$$

The following are properties of the density function:

- $p(x) \geq 0$: which implies that the density probability can also be greater than 1;
- $\int_{-\infty}^{\infty} p(x)dx = 1$: even if the density function can be grater than 1, the total integral over the possible values is 1.

Let's consider the following density distribution, the uniform distribution over $[a, b]$:

$$p(x) = Unif(x; a, b) = \frac{1}{b-a}$$

For $a = 0, b = {}^1/_2$, then $\forall x \in [0, {}^1/_2], p(x) = 2$. Let's now check the value of the integral:

$$F(0 < x \leq {}^1/_2) = \int_{-\infty}^{+\infty} p(x)dx = \int_{a}^{b} p(x)dx = \int_{0}^{\frac{1}{2}} 2dx = [2x]_0^{1/2} = 2 * \frac{1}{2} - 2 * 0 = 1$$

The expected value of a continuous variable, is computed as the integral of the product of the variable value by its probability:

$$\mathrm{E}[x] = \mu = \int_{-\infty}^{\infty} xp(x)dx$$

Notice the similarity with the discrete value, only instead of the sum, an integral is used, as can be expected. It should come as no surprise then, the expression for the variance:

$$\mathrm{Var}[x] = \sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx$$

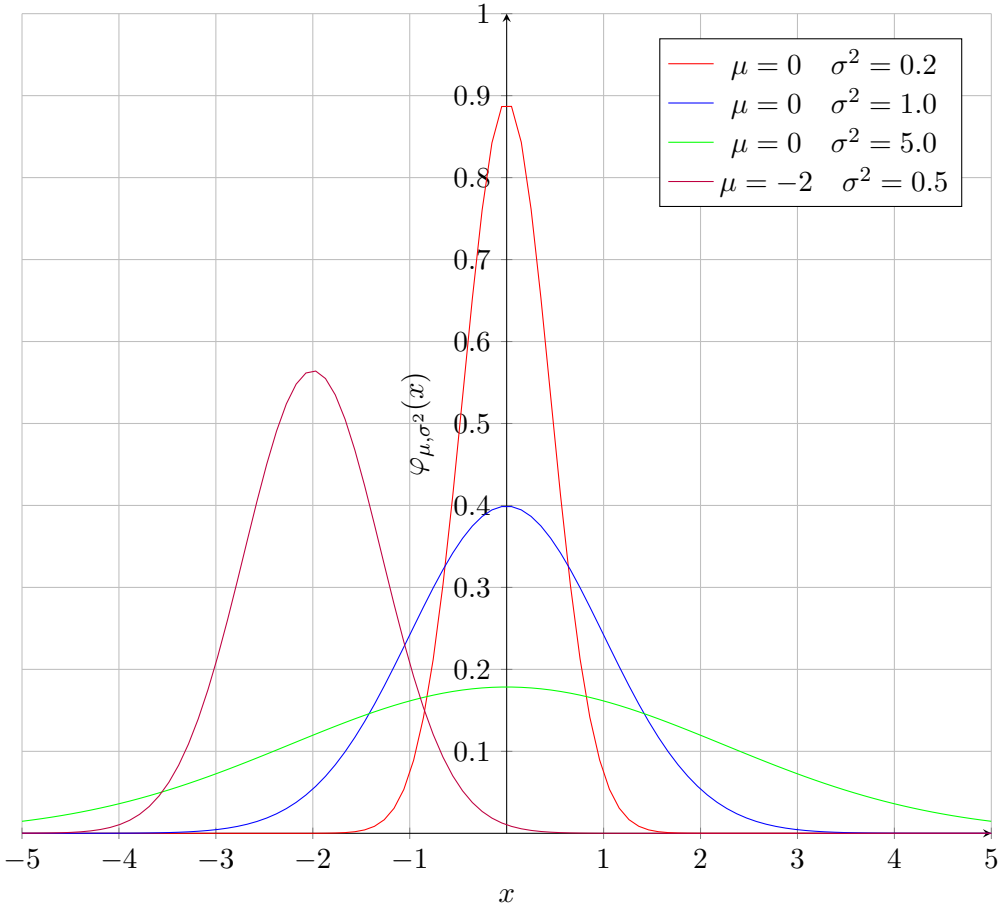
5.4.1 Gaussian – Normal

It is described in terms of μ and σ^2 which are its parameters.
It's density function is:

$$p(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} exp - \frac{(x-\mu)^2}{2\sigma^2} \tag{5.6}$$

Where the term $\frac{(x-\mu)^2}{2\sigma^2}$ is called standardisation of a normal distribution: $N(\mu,\sigma^2)$, and what it does is basically translate the belle shape and normalize it.
The standard normal distribution is a Gaussian distribution with mean 0 and variance 1.
As it can be expected, the expected value and the variance of the gaussian are μ and σ^2 respectively:

$$\mathrm{E}[x] = \mu \qquad \mathrm{Var}[x] = \sigma^2$$



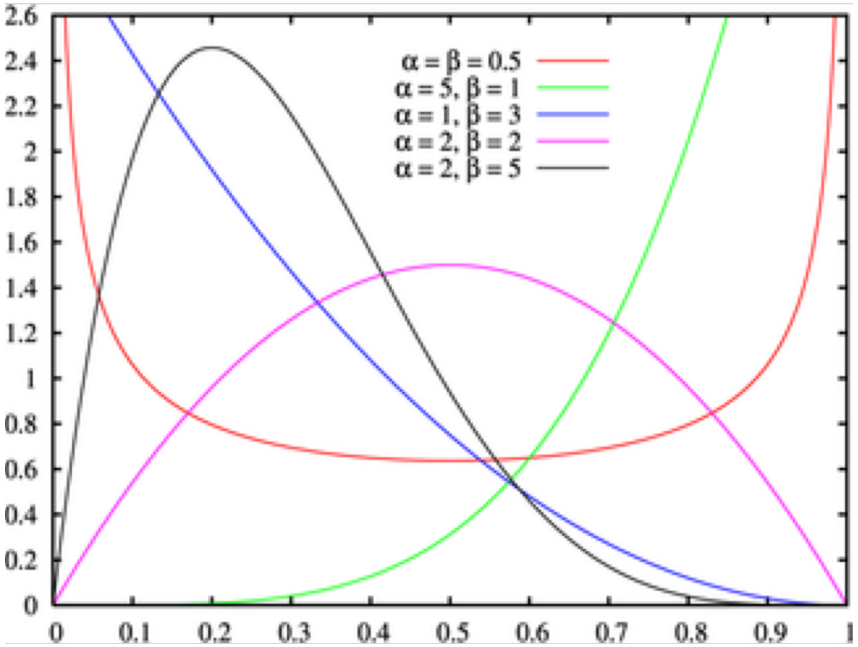
5.4.2 Beta Distribution

The Beta[2] distribution is defined inside the interval $[0, 1]$.
It is based on parameters α, β which let the density function vary as follows:

$$p(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} \tag{5.7}$$

This distribution can model for example the following scenario: we have a Bernoulli distribution, but no probability distribution for the event to happen is given. Such distribution can be modelled via a Beta distribution, making this a probability of the second order.
Notice that the terms $x^{\alpha-1}, (1-x)^{\beta-1}$ reminds of the binomial one, the beta distribution models the posterior distribution of parameter p of a binomial distribution after observing $\alpha - 1$ independent events with probability p and $\beta - 1$ events with probability $1 - p$.
The expected value and variation are:

$$\mathrm{E}[x] = \frac{\alpha}{\alpha + \beta} \qquad \mathrm{Var}[x] = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)} \tag{5.8}$$



5.4.3 Multivariate Normal Distribution

This is a normal distribution for d-dimensional vectorial data. For this reason the parameters are μ, a vector containing the means, Σ a covariance matrix.
It's possible to rewrite the gaussian distribution with the new data by exploiting matrices:

$$p(\boldsymbol{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} \tag{5.9}$$

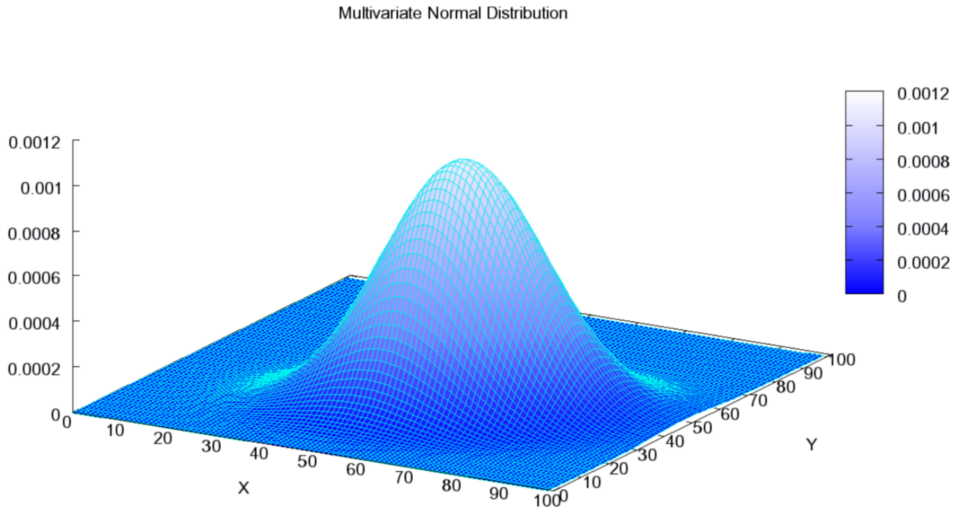
The term

$$\boldsymbol{r}^2 = (\boldsymbol{x} - \boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu}) \tag{5.10}$$

is called Mahalanobis distance and is the generalization of the euclidean distance with respect to the covariance of data. r^2 measures the distance from the mean $\boldsymbol{\mu}$.

[2]Pronounced /bit/.

The expected value is actually the expected one: $\mathrm{E}[x] = \mu$, while the variance is actually the covariance parameters: $\mathrm{Var}[x] = \Sigma$.



5.4.4 Dirichlet Distribution

The beta distribution is used to model for example the probability of a binary event. Obviously it can be generalized to the Dirichlet distribution, which is nothing less the continuous version of the multinomial distribution. The Dirichlet distribution allows to model the probability of an event with more than two possible results.

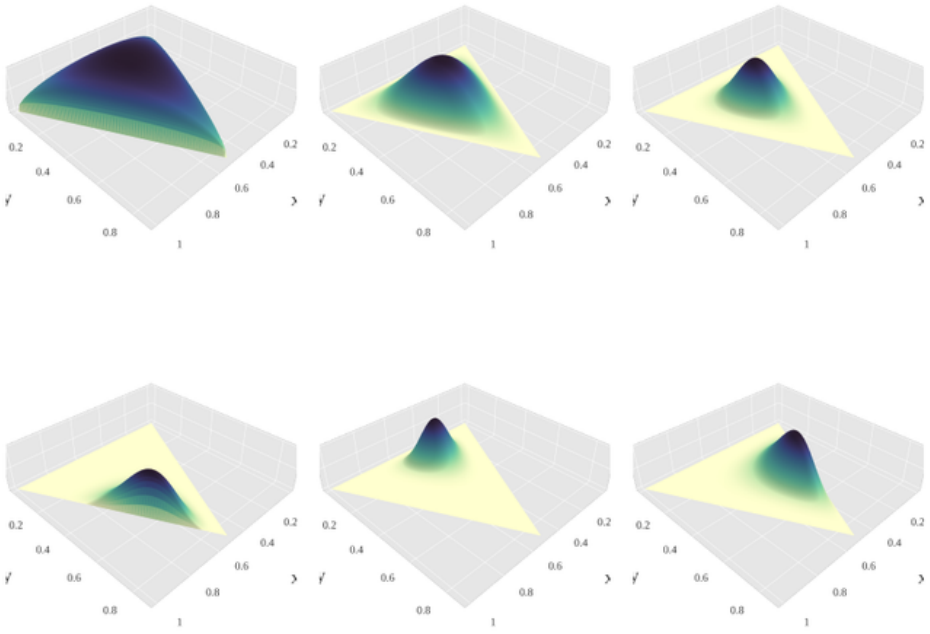
The Dirichlet distribution is defined for $\boldsymbol{x} \in [0,1]^m, \sum_{i=1}^{m} x_i = 1$, that is, given m possible outcomes, $x_1, \ldots, x_m$, the sum of all the outcomes must be 1.
It is parametrized over $\boldsymbol{\alpha} = \alpha_1, \ldots, \alpha_m$.
The probability density function is:

$$p(x_1, \ldots, x_m; \boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^{m} \Gamma(\alpha_i)} \prod_{i=1}^{m} x_i^{\alpha_i - 1} \tag{5.11}$$

Where $\alpha_0 = \sum_{j=1}^{m} \alpha_j$. The expected value and variance are the followings:

$$\mathrm{E}[x_i] = \frac{\alpha_i}{\alpha_0} \qquad \mathrm{Var}[x_i] = \frac{\alpha_i(\alpha_0 - \alpha_i)}{\alpha_0^2(\alpha_0 + 1)} \qquad \mathrm{Cov}[x_i, x_j] = \frac{-\alpha_i \alpha_j}{\alpha_0^2(\alpha_0 + 1)} \tag{5.12}$$

This distribution models the posterior distribution of parameters $\boldsymbol{p}$ of a multinomial distribution after observing $\alpha_i - 1$ times each mutually exclusive event.



5.5 Probability Laws

5.5.1 Expectation and Variance of an Average

Consider a sample of $X_1, \ldots, X_n$ *independent and identical distributed* (iid) instances drawn from a distribution with mean μ and variance σ^2.
Consider the random variable $\overline{X}_n$ measuring the sample average:

$$\overline{X}_n = \frac{X_1 + \ldots + X_n}{n}$$

Its expectation is computed as:

$$\mathrm{E}\left[\overline{X}_n\right] = \mathrm{E}\left[\frac{1}{n}(X_1 + \ldots + X_n)\right]$$

Now recall that the expected value is linear (Equation 5.1), hence it can be rewritten has:

$$\mathrm{E}\left[\overline{X}_n\right] = \frac{1}{n}(\mathrm{E}\left[X_1\right] + \ldots + \mathrm{E}\left[X_n\right])$$

Since $\mathrm{E}\left[X_i\right] = \mu$, then we have:

$$\mathrm{E}\left[\overline{X}_n\right] = \frac{1}{n}(\mu + \ldots + \mu) = \frac{1}{n} * n\mu = \mu$$

That is the expectation of an average is the true mean of the distribution.
Now let's consider the variance which though is not linear with respect to scalars (Equation **??**), while it behaves linear when summing variances (Equation 5.3). It's possible to write the variance of $\overline{X}_n$ as:

$$\mathrm{Var}[\overline{X}_n] = \frac{1}{n^2}(\mathrm{Var}[X_1] + \ldots + \mathrm{Var}[X_n]) = \frac{\sigma^2}{n}$$

From this is possible to notice that the variance of the average decreases with the number of observation: the more examples are taken into consideration, the more likely estimating the correct average is.

5.5.2 Chebyshev's Inequality

Consider a random variable X with mean μ and variance σ^2. The Chebyshev's inequality states that for all $a > 0$:

$$\Pr[|X - \mu| \geq a] \leq \frac{\sigma^2}{a^2}$$

Replacing a with $k\sigma, k > 0$ we obtain:

$$\Pr[|X - \mu| \geq k\sigma] \leq \frac{1}{k^2}$$

Chebyshev's inequality shows that most of the probability mass of a random variable stays within few standard deviations from its mean.

5.6 The Law of Large Numbers

Let's suppose to have some phenomenon to model and the distribution was not observed in its entirety, but only n events, for example we register only if rains for some days. Let's suppose also that the events are *identically distributed* since they come from the same phenomenon and that they are also independent which implies that the outcome of an event does not affect other events.
Our goal is to evaluate the average and the variance observing the data at our disposal.
Consider a sample of $X_1, \ldots, X_n$ iid instances drawn from a distribution with mean μ and variance σ^2.
We have already shown in the previous sections that the expected value of iid samples of mean μ is actually μ, now it's possible to show through the Chebyshev's inequality that the probability that the sample-mean drifts away from the actual mean of a little value
epsilon decreases at the increasing of the number of samples. This implies that the accuracy of an empirical statistic increases with the number of samples.

$$\Pr[|\overline{X}_n - \mathrm{E}\left[\overline{X}_n\right]| \geq \epsilon] \leq \frac{\sigma^2}{n\epsilon^2}$$

$$\lim_{n\to\infty} \Pr[|\overline{X}_n - \mu| \geq \epsilon] = 0$$

5.6.1 Central Limit Theorem

Definition 5.12: Central Limit Theorem

The sum of a sufficiently large sample of iid random measurement is approximately normally distributed.

This means that if we have a number n of samples sufficiently large, with mean μ and variance σ^2, we don't need to know the form of their distribution, because they can be modelled through a normal gaussian.
Questo significa che se abbiamo molti dati, se non conosciamo la distribuzione dei dati, allora la possiamo considerare questa distribuzione come una normale.

5.7 Information theory

5.7.1 Entropy

Consider a set of symbols $\mathcal{V} = \{v_1, \dots, v_n\}$ with mutually exclusive probabilities $P(v_i)$.
The goal is to design a binary code for each symbol minimizing the average length of messages.
In 1949 Shannon and Weaver proved that the optimal code assigns to each symbol v_i a number of bits equal to:

$$-\log P(v_i)$$

Definition 5.13: Entropy

The entropy of a set of symbols is the **expected value** length of a message encoding a symbol assuming such optimal coding:

$$H[\mathcal{V}] = \mathrm{E}\left[-\log P(v)\right] = -\sum_{i=1}^{n} P(v_i) \log P(v_i)$$

If all symbols have the same probability, then the entropy is maximized, while if the symbols have very much different probabilities, then the entropy becomes 0.

5.7.2 Cross entropy

Definition 5.14: Cross Entropy

Given two distributions P and Q over a variable X, the cross entropy between P and Q measures the expected number of bits needed to code a symbol sampled from P using Q instead:

$$H(P;Q) = \mathrm{E}_P[-\log Q(v)] = -\sum_{i=1}^{n} P(v_i) \log Q(v_i)$$

This is often used as a loss function for binary classification, with P (empirical) true distribution and Q (empirical) predicted distribution.

5.7.3 Relative Entropy

Definition 5.15: Relative Entropy

Given two distribution P, Q over a variable X, the relative entropy (or *Kullback-Leibler divergence)* measures the expected length difference when coding instances samples from P using Q instead:

$$D_{KL}(p\|q) = \sum_{i=1}^{n} P(v_i) \log \frac{P(v_i)}{Q(v_i)}$$

This can be derived as follows:

$$
\begin{aligned}
D_{KL}(p\|Q) &= H(P;Q) - H(P) \\
&= \mathrm{E}_P[-\log Q(v)] - \mathrm{E}\left[-\log P(v)\right] \\
&= -\sum_{i=1}^{n} P(v_i)\log Q(v_i) + \sum_{i=1}^{n} P(v_i)\log P(v_i) \\
&= \sum_{i=1}^{n} \left[P(v_i)\log P(v_i) - P(v_i)\log Q(v_i)\right] \\
&= \sum_{i=1}^{n} P(v_i)\left(\log P(v_i) - \log Q(v_i)\right) \\
&= \sum_{i=1}^{n} P(v_i)\log\frac{P(v_i)}{Q(v_i)}
\end{aligned}
$$

5.7.4 Conditional entropy

Definition 5.16: Conditional Entropy

Given two variables V, W with possibly different distributions P, the conditional entropy is the entropy remaining for variable W once V is known:

$$
\begin{aligned}
H(W|V) &= \sum_{v} P(v)H(W|V=v) \\
&= -\sum_{v} P(v)\sum_{w} P(w|v)\log P(w|v)
\end{aligned}
$$

This says that, the more we know about V, the more is the entropy of W, and viceversa.

5.7.5 Mutual Information – Information Gain

Definition 5.17: Information Gain

Given two variables V, W with (possibly different) distributions P, the mutual information, or information gain, is the reduction in entropy for W once V is known:

$$
\begin{aligned}
I(W;V) &= H(W) - H(W|V) \\
&= -\sum_{w} p(w)\log p(w) + \sum_{v} P(v)\sum_{w} P(w|v)\log P(w|v)
\end{aligned}
$$

The information gain is the reduction of entropy on W once V is known. This can be used, for example, when deciding which attribute is best to select when building a decision tree, where V is the attribute and W is the label.

6 Bayesian Decision Theory

We are not yet to the point of taking decision. Bayesian decision *theory* allows to take optimal decisions in a fully probabilistic way.
It allows to provide an upper bounds on achievable errors and evaluate classifiers accordingly. Moreover bayesian reasoning can be generalized to cases when the probabilistic structure is not entirely known.
From now on we will be using $P(x)$ for mass functions and $p(x)$ for distribution functions or unknowns.

6.1 Bayes decision Rule

Let's consider a binary classification. Assume of having examples $(x, y) \in \mathcal{X} \times \{-1, 1\}$ drawn from a known distribution $p(x, y)$. The task is predicting the class y of examples given the input x, which can be done via Bayes' rule (Definition 5.9):

$$P(y|x) = \frac{p(x|y)P(y)}{p(x)}$$

Bayes rule allows to compute the posterior probability given likelihood, prior and evidence:

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

Where:

- **posterior** $P(y|x)$ is the probability that the class is y given that x was observed;
- **Likelihood** $p(x|y)$ is the probability of observing x given that its class is y;
- **Prior**: $P(y)$: is the prior probability of the class, without any evidence;
- **Evidence**: $p(x)$ is the probability of the observation, and by the law of total probability -sum rule (Definition 5.8), and the product rule (Definition 5.6), it can be computed as:

$$p(x) = \sum_{i=1}^{2} p(x|y)P(y)$$

When taking decision it's possible to commit mistake, for this reason we want to know the probability of making mistakes.
The probability of an error can be expressed as the probability of the error given x on all possible values of y. For example when considering the binomial, then

$$P(error|x) = \begin{cases} P(y_2|x) & \text{if we decide } y_1 \\ P(y_1|x) & \text{if we decide } y_2 \end{cases}$$

For the sum rule (Definition 5.8), the average error can be expressed as:

$$P(error) = \sum_x P(error, x)$$

And by the product rule (Definition 5.6), it can be expanded to:

$$P(error) = \sum_x P(error|x)p(x)$$

This is for the discrete case, if the variable were to be continuous, then the formula would be:

$$P(error) = \int_{-\infty}^{\infty} P(error|x)p(x)dx$$

Based on the error probability value, a decision can be made in order to minimize the error maximize the probability of success. This rule is called Bayes Decision Rule and is divided in two cases; the first one is for binary classification:

$$y_B = \text{argmax}_{y_i \in \{-1,1\}} P(y_i|x) = \text{argmax}_{y_i \in \{-1,1\}} p(x|y_i)P(y_i) \tag{6.1}$$

Mind that by the Bayes Rule (Definition 5.9) the last expression should have $p(x)$ at the denominator, but this doesn't change with the changing of y_i, hence it can be removed.
A second case is the multiclass case:

$$y_B = \text{argmax}_{y_i \in \{1,\dots,c\}} P(y_i|x) = \text{argmax}_{y_i \in \{1,\dots,c\}} p(x|y_i)P(y_i) \tag{6.2}$$

The **optimal rule** says that the probability of error given x:

$$P(error|x) = 1 - P(y_B|x)$$

Hence, the Bayes Decision Rule *minimizes* the probability of error.

6.2 Representing Classifiers

So when we are making a decision, that is trying to classify an input, we are trying to maximize the probability a posteriori. To execute the classification, a *classifier* is used, that is a function, or more that we hope are as similar to the reality as they can be.
A classifier can be represented as a set of **discriminant functions** $g_i(\boldsymbol{x}), i\{\in 1, \dots, c\}$, where c is the number of classes. For such reason, the Bayes optimal rule can be rewritten as:

$$y = \text{argmax}_{i \in \{1,\dots,c\}} g_i(\boldsymbol{x})$$

By comparison with Equation 5.3, we have:

$$\begin{aligned} g_i(\boldsymbol{x}) = &\quad P(y_i|\boldsymbol{x} = \frac{p(\boldsymbol{x}|y_i)P(y_i)}{p(\boldsymbol{x})} \\ = &\quad p(\boldsymbol{x}|y_i)P(y_i) \end{aligned}$$

Which by the rule of logarithms can be rewritten as:

$$g_i(\boldsymbol{x}) = \ln p(\boldsymbol{x}|y_i) + \ln P(y_i)$$

With these discriminant functions, the features space is divided into decisions regions $\mathcal{R}_1, \dots, \mathcal{R}_c$ such that:

$$\boldsymbol{x} \in \mathcal{R}_i \quad \text{if } g_i(\boldsymbol{x}) > g_j(\boldsymbol{x}) \ \forall j \neq i$$

Decision regions are separated by **decision boundaries**, that is regions in which ties occur among the the discriminant functions and hence the classifier is ambiguous.
Let's suppose to have two features (ω_1 and ω_2) and to divide the space via two discriminant functions, hence binary. The most common choice to model likelihood is using gaussians which allows to have a multivariate normal distribution. From Figure 6.1 it's possible to notice the gaussians functions which has two peaks which represents the values for which the class will be chosen. For some values of x, the value of g will be the same, hence the decision boundaries. Let's first recall the equation for the multivariate normal density:

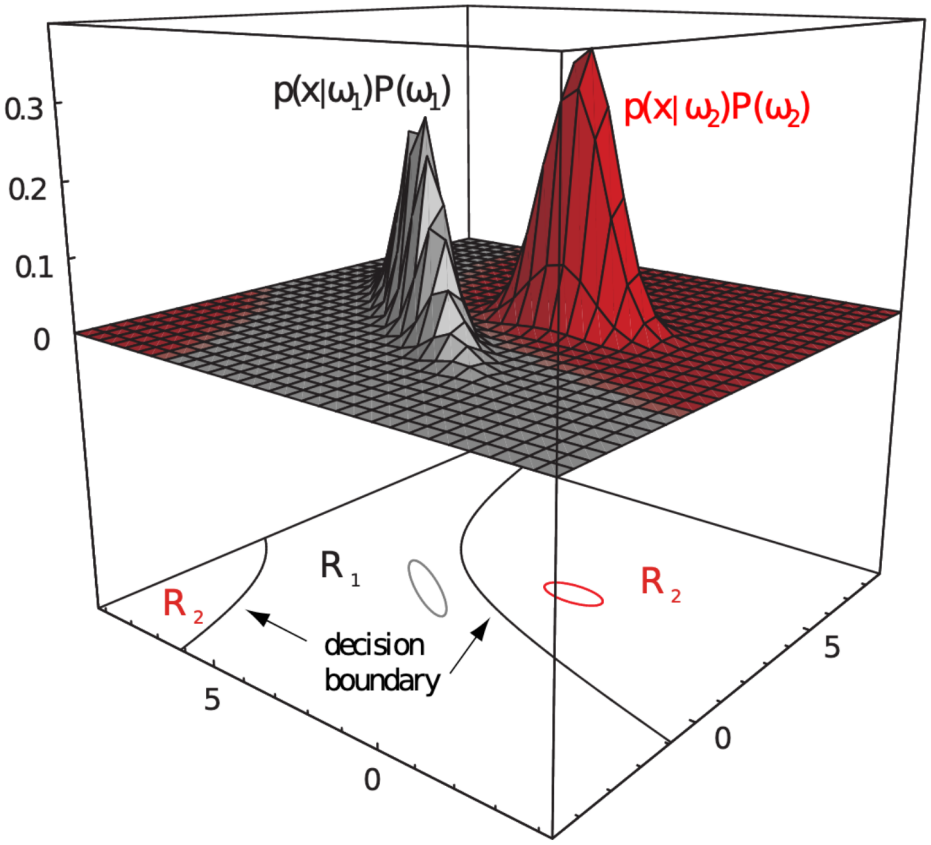


Figure 6.1: Example of classifiers.

$$p(\boldsymbol{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

And let's remember that the covariance matrix Σ is always symmetric and positive semi-definite, and becomes strictly positive if the dimension of the feature space is f.
This distribution can map the probability of $\boldsymbol{x}$ given $\boldsymbol{y}_i$:

$$p(\boldsymbol{x}|y_i) = \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)}$$

Given a multivariate normal distribution, the location of points of constant density are hyper-ellipsoids of constant Mahalanobis distance (Equation 5.10) from $\boldsymbol{x}$ to $\boldsymbol{\mu}$.

6.2.1 Discriminant function

Let's know take expression from before describing discriminant functions:

$$g(x) = \ln P(x|y_i) + \ln P(y_i)$$

And substitute the multivariate normal distribution inside:

$$g_i(\boldsymbol{x}) = \ln \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\Sigma_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)} + \ln P(y_i)$$

$$g_i(\boldsymbol{x}) = \ln \left[\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}}\right] + \left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right] + \ln P(y_i)$$

$$g_i(\boldsymbol{x}) = -\frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_i| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) + \ln P(y_i)$$

Since $\frac{-}{d}2ln2\pi$ does not depend on i then we can erase it:

$$g_i(\boldsymbol{x}) = -\frac{1}{2}\ln|\Sigma_i| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) + \ln P(y_i) \tag{6.3}$$

6.2.1.1 $\Sigma_i = \sigma^2 I$

All features are statistically independent which means that they have the same variance σ^2.
Now the covariance determinant $|\Sigma_i| = \sigma^{2d}$ is independent of i and can hence be ignored from the Equation 6.3.
The inverse of the covariance is $\Sigma_i^{-1} = ({}^1/_{\sigma^2})I$ and even though is not dependent of i, it cannot be cancelled because part of another term. The new discriminant functions become:

$$g_i(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\frac{I}{\sigma^2}(\boldsymbol{x}-\boldsymbol{\mu}_i) + \ln P(y_i)$$

Which equals to:

$$g_i(\boldsymbol{x}) = -\frac{||\boldsymbol{x}-\boldsymbol{\mu}_i||^2}{2\sigma^2} + \ln P(y_i)$$

Expanding the quadratic term, we obtain:

$$g_i(\boldsymbol{x}) = -\frac{1}{2\sigma^2}\left[\boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{\mu}_i^T\boldsymbol{x} + \boldsymbol{\mu}_i^T\boldsymbol{\mu}_i\right] + \ln P(y_i)$$

Discarding terms which are independent of i we obtain *linear discriminant functions*:

$$g_i(\boldsymbol{x}) = \underbrace{\frac{1}{\sigma^2}\boldsymbol{\mu}_i^T}_{\boldsymbol{w}_i^T}\boldsymbol{x} - \underbrace{-\frac{1}{2\sigma^2}\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i + \ln P(y_i)}_{w_{i0}} \tag{6.4}$$

This is in fact linear: said $\boldsymbol{w}_i^T = {}^1/_{\sigma^2}\boldsymbol{\mu}_i^T\boldsymbol{x}$ and $w_{i0} = -{}^1/_{2\sigma^2}\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i + \ln P(y_i)$, none of them depend on $\boldsymbol{x}$, hence $g_i(\boldsymbol{x})$ can be rewritten has:

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0}$$

which is indeed a line.
We have said that the boundaries are the region where the discriminant functions intersect, that is when they are equal $g_i(\boldsymbol{x}) = g_j(\boldsymbol{x})$. It's possible to find the equation of the decision boundaries:

$$g_i(\boldsymbol{x}) = g_j(\boldsymbol{x})$$
$$\frac{\boldsymbol{\mu}_i^T \boldsymbol{x}}{\sigma^2} - \frac{\boldsymbol{\mu}_i^T \boldsymbol{\mu}_i}{2\sigma^2} + \ln P(y_i) = \frac{\boldsymbol{\mu}_j^t \boldsymbol{x}}{\sigma^2} - \frac{\boldsymbol{\mu}_j^t \boldsymbol{\mu}_j}{2\sigma^2} + \ln P(y_j)$$
$$\underbrace{\frac{(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T}{\sigma^2}}_{\boldsymbol{w}^T} \boldsymbol{x} \underbrace{- \frac{\boldsymbol{\mu}_i \boldsymbol{\mu}_j}{\sigma^2} + \frac{\boldsymbol{\mu}_j^t \boldsymbol{\mu}_j}{\sigma^2} + \ln \frac{P(y_i)}{P(y_j)}}_{\boldsymbol{x}_0} = 0 \tag{6.5}$$

Which do not depend on $\boldsymbol{x}$ again hence making it linear again:

$$\boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{x}_0$$

This line (which is an hyperplane btw) is orthogonal to the vector $\boldsymbol{w}$, that is to the distance between $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$ since $\boldsymbol{w}$ is transposed with respect to $\boldsymbol{x}$.
Moreover the hyperplane passes through $\boldsymbol{x}_0$ which is based on the prior probabilities of classed being equal. If they are, then $\boldsymbol{x}_0$ is halfway between the means, while if they are not, $\boldsymbol{x}_0$ shifts away from the more likely mean. This can be seen in the logarithm: if $y_i = y_j$, then it would be $\ln 1 = 0$.

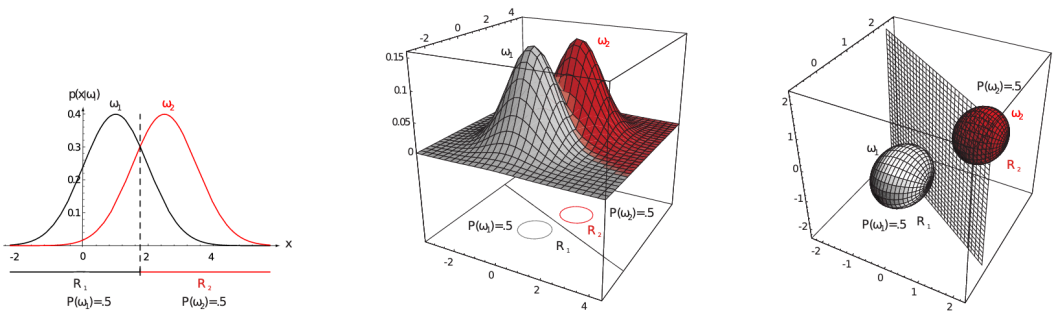


Figure 6.2: Case in which $P(y_i) = P(y_j)$. The discriminant function is a line which pass right through the middle of the distance of the two regions.

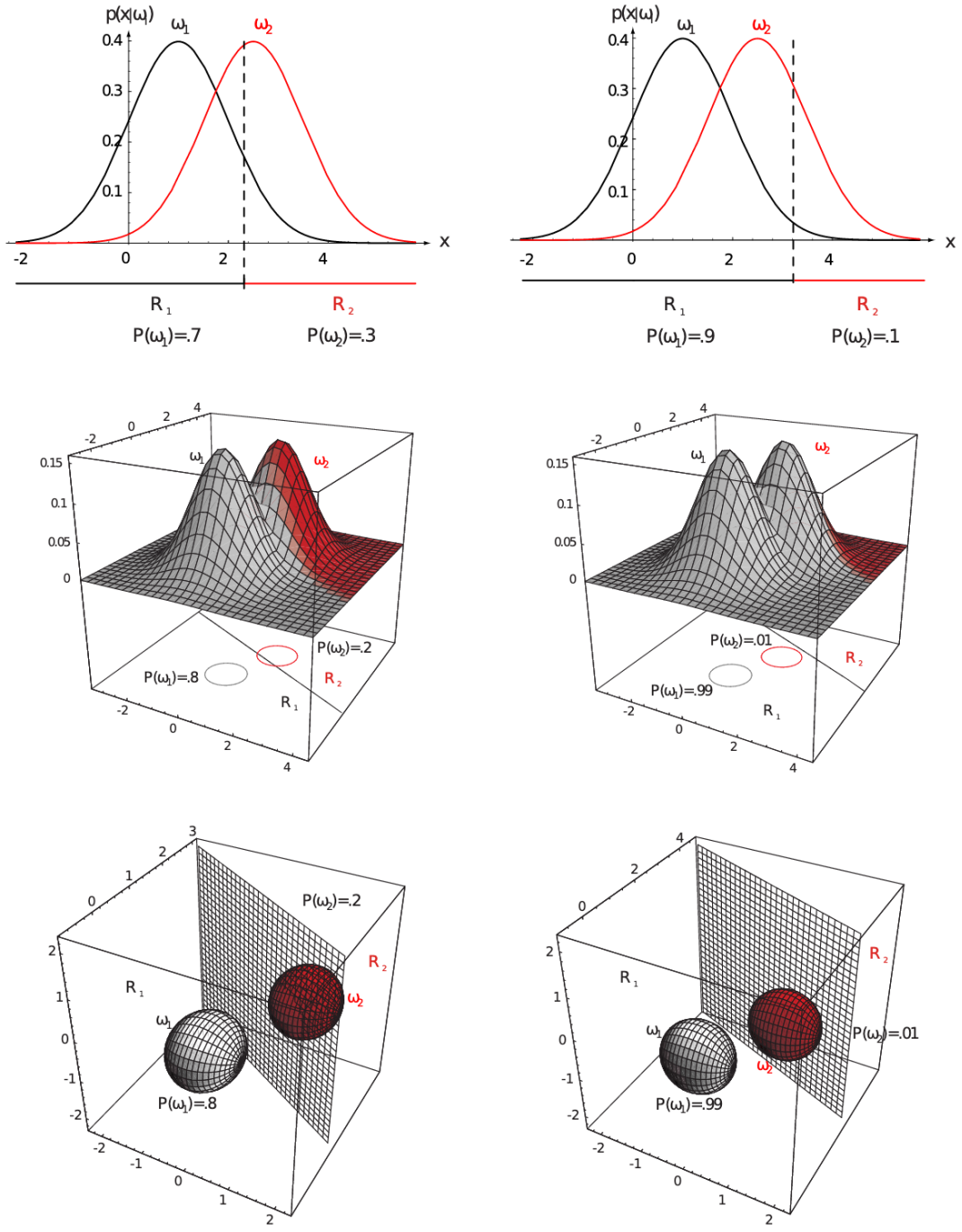


Figure 6.3: Case in which $P(y_i) \neq P(y_j)$. It's possible to notice that the more the probability difference increases, the more the line drifts away from that probability.

7 Maximum-Likelihood and Bayesian Parameter Estimation

Until now we have seen cases in which the parameters of the distributions were known. Now we'll look on how to estimate also this data.
Let's suppose to have data sampled from a probability distribution $p(x, y)$, but while the form of the distribution is known, the parameters of such are not. Let's suppose also to have a training set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ of examples sampled independently and identically distributed[1] according to $p(x, y)$. The goal is to estimate the unkown parameters of p from the training data $\mathcal{D}$.
Let's consider a multiclass classification problem: the training set can be divided into $\mathcal{D}_1, \ldots, \mathcal{D}_c$ subsets, one for each class. Mind that each subset contains iid examples for target class y_i: $\mathcal{D}_i = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$.
If the goal is to compute the probability of a class for a new example $\boldsymbol{x}$, then we can compute the probability of a class given the new example and all the dataset as described by Bayes rule, Definition 5.9:

$$P(y_i|\boldsymbol{x}, \mathcal{D}) = \frac{p(\boldsymbol{x}|y_i, \mathcal{D})p(y_i|\mathcal{D})}{p(\boldsymbol{x}|\mathcal{D})}$$

To really get the class of the new example we are going to compute the probability for all possible classes and then maximize over it.
Since $\boldsymbol{x}$ will be in a class y_i, it's safe to assume that it will be independent of $D_j, i \neq j$. It's possible to notice that the term $p(y_i|D)$ is the probability of finding a class inside the dataset, for example if the dataset contained students, and the class were to be Italian students, so it simply is:

$$p(y_i|\mathcal{D}) = \frac{|D_i|}{|D|}$$

As for the denominator, it's possible to marginalize (Sum rule 5.8) $p(\boldsymbol{x}|y_i, \mathcal{D}_i)p(y_i|\mathcal{D})$ over all possible classes:

$$p(\boldsymbol{x}|\mathcal{D}) = \sum_{y_i} p(\boldsymbol{x}|y_i, \mathcal{D}_i)p(y_i|\mathcal{D})$$

The idea of this step is that y_i does not depend on all $\mathcal{D}$, but only on $\mathcal{D}_i$.
The goal is to estimate *class-dependent* parameters θ_i for $p(\boldsymbol{x}|y_i, \mathcal{D}_i)$. There are two ways to solve this problem:

- **Bayesian estimation**: assumes that parameters θ_i are random variables with some known prior distribution. Once the example has been observed, the probability becomes a posterior distribution. Predictions for new examples are obtained integrating over all possible values for the parameters:

[1] *Independent*: each example is sampled independently from the others; *identically distributed*: all examples are sampled from the same distribution.

$$p(\boldsymbol{x}|y_i, \mathcal{D}_i) = \int_{\theta_i} p(\boldsymbol{x}, \theta_i|y_i, \mathcal{D}_i) d\theta_i$$

- **Maximum likelihood**: The parameters are computed as those maximising the probability of the observed examples $\mathcal{D}_i$, that is the training set for the class. For such reason they can be used to compute the probability for the new examples:

$$p(\boldsymbol{x}|y_i, \mathcal{D}_i) \approx p(\boldsymbol{x}|\theta_i)$$

7.1 Maximum likelihood

maximum likelihood Maximum likelihood If we can assume a prior distribution for the parameters $p(\theta_i)$, then maximum likelihood becomes maximum a-posteriori estimation and can be summarized as:

$$\theta_i^* = \text{argmax}_{\theta_i} p(\theta_i|\mathcal{D}_i, y_i) = \text{argmax}_{\theta_i} p(\mathcal{D}_i, y_i|\theta_i) p(\theta_i)$$

If a priori distribution is not available, then the solution is indeed maximum likelihood, that is maximizing the likelihood of the parameters with respect to the training samples:

$$\theta_i^* = \text{argmax}_{\theta_i} p(\mathcal{D}_i, y_i|\theta_i)$$

From now on, for simplicity, since each class y_i is trated independentely, we'll write $y_i, \mathcal{D}_\rangle$ as simply $\mathcal{D}$.
Given a training dataset $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ of iid examples for the target class y, assumed that the parameter vector $\boldsymbol{\theta}$ has a fixed but unknown value, we estimate such value by maximizing its likelihood with respect to the training data:

$$\theta^* = \text{argmax}_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta})$$

Since the examples are iid, the joint probability over $\mathcal{D}$ can be decomposed into a product:

$$\theta^* = \text{argmax}_{\boldsymbol{\theta}} \prod_{j=1}^{n} p(\boldsymbol{x}_j|\boldsymbol{\theta})$$

Moreover we don't generally like products, and since the goal is to maximize, it is possible to maximize over the logarithm of the product which equals to the sum of the logarithms. This is possible because the logarithm function is monotonic:

$$\theta^* = \text{argmax}_{\boldsymbol{\theta}} \ln p(\mathcal{D}|\boldsymbol{\theta}) = \text{argmax}_{\boldsymbol{\theta}} \ln \prod_{j=1}^{n} p(\boldsymbol{x}_j|\boldsymbol{\theta})$$

$$\theta^* = \text{argmax}_{\boldsymbol{\theta}} \sum_{j=1}^{n} \ln p(\boldsymbol{x}_j|\boldsymbol{\theta})$$

This is also called *maximizing the log-likelihood*.
The maxima can be obtained zeroing the gradient with respect to $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} = \sum_{j=1}^{n} \ln p(\boldsymbol{x}_j|\boldsymbol{\theta}) = \mathbf{0}$$

Since points zeroing the gradient can be local or global maxima.

7.1.1 Example – Univariate Gaussian

Let's consider a Gaussian distribution (Equation 5.6) with unknown μ and σ^2, then the log-likelihood $\mathcal{L}$ is:

$$\begin{aligned}\mathcal{L} = & \sum_{j=1}^{n} \ln \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} \\ = & \sum_{j=1}^{n} -\frac{1}{2\sigma^2}(x_j-\mu)^2 - \frac{1}{2}\ln 2\pi\sigma^2\end{aligned}$$

Now let's do the gradient with respect to θ, that is derive first for μ and then for σ^2:

$$\nabla_\theta = \begin{cases} \dfrac{\partial \mathcal{L}}{\partial \mu} \\ \dfrac{\partial \mathcal{L}}{\partial \sigma^2} \end{cases}$$

Let's do the one respect to μ first:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mu} = & -\sum_{j=1}^{n} \frac{1}{2\sigma^2}(2\mu - 2x_j) + 0 \\ = & \sum_{j=1}^{n} \frac{1}{\sigma^2}(x_j - \mu)\end{aligned}$$

Let's then do the one respect to σ^2:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \sigma^2} = & -\sum_{j=1}^{n} \frac{1}{2\sigma^2}(2\mu - 2x_j) - \frac{1}{2}\ln 2\pi\sigma^2 \\ = & -\sum_{j=1}^{n} \left[\frac{1}{2}(x_j-\mu)^2 \frac{\partial}{\partial\sigma^2}\frac{1}{\sigma^2}\right] - \frac{n}{2}\frac{\partial}{\partial\sigma^2}\ln 2\pi\sigma^2 \\ = & -\frac{n}{2}2\pi\frac{1}{2\pi\sigma^2} - \sum_{j=1}^{n}\frac{(x_j-\mu)^2}{2}(-1)\frac{1}{\sigma^4} \\ = & -\frac{n}{2\sigma^2} + \sum_{j=1}^{n}\frac{(x_j-\mu)^2}{2\sigma^4}\end{aligned}$$

The gradient with respect to the parameters is:

$$\nabla_\theta = \begin{cases} \dfrac{\partial \mathcal{L}}{\partial \mu} = \displaystyle\sum_{j=1}^{n} \frac{1}{\sigma^2}(x_j - \mu) \\ \dfrac{\partial \mathcal{L}}{\partial \sigma^2} = -\dfrac{n}{2\sigma^2} + \displaystyle\sum_{j=1}^{n} \frac{(x_j-\mu)^2}{2\sigma^4} \end{cases}$$

Now we need to zero the gradient. First for μ:

$$\begin{aligned}\sum_{j=1}^{n} \frac{1}{\sigma^2}(x_j - \mu) = 0 \\ \sum_{j=1}^{n}(x_j - \mu) = 0\end{aligned}$$

$$\sum_{j=1}^{n} x_j - \sum_{j=1}^{n} \mu = 0$$
$$\sum_{j=1}^{n} x_j = \sum_{j=1}^{n} \mu$$
$$\sum_{j=1}^{n} x_j = n\mu$$
$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j$$

Now let's do the one for σ^2:

$$-\frac{n}{2\sigma^2} + \sum_{j=1}^{n} \frac{(x_j - \mu)^2}{2\sigma^4} = 0$$
$$\sum_{j=1}^{n} \frac{(x_j - \mu)^2}{2\sigma^4} = \frac{n}{2\sigma^2}$$
$$\frac{1}{2\sigma^4}\sum_{j=1}^{n}(x_j - \mu)^2 = \frac{n}{2\sigma^2}$$
$$\frac{1}{\sigma^2}\sum_{j=1}^{n}(x_j - \mu)^2 = n$$
$$\sigma^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j - \mu)^2$$

So the maximum-likelihood estimates are:

$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j$$

$$\sigma^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j - \mu)^2$$

7.1.2 Example – Multivariate Gaussian

Let's recall the multivariate distribution (Equation 5.9):

$$p(\boldsymbol{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

The log-likelihood is:

$$\sum_{j=1}^{n} -\frac{1}{2}(\boldsymbol{x}_j - \boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}_j - \boldsymbol{\mu}) - \frac{1}{2}\ln(2\pi)^d |\Sigma|$$

And the maximum -likelihood estimates are:

$$\mu = \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{x}_j$$
$$\Sigma = \frac{1}{n}\sum_{j=1}^{n} (\boldsymbol{x}_j \boldsymbol{\mu})(\boldsymbol{x}_j - \boldsymbol{\mu})^T$$

It's possible to notice then, that for Gaussian distributions, the parameters are simply their empirical estimates over the sample:

- Gaussian mean is the sample mean;
- Gaussian covariance matrix is the mean of the sample covariances.

7.2 Bayesian estimation

Let's assume that parameters θ_i are *random variables* with some known *prior* distribution.
Predictions for new examples are obtained integrating over all possible values for the parameters:

$$p(\boldsymbol{x}|y_i, \mathcal{D}_i) = \int_{\boldsymbol{\theta}_i} p(\boldsymbol{x}, \boldsymbol{\theta}_i|y_i, \mathcal{D}_i) d\boldsymbol{\theta}_i$$

Probability of $\boldsymbol{x}$ given each class y_i is independent of the other classes y_i, for simplicity we can again write:

$$p(\boldsymbol{x}|y_i, \mathcal{D}_i) \rightarrow p(\boldsymbol{x}|\mathcal{D}) = \int_{\boldsymbol{\theta}} p(\boldsymbol{x}, \boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta}$$

Where $\mathcal{D}$ is a dataset for a certain class y, and $\boldsymbol{\theta}$ the parameters of the distribution.

$$p(\boldsymbol{x}|\mathcal{D}) = \int_{\boldsymbol{\theta}} p(\boldsymbol{x}, \boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} = \int_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta}, D) p(\boldsymbol{\theta}|D) d\boldsymbol{\theta} = \int_{\boldsymbol{\theta}} p(\boldsymbol{x}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} \tag{7.1}$$

Where $p(\boldsymbol{x}|\mathcal{D})$ can be easily computed since we have both form and parameters of the distribution, e.g., if it were a gaussian the form is Equation 5.6, while the parameters are μ and σ^2.
Since our goal is to estimate the parameters' posterior density given a training set $p(\boldsymbol{\theta}|\mathcal{D})$ by using Bayes rule (Definition 5.9), then we can write:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{D})} = \frac{\left(\prod_{j=i}^{n} (P(\boldsymbol{x}_i|\boldsymbol{\theta}) \right) P(\boldsymbol{\theta})}{P(\mathcal{D})}$$

$P(\mathcal{D}|\boldsymbol{\theta})$ is the likelihood probability that has been used also in the maximum likelihood method. In this case though, the parameters are weighted with respect to their probability avoiding overfitting which is a problem of maximum likelihood.
Mind that $P(\mathcal{D})$ is a constant independent of θ, hence it will have no influence in the final Bayesian decision, therefore it can be neglected.
At last, if final probability is needed, then $P(\mathcal{D})$ is computable via:

$$p(\mathcal{D}) = \int_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

7.2.1 Example – Univariate Normal, unknown μ, known σ^2

Example are drawn from a normal distribution, hence:

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = p(x|\mu) \sim N(\mu, \sigma^2)$$

Moreover the Gaussian mean distribution is itself a normal distribution:

$$P(\mu) \sim N(\mu_0, \sigma_0^2)$$

So the distribution of the parameter's posterior distribution can be computed as:

$$p(\mu|\mathcal{D}) = \frac{p(\mathcal{D}|\mu)p(\mu)}{p(\mathcal{D})} = \frac{1}{p(\mathcal{D})}\left(\prod_{j=1}^{n} p(x_j|\mu)\right)p(\mu)$$

Where we said that $p(x_j|\mu) \sim N(\mu, \sigma^2)$ and $p(\mu) \sim N(\mu_0, \sigma_0^2)$:

$$p(\mu|\mathcal{D}) = \frac{1}{p(\mathcal{D})}\left(\prod_{j=1}^{n} \underbrace{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}}}_{p(x_j|\mu)}\right) \underbrace{\frac{1}{\sigma_0\sqrt{2\pi}} e^{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}}}_{p(\mu)}$$

Since $p(\mathcal{D})$ is constant, it can be moved out of the product, and we'll call it $\alpha = {}^{1}/_{p(\mathcal{D})}$:

$$p(\mu|\mathcal{D}) = \alpha \prod_{j=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} \frac{1}{\sigma_0\sqrt{2\pi}} e^{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}}$$

Now we could use the exponentiation's properties:

$$e^x \times e^y = e^{x+y}$$

$$\prod_i e^{x_i} = e^{\Sigma_i x_i}$$

Before though, the constant parts could be moved outside the product and we'll call the new constant part outside the product as α':

$$p(\mu|\mathcal{D}) = \alpha \left(\frac{1}{2\sigma\sigma_0\pi}\right)^n \left(\prod_{j=1}^{n} \exp\left\{-\frac{(x_j-\mu)^2}{2\sigma^2}\right\}\right) \exp\left\{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}\right\}$$

$$p(\mu|\mathcal{D}) = \alpha' \exp\left\{\sum_{j=1}^{n} -\frac{(x_j-\mu)^2}{2\sigma^2}\right\} \exp\left\{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}\right\}$$

$$p(\mu|\mathcal{D}) = \alpha' \exp\left\{\frac{1}{2}\left(\sum_{j=1}^{n} \frac{(x_j-\mu)^2}{\sigma^2} + \frac{(\mu-\mu_0)^2}{\sigma_0^2}\right)\right\}$$

Let's now focus only on the exponent[2]:

$$-\frac{1}{2}\left(\sum_{j=1}^{n} \frac{(\mu-x_j)^2}{\sigma^2} + \frac{(\mu-\mu_0)^2}{\sigma_0^2}\right)$$

Now it's possible to expand the exponentials:

$$-\frac{1}{2}\left(\sum_{j=1}^{n} \frac{\mu^2 + x_j^2 - 2\mu x_j}{\sigma^2} + \frac{\mu^2 + \mu_0^2 - 2\mu\mu_0}{\sigma_0^2}\right)$$

And move the constant term wrt to the sum outside the sum:

$$-\frac{1}{2}\left(\frac{n\mu^2}{\sigma^2} + \frac{1}{\sigma^2}\sum_{j=1}^{n} x_j^2 - \frac{1}{\sigma^2}\sum_{j=1}^{n} 2\mu x_j + \frac{\mu^2}{\sigma^2} + \frac{\mu_0^2}{\sigma^2} - \frac{2\mu\mu_0}{\sigma_0^2}\right)$$

[2] $(x_j - \mu)^2 = (\mu - x_j)^2$.

First of all let's group the common terms:

$$-\frac{1}{2}\left(\mu^2\left(\frac{n}{\sigma^2}+\frac{1}{\sigma^2}\right)+\frac{\mu_0^2}{\sigma^2}+\frac{1}{\sigma^2}\sum_{j=1}^{n}x_j^2-2\mu\left(\frac{1}{\sigma^2}\sum_{j=1}^{n}x_j-\frac{\mu_0}{\sigma_0^2}\right)\right)$$

Let's now consider all the equation obtained:

$$p(\mu|\mathcal{D})=\alpha'\exp\left\{-\frac{1}{2}\left(\mu^2\left(\frac{n}{\sigma^2}+\frac{1}{\sigma^2}\right)+\frac{\mu_0^2}{\sigma^2}+\frac{1}{\sigma^2}\sum_{j=1}^{n}x_j^2-2\mu\left(\frac{1}{\sigma^2}\sum_{j=1}^{n}x_j-\frac{\mu_0}{\sigma_0^2}\right)\right)\right\}$$

Mind now that this is, by the exponentials properties, the same as:

$$p(\mu|\mathcal{D})=\alpha'\exp\left\{-\frac{1}{2}\left(\mu^2\left(\frac{n}{\sigma^2}+\frac{1}{\sigma^2}\right)-2\mu\left(\frac{1}{\sigma^2}\sum_{j=1}^{n}x_j-\frac{\mu_0}{\sigma_0^2}\right)\right)\right\}\exp\left\{+\frac{\mu_0^2}{\sigma^2}+\frac{1}{\sigma^2}\sum_{j=1}^{n}x_j^2\right\}$$

Where the second exponential is actually not dependent on μ, hence it can be moved insider α' which then becomes $\alpha''=\alpha'\times\exp\left\{+\frac{\mu_0^2}{\sigma^2}+\frac{1}{\sigma^2}\sum_{j=1}^{n}x_j^2\right\}$:

$$p(\mu|\mathcal{D})=\alpha''\exp\left\{-\frac{1}{2}\left(\mu^2\left(\frac{n}{\sigma^2}+\frac{1}{\sigma^2}\right)-2\mu\left(\frac{1}{\sigma^2}\sum_{j=1}^{n}x_j-\frac{\mu_0}{\sigma_0^2}\right)\right)\right\}$$

It's possible to notice that this posterior probability is quite similar with a normal, let's describe it as:

$$p(\mu|\mathcal{D})=\frac{1}{\sqrt{2\pi}\sigma_n}\exp\left\{-\frac{1}{2}\left(\frac{\mu-\mu_n}{\sigma_n}\right)^2\right\} \tag{7.2}$$

$$p(\mu|\mathcal{D})=\frac{1}{\sqrt{2\pi}\sigma_n}\exp\left\{-\frac{1}{2}\left(\frac{\mu^2}{\sigma_n^2}+\frac{\mu_n^2}{\sigma_n^2}-\frac{2\mu\mu_n}{\sigma_n^2}\right)\right\}$$

$$p(\mu|\mathcal{D})=\frac{1}{\sqrt{2\pi}\sigma_n}\exp\left\{\frac{\mu_n^2}{\sigma_n^2}\right\}\exp\left\{-\frac{1}{2}\left(\frac{\mu^2}{\sigma_n^2}-\frac{2\mu\mu_n}{\sigma_n^2}\right)\right\}=\alpha'''\exp\left\{-\frac{1}{2}\left(\frac{\mu^2}{\sigma_n^2}-\frac{2\mu\mu_n}{\sigma_n^2}\right)\right\}$$

By comparison with the equation obtained before it's possible to notice that:

$$\frac{1}{\sigma_n^2}=\frac{n}{\sigma^2}+\frac{1}{\sigma_0^2}\qquad\frac{\mu_n}{\sigma_n^2}=\frac{\mu_0}{\sigma_0^2}+\sum_{j=1}^{n}x_j$$

Let's first focus on the second equation. By multiplying and dividing for n the sum, it's possible to obtain the sample mean $\widehat{\mu}_n$:

$$\frac{\mu_n}{\sigma_n^2}=\frac{\mu_0}{\sigma_0^2}+n\frac{1}{n}\sum_{j=1}^{n}x_j=\frac{\mu_0}{\sigma_0^2}+n\widehat{\mu}_n$$

Now let's solve for σ_n first since μ_n depends on σ_n:

$$\frac{1}{\sigma_n^2}=\frac{n\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}$$

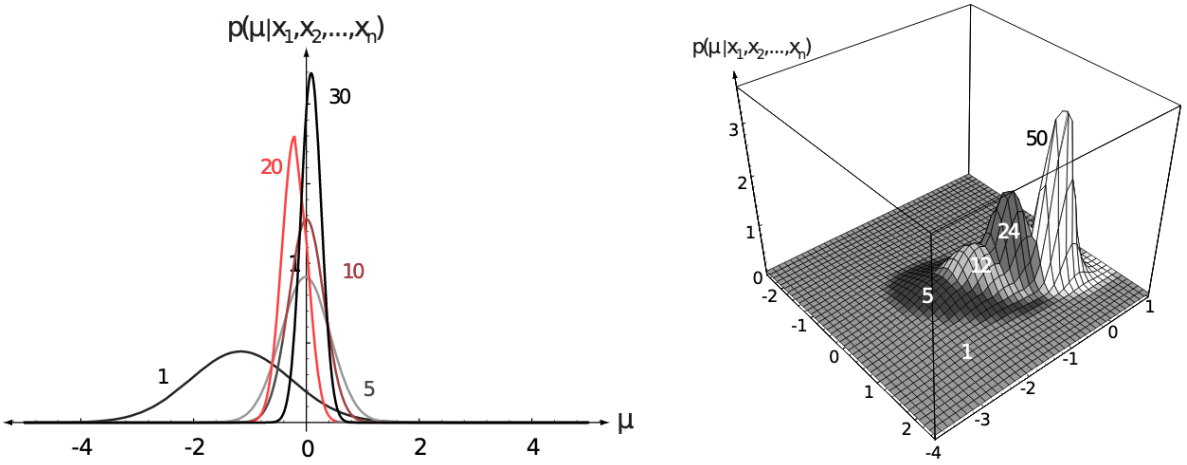
$$\sigma_n^2=\frac{\sigma^2\sigma_0^2}{n\sigma_0^2+\sigma^2}$$

It's now possible to solve for μ_n:

$$\mu_n = n\widehat{\mu}_n\sigma_n^2 + \frac{\mu_0\sigma_n^2}{\sigma_0}$$

$$\mu_n = n\widehat{\mu}_n\frac{\sigma^2\sigma_0^2}{n\sigma_0^2+\sigma^2} + \mu_0\frac{\sigma^2}{n\sigma_0^2+\sigma^2}$$

Notice that the men μ_n is a linear combination of the prior μ_0 and the sample mean $\widehat{\mu}_n$. If the number of samples increases, the sample mean starts to dominate over the prior mean and the variance decreases, making the distribution sharply peaked over its mean.



We stil haven't computed the class conditional density $p(x|\mathcal{D})$. Remember Equation 7.1:

$$P(x|\mathcal{D}) = \int_\mu P(x|\mu)P(\mu|\mathcal{D})\mu$$

Since we said that $p(x|\mu) \sim N(\mu, \sigma^2)$, and from what was achieved in Equation 7.2, it's possible to write:

$$p(x|\mathcal{D}) = \int_\mu N(\mu,\sigma^2)N(\mu_n,\sigma_n^2)d\mu$$

$$p(x|\mathcal{D}) = \int_\mu \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}\frac{1}{\sqrt{2\pi}\sigma_n}\exp\left\{-\frac{1}{2}\left(\frac{\mu-\mu_n}{\sigma_n}\right)^2\right\}d\mu$$

Let's now define $f(\sigma,\sigma_n)$:

$$f(\sigma,\sigma_n) = \int_\mu \exp\left\{-\frac{1}{2}\frac{\sigma^2+\sigma_n^2}{\sigma^2\sigma^n}\left(\mu - \frac{\sigma_n^2 x + \sigma^2\mu_n}{\sigma^2+\sigma_n^2}\right)^2\right\}d\mu$$

Then we could rewrite the previous probability as:

$$p(x|\mathcal{D}) = \int_\mu \frac{1}{2\pi\sigma\sigma_n}\underbrace{\exp\left\{-\frac{1}{2}\frac{(x-\mu_n)^2}{\sigma^2+\sigma_n^2}\right\}}_{\beta} f(\sigma,\sigma_n)$$

If we consider $p(x|\mathcal{D})$ as a function of x, then it is proportional to β in the previous equation and hence $p(x|\mathcal{D})$ is normally distributed with mean μ_n and variance $\sigma^2+\sigma_n^2$:

$$p(x|\mathcal{D}) \sim N(\mu_n, \sigma^2+\sigma_n^2)$$

This means that the probability of x given the dataset for the class is a Gaussian with mean equal to the posterior mean, and the variance equal to the sum of the known variance (σ^2) and the additional variance (σ^2)due to the uncertainty of the mean.
The same thing happens with the multivariate. Known that the examples are drawn from a multivariate and that also distribution of the mean for the multivariate is still a multivariate:

$$p(\boldsymbol{x}|\boldsymbol{\mu}) \sim N(\boldsymbol{\mu}, \Sigma)$$

$$p(\boldsymbol{\mu}) \sim N(\boldsymbol{\mu}_0, \Sigma_0)$$

It's possible to obtain the posterior and the class-conditional distributions:

$$p(\boldsymbol{\mu}|\mathcal{D}) \sim N(\boldsymbol{\mu}_n, \Sigma_n)$$

$$p(\boldsymbol{x}|\mathcal{D}) \sim N(\boldsymbol{\mu}_n, \Sigma + \Sigma_n)$$

7.3 Sufficient statics

Definition 7.1: Statistic

A function on a set of samples $\mathcal{D}$ is called statistic.

One example of statistics is the mean. We'll use statistics to evaluate the forecast.

Definition 7.2: Sufficient Statistic

A statistic $\phi(\mathcal{D})$is said to be *sufficient* for some parameters $\boldsymbol{\theta}$ if all we need to estimate the parameter is the statistic:

$$P(\mathcal{D}|\boldsymbol{s}, \boldsymbol{\theta}) = P(\mathcal{D}|\boldsymbol{s})$$

If $\boldsymbol{\theta}$ is a random variable, a sufficient statistic contains all relevant information $\mathcal{D}$ has for estimating it:

$$p(\boldsymbol{\theta}|\mathcal{D},) = \frac{p(\mathcal{D}|\theta, \boldsymbol{s})p(\boldsymbol{\theta}|\boldsymbol{s})}{p(\mathcal{D}|\boldsymbol{S})} = p(\boldsymbol{\theta}|\boldsymbol{s})$$

For example if we consider the Gaussian, the sample mean was all one would need from the dataset, the exact values are meaningless.

8 10/10/2019

8.1 Conjugate

Definition 8.1: Conjugate priors

Given a likelihood function $p(x|\theta)$, given a prior distribution $p(\theta)$, $p(\theta)$ is said to be a conjugate prior for $p(x|\theta)$ if the posterior distribution $p(\theta|x)$ is the same family as the prior $p(\theta)$.

Over what was seen before about the univariate normal case, other examples are described in Table 8.1. For example if we are trying to estimate samples taken from a Normal distribution ($p(x\boldsymbol{\theta}) \sim N(\mu, \sigma^2)$), and the modelled parameter is the mean, then the conjugate prior will still be a normal distribution.

Likelihood	Parameters	Conjugate Prior
Binomial	p (probability)	Beta
Multinomial	$\boldsymbol{p}$ (probability vector)	Dirichlet
Normal	μ (mean)	Normal
Multivariate Normal	$\boldsymbol{\mu}$ (mean vector)	Normal

Table 8.1: Table showing the conjugate prior of a likelihood depending on the parameter.

8.1.1 Bernoulli Distribution

This distribution models boolean events, either they are "success", or they are "failure". The parameter θ of the Bernoulli is simply the probability of success. The mass function (which will be used in the likelihood) is:

$$P(x|\theta) = \theta^x(1-\theta)^{1-x}$$

The conjugate prior is a Beta distribution:

$$P(\theta|\psi) = P(\theta|\alpha_h, \alpha_t) = \frac{\Gamma(\alpha)}{\Gamma(\alpha_h)\Gamma(\alpha_t)}\theta^{\alpha_h - 1}(1-\theta)^{\alpha_t - 1}$$

8.1.1.1 Example

Let's consider a dataset made out of the results of tossing a coin:

$$\mathcal{D} = \{H, H, T, T, T, H, H\}$$

The parameter θ indicates the probability of tossing head. The likelihood function becomes:

$$p(\mathcal{D}|\theta) = \theta \cdot \theta \cdot (1-\theta) \cdot (1-\theta) \cdot (1-\theta) \cdot \theta \cdot \theta = \theta^h(1-\theta)^t$$

First let's try to estimate θ via the maximum likelihood method seen in Section 7.1, that is basically deriving the log likelihood and seeing for what values it goes to zero.

$$\frac{\partial}{\partial\theta}\ln p(\mathcal{D}|\theta) = 0$$

$$\frac{\partial}{\partial\theta}\ln \theta^h(1-\theta)^t = 0$$

$$\frac{\partial}{\partial\theta}h\ln\theta + t\ln(1-\theta) = 0$$

$$h\frac{1}{\theta} + t\frac{1}{1-\theta}(-1) = 0$$

$$\frac{h}{\theta} = \frac{t}{1-\theta}$$

$$h - \theta h = t\theta$$

$$\theta = \frac{h}{h+t}$$

This tells us that h, t, that is the number of heads and tails in the dataset $\mathcal{D}$ respectively, are the sufficient parameters. This implies that for example we don't care about the order of the outcomes.
Let's now try to estimate the value of the parameter θ by using the Bayesian estimation. We have seen in Section 7.2 that the parameter posterior is proportional to:

$$P(\theta|\mathcal{D},\psi) \propto P(\mathcal{D}|\theta)P(\theta|\psi) \propto \theta^h(1\theta)^t\theta^{\alpha_h-1}(1-\theta)^{\alpha_t-1} = \theta^{h+\alpha_h-1}(1-\theta)^{t+\alpha_t-1}$$

That is the posterior is proportional to a Beta distribution with parameters $h+\alpha_h, t+\alpha_t$.
The prediction for a new event it's basically the expected value of the posterior Beta:

$$P(x|\mathcal{D}) = \int P(x|\theta)P(\theta|\mathcal{D},\psi)d\theta$$

The probability $P(x|\theta)$ is actually θ since the parameter represents the probability of success for an event.

$$P(x|\mathcal{D}) = \int \theta P(\theta|\mathcal{D},\psi)d\theta$$

Such integral is just the expected value of the Beta (Equation 5.8):

$$P(x|\mathcal{D}) = \mathrm{E}_{P(\theta|\mathcal{D},\psi)}[\theta] = \frac{h+\alpha_h}{h+t+\alpha_h+\alpha_t}$$

Our prior knowledge is encoded as a number $\alpha = \alpha_h + \alpha_t$ of imaginary experiments. α is called equivalent sample size. Notice that if $\alpha \to 0$, then the estimation is reduced to the classical maximum likelihood approach.

8.1.2 Multinomial distribution

Let's now consider an event with r states, that is r outcomes: $x \in \{x^1, \ldots, x^r\}$. For example tossing a six face dice is such an event with $r = 6$ states.
Such event can be modelled via a Multinomial distribution. To represent the outcomes we could use the one-hot encoding:

$$\boldsymbol{z}(x) = [z_1(x), \ldots, z_r(x)], z_k(x) = \begin{cases} 1 \text{ if } x = x^k \\ 0 \text{ otherwise} \end{cases}$$

For example if the outcome of rolling a dice was a 2, then $\boldsymbol{z}(x) = [0, 1, 0, 0, 0, 0]$.
The parameter is the vector $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_r]$ which contains the probability for each outcome.
Finally the probability mass function can be written as (Equation 5.4):

$$P(x|\boldsymbol{\theta}) = \prod_{k=1}^{r} \theta_k^{z_k(x)}$$

As from Table 8.1, the conjugate prior is a Dirichlet distribution (Equation 5.11):

$$P(\boldsymbol{\theta}|\psi) = P(\boldsymbol{\theta}|\alpha_1, \ldots, \alpha_r) = \frac{\Gamma(\alpha)}{\prod_{k=1}^{r} \Gamma(\alpha_k)} \prod_{k=1}^{r} \theta_k^{\alpha_k - 1}$$

Given a dataset $\mathcal{D}$ of N realizations, e.g., results of tossing a dice N times, then the likelihood function is:

$$P(\mathcal{D}|\boldsymbol{\theta}) = \prod_{j=1}^{N} \prod_{k=1}^{r} \theta_k^{z_k(x_j)} = \prod_{k=1}^{r} \theta_k^{N_k}$$

For example let's consider instead a dataset containing RGB values: $\mathcal{D} = \{R, R, R, G, B, G, B, R, B, G\}$.
The likelihood can be written as:

$$P(\mathcal{D}|\boldsymbol{\theta}) = \theta_R^4 \theta_G^3 \theta_B^3$$

Let's first apply maximum likelihood estimation:

$$\frac{\partial}{\partial \theta_k} \ln \prod_{k=1}^{r} \theta_k^{N_k} = 0$$

From which we obtain that:

$$\theta_k = \frac{N_K}{N}$$

Let's then apply Bayesian estimation. The parameter posterior is proportional to:

$$P(\boldsymbol{\theta}|\mathcal{D}, \psi) \propto P(\mathcal{D}|\boldsymbol{\theta}) P(\theta|\psi) \propto \prod_{k=1}^{r} \theta_k^{N_k + \alpha_k - 1}$$

From this is possible to observe that the posterior has a Dirichlet as expected from Table 8.1.
Finally let's compute the probability of a new event via Bayesian estimation:

$$P(x_k|\mathcal{D}) = \int \theta_k P(\boldsymbol{\theta}|\mathcal{D}, \psi) d\boldsymbol{\theta}$$

Which is exactly as the expected value for the Dirichlet distribution (Equation 5.12:

$$P(x_k|\mathcal{D}) = \mathrm{E}_{P(\boldsymbol{\theta}|\mathcal{D},\psi)}[\theta_k] = \frac{\alpha_k + N_k}{\sum_{i=0}^{r} (\alpha_i + N_i)} = \frac{N_k + \alpha_k}{N + \alpha}$$

9 Bayesian Networks

Bayesian Networks are a fast and easy way to create graphical models to represent more variables and their relations.
In general *probabilistic graphical models* are graphical representation of the *qualitative*[1] aspects of probability distributions allowing to:

- Visualize the structure of a probabilistic model in a simple and intuitive way, in particular the relations between the variables.
- Detect dependency or independency between variables without having to apply derivation rules.
- Express complex computations for inference and learning in terms of graphical manipulations.
- Represent multiple probability distributions with the same graph, abstracting from their quantitative aspects (e.g. discrete vs continuous distributions).

9.1 Structure

A Bayesian Network structure $\mathcal{G}$ is directed graph (graphical model).
Each node represents a random variable and each edge represents direct dependency between two random variables, that is one variable that is influenced by the other. This implies also that the father depends on the child, while the second does not depend on the first.
The structure hence can be described as encoding the independences assumptions:

$$\mathcal{I}_\ell(\mathcal{G}) = \{\forall i \; x_i \perp NonDecendants_{x_i} | Parents_{x_i}\}$$

NonDecendants are all those nodes that are not directly reachable from a node, that is the parents plus all the nodes that are not in its subtree. Let's consider node x_4 in Figure 9.1, its *NonDecendants* are its parents $\{x_1, x_2, x_3\}$ plus x_5 which is a non reachable node from x_4. In total the *NonDecendants* nodes are: $\{x_1, x_2, x_3, x_5\}$ The $\mathcal{I}_\ell$ dependency is said to be local ($_\ell$) since it's defined only wrt to a single variable.
A part from the structure, we need also to have a probability distribution. Let's consider a dataset $\mathcal{D}$ in which all variables are tied with the other by a distribution p which is a joint distribution. We'd like to represent qualitatively with a graph such distribution.
Since the Bayesian Network depends on the independences, it's possible to create a set $\mathcal{I}(p)$ of these by looking at the distribution p.
$\mathcal{G}$ is an *independency map* (**I-map**) for p if p satisfies the local independencies in $\mathcal{G}$:

$$\mathcal{I}_\ell(\mathcal{G}) \subseteq \mathcal{I}(p)$$

[1] The quantitative aspect is represented by the probabilistic distributions.

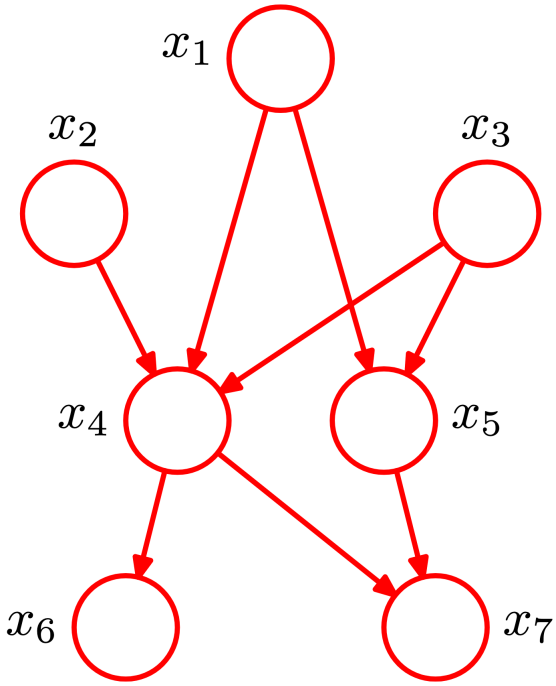


Figure 9.1: Example of Bayes Network.

The sufficient condition for $\mathcal{G}$ to be valid is that all the independences are also in p, while the opposite is not always true. Indeed, if some independence from p cannot be modelled into $\mathcal{G}$ via qualitative definition, then they must be modelled quantitatively.
Now we can describe p in terms of the graphical model, that is we can **factorize** the distribution based on the structure of the model.

Theorem 9.1

p is said to factorize according to $\mathcal{G}$ if:

$$p(x_1, \ldots, x_m) = \prod_{i=1}^{m} p(x_i | Parents_{x_i}) \tag{9.1}$$

Mind that this is a double implication: if $\mathcal{G}$ is an I-map for p, then p factorizes according to $\mathcal{G}$, then it's true also that if p factorizes according to $\mathcal{G}$, then $\mathcal{G}$ is an I-map for p. This can be proven as follows.

Proof. I-map⇒ factorization
If $\mathcal{G}$ is an I-map for p, then p satisfies at least these local independences:

$$\{\forall i \; x_i \perp NonDecendants_{x_i} | Parents_{x_i}\}$$

It's possible to order the variable in a topological order relative to $\mathcal{G}$, i.e.:

$$x_i \to x_j \Rightarrow i < j$$

That is the parents have a lower id than the children. Let us now decompose the joint probability using the chain rule as:

$$p(x_1, \ldots, x_m) = \prod_{i=1}^{m} p(x_i | x_1, \ldots, x_{i-1})$$

Finally local independences imply that for each x_i:

$$p(x_i|x_1,\dots,x_{i-1}) = p(x_i|Parents_{x_i})$$

Which by substitution:

$$p(x_1,\dots,x_m) = \prod_{i=1}^{m} p(x_i|Parents_{x_i})$$

□

Now we need to prove the inverse:

Proof. Factorization⇒ I-map
If p factorizes according to $\mathcal{G}$, the joint probability can be written as

$$p(x_1,\dots,x_m) = \prod_{i=1}^{m} p(x_i|Parents_{x_i})$$

Said $\mathcal{X}$ the set of all variables x_i: $\mathcal{X} = \{x_1,\dots,x_m\}$, let's consider variable x_m (repeat the following steps for the other variable), it's possible to write by the product and sum rules:

$$p(x_i|x_1,\dots,x_{m-1}) = \frac{p(x_1,\dots,x_m)}{p(x_1,\dots,x_{m-1})} = \frac{p(x_1,\dots,x_m)}{\sum_{x_m} p(x_1,\dots,x_m)}$$

Applying factorisation and isolating the only term containing x_m we get:

$$p(x_i|x_1,\dots,x_{m-1}) = \frac{\prod_{i=1}^{m} p(x_i|Parents_{x_i})}{\sum_{x_m}\prod_{i=1}^{m} p(x_i|Parents_{x_i})} =$$

$$= \frac{p(x_m|Parents_{x_m})\cancel{\prod_{i=1}^{m-1} p(x_i|Parents_{x_i})}}{\cancel{\prod_{i=1}^{m-1} p(x_i|Parents_{x_i})}\cancel{\sum_{x_m} p(x_m|Parents_{x_m})}}$$

The sum cancels out because summing all the value of a variable equals to 1.
Hence:

$$p(x_i|x_1,\dots,x_{m-1}) = p(x_m|Parents_{x_m})$$

□

For example consider Figure 9.1. It's possible to write:

$$p(x_1,\dots x_7) = p(x_1)p(x_2)p(x_3)p(x_4|x_1,x_2,x_3)p(x_5|x_2,x_3)p(x_6|x_4)p(x_7|x_4,x_5)$$

Mind that the joint probability $p(x_1,\dots,x_7)$ is largely influenced by the probability of node 4: $p(x_4|x_1,x_2,x_3)$. Now let's suppose that each variable defined a simple binary value, then the total number of possibility would be 2^7, against the 2^4 we found right now. This allows to be able to work with a lot of variables.

Definition 9.1: Bayesian Network

A Bayesian Network is a pair $(\mathcal{G}, p)$ where p factorizes over $\mathcal{G}$ and its represents as a set of conditional probability distribution associated with the nodes of $\mathcal{G}$

9.1.1 Example

Let's consider another example: genes A and B have independent prior probabilities and gene C can be enhanced by both A and B, with the following probability tables:

Gene	Value	$P(value)$
A	Active	0.3
A	Inactive	0.7

Gene	Value	$P(value)$
B	Active	0.3
B	Inactive	0.7

		A			
		Active		Inactive	
		B		B	
		Active	Inactive	Active	Inactive
C	Active	0.9	0.6	0.7	0.1
C	Inactive	0.1	0.4	0.3	0.9

Table 9.1: Table showing $p(A, B, |C)$.

9.1.2 Conditional Independence

First of let's recall Definition 5.7: two variables a, b are said to be independent written $a \perp b|\emptyset$ if:

$$p(a, b) = p(a)p(b)$$

Definition 9.2: Conditional Independence

Two variables a, b are conditionally independent given c, written $a \perp b|c$ if:

$$p(a, b|c) = p(a|c)p(b|c)$$

Graphical models allow to directly verify them through the **d-separation** criterion.

9.2 D-separation

Looking at some simpler graphs, some dependency rules can be inferred and then used on larger graphs by applying them to subgraphs.

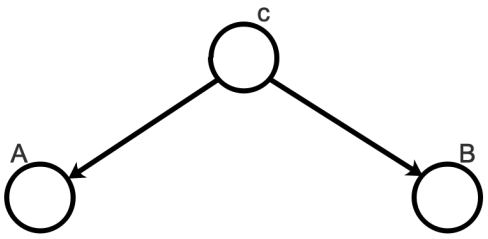
9.2.1 Two Nodes

Given two nodes, or they are independent, hence they are not connected, or they are dependent, hence they are connected.

9.2.2 Three Nodes

9.2.2.1 Tail-To-Tail

Also known as *common cause*.



As for the factorization Equation 9.1, the joint distribution can be expressed as:

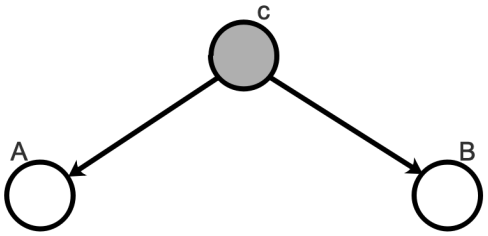
$$p(a, b, c) = p(a|c)p(b|c)p(c)$$

The tail-to-tail case says that a and b are *not independent*, written $a \not\perp b$. If c is not given then we have that:

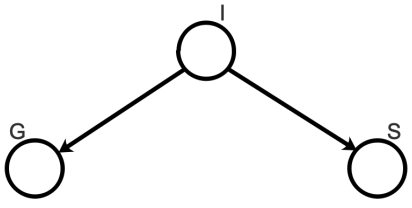
$$p(a, b) = \sum_c p(a|c)p(b|c)p(c) \neq p(a)p(b)$$

On the contrary, if c is given, then they are *conditionally independent*:

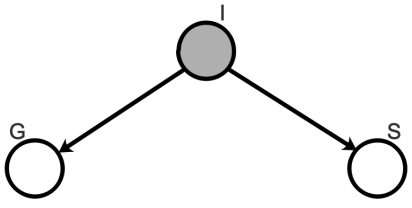
$$p(a, b|c) = \frac{p(a, b, c)}{p(c)} = p(a|c)p(b|c)$$



Let's consider an example: a company needs to choose the most intelligent (I) student between more students. For each student the variables grade (G) and score (S) are given. The following graph can be drawn since if the student is intelligent, then it should be seen in both grades and scores.



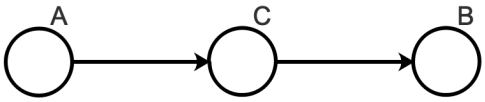
G and S are directly correlated, for example observing high scores could be indication of having a higher intelligence and hence also higher grades.



G and S are not directly dependent since once intelligence has been observed, the correlation disappear and nor G gives more information S, nor S gives more information about G than the what we already know.

9.2.2.2 Head-To-Tail

Also known as *indirect causal effect*.



The joint distribution can be expressed as:

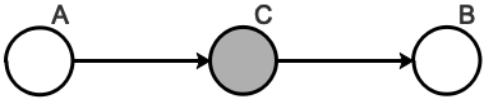
$$p(a, b, c) = p(b|c)p(c|a)p(a) = p(b|c)p(a|c)p(c)$$

If c is not given, then then a and b are *not independent*:

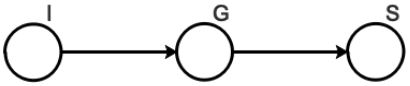
$$p(a, b) = p(a) \sum_c p(b|c)p(c|a) \neq p(a)p(b)$$

If instead c is given, then a and b are conditionally independent:

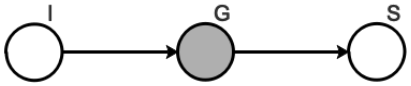
$$p(a, b|c) = \frac{p(b|c)p(a|c)p(c)}{p(c)} = p(b|c)p(a|c)$$



As before, let's consider the example of the intelligence of a student.



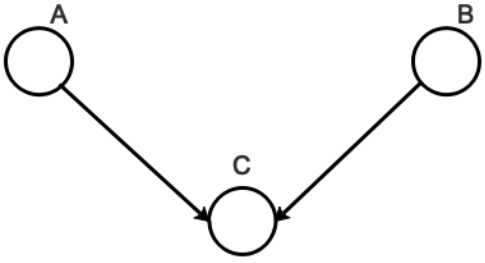
Intuitivly, if we observe that the student is intelligent, then we are more inclined to believe that it's grades G are good and that they will have a better score at the interview S, that is the probability of these latter events is higher conditioned on the observation that the student is intelligent.



Instead if we assume that G is observed, then it's intelligence no longer influences the score of the interview.

9.2.2.3 Head-To-Head

Also known as *common effect*.



The joint distribution can be expressed as:

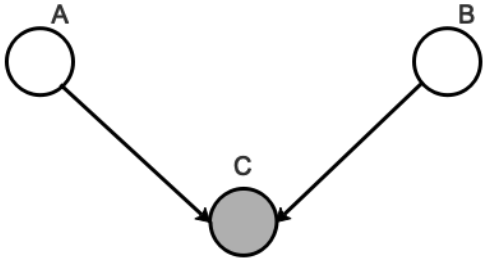
$$p(a, b, c) = p(c|a, b)p(a)p(b)$$

If c is not given, then then a and b are *independent*:

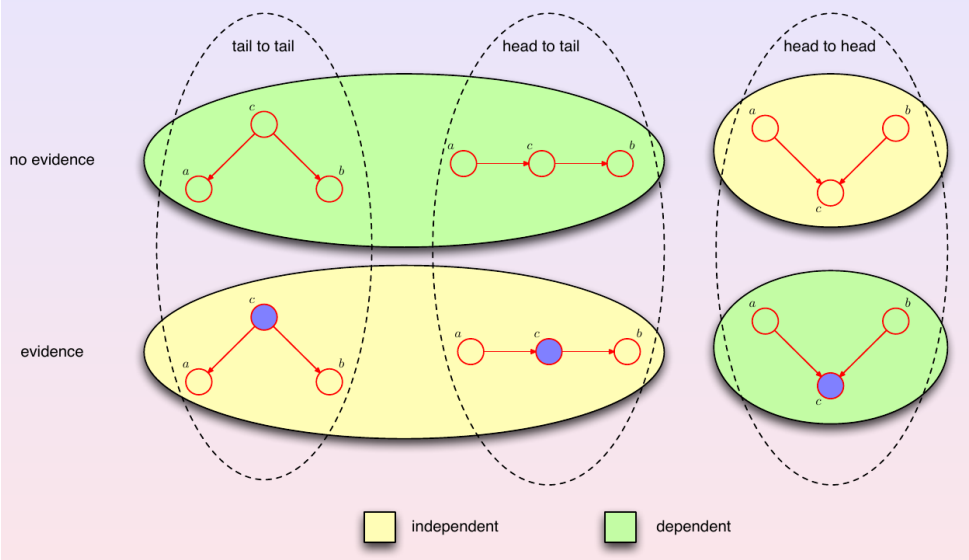
$$p(a,b) = \sum_c p(c|a,b)p(a)p(b) = p(a)p(b)$$

If instead c is given, then a and b are not conditionally independent, hence they are conditionally dependent:

$$p(a,b|c) = \frac{p(c|a,b)p(a)p(b)}{p(c)} \neq p(a|c)p(b|c)$$



Let's consider one last time the example of the student that is to be hired from a company. We have said that if c is not given, then a and b are independent, while if c is given they are dependent. Indeed, if c was the score of a test, then the student (G) and it was low, we could think that they are actually not that smart, but then if we were to observe that the test was difficult, then we could think that actually they are not *not* intelligent. Hence, a and b are actually correlated if c is given.

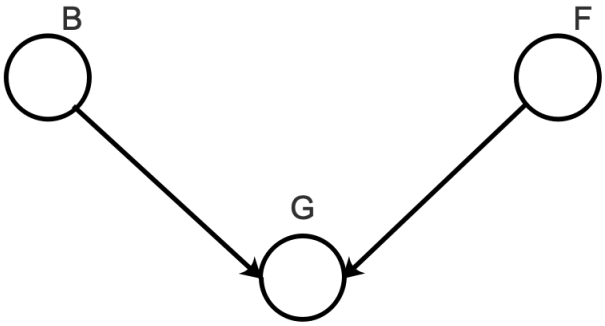


9.2.2.3.1 Example Head-To-Head Example

Let's consider a fuel system of a car. The fuel system is make of:

- Battery B: can either be charged ($B = 1$) or flat ($B = 0$);
- Fuel tank F: can either be full ($F = 1$) or empy ($F = 0$);
- Electric fuel gauge G: either full ($G = 1$) or empty ($G = 0$).

Let's now say that the probability of the battery to be charged is $P(B = 1) = 0.9$ and the probability for the fuel tank to be full is $P(F = 1) = 0.9$.
The electric fuel gauge is conditioned on both:



$$P(G=1|B=1,F=1)=0.8 \quad P(G=1|B=1,F=0)=0.2$$
$$P(G=1|B=0,F=1)=0.2 \quad P(G=1|B=0,F=0)=0.1$$

We want to compute the probability of having an empty tank:

$$P(F=0)=1-P(F=1)=0.1$$

What would happen to the probability if we were to notice that the fuel gauge was empty? In particular let's compute the probability of the fuel tank to be empty given the fact that the fuel gauge is empty $P(F=0|G=0)$. Such probability is not given, hence we need to find it. To compute $P(F=0|G=0)$ we could use Bayes:

$$P(F=0|G=0)=\frac{P(G=0|F=0)P(F=0)}{P(G=0)}$$

We need to compute $P(G=0|F=0)$ and $P(G=0)$ but they are easier to compute by the product rule:

$$\begin{aligned}
P(G=0|F=0) &= \sum_{B\in\{0,1\}} P(G=0,B|F=0) \\
&= \sum_{B\in\{0,1\}} P(G=0|B,F=0)P(B|F=0) \\
&= \sum_{B\in\{0,1\}} P(G=0|B,F=0)P(B) \\
&= P(G=0|B=0,F=0)P(B=0)+P(G=0|B=1,F=0)P(B=1) \\
&= 0.9\cdot 0.1+0.8\cdot 0.9=0.81
\end{aligned}$$

Notice that the passage from the second to the third equation it's possible since B and F are independent, hence $P(B|F=0)=P(B)$.
Similarly $P(G=0)$ is given by:

$$\begin{aligned}
P(G=0) &= \sum_{B\in\{0,1\}}\sum_{F\in\{0,1\}} P(G=0,B,F) \\
&= \sum_{B\in\{0,1\}}\sum_{F\in\{0,1\}} P(G=0|B,F)P(B)P(F)=0.315
\end{aligned}$$

In the end it's possible to see that by observing the fact that the fuel gauge is empty, then the probability of the fuel tank of being actually empty increases:

$$P(F=0|G=0)=\frac{P(G=0|F=0)P(F=0)}{P(G=0)}\simeq 0.257$$

Not as much as one would expect, but this is due to strong prior and an unreliable gauge.
Now let's consider the case in which we knew that also the battery is flat. How does the

probability $P(F = 0|G = 0, B = 0)$ change knowing this information?
It's possible to apply again Bayes Theorem and have:

$$P(F = 0|G = 0, B = 0) = \frac{P(G = 0|F = 0, B = 0)P(F = 0|B = 0)}{P(G = 0|B = 0)}$$

Definition 9.3: Active Trail

Let's consider a graph with three node X, Y, Z connected as in one of the cases before. The trail $X \rightleftharpoons Y \rightleftharpoons Z$ is said to be *active* if information can flow from X to Y via Z.

- Head-to-Tail $X \to Z \to Y$ is active if and only if Z **is not** observed;
- Tail-to-Tail $X \leftarrow Z \to Y$ is active if and only if Z **is not** observed;
- Head-to-Head $X \to Z \leftarrow Y$ is active if and only if Z **is** observed;

9.2.3 d-Separation – General Case

Obviously the majority of the graphs is not made of three nodes, but has many nodes. Suppose that we want to find a path that allows to go from one group nodes holding evidence to another. We can say whether this is possible or not by applying the before rules and by adding the following rule: a head-to-head node c unblocks the dependency path between its parents if either itself or *any of its descendants* receives evidence.

Definition 9.4: Descendant

A descendant of a node x is any node which can be reached from x with a path following the direction of the arrows.

Theorem 9.2: d-Separation General Case

Let $\mathcal{G}$ be a Bayesian Network structure and $X_1 \rightleftharpoons \ldots \rightleftharpoons X_n$ a trail in $\mathcal{G}$. Let $\boldsymbol{Z}$ be a subset of observed variables. The trail $X_1 \rightleftharpoons \ldots \rightleftharpoons X_n$ is active given $\boldsymbol{Z}$ if:

- Whenever we have a head-to-head structure $X_{i-1} \to X_i \leftarrow X_{i+1}$, then X_i or one of its descendants are in $\boldsymbol{Z}$;
- No other node along the trail is in $\boldsymbol{Z}$.

Note that if X_1 or X_n are in $\boldsymbol{Z}$, then the trail is not active.

Definition 9.5: Path Blocked

A path is said to be blocked if it includes at least one node such that either:

- The arrows on the path meet tail-to-tail or head-to-tail at the node and it is in C, or
- The arrows on the path meet head-to-head at the node and neither it nor any of its descendants is in C.

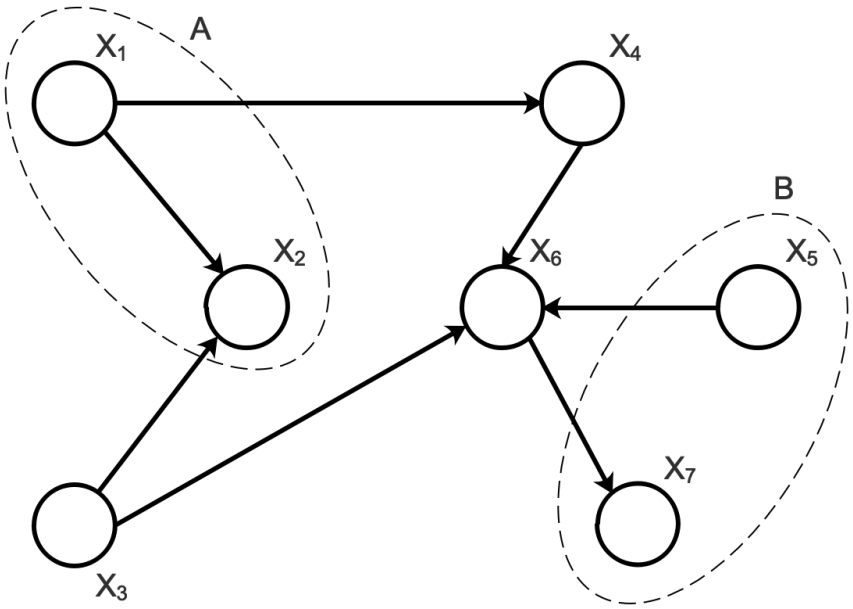
Definition 9.6: General d-separation Criterion

Given a generic Bayesian Network and nonintersecting arbitrary sets of nodes A, B, C, the sets A and B are d-separated by C, written $\text{dsep}(A; B|C)$ if all paths from any node in A to any node in B are blocked.

Note that d-separation *implies conditional independence*: the sets A and B are independent given C, written $A \perp B|C$ if they are d-separated by C.

9.2.3.1 Example 1

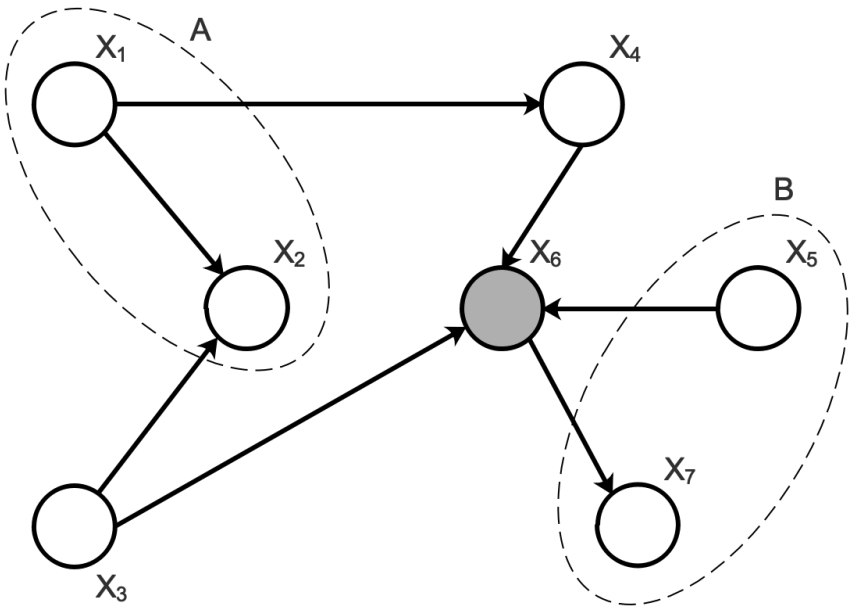
Let's consider the following graph:



and three independent sets: $A = \{X_1, X_2\}, B = \{X_5, X_7\}, C = \emptyset$. By applying the rules we need to find a possible path from one set to another. For example let's consider a first possibility made of the path $X_1 \to X_4 \to X_6 \leftarrow x_5$: the first three nodes are in a head-to-tail structure, we do not have any information regarding X_4, hence the path is active till this point. $X_4 \to X_6 \leftarrow X_5$ is a head-to-head structure and there is no evidence on X_6, hence the path is blocked.
Let's instead consider the path: $X_1 \to X_4 \to X_6 \to X_7$. We saw that the first three nodes are in a head-to-tail structure and there is no evidence, so the trail is active. Let's then look at the last three nodes: $X_4 \to X_6 \to X_7$: this is head-to-tail structure and there is no evidence on X_6, hence the trail is active and an actual path from the set A to the set B could be found.
It's possible to find also a path starting from X_2: $X_2 \leftarrow X_3 \to X_6 \to X_7$ which actually works because the first three nodes are in a tail-to-tail structure and there is no evidence on X_3, hence the trail is active, and the last three nodes are in a head-to-tail structure with no evidence on X_6.
Similarly to before, the path $X_2 \leftarrow X_3 \to X_6 \to X_5$ does not work because the last three nodes are in a head-to-head structure and there is no evidence on X_6.

9.2.3.2 Example 2

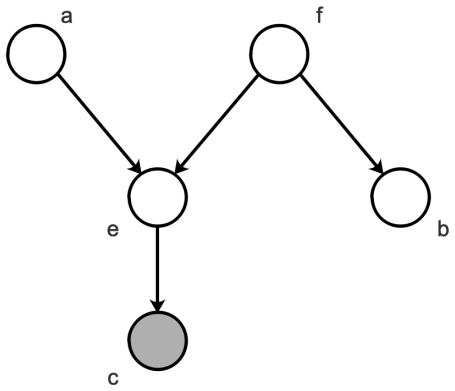
Let's consider the case in which X_6 is given evidence:

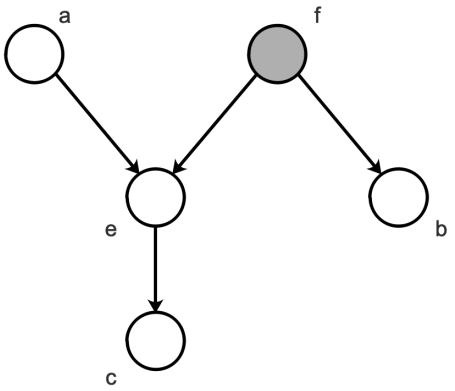


In this case the path $X_1 \to X_4 \to X_6 \leftarrow X_5$ is a correct path since X_4 holds no evidence hence the trail $X_1 \to X_4 \to X_6$ is active and X_6 holds evidence, so the structure $X_4 \to X_6 \leftarrow X_5$ is active too.

9.2.3.3 Example 3

Let's consider the following cases and decide whether a and b are d-separated:





In this case, the first three nodes meet a head-to-head structure and we have that c, which is a descendant of e, has evidence, hence the trail is active. The last three nodes are in a tail-to-tail structure, f does not have any evidence, hence the path is valid and a and b are not d-separated since there exists a path from a to b.

It's immediate to see that f is observed, hence the trail $e \leftarrow f \to b$ is not active, and also there is no evidence for e or any of its descendants. By Definition 9.6, a and b are d-separated given f: $a \perp b | f$.

9.3 Bayesian Networks independences

We saw how local independences work, that is independences between nodes, now we are going to define also global dependencies.

Definition 9.7: Local Independece

A Bayesian Networks structure $\mathcal{G}$ encodes a set of local independence assumptions:

$$\mathcal{I}_l(\mathcal{G}) = \{\forall i \; x_i \perp \text{nondescendants}_{x_i} | \text{parents}_{x_i}\}$$

Definition 9.8: Global Independence

A Bayesian Network structure $\mathcal{G}$ encodes a set of global (Markov) independence assumptions:

$$\mathcal{I}(\mathcal{G}) = \{(A \perp B|C) : \text{dsep}(A; B|C)\}$$

Notice that the global independences contain the local independences.

9.4 l-Equivalence

We can notice that different structures actually encode the same set of independences assumptions. Let's consider for example head-to-head in one direction or in the other: if the intermediate node does not present evidence, then the two structures encode the same independence. In Figure 9.2 are shown two equivalence classes.

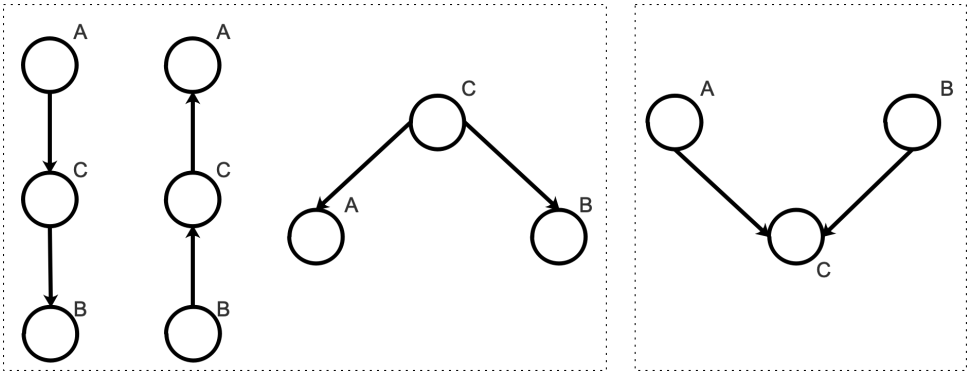


Figure 9.2: Two equivalence classes.

Definition 9.9: l-Equivalence

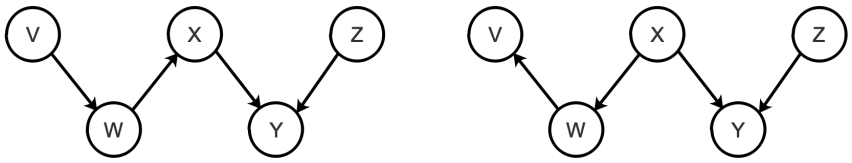
Two Bayesian Networks structures $\mathcal{G}$ and $\mathcal{G}'$ are l equivalent if $\mathcal{I}(\mathcal{G}) = \mathcal{I}(\mathcal{G}')$.

The space of Bayesian Network structures over the set of nodes $\mathcal{X}$ is partitioned into a set of mutually exclusive and exhaustive l-equivalence classes.
Note that l-equivalence of two graphs implies that any distribution P that can be factorized over on of these graphs can be factorized over the other too. Furthermore, there is no intrinsic property P that would allow us to associate it with one graph rather than an equivalent one. In other words, although we can determine, for a distribution $P(X, Y)$, whether X and Y are correlated, there is nothing in the distribution that can help us determine whether the correct structure is $X \to Y$ or $Y \to X$.

Definition 9.10: Skeleton

The skeleton of a Bayesian network graph $\mathcal{G}$ over a set of nodes $\mathcal{X}$ is an undirected graph over $\mathcal{X}$ that contains an edge $\{X, Y\}$ for every edge (X, Y) in $\mathcal{G}$.

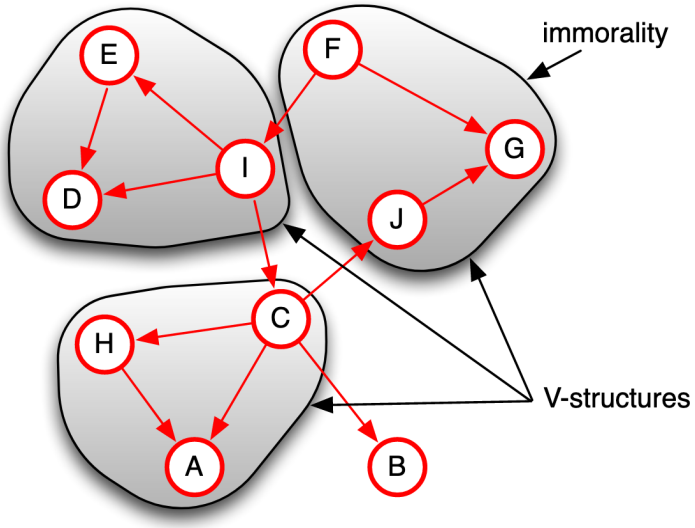
It's possible to prove that if two structures $\mathcal{G}$ and $\mathcal{G}'$ have the same skeleton and the same set of v-structures, then they are I-equivalent.



Considering the above structures, they are I-equivalent since they have the same skeleton and the same v-structure $(X \to Y \leftarrow Z)$.
This is actually a *sufficient condition*. Instead it's possible to have a *necessary and sufficient condition* by considering also **immoralities**.

Definition 9.11: Immorality

A v-structure $X \to Z \leftarrow Y$ is an immorality if there is no direct edge between X and Y. If there is such an edge, it is called a **covering edge** for the v-structure.



It's possible to notice that the v-structure $F \to G \leftarrow J$ is different from the other two since there is no edge between F and J. This implies that such v-structure is an immorality, while the others are not immoralities.
The following necessary and sufficient condition can be expressed an proven:

Theorem 9.3

Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two graphs over a set of nodes $\mathcal{X}$. Then $\mathcal{G}_1$ and $\mathcal{G}_2$ have the same skeleton and the same set of immoralities if and only if they are I-equivalent.

9.4.1 l-Maps

There can be two types of of maps: *minimal l-maps* and *P-maps*, that is perfect maps.
Notice that for a structure $\mathcal{G}$ to be an l-map for a distribution p, it does not need to encode all its independences, e.g., a fully connected graph is an l-map of any p defined over its variables.

Definition 9.12: Minimal l-Map

A minimal l-map for a distribution p is an l-map $\mathcal{G}$ which can't be reduced into a $\mathcal{G}' \subset \mathcal{G}$ that is also an l-map for p.

To reduce an l-map to another l-map means that by removing some edges we obtain again a valid l-map.
Note that by this definition, a minimal l-map for p does not necessarily capture all the independences in p. On the contrary a perfect P-map does:

Definition 9.13: P-Map

A structure $\mathcal{G}$ is a perfect map P-map for a distribution p if it captures all (and only) its independences:

$$\mathcal{I}(\mathcal{G}) = \mathcal{I}(p)$$

Notice that a P-map is also a minimal l-map, while the opposite is not valid.
There exists an algorithm that allows one to find a P-map for a distribution p, but it's exponential w.r.t. the *in-degree* of the P-map, that is the maximum number of connections of the nodes. Moreover the algorithms returns an equivalence class rather than a single structure.
Moreover there are some distributions for which is not possible to find P-maps. In these cases, indirect graphs called Markov Networks are used.

9.5 Making Bayesian Networks

The first step is to find a field-expert who can tell us the variables that we may need. For example, if we are trying to develop an algorithm to predict a disease, we need to know all the possible symptoms.
The next step is to actually define the variables that can be observed or that you can be interested in predicting, also latent variables can be useful.
By following *causality* considerations, we try to add edges to the variables. This would allow us to have a more interpretable network and also a sparser one.
Lastly we start to add the probabilities for the configurations and its best to (almost) never assign zero probabilities. If any data are available, we should use them to help in learning parameters, and also the structure.

9.6 Inference in Graphical Models

Bayesian Networks, as all the graphical models, actually models a joint probability.
Assume we have evidence $\boldsymbol{e}$ on the state of a subset of variables $\boldsymbol{E}$ in the model. Inference amounts at computing the posterior probability of a subset $\boldsymbol{X}$ of the non-observed variables given the observations:

$$P(\boldsymbol{X}|\boldsymbol{E} = \boldsymbol{e}) = \frac{P(\boldsymbol{X}, \boldsymbol{E})}{P(\boldsymbol{E} = \boldsymbol{e})}$$

The problem consists in estimating such joint probabilities when dealing with a large number of variables: directly working on the full joint probabilities would require time exponential in the number o variables. For instance, if all N variables are discrete and take one of K possible values, a joint probability table has K^N entries.

$$P(\boldsymbol{X}|\boldsymbol{E}=\boldsymbol{e}) = \frac{P(\boldsymbol{X},\boldsymbol{E})}{P(\boldsymbol{E}=\boldsymbol{e})} = \sum_{Y_1,\ldots,Y_n} P(\boldsymbol{X}, Y_1, \ldots, Y_n, \boldsymbol{E}=\boldsymbol{e})$$

Our goal is to exploit the structure in graphical models to do inference more efficiently.

9.6.1 Inference – Chain

The most simple structure we have seen is a chain of nodes where each variable has a connection to the following one.

In this way the probability of all variables in the chain becomes:

$$P(X) = P(X_1) \cdot P(X_2|X_1) \cdot \ldots \cdot P(X_N|X_{N-1})$$

The marginal probability of an arbitrary variable X_i is:

$$P(X_i) = \sum_{X_1}\sum_{X_2}\ldots\sum_{X_{i-1}}\sum_{X_{i+1}}\ldots\sum_{X_n} P(\boldsymbol{X})$$

Suppose that $i = N$, then it's possible to see that only $P(X_N|X_{N-1})$ is involved in the last summation which can be computed first giving a function of X_{N-1}:

$$\mu_\beta(X_{N-1})- = \sum_{X_N} P(X_N|X_{N-1})$$

Noe let's suppose that $i = N-1$, then the marginalization can be iterated as:

$$\mu_\beta(X_{N-1}) = \sum_{X_{N-1}} P(X_{N-1}|X_{N-2})\mu_\beta(X_{N-1})$$

One could decrease the value of i and obtain the general function:

$$\mu_\beta(X_i) = \sum_{X_{i+1}} P(X_{i+1}|X_i)\mu_\beta(X_{i+1})$$

Now let's try to do the opposite and start from the front:

$$\mu_\alpha(X_2) = \sum_{X_1} P(X_1)P(X_2|X_1)$$

$$\mu_\alpha(x_3) = \sum_{x_2} P(x_3|x_2)\mu_\alpha(x_2)$$

It's possible to see the marginalization of the chain as a product of μ_α and μ_β. Indeed, we could write:

$$P(X_i) = \mu_\alpha(X_i)\mu_\beta(X_i)$$

We could think of $\mu_\alpha(X_i)$ and $\mu_\beta(X_i)$ as messages passing from X_{i-1} to X_i and from X_{i+1} to X_i respectively[2]:

$$\mu_\alpha = \sum_{X_{i-1}} P(X_i|X_{i-1})\mu_\alpha(X_{i-1})$$

$$\mu_\beta = \sum_{X_{i+1}} P(X_{i+1}|X_i)\mu_\beta(X_{i+1})$$

Each outgoing message is obtained by multiplying the incoming message by the local probability and summing over the node values. Suppose we want to compute the marginal probabilities for a number of different variables $\boldsymbol{X}_i$. For each variable we should send a message from the beginning to the variable and from the variable to the end of the chain. Since the probabilities at each node is not going to change, then one could simply send a message from the beginning to the end and one from the end to the beginning. If all nodes store messages, we can compute any marginal probability as:

$$P(\boldsymbol{X}_i) = \mu_\alpha(\boldsymbol{X}_i)\mu_\beta(\boldsymbol{X}_i)$$

This does not count for evidence. If we were to consider also the case in which some nodes $\boldsymbol{X}_e$ are observed, then the formula would change and instead of summing over all possible values, we'd just use the observed values. For instance, let's consider the case in which we have 4 variables: $\boldsymbol{X} = \{X_1, X_2, X_3, X_4\}$, we could compute the probability of $\boldsymbol{X}$ as:

$$P(\boldsymbol{X}) = P(X_1)P(X_2|X_1)P(X_3|X_2)P(X_4|X_3)$$

The marginal probability of X_2 and observations $X_1 = x_{e_1}, X_3 = x_{e_3}$ is:

$$P(X_2, X_1 = x_{e_1}, X_3 = x_{e_3}) = \sum_{X_1}\sum_{X_3}\sum_{X_4} P(\boldsymbol{X})$$

$$P(X_2, X_1 = x_{e_1}, X_3 = x_{e_3}) = \sum_{X_1}\sum_{X_3}\sum_{X_4} P(X_1)P(X_2|X_1)P(X_3|X_2)P(X_4|X_3)$$

Since $X_1 = x_{e_1}$ and $X_3 = x_{e_3}$, we have:

$$P(X_2, X_1 = x_{e_1}, X_3 = x_{e_3}) = P(X_1 = x_{e_1})P(X_2|X_1 = x_{e_1})P(X_3 = x_{e_3}|X_2)\sum_{X_4} P(X_4|X_3 = x_{e_3})$$

Whne adding evidence, the message passing procedure computes the joint probability of the variable and the evidence, hence it has to be normalized to obtain the conditional probability given the evidence:

$$P(X_n|\boldsymbol{X}_e = \boldsymbol{x}_e) = \frac{P(X_n, \boldsymbol{X}_e = \boldsymbol{x}_e)}{\sum_{X_n} P(X_n, \boldsymbol{X}_e = \boldsymbol{x}_e)}$$

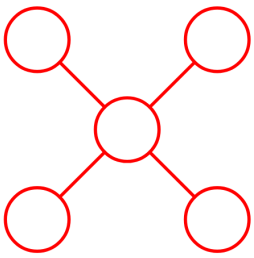
9.6.2 Inference – Trees

Obviously chains are not the only existing structure, but trees are another type of structure. In this case we can consider three cases:

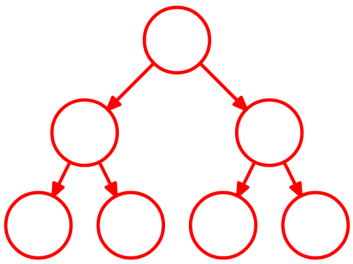
- **Undirected trees**: undirected graphs with a single path for each pair of nodes.
- **Directed trees**: directed graphs with a single node with no parentes (root) and all the other nodes with a single parent.

[2] α is passing the message forward, β is passing the message backwards.

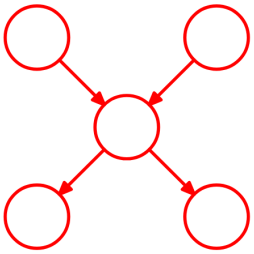
- **Directed polytrees**: directed graphs with multiple parents for node and multiple roots, but still a single (undirected) path between each pair of nodes.



Undirected trees.



Directed trees.



Directed polytrees.

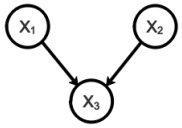
Since explaining these structures per se is quite troublesome, we are introducing **facto graphs**.

9.6.3 Inference – Factor Graph

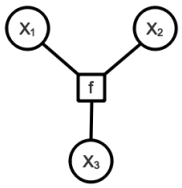
Factor graphs are an alternative graphical representation which is useful to better explain inference algorithms.
A factor graph is a graphical representation of a graphical model highlighting its factorization, i.e., its conditional probabilities. A factor graph has one node for each node in the original graph, one additional node for each factor (evidence) and undirected links to each of the node variables in the factor.
Mind that each node factor needs to be linked to all the nodes from which the decomposition comes from.
Let's consider for example the following probability distribution:

$$P(X_3|X_1, X_2)P(X_1)P(X_2)$$

We know that a Bayesian Network representation can be:



We could notice that actually X_3 is conditioned on X_1 and X_2 and hence we could create a factor node f as follows:

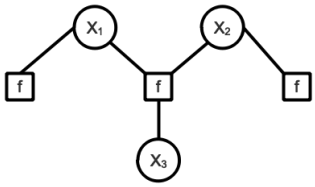


Actually f can be thought as a function:

$$f(X_1, X_2, X_3) = P(X_3|X_1, X_2)P(X_1)P(X_2)$$

Now then it's possible to add also other two factor nodes which would express the probability of the nodes X_1 and X_2.

$$f_c(X_1, X_2, X_3) = P(X_3|X_1, X_2)$$
$$f_a(X_1) = P(X_1)$$
$$f_b(X_2) = P(X_2)$$



9.6.4 Sum-Product Algorithm

The sum-product algorithm is an efficient algorithm for exact inference on tree-structured graphs. As the version for the chains, also this is a message passing algorithm.
This algorithm is applicable to undirected models, i.e., Markov Networks, as well as directed ones, i.e., Bayesian Networks. We will now present it on factor graphs assuming a tree-structured graph giving rise to a factor graph which is a tree.

9.6.4.1 Computing Marginals

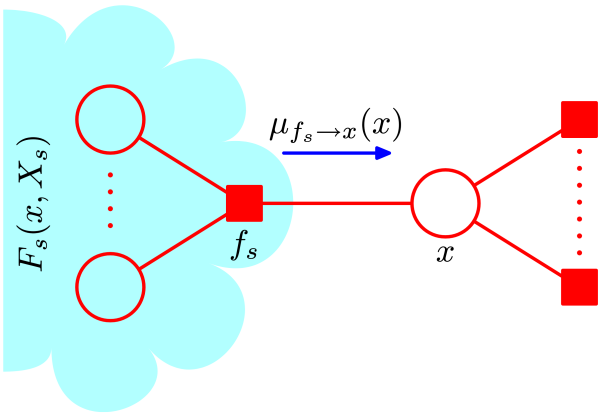
To compute the probability of a single variable, one sums over the probabilities of all variables except the one for which they wish to compute the marginal.

$$P(X) = \sum_{\boldsymbol{X} \setminus X} P(\boldsymbol{X})$$

Since X is a node, then the probability $P(X)$ of an event is the product of all the incoming messages from the neighbouring factors f_s:

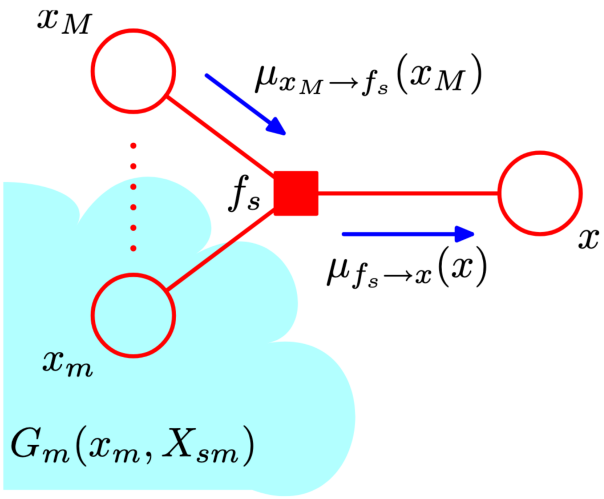
$$P(X) = \prod_{f_s \in \text{ne}(X)} \mu_{f_s \to X}(X)$$

where $\text{ne}(X)$ are the neighbours of X.



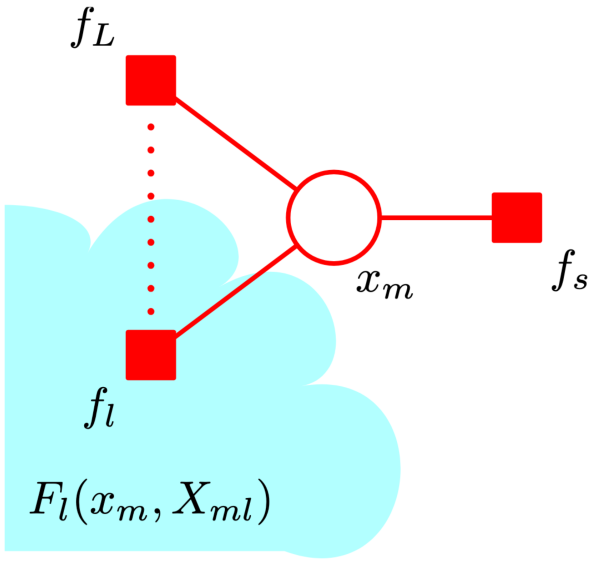
Each factor message is the product of messages coming from nodes other than X, times the factor, summed over all possible values of the factor variables other than X:

$$\mu_{f_s \to X}(X) = \sum_{X_1} \dots \sum_{X_M} f_s(X, X_1, \dots, X_M) \prod_{X_m \in \text{ne}(f_s) \setminus X} \mu_{X_m \to f_s}(X_m)$$



Now it's possible to notice that each message from node X_m to factor f_s is the product of the factor message to X_m coming from factors other than f_s:

$$\mu_{X_m \to f_s}(X_m) = \prod_{f_l \in \text{ne}(X_m) \setminus f_s} \mu_{f_l \to X_m}(X_m)$$



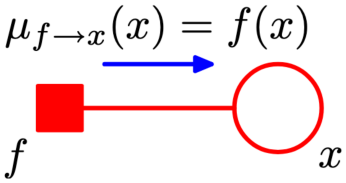
Knowing these informations, we can move to the algorithm. First of all the message passing starts from the leaves, either factors or nodes.
If the leaf is a factor, the message is initialized to the factor itself, indeed there would be no X_m different from the destination on which to sum over:

$$\mu_{f \to x}(x) = f(x)$$

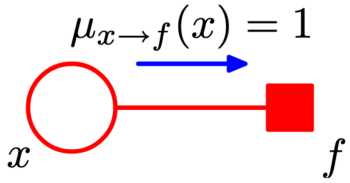
If the leaf is a node, the message is initialized to 1:

$$\mu_{x \to f}(x) = 1$$

Now the node X whose marginal has to be computed is designed as root. Messages are sent from all leaves to their neighbours. Each internal node sends its message towards the root as soon as it has received the messages from all the other neighbours. Once the root has collected all messages, the marginal can be computed as the product of them.



If the leaf is a factor, then the message is initialized to the factor itself.

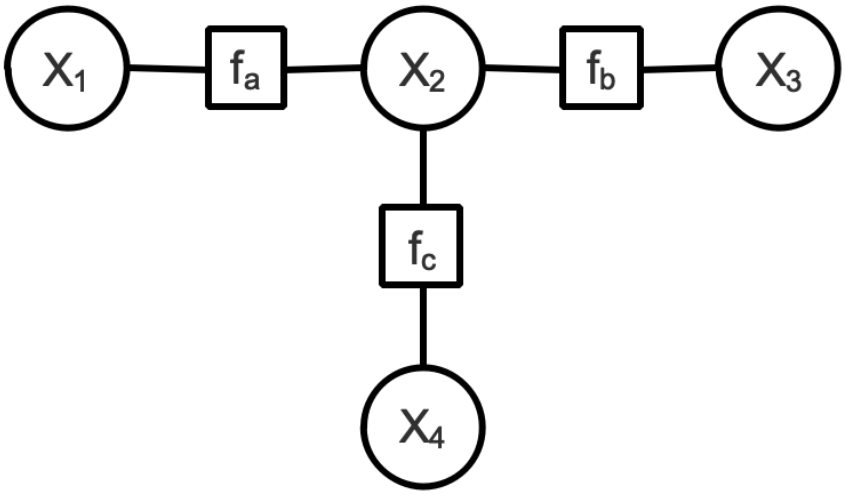


If the leaf is a node, then the message is initialized to 1.

One should remember that for the chain structure the best approach was to send a message from one end to the other and another one on the opposite direction. Similarly, this can be done also for a tree structure: once the root as computed the marginal, send the value back to all the leaves. This allows to pass all messages in all directions using only twice the number of computations used for a single marginal.

9.6.4.2 Example

Let's consider the following factor graph:



Which has the following joint distribution:

$$P(\boldsymbol{X}) = f_a(X_1, X_2) f_b(X_2, X_3) f_c(X_2, X_4)$$

Let's consider the case in which we choose X_3 has the root of the tree, hence X_1 and X_4 are the leafs. Since they are nodes, then the messages are initialized to 1:

$$\mu_{X_1\to f_a}(X_1) = 1$$

$$\mu_{X_4\to f_c}(X_4) = 1$$

By definition, the message from factors to nodes is the product of the messages coming from nodes other then X_i, times the factor, summed over all possible values of the factor variables other than X_i. Since for both factors f_a and f_c there is only one node before, hence one single message which value is 1, then it's possible to write:

$$\mu_{f_a\to X_2}(X_2) = \sum_{X_1} f_a(X_1, X_2)$$

$$\mu_{f_c\to X_2}(X_2) = \sum_{X_4} f_a(X_2, X_4)$$

Now we need to send a message from the node X_2 to the factor f_b, and by definition, such message is the product of the message arriving to X_2:

$$\mu_{X_2 \to f_b}(X_2) = \mu_{f_a \to X_2}(X_2)\mu_{f_c \to X_2}(X_2)$$

Then the message from f_b to X_3 can be written as:

$$\mu_{f_b \to X_3}(X_3) = \sum_{X_2} f_b(X_2, X_3)\mu_{X_2 \to f_b}(X_2)$$

Now we need to do the opposite, that is, start from X_3 and go towards the leaves. Since X_3 is a node, then the message is initialized to 1:

$$\mu_{X_3 \to f_b}(X_3) = 1$$

From factor f_b to factor node X_2 we have:

$$\mu_{f_b \to X_2}(X_2) = \sum_{X_3} f_b(X_2, X_3)$$

From X_2 to factors f_a and f_c:

$$\mu_{X_2 \to f_a}(X_2)) = \mu_{f_b \to X_2}(X_2)\mu_{f_c \to X_2}(X_2)$$

$$\mu_{X_2 \to f_c}(X_2)) = \mu_{f_b \to X_2}(X_2)\mu_{f_a \to X_2}(X_2)$$

Finally the message to the leaves are:

$$\mu_{f_a \to X_1}(X_1)) = \sum_{X_2} f_a(X_1, X_2)\mu_{X_2 \to f_a}(X_2)$$

$$\mu_{f_c \to X_4}(X_4)) = \sum_{X_2} f_c(X_2, X_4)\mu_{X_2 \to f_c}(X_2)$$

Now that we have all the information that we need, we could for example decide to compute the marginal probability of X_2. We know that the marginal can be computed as the product of all the incoming messages:

$$P(\boldsymbol{X}) = \mu_{f_a \to X_2}(X_2)\mu_{f_b \to X_2}(X_2)\mu_{f_c \to X_2}(X_2)$$

By expanding each message as written before, we obtain:

$$= \left[\sum_{X_1} f_a(X_1, X_2)\right]\left[\sum_{X_3} f_b(X_2, X_3)\right]\left[\sum_{X_4} f_c(X_2, X_4)\right]$$

$$= \sum_{X_1}\sum_{X-3}\sum_{X_4} f_a(X_1, X_2)f_b(X_2, X_3)f_c(X_2, X_4)$$

But the product of all factors is nothing but the initial distribution:

$$= \sum_{X_1}\sum_{X-3}\sum_{X_4} P(\boldsymbol{X})$$

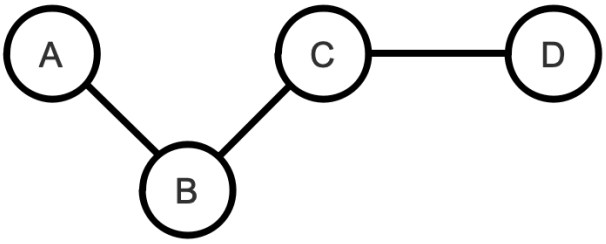
This "proves" that we could use just two passing of messages to compute the marginal probabilities.

9.6.4.3 Example – Adding Evidence

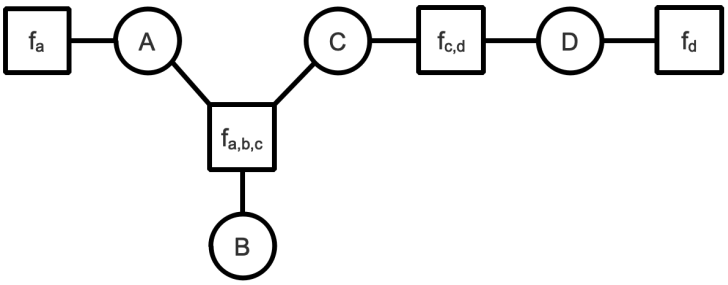
As said before, if any node is observed, then we simply use their observed values instead of summing over all possible values when computing their messages. After normalization this gives the conditional probability given the evidence.
Consider for example the following probability distribution:

$$P(B|A,C)P(A)P(C|D)P(D)$$

Let's now represent this via a Bayesian Network:



We could convert it to a factor graph by looking at its probability distribution and adding a function for each probability:



Let's consider the case in which we'd like to compute the marginal of B. Notice that when we know which node we want to compute the probability of, then we can just use one passage of messages. So let's make B the root of the tree and start from f_a and f_d which are the leaves. Since the leaves are factors, then we can use directly their probability:

$$\mu_{f_A \to A} = P(A)$$

$$\mu_{f_D \to D} = P(D)$$

This are the incoming messages to A and D respectively. Now let's compute the outgoing messages; since both A and B do not have any factor except for f_A and f_D respectively, then their messages are:

$$\mu_{A \to f_{A,B,C}}(A) = \mu_{f_A \to A} = P(A)$$
$$\mu_{D \to f_{C,D}}(D) = \mu_{f_D \to D} = P(D)$$

Now let's wait for the branch of A and focus on the branch of C and D. We'll send a message from factor $f_{C,D}$ to node C. In this case we'll have to sum over all possible values of D:

$$\mu_{f_{C,D} \to C}(C) = \sum_D P(C|D)\mu_{D \to f_{C,D}}(D) = \sum_D P(C|D)P(D)$$

We can now also send the message from C to factor $f_{A,B,C}$:

$$\mu_{C \to f_{A,B,C}}(C) = \mu_{f_{C,D} \to C}(C) = \sum_D P(C|D)P(D)$$

At last the factor $f_{A,B,C}$ sends a message to the node B:

$$\begin{aligned}\mu_{f_{A,B,C}\to B}(B) &= \sum_A \sum_C P(B|A,C)\mu_{C\to f_{A,B,C}}(C)\mu_{A\to f_{A,B,C}}(A) \\ &= \sum_A \sum_C P(B|A,C)P(A)\sum_D P(C|D)P(D)\end{aligned}$$

Since the marginal probability of a node is the product of all incoming messages to the node, we have:

$$\begin{aligned}P(B) = \mu_{f_{A,B,C}\to B}(B) &= \sum_A \sum_C P(B|A,C)P(A)\sum_D P(C|D)P(D) \\ &= \sum_A \sum_C \sum_D P(B|A,C)P(A)P(C|D)P(D) \\ &= \sum_A \sum_C \sum_D P(A,B,C,D)\end{aligned}$$

9.6.5 Most Probable Configuration

Consider the following case: we are given a joint probability distribution $p(\boldsymbol{X})$ and our goal is to find the configuration for variables $\boldsymbol{X}$ having the highest probability:

$$\boldsymbol{X}^{\max} = \text{argmax}_{\boldsymbol{X}} P(\boldsymbol{X})$$

for which the probability is:

$$P(\boldsymbol{X}^{\max}) = \max_{\boldsymbol{X}} P(\boldsymbol{X})$$

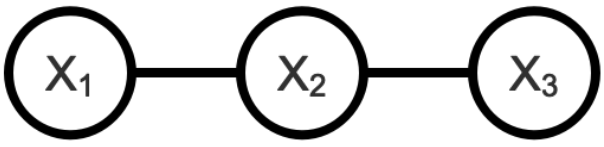
For instance, we are not interested in computing the probability that in the next days it'll rain, but in knowing which is the most probable weather during the next week.
Notice that we are trying to find the configuration which is jointly maximal for all variables, that is if the variables are not independent, we cannot simply compute $P(X_i)$ for each i and then maximize.
As for the sum-product algorithm, we can exploit the distribution factorization to efficiently compute the maximum:

$$P(\boldsymbol{X}^{\max}) = \max_{\boldsymbol{X}} P(\boldsymbol{X}) = \max_{X_1} \dots \max_{X_M} P(\boldsymbol{X})$$

All we have done is to replace the sum from the sum-product algorithm with a max function. When doing these maximizations, we are actually ignoring all the variables that are not the variable we are trying to maximize.

9.6.5.1 Chain Example

Let's consider a simple chain:



And let's say that the variables can take the following values: $X_1 = \{\text{Rainy, Cloudy, Sunny}\}$, $X_2 = \{T, F\}$, $X_3 = \{T, F\}$. The probability for X_1 and the conditional probabilities are the following:

X_1	
R	0.2
C	0.2
S	0.6

X_2	X_1 = R	X_1 = C	X_1 = S
T	0.2	0.4	0.8
F	0.8	0.6	0.2

X_3	X_2 = T	X_2 = F
T	0.8	0.4
F	0.2	0.6

The joint probability of a chain is:

$$P(\boldsymbol{X}) = P(X_3|X_2)P(X_2|X_1)P(X_1)$$

Our goal is to find the values for X_1, X_2, X_3 that maximize the joint probability.
Each maximization can be seen as an outgoing message (μ_β) where instead of having a sum we are actually maximizing. For instance:

$$\mu_\beta(X_2) = \max_{X_3} P(X_3|X_2) = \begin{cases} 0.8 & \text{if } X_2 = T \\ 0.6 & \text{if } X_2 = F \end{cases}$$

The result is due to the fact that, looking at the rightmost table, fixating a value for X_2, it's possible to find the maximum probability, for instance, if X_2 is T, then the $P(X_3 = T|X_2 = T) > P(X_3 = F|X_2 = T)$.
The same can be done to compute the value for X_2:

$$\mu_\beta(X_1) = \max_{X_2} P(X_2|X_1)\mu_\beta(X_2) = \begin{cases} 0.48 & \text{if } X_1 = R \\ 0.36 & \text{if } X_1 = C \\ 0.64 & \text{if } X_1 = S \end{cases}$$

In order to compute this we have to look at the center table above. First let's consider the case in which $X_1 = R$, then we could have $P(X_1 = R|X_2 = T) = 0.2$, or $P(X_1 = R|X_2 = F) = 0.8$. We want to check both of these possibilities and take the highest value:

$$P(X_1 = R|X_2 = T)\mu_\beta(X_2 = T) = 0.2 \cdot 0.8 = 0.16$$

$$P(X_1 = R|X_2 = F)\mu_\beta(X_2 = F) = 0.8 \cdot 0.6 = 0.48$$

Then we choose the highest value, that is 0.48. The same reasoning is applied for $P(X_1 = C|X_2)$ and $P(X_1 = S|X_2)$

$$P(X_1 = C|X_2 = T)\mu_\beta(X_2 = T) = 0.4 \cdot 0.8 = 0.48$$

$$P(X_1 = C|X_2 = F)\mu_\beta(X_2 = F) = 0.6 \cdot 0.6 = 0.36$$

$$P(X_1 = S|X_2 = T)\mu_\beta(X_2 = T) = 0.8 \cdot 0.8 = 0.64$$

$$P(X_1 = S|X_2 = F)\mu_\beta(X_2 = F) = 0.2 \cdot 0.6 = 0.12$$

At last we can compute the maximum configuration probability:

$$P(\boldsymbol{X}^{\max}) = \max_{X_1} P(X_1)\mu_\beta(X_1) = \max_{X_1} \begin{bmatrix} 0.48 \cdot 0.2 \\ 0.36 \cdot 0.2 \\ 0.64 \cdot 0.6 \end{bmatrix} = 0.64 \cdot 0.6 = 0.384$$

The most likely configuration for X_1 is:

$$\text{argmax}_{X_1} P(X_1)\mu_\beta(X_1) = S$$

Then to know the most likely configuration for X_2 all we need to do is to go back to the computation of $\mu_\beta(X_1)$ and look to which value gave the highest probability when $X_1 = S$, that is when $X_2 = T$, $P(X_1 = S|X_2 = T)\mu_\beta(X_2) = 0.64$.
The same is done for X_3 keeping in mind that now we know that $X_2 = T$: given such information, the highest probability is obtained for $X_3 = T$, $P(X_3 = T|X_2 = T) = 0.8$.
This action of retrieving the correct configuration going backwards is called **backpropagation**.

9.6.5.2 Tree

As we saw in the example before, the max-product algorithm can be seen as a message passing over the graph. The algorithm is thus easily applied to tree-structured graphs via their factor trees: all one needs to do is to change the sum with a maximization:

$$\mu_{f\to X}(X) = \max_{X_1,\ldots,X_M}\left[f(X,X_1,\ldots,X_M)\prod_{X_m\in \mathrm{ne}(f)\setminus X}\mu_{X_m\to f}(X_m)\right]$$

$$\mu_{X\to f}(X) = \prod_{f_l\in \mathrm{ne}(X)\setminus f}\mu_{f_l\to X}(X)$$

As the sum-product algorithm, once a root X_r is chosen, the leaves send messages towards the root. The probaility of the maximal configuration is readily obtained as:

$$P(\boldsymbol{X}^{\max}) = \max_{X_r}\left[\prod_{f_l\in \mathrm{ne}(X_r)}\mu_{f_l\to X_r}(X_r)\right]$$

The maximal configuration for the root is obtained as:

$$X_r^{\max} = \mathrm{argmax}_{X_r}\left[\prod_{f_l\in \mathrm{ne}(X_r)}\mu_{f_l\to X_r}(X_r)\right]$$

We now need to recover the maximal configuration for the other variables. When sending a message towards x, each factor node should store the configuration of the other variables which gave the maximum:

$$\phi_{f\to X}(X) = \mathrm{argmax}_{X_1,\ldots,X_M}\left[f(X,X_1,\ldots,X_M)\prod_{X_m\in \mathrm{ne}(f)\setminus X}\mu_{X_m\to f}(X_m)\right]$$

When the maximal configuration for the root node X_r has been obtained, it can be used to retrieve the maximal configuration for the variables in neighbouring factors from:

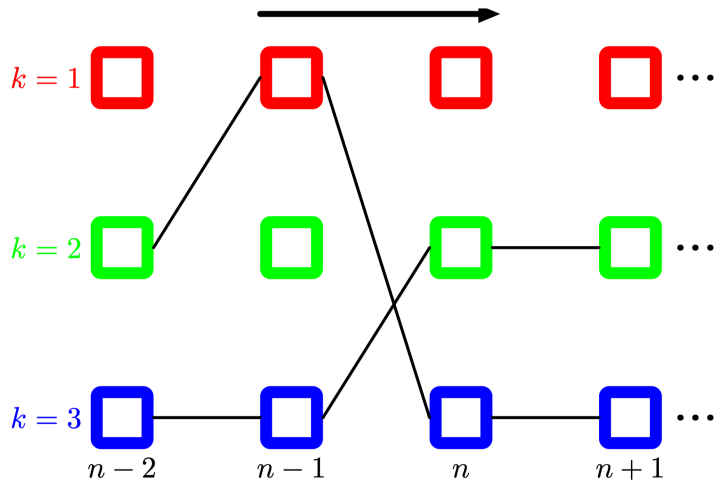
$$X_1^{\max},\ldots,X_M^{\max} = \phi_{f\to X_r}\left(X_r^{\max}\right)$$

The procedure can be repeated *back-tracking* to the leaves, retrieving maximal values for all variables.
For the chain we saw before, back-tracking resolves in:

$$\begin{aligned}
X_N^{\max} &= \mathrm{argmax}_{X_N}\mu_{f_{N-1,N}\to X_N}(X_N)\\
X_{N-1}^{\max} &= \phi_{f_{N_1,N}\to X_N}(X_N^{\max})\\
X_{N-2}^{\max} &= \phi_{f_{N_2,N-1}\to X_{N-1}}(X_{N-1}^{\max})\\
&\vdots\\
X_1^{\max} &= \phi_{f_{1,2}\to X_2}(X_2^{\max})
\end{aligned}$$

9.6.5.3 Trellis

It's possible to visualize the best configuration by using a trellis diagram which shows the K possible states of each variable X_n one per row. For each state k of a variable X_n, $\phi_{f_{n-1,n}\to X_n}(X_n)$ defines a unique (maximal) previous state, linked by an edge in the diagram.
Once the maximal state for the last variable X_N is chosen, the maximal states for other variables are recovered following the edges backwards.


$k=1$
$k=2$
$k=3$
$n-2$
$n-1$
n
$n+1$

10 30/10/2019

10.0.0.1 Underflow

The max-product algorithm relies on products, but the multiplication of small numbers (probabilities) can lead to underflow problems. This can be solved by considering $\log(\max(P(\boldsymbol{X})))$ instead of simply $\max(P(\boldsymbol{X}))$.
Since the logarithm is monotonic, we have that the proper maximal configuration is recovered:

$$\log\left(\max_{\boldsymbol{X}} P(\boldsymbol{X})\right) = \max_{\boldsymbol{X}} \log(P(\boldsymbol{X}))$$

Moreover we know that, by the properties of logarithms, the multiplication of logarithms can be converted into a sum, giving the *max-sum* algorithm.

10.0.1 Exact Inference – Junction Tree Algorithm

Let's see how to deal with inference when the network structure is not as nice as the ones we have seen until now. Indeed many applications require graphs with (undirected loops), so we acutally need an extension of the max-product algorithm to generic graphs.
Such new algorithm is called **junction tree** algorithm. As the name suggests, this algorithm does not work on factor graphs, but on junction trees, i.e., tree-structured graphs with nodes containing clusters of variables of the original graph. Then a message passing scheme similar to the sum-product and max-product algorithms is run on the junction tree.
Notice though that such an algorithm is exponential on the maximal number of variables in a cluster, making it intractable for large complex graphs.

10.0.2 Approximate Inference

If the graph is too large or complex and the junction tree algorithm is not usable, then we can resort to *approximate* techniques such as:

- *Loopy belief propagation*: message passing on the original graph even if it contains loops;
- *Variational methods*: deterministic approximations, assuming the posterior probability (given the evidence) factorizes in a particular way;
- *Sampling methods*: approximate posterior is obtained sampling from the network.

10.0.2.1 Loopy Belief Propagation

The idea is simply to apply the sum-product algorithm even if it is not guaranteed to provide an exact solution.
We assume all nodes are in condition of sending messages, i.e., they already received a constant

1 message from all neighbours.
A message passing schedule is chosen in order to decide which nodes start sending messages, e.g., flooding.
The information than flows many times around the graph (because of loops), each message on a link replaces the previous one and is only based on the most recent messages received from the other neighbours.
The algorithm can eventually converge depending on the specific model over which it is applied.

10.0.2.2 Sampling Methods

Given the joint probability distribution $P(\boldsymbol{X})$, we want to compute $P(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{E} = \boldsymbol{e})$. A sample from the distribution is an instatiation of all the variables $\boldsymbol{X}$ according to the probability P, that is the "output" of the distribution probability P applied to some input $\boldsymbol{X}$.
Samples can be used to approximate the probability of a certain assignment $\boldsymbol{Y} = \boldsymbol{y}$ for a subset $\boldsymbol{Y} \subset \boldsymbol{X}$ of the variables: we simply count the fraction of samples which are consistent with the desired assignment.
We usually need to sample from a posterior distribution given some evidence $\boldsymbol{E} = \boldsymbol{e}$. The problem is that sampling from a distribution is difficult. We need to build to a process that allows us to do this and this is, for example the *Markov Chain Monte Carlo*.

10.0.2.2.1 Markov Chain Monte Carlo

We usually cannot directly sample from the posterior distribution since this would be too expensive. What we can do instead is to build a random process which gradually samples from distribution closer and closer to the posterior.
The state of the process is an instatiation of all the variables.
The process can be seen as randomly traversing the graph of states moving from one state to another with a certain probability. After enough time, the probability of being in any particular state is the desired posterior.
In this specific case, the random process is a Markov Chain.

10.0.2.3 Sampling Methods – Markov Chain

The state graph is very different from the graphical model of the joint distribution:

- Nodes in the graphical model are variables;
- Nodes in the state graph are instatiations of all the variables.

Definition 10.1: Markov Chain

A Markov Chain consists of:

- A space $Val(\boldsymbol{X})$ of possible states, one for each possible instatiation of all variables $\boldsymbol{X}$;
- A *transition probability model* defining for each state $\boldsymbol{x} \in Val(\boldsymbol{X})$ a *next-state* distribution over $Val(\boldsymbol{X})$:

$$\mathcal{T}(\boldsymbol{x} \to \boldsymbol{x}')$$

A Markov chain *defines a stochastic process that evolves from state to state.*

Definition 10.2: Homogeneous Chains

A Markov chain is said to be homogeneous if its transition probability model does not change over time.

Transition probabilities between states do not depend on the particular time instant in the process: such are the chains we'll use for sampling.
Consider a sequence of states $\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots$ of the random process.
Being random, the state of the process at time t can be seen as a random variable $\boldsymbol{X}^{(t)}$. Assume the initial state $\boldsymbol{X}^{(0)}$ is distributed according to some initial probability $p^{(0)}\left(\boldsymbol{X}^{(0)}\right)$. The distribution over subsequent states can be defined using the chain dynamics as:

$$p^{(t+1)}\left(\boldsymbol{X}^{(t+1)} = \boldsymbol{x}'\right) = \sum_{\boldsymbol{x} \in Val(\boldsymbol{X})} p^{(t)}\left(\boldsymbol{X}^{(t)} = \boldsymbol{x}\right) \mathcal{T}(\boldsymbol{x} \to \boldsymbol{x}')$$

The probability of being in a certain state $\boldsymbol{x}$ at time t is the sum of the probabilities of being in any possible state $\boldsymbol{x}'$ at time $t-1$ times the probability of a transition from $\boldsymbol{x}'$ to $\boldsymbol{x}$.
As the process converges, we expect that $p^{(t+1)}$ becomes close to $p^{(t)}$:

$$p^{(t)}(\boldsymbol{x}') \approx p^{(t+1)}(\boldsymbol{x}') = \sum_{\boldsymbol{x} \in Val(\boldsymbol{X})} p^{(t)}(\boldsymbol{x}) \mathcal{T}(\boldsymbol{x} \to \boldsymbol{x}')$$

At convergence, we expect to have reached an equilibrium distribution $\pi(\boldsymbol{X})$: the probability of being in a state is the same as that of transitioning into it from a random predecessor.

Definition 10.3: Stationary Distribution

A distribution $\pi(\boldsymbol{X})$ is said to be a stationary distribution for a Markov chain $\mathcal{T}$ if:

$$\pi(\boldsymbol{X} = \boldsymbol{x}') = \sum_{\boldsymbol{x} \in Val(\boldsymbol{X})} \pi\left(\boldsymbol{X} = \boldsymbol{x}\right) \mathcal{T}\left(\boldsymbol{x} \to \boldsymbol{x}'\right)$$

We are interested only in Markov Chains with a unique stationary distribution which is reachable from any starting distribution $p^{(0)}$. This can be guaranteed via various conditions, for example ergodicity. Notice that for a Markov Chain with a finite state space $Val(\boldsymbol{X})$, **regularity** is a sufficient and necessary condition.

Definition 10.4: Regularity

A Markov chain is regular if there exists a number k such that for all state pairs $\boldsymbol{x}, \boldsymbol{x}' \in Val(\boldsymbol{X})$, the probability of getting from $\boldsymbol{x}$ to $\boldsymbol{x}'$ in exactly k steps is greater than zero.

This can be proven via 2 conditions:

- It is possible to get from any state to any other state using a positive probability path in the state graph;
- For every state $\boldsymbol{x}$ there is a positive probability of transitioning from $\boldsymbol{x}$ to $\boldsymbol{x}$ in one step (self loop).

10.0.2.3.1 Markov Chains for Graphical Models

The typical use is estimating the probability of a certain instatiation of $\boldsymbol{x}$ of the variables $\boldsymbol{X}$ given some evidence $\boldsymbol{E} = \boldsymbol{e}$. This requires sampling from the posterior distribution $P(\boldsymbol{X}|\boldsymbol{E} = \boldsymbol{e})$.
We wish to define a chain for which $P(\boldsymbol{X}|\boldsymbol{E} = \boldsymbol{e})$ is the stationary distribution.
States in the chain should be the subset of all possible instatiations which is consistent with the evidence $\boldsymbol{e}$:

$$\boldsymbol{x} \in Val(\boldsymbol{X}) : \boldsymbol{x}_{\boldsymbol{E}} = \boldsymbol{e}$$

A simple transition model that we can think consists in updating variables in $\widehat{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{E}$ one at a time. Such model can be seen as made of a set of $k = |\widehat{\boldsymbol{X}}|$ local transition model $\mathcal{T}_i$, one for each variable $X_i \in \widehat{\boldsymbol{X}}$. Let $\boldsymbol{U}_i = \boldsymbol{X} - X_i$ and let $\boldsymbol{u}_i$ be a possible instatiation of $\boldsymbol{U}_i$; then the local transition model $\mathcal{T}_i$ defines transitions between states $(\boldsymbol{u}_i, x_i)$ and $(\boldsymbol{u}_i, x_i')$:

$$\mathcal{T}_i : (\boldsymbol{u}_i, x_i) \mapsto (\boldsymbol{u}_i, x_i')$$

By defining a transition as a sequence of k local transitions, one for each variable X_i, we obtain a homogeneous, i.e., not time-dependent, global transition model.

10.0.2.3.2 Gibbs Sampling

Gibbs sampling is an efficient and effective simple Markov chain for factored state spaces (like the ones in graphical models).
Local transitions $\mathcal{T}_i$ are defined "forgetting" about the value of X_i in the current state: this allows to sample a new value according to the posterior probability of X_i given the rest of the current state:

$$\mathcal{T}_i\left((\boldsymbol{u}_i, x_i) \mapsto (\boldsymbol{u}_i, x_i')\right) = P(x_i'|\boldsymbol{u}_i)$$

The Gibbs chain samples a new state performing k subsequent local moves and we only take samples after a full round of local moves.
Gibbs chains are ensured to be regular if the distribution is strictly positive: every value of X_i given any instatiation of $\boldsymbol{U}_i$ has non-zero probability. If this is the case, then we can get from any state to any state in at most $k = |\widehat{\boldsymbol{X}}|$ local steps.
Gibbs chains have the desired posterior $P(\boldsymbol{X}|\boldsymbol{E} = \boldsymbol{e})$ as stationary distribution.
To generate samples from the correct posterior distribution, we need to wait until the Markov chain has converged to the stationary distribution. Once at convergence, all subsequent samples could be used to estimate the desired probability.
Mind though that consecutive samples are definitely not independent: a possible approach consists of letting an interval of d samples before collecting the next sample.

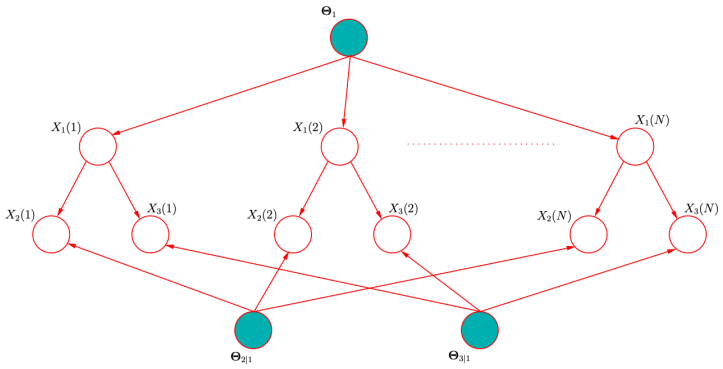
10.1 Learning in Graphical Models

We'll see how to apply what we have seen for parameters learning to Bayesian networks.
We are going to assume that the structure of the model is given and that we are also given a dataset of examples $\mathcal{D} = \{\boldsymbol{x}(1), \ldots, \boldsymbol{x}(N)\}$. Each example $\boldsymbol{x}(i)$ is a configuration for all *complete data* or some *incomplete data* variables in the model.
As said, our goal is to estimate the parameters of the model, in form of conditional probability distributions (CPD), from the data. The simplest approach is to learn the parameters by maximizing the *likelihood* of the data:

$$\boldsymbol{\theta}^{\max} = \text{argmax}_{\boldsymbol{\theta}} P(\boldsymbol{D}|\boldsymbol{\theta}) = \text{argmax}_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}, \boldsymbol{\theta})$$

Bayesian Network can be used not only to represent configuration of variables, but also to reason about what we want to learn. Suppose we have a simple graph with tail to tail connections and

a dataset of N examples, which can be specified by representing the network N times, where each copy refers to the network for a certain example:



The parameters $\boldsymbol{\theta}$ of the network are the ones in blue, and they are connected to the nodes that depend on the parameters. Now we can compute the likelihood as follows since the examples are independent given $\boldsymbol{\theta}$:

$$P(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{N} P(\boldsymbol{x}(i)|\boldsymbol{\theta})$$

Then if we consider the factorization of the Bayesian Network we have

$$P(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{N}\prod_{j=1}^{m} P(\boldsymbol{x}_j(i)|\mathrm{pa}_j(i), \boldsymbol{\theta})$$

As it's possible to see from the previous graph, if, for instance, we'd want to compute the distribution of $\boldsymbol{\theta}_{3|1}$, there is no reason for which we should also keep in consideration $x_2(2)$ since they are not connected:

$$P(\mathcal{D}\boldsymbol{\theta}) = \prod_{i=1}^{N}\prod_{j=1}^{m} P(\boldsymbol{x}_j(i)|\mathrm{pa}_j(i), \boldsymbol{\theta}_{X_j|\mathrm{pa}_j}$$

Giving the disjoint conditional probability distribution.
The parameters of CPD can be estimated independently, the estimate of the parameter X_j by the maximum likelihood method is its maximum value:

$$\boldsymbol{\theta}^{\max}_{X_j|\mathrm{Pa}_j} = \mathrm{argmax}_{\boldsymbol{\theta}_{X_j|\mathrm{Pa}_j}} \prod_{i=1}^{N} P(\boldsymbol{x}_j(i)|\mathrm{Pa}_j(i), \boldsymbol{\theta}_{X_j|\mathrm{Pa}_j})$$

The second part is nothing but the likelihood of the parameters and the dataset:

$$\boldsymbol{\theta}^{\max}_{X_j|\mathrm{Pa}_j} = \mathrm{argmax}_{\boldsymbol{\theta}_{X_j|\mathrm{Pa}_j}} \prod_{i=1}^{N} P(\boldsymbol{x}_j(i)|\mathrm{Pa}_j(i), \boldsymbol{\theta}_{X_j|\mathrm{Pa}_j}) = \mathcal{L}(\boldsymbol{\theta}_{X_j|\mathrm{Pa}_j}, \mathcal{D})$$

A discrete CPD $P(X|\boldsymbol{U})$ can be represented as a table with:

- A number of rows equal to the number $\mathrm{Val}\,(X)$ of configurations for X, that is as many rows as the number of states of X;
- A number of columns equal to the number $\mathrm{Val}\,(\boldsymbol{U})$ of configurations for its parents $\boldsymbol{U}$, that is as many columns as the number of possible parents times the number of the values that they can get;

- Each table entry $\theta_{x|\boldsymbol{u}}$ indicating the probability of a specific configuration $X = x$ and its parents $\boldsymbol{U} = \boldsymbol{u}$

Let's say we have the following table:

$X_2 \mid X_1$	Y	N
R	$\theta^{R\mid Y}$	$\theta^{R\mid N}$
G	$\theta^{G\mid Y}$	$\theta^{G\mid N}$
B	$\theta^{B\mid Y}$	$\theta^{B\mid N}$

Each cell of the table indicates the probability of a specific configuration, i.e., $P(X = x|\boldsymbol{U} = \boldsymbol{u})$, and the value $\theta^{X_2=x_2|X_1=x_1}$ inside the cell is the probability of finding such configuration in the dataset. Each column of the table containes a multinomial distribution of the value of X for a certain configuration. Mind then that the sum of each value in column must return 1: $\sum_X \theta^{X|\boldsymbol{U}} = 1$.
If we were to replace $P(X(i)|\mathrm{Pa}(i))$ with $\theta_{x(i)|\boldsymbol{u}(i)}$ the local likelihood of a single conditional probability distribution becomes:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}_{X|\mathrm{Pa}}, \mathcal{D}) &= \prod_{i=1}^{N} P(x(i)|\mathrm{Pa}(i), \boldsymbol{\theta}_{X|\mathrm{Pa}_j}) \\ &= \prod_{i=1}^{N} \theta_{x(i)|\boldsymbol{u}(i)} \\ &= \prod_{\boldsymbol{u}\in\mathrm{Val}(\boldsymbol{U})} \left[\prod_{x\in\mathrm{Val}(X)} \theta_{x|\boldsymbol{u}}^{N_{\boldsymbol{u},x}} \right]\end{aligned}$$

Where $N_{\boldsymbol{u},x}$ is the number of times the specific configuration $X = x, \boldsymbol{U} = \boldsymbol{u}$ was found in the data. Parents of different columns can be estimated independently.
For each multinomial distribution, zeroing the gradient of the maximum likelihood and considering the normalization constraint, we obtain:

$$\theta_{x|\boldsymbol{u}}^{\max} = \frac{\boldsymbol{N}_{\boldsymbol{u},x}}{\sum_X \boldsymbol{N}_{\boldsymbol{u},x}}$$

The maximum likelihood parameters are simply the fraction of times in which the specific configuration was observed in the data.

10.1.1 Adding Prior

Machine learning estimations tend to overfit the training set, since all the configurations not appearing in the training set will receive zero probability.
A common approach consists of combining machine learning with a prior probability on the parameters, achieving a maximum-a-posteriori estimate:

$$\boldsymbol{\theta}^{\max} = \mathrm{argmax}_{\boldsymbol{\theta}} P(\mathcal{D}|\boldsymbol{\theta}) P(\boldsymbol{\theta})$$

10.1.1.1 Dirichlet Priors

The conjugate prior for a multinomial distribution is a Dirichlet distribution with parameters $\alpha_{x|\boldsymbol{u}}$ for each possible value of x.

The resulting maximum-a-posteriori estimate is:

$$\theta_{x|\boldsymbol{u}} = \frac{\boldsymbol{N}_{\boldsymbol{u},x} + \alpha_{x|\boldsymbol{u}}}{\sum_x \left(\boldsymbol{N}_{\boldsymbol{u},x} + \alpha_{x|\boldsymbol{u}}\right)}$$

The prior is like having observed $\alpha_{x|\boldsymbol{u}}$ imaginary samples with configuration $\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{U} = \boldsymbol{u}$.

10.1.2 Incomplete data

With incomplete data, some of the examples miss evidence on some of the variables. Counting the occurrences different configurations cannot be computed if not all data are observed.
The full Bayesian approach of integrating over missing variables is often intractable in practice, hence we need approximate methods to deal with the problem.
Usually the missing data are filled in with inference using known parameters: one computes the parameters with the maximum likelihood criteria (or with maximum-a-posteriori) and then the quality of the parameters can be improved by iterating and reaching convergence. An example is the Expectation-Maximization Algorithm.

10.1.3 Expectation-Maximization Algorithm

This algorithm is composed of two steps:

- **E-step**: the expectation step. It consists in computing the expected sufficient statistics for the complete dataset, with expectation taken w.r.t. the joint distribution for $\boldsymbol{X}$ conditioned of the current value of $\boldsymbol{\theta}$ and the known data $\mathcal{D}$:

  $$\mathrm{E}_{P(\boldsymbol{X}|\mathcal{D},\boldsymbol{\theta})}\left[\boldsymbol{N}_{ijk}\right] = \sum_{l=1}^{n} P(X_i(l) = x_k, \mathrm{Pa}_i(l) = \mathrm{pa}_j | \boldsymbol{X}_l, \boldsymbol{\theta})$$

  Where $\boldsymbol{N}_{ijk}$ is the entry in the table of node i, row j and column k.
  If $X(l)$ and $\mathrm{Pa}_i(l)$ are observed for $\boldsymbol{X}_l$, then the expected value is either zero or one. Otherwise we need to run Bayesian inference to compute probabilities from observed variables.

- **M-step**: the maximization step. It consists in computing the parameters maximizing likelihood of the complete dataset $\mathcal{D}_c$ using expected counts:

  $$\boldsymbol{\theta}^* = \mathrm{argmax}_{\boldsymbol{\theta}} P(\mathcal{D}_c | \boldsymbol{\theta})$$

  which for each multinomial parameter evaluates to:

  $$\theta^*_{ijk} = \frac{\mathrm{E}_{P(\boldsymbol{X}|\mathcal{D},\boldsymbol{\theta})}\left[\boldsymbol{N}_{ijk}\right]}{\sum_{k=1}^{r_i} \mathrm{E}_{P(\boldsymbol{X}|\mathcal{D},\boldsymbol{\theta}}\left[\boldsymbol{N}_{ijk}\right]}$$

  Notice that maximum likelihood estimation can be replaced by maximum-a-posteriori estimation giving:

  $$\theta^*_{ijk} = \frac{\alpha_{ijk} + \mathrm{E}_{P(\boldsymbol{X}|\mathcal{D},\boldsymbol{\theta},S}\left[\boldsymbol{N}_{ijk}\right]}{\sum_{k=1}^{r_i} \mathrm{E}_{P(\boldsymbol{X}|\mathcal{D},\boldsymbol{\theta}}\left[\boldsymbol{N}_{ijk}\right]}$$

10.2 Naïve Bayes Classifier

Each instance x is described by a conjunction of attribute values $< a_1, \ldots, a_m >$. The target function can take any value from a finite set $\mathcal{Y}$.
The task is predicting the maximum-a-posteriori target value given the instance:

$$\begin{aligned} \boldsymbol{y}^* = \text{argmax}_{y_i \in \mathcal{Y}} P(y_i|x) &= \text{argmax}_{y_i \in \mathcal{Y}} \frac{P(a_1, \ldots, a_m|y_i)P(y_i)}{P(a_1, \ldots, a_m)} \\ &= \text{argmax}_{y_i \in \mathcal{Y}} P(a_1, \ldots, a_m|y_i)P(y_i) \end{aligned}$$

Class conditional probabilities $P(a_1, \ldots, a_m|y_i)$ are hard to learn, as the number of terms is equal to the number of possible instances times the number of target values. We can then simplify the assumptions by assuming that the attribute values are independent of each other given the target value:

$$P(a_1, \ldots, a_m|y_i) = \prod_{j=1}^{m} P(a_j|y_i)$$

So a good estimator is:

$$y^* = \text{argmax}_{y_i \in \mathcal{Y}} \prod_{j=1}^{m} P(a_j|y_i)P(y_i) \tag{10.1}$$

10.2.1 Single Distribution Case

Assume that all attribute values come from the same distribution. The probability of an attribute value given the class can be modeled as a multinomial distribution over the K possible values:

$$P(a_j|y_i = \prod_{k=1}^{K} \theta_{ky_i}^{z_k(a_j)}$$

10.2.2 Parameters Learning

Target priors $P(y_i)$ can be learned as the fraction of training set instances having each target value.
The maximum-likelihood estimate for the parameter θ_{kc}, that is the probability of value v_k given class c) is the fraction of times the value was observed in training examples of class c:

$$\theta_{kc} = \frac{N_{kc}}{N_c}$$

Assume a Dirichlet prior distribution (with parameters $\alpha_{1c}, \ldots, \alpha_{Kc}$) for attribute parameters. The posterior distribution for attribute parameters is again a multinomial:

$$\theta_{kc} = \frac{\boldsymbol{N}_{kc} + \alpha_{kc}}{\boldsymbol{N}_c + 1\alpha_c}$$

10.2.3 Example – Text Classification

The task is to classify documents in one of C possible classes. Each document is represented as the bag-of-words it contains, the position of them is not important. Let V be the vocabulary of all possible words and a dataset of labeled documents $\mathcal{D}$ is available.
We shall first use Naive Bayes to learn: we'll compute the prior probabilities of classes as:

$$P(y_i) = \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$$

Then we'll model the attributes with a multinomial distribution with $K = |V|$ possible states (words). At last we'll compute the probability of work w_k given class c as the fraction of times the word appear in documents of class y_i, wr.t. all words in documents of class c:

$$\theta_{kc} = \frac{\sum_{\boldsymbol{X} \in \mathcal{D}_c} \sum_{j=1}^{|\boldsymbol{X}|} z_k(x[j])}{\sum_{\boldsymbol{x} \in \mathcal{D}_c} |\boldsymbol{X}|}$$

Then the classification is done as follows:

$$\begin{aligned} \boldsymbol{y}^* &= \text{argmax}_{y_i \in \mathcal{Y}} \prod_{j=1}^{|\boldsymbol{X}|} P(x[j]|y_i) P(y_i) \\ &= \text{argmax}_{y_i \in \mathcal{Y}} \prod_{j=1}^{|\boldsymbol{X}|} \theta_{ky_i}^{z_k(x[j])} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \end{aligned}$$

10.2.4 Simplification

Note that we are making the simplifying assumption that all attributer values come from the same distribution. If this were not to be the case, then attributes from different distributions have to be considered separately for parameter estimation.
For example assume that each instance x is represented as a vector of l attributes. Assume that the j^{th} attribute ($j \in [1, l]$) can take $\{v_{j1}, \ldots, v_{jK_j}\}$ possible values. The parameter θ_{jkc} representing the probability of observing value v_{jk} for the j^{th} attribute given class c is estimated as:

$$\theta_{jkc} = \frac{\sum_{\boldsymbol{X} \in \boldsymbol{\mathcal{D}_c}} z_{jk}(x[j])}{|\mathcal{D}_c|}$$

10.2.5 Naïve Bayes Recap

It's possible to describe this classifier as a Bayesian Network where the class to be predicted is dependent on all the other:

$$\boldsymbol{y}^* = \text{argmax}_{y_i \in \mathcal{Y}} P(y_i|x) = \text{argmax}_{y_i \in \mathcal{Y}} \frac{P(\boldsymbol{X}|Y)P(Y)}{P(\boldsymbol{X})}$$

where:

- $P(Y) = \frac{|Y_n|}{|Y}$ computes the number of elements in a class;
- $P(\boldsymbol{X}|Y)$ its a likelihood measure using the k nearest samples in the training set.
- $P(\boldsymbol{X}, Y)$ it's the a posteriori probability to classify the sample.

The estimated output is the one that gives the bigger posteriori probability:

$$\boldsymbol{y}^* = \text{argmax}_{y_i \in \mathcal{Y}} P(\boldsymbol{X}|Y)P(Y) = \text{argmax}_{y_i \in \mathcal{Y}} \prod_{j=1}^{m} P(a_j|y_i)P(y_i)$$

11 06/11/2019

12 Evaluation

12.1 Introduction

Evaluation requires to define one or more measure which have to be optimized. For such reason, they need to be quantitative.
In principle, when training a model we'd like to know its performance on the entire domain where it's going to be applied, but this is not possible due to the generalization error[1], hence an approximation is needed.
Performance evaluation is needed for:

- Tuning hyperparameters[2] of learning method;
- Evaluating quality of learned predictor;
- Computing statistical significance of difference between learning algorithms.

Notice that we cannot evaluate on the training set since, if were to do so, we would end up with overfitting.
Mind that hyperparameters can be machine learned by trying different possibilities and automatically choosing the best one.
When we talked about parameters estimation, we talked about, e.g., maximum likelihood and maximum a posteriori. This functions allows to learn good values for the weights. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called *empirical risk minimization*. **Loss** is the penalty for a bad prediction, that is, loss is a number indicating how bad the model's prediction was on a single example: if the model's prediction is perfect, then loss is zero, otherwise is grater. The goal of training a model is to find a set of weights that have low loss, on average, across all examples.
In general we can say that the training loss function measures the cost paid for predicting $f(\boldsymbol{x})$ for output y.
Notice that training loss and performance measure are two different things, the first one is used in training, the second is used to evaluate the performance of the model.
Multiple performance measures could be used to evaluate different aspects of a learner.

12.2 Binary classification – Confusion Matrix

In order to evaluate binary classification, one could use a confusion matrix, that is a matrix that reports the true labels (on rows) and the predicted labels (on columns). Each cell tells how

[1] Since learning algorithms are evaluated on finite samples, the evaluation of a learning algorithm may be sensitive to sampling error, i.e., measurements of prediction error on the current data may not provide much information about predictive ability on new data.

[2] All those parameters that are not learned, but are set at the beginning by the programmer, e.g. the type of kernel and parameters or the learning rate of a perceptron.

many value have been predicted correctly, and if not what was the mistake.
Each can belong to one of the following cases:

- True positives: positives that were predicted positive;
- True negatives: negatives that were predicted negative;
- False positives: negatives that were predicted positive;
- False negatives: positives that were predicted negative.

True \ Pred	Positive	Negative
Positive	True Positive	False Negative
Negative	False Positive	True Negative

This matrix is useful because ti gives a picture of what the predictor does. For example consider the following table:

True \ Pred	Pos	Neg
Pos	0	10
Neg	0	90

It's possible to see that the predictor is biased in not predicting any positive: 1 out of 10 samples where predicted false but actually were positive, hence they were false negatives.
Over the confusion matrix we can compute the following metrics:

- **Accuracy**: it says how many samples were predicted correctly.
- **Precision**: it says how accurate the model is when predicting positive;
- **Recall** or **sensitivity**: how many positive example we got with respective to the true positive.
- **Aggregate measures**: for instance, *F-Measure* that combines precision and recall.

12.2.1 Accuracy

Accuracy is the fraction of correctly labelled examples among all predictors.

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

It's a scoring function, so the bigger the value the better. Notice that the accuracy is actually one minus the misclassification cost.
Two bigger problems are bound to the accuracy:

- It cannot be used for training;
- In strongly unbalanced datasets (typically when negatives are much more than positives) it is not informative: predictions are dominated by the larger class. Considering the example before, the accuracy was 0.9, but looking at all the table it's possible to see how it learned only to predict negatives. It's possible to rebalance the costs, e.g. a single positive counts as $^N/_P$ where $N = TN + FP$ and $P = TP + FN$.

This cannot be used for training, its also problematic as an evaluation metric as some datasets are too unbalanced as the table before. The accuracy of the previous was 0.9, but this has learned only to predict negatives. If we hadn't looked at the confusion matrix we wouldn't have noticed.

12.2.2 Precision

Precision is the fraction of positives among examples predicted as positives, so it measures the precision of a learner when predicting positive.

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

12.2.3 Recall

It is the fraction of positive examples predicted as positives, so it measures the coverage of the learner in returning positive examples.

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

the complement of precision: This two measures are contradictory: if we want to be really precise we'll predict positive less frequently and so recall will be lower.

12.2.4 Aggregate Metrics – F-Measure

Precision and recall are complementary, so increasing the precision typically reduces the recall. F-measure combines the two measures balancing the two aspects:

$$F_\beta = \frac{(1 + \beta^2)(Pre \times Rec)}{\beta^2 Pre + Rec}$$

β is the parameter that trades-off precision and recall.

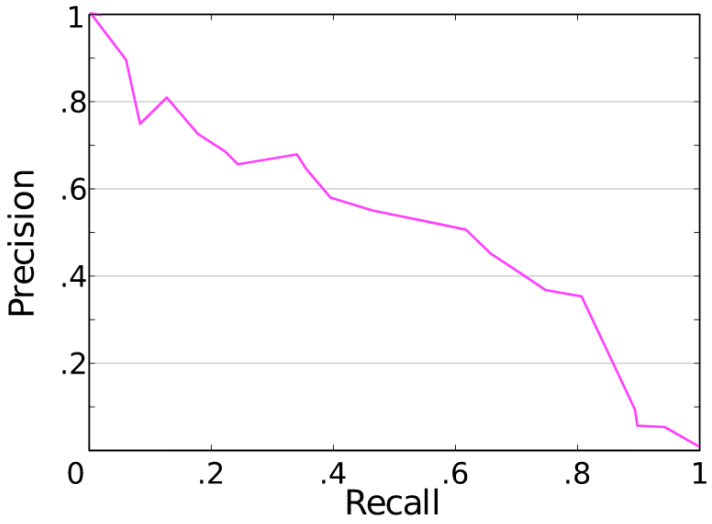
12.2.4.1 F_1

If $\beta = 1$, we have F_1-measure. This is actually the harmonic mean of precision and recall:

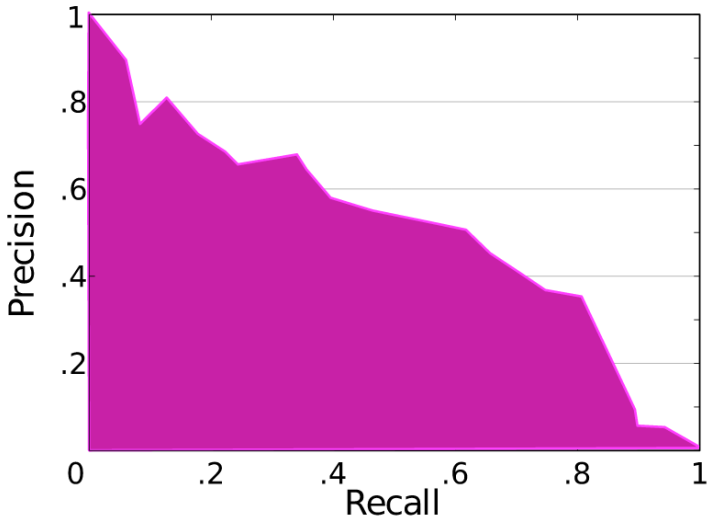
$$F_1 = \frac{2(Pre \times Rec)}{Pre + Rec}$$

12.2.5 Precision-Recall Curve

Classifiers often provide a confidence in the prediction. A hard decision is made setting a threshold on the classifier. When we compute one of the above measures, we are actually computing it for a specific threshold. It is possible to change the threshold if interested in maximizing a specific performance, e.g. recall.
We can do so by plotting a graph representing the values for each threshold:



It is possible to investigate the performance of the learner in different scenarios.
A single aggregate value can be obtained by taking the area under the curve. A high value means both high precision and recall, while a lower value doesn't give information regarding each one of the measures.



Notice that when recall is 0 also precision is 0, and vice versa, so we have a discontinuity.

12.3 Multiclass Classification

Binary matrix can be extent to multi-class.

True \ Pred	y_1	$\dots$	y_M
y_1	n_{11}	$\dots$	n_{1M}
	$\vdots$	$\vdots$	$\vdots$
y_M	n_{M1}	$\dots$	n_{MM}

n_{ij} is the number of examples with class y_i predicted as y_j.
The main diagonal contains the true positives for each class, while false positives FP_i are given

by the sum of off-diagonal elements along a column and false negatives FN_i are given by the sum of off-diagonal elements along a row:

$$FP_i = \sum_{j \neq i} n_{ji} \qquad FN_i = \sum_{j \neq i} n_{ij}$$

Even the metrics are computed per class, for example:

$$Pre_i = \frac{n_{ij}}{n_{ij} + FP_i} \qquad Rec_i = \frac{n_{ij}}{n_{ij} + FN_i}$$

There is then a global metric that's called *multiclass accuracy* that is the sum of true positives divided by the sum of all elements in the matrix:

$$MAcc = \frac{\sum_i n_{ij}}{\sum_i \sum_j n_{ij}}$$

12.4 Regression Measures

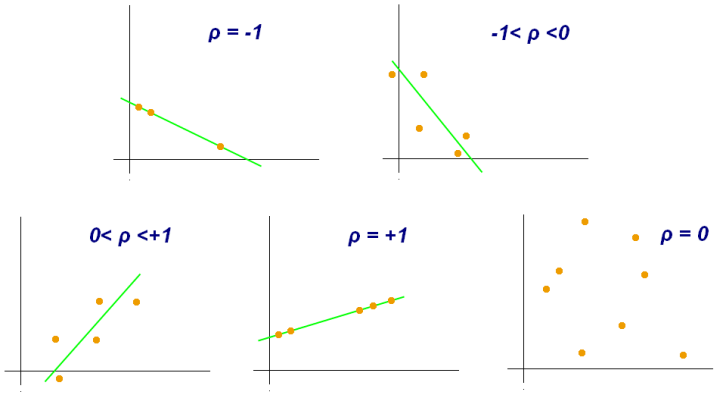
12.4.1 Root Mean Squared Error

The root mean squared error ($RMSE$) it's a measure frequently used to comput the differences between the values predicted by the model and the observed values. Given a dataset $\mathcal{D}$ with $n = |\mathcal{D}|$ we have:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (f(\boldsymbol{x}_i) - y_i)^2}$$

12.4.2 Pearson Correlation Coefficient

The Pearson correlation coefficient ρ is a statistic that measures linear correlation between two variables X and Y. It takes values in $[-1, 1]$ where 1 is total positive linear correlation, 0 is no linear correlation and -1 is total negative linear correlation.



$$\rho = \frac{\mathrm{Cov}[X, Y]}{\sigma_X \sigma_Y} = \frac{\mathrm{E}\left[(X - \overline{X})(Y - \overline{Y})\right]}{\sqrt{\mathrm{E}\left[(X - \overline{X})^2\right] \mathrm{E}\left[(Y - \overline{Y})^2\right]}}$$

If instead of two random variables we are given a dataset $\mathcal{D}$, then the formula becomes:

$$\rho = \frac{\sum_{i=1}^{n} \left(f(\boldsymbol{x}_i) - \overline{f}(\boldsymbol{x}_i)\right)(y_i - \overline{y}_i)}{\sqrt{\sum_{i=1}^{n} \left(f(\boldsymbol{x}_i) - \overline{f}(\boldsymbol{x}_i)\right)^2 \sum_{i=1}^{n} (y_i - \overline{y}_i)^2}}$$

12.5 Hold-out

Since computing performance measure on the training set would be optimistically biased, we need to retain an independent set on which to compute performance. Moreover we'd like to have:

- **Validation set**: when used to estimate performance of different algorithm settings, for instance when tuning the hyperparameters;
- **Test set**: when used to estimate the final performance of selected model.

One simple way to obtain this datasets is to divide the initial dataset into the three, usually with percentage as $40\%, 30\%, 30\%$ for training, validation and testing. One problem with this procedure though is that the results depend on the specific test (and validation) set chosen, especially for small datasets.

12.6 K-Fold Cross Validation

The hold-out method is used for big datasets, where we have a large amount of data and is less probable to obtain biased datasets.
When such method is unusable, one could resolve to k-fold cross validation. The procedure is the following:

- Split $\mathcal{D}$ into k *equal sized disjoint* subsets $\mathcal{D}_i$;
- For $i \in [1, k]$
  - Train the predictor on $\mathcal{T}_i = \frac{\mathcal{D}}{\mathcal{D}_i}$
  - Compute score S on predictor $L(\mathcal{T}_i)$ on test set $\mathcal{D}_i$

$$S_i = S_{\mathcal{D}_i}[L(\mathcal{T}_i)]$$

- Return the average across folds:

$$\overline{S} = \frac{1}{k} \sum_{i=1}^{k} S_i$$

Basically is like saying that we do not trust the value that a hold-out procedure would return, but instead we prefer to have some kind of average.
The idea is to split the dataset $\mathcal{D}$ into k subsets, then to iterate through the various subsets keeping one for validation and the rest of the sub-datasets for training. Each time we'll compute

the score and at the end average over all the scores.
A part from the mean $\overline{S}$, one would like to compute also the variance of the mean:

$$\text{Var}[\overline{S}] = \text{Var}[\frac{S_1 + \ldots + S_k}{k}] = \frac{1}{k^2}\sum_{j=1}^{k}\text{Var}[S_j]$$

The problem though is that we could compute $\text{Var}[S_j]$ exactly only if all the training sets were independent, but this is not true since they have samples in common, so we have to approximate it with the unbiased variance across folds:

$$\text{Var}[S_j] = \text{Var}[S_h] \approx \frac{1}{k-1}\sum_{i=1}^{k}\left(S_i - \overline{S}\right)^2$$

Giving:

$$\text{Var}[\overline{S}] \approx \frac{1}{k^2}\sum_{j=1}^{k}\left(\frac{1}{k-1}\sum_{i=1}^{k}\left(S_i - \overline{S}\right)^2\right) = \frac{1}{k-1}\sum_{i=1}^{k}\left(S_i - \overline{S}\right)^2$$

12.7 Comparing Different Learning Algorithms

Let's say we have two learning algorithms and we want to compare them. One possible way would just be to do cross validation and see which have the best statistics. The problem is that we want to be sure that the differences we see are *statistically significant* and not due to some noisy evaluation.
The most precise way to choose between two algorithms is to use **hypothesis testing** which allows to test the statistical significance of a hypothesis.
In hypothesis testing we have a statement, called **null hypothesis** H_0, of which we want to prove the truthfulness, especially we want to provide *prof* to reject the hypothesis. This is done with *test statistic*: given a sample of k realizations of random variables $X_1, \ldots, X_k$, a test statistic is a statistic $T = h(X_1, \ldots, X_k)$ whose value is used to decide whether to reject H_0 or not. Mind that not rejecting the hypothesis does not prove it to be true.

Definition 12.1: Tail Probability

The probability that T is at least as great (right tail) or at least as small (left tail) as the observed value t.

Definition 12.2: p-Value

The probability of obtaining a value T at least as extreme as the one observed t, in case H_0 is true.

The p-value is also the value that allows to decide whether to accept or reject the null hypothesis. Given a threshold α, if the p-value p is higher than it, then the hypothesis may be accepted, otherwise it is rejected.

Definition 12.3: Critical Region & Critical Values

The critical region is a set of values of T for which we reject the null hypothesis. The critical values are the values on the boundary of the critical region.

We can define two possible errors:

- Type I error: when we reject the null hypothesis when it's actually true;
- Type II error: when we accept the null hypothesis when it's actually false.

The **significance level** is the largest acceptable probability for committing a type I error.

12.7.1 T-Test

The test statistic is given by the standardized (also called *studentized*) mean:

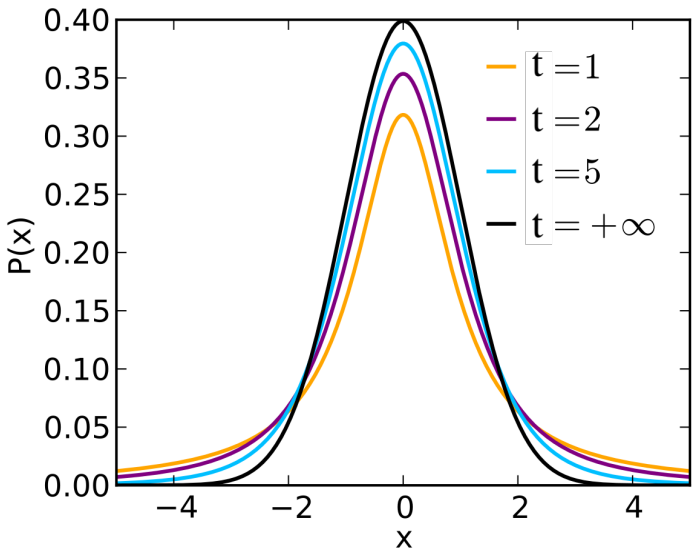
$$T = \frac{\overline{X} - \mu_0}{\sqrt{\tilde{\text{Var}}[\overline{X}]}}$$

where $\tilde{\text{Var}}[\overline{X}]$ is the approximate variance and μ_0 is the mean of the null distribution.
Assuming all the samples come from an unknown Normal distribution, the test statistics has is a t-distribution which is a k-parametrized distribution with $k-1$ degree of freedom and with k as the number of folds. The statistics is said to have a t_{k-1} distribution under the null hypothesis. Given a significance level α, the null hypothesis can be rejected if:

$$T \leq -t_{k-1,\frac{\alpha}{2}}, \qquad T \geq t_{k-1,\frac{\alpha}{2}}$$

12.7.1.1 t_{k-1}-Distribution

It's a bell-shaped distribution similar to the Normal one, though, it's wider and shorter: it reflects greater variance due to using $\tilde{\text{Var}}[\overline{X}]$ instead of the true unknown variance of the distribution. t_{k-1} tends to the standardized normal z for $k \to \infty$, that is, the more number of folds we have, the more the t-distribution becomes similar to the Normal distribution.



12.7.2 How To

Let's consider the case in which we have 2 algorithms A and B and we want to know which one works better.
First of all, we shall run k-fold cross validation for algorithms A and B. Then we'll compute the mean performance difference for the two algorithms:

$$\widehat{\delta} = \frac{1}{k}\sum_{i=1}^{k}\delta_i = \frac{1}{k}\sum_{i=1}^{k} S_{\mathcal{D}_i}[L_A(\mathcal{T}_i)] - S_{\mathcal{D}_i}[L_B(\mathcal{T}_i)]$$

where $S_{\mathcal{D}_)}[]$ is the score function chosen.
Let's say that the null hypothesis is that the mean difference is 0.
Let:

$$\sqrt{\tilde{\text{Var}}[\overline{\delta}]} = \sqrt{\frac{1}{k(k-1)}\sum_{i=1}^{k})\delta_i - \overline{\delta})^2}$$

Given a significance level α, i.e., the probability of resulting a type I error, we accept the null hypothesis if:

$$\frac{\overline{\delta}}{\sqrt{\tilde{\text{Var}}[\overline{\delta}]}} \leq -t_{k-1,\frac{\alpha}{2}} \quad or \quad \frac{\overline{\delta}}{\sqrt{\tilde{\text{Var}}[\overline{\delta}]}} \geq t_{k-1,\frac{\alpha}{2}}$$

Basically if the probability of type 1 error is on the tails of the distribution
Note that the smaller α, e.g. 5%, the smaller the probability of having an error.
The T-test is called **pair test** because when we compute the two sample estimate, they come from the same fold, so we could compare the statistic on a certain fold. Moreover, it's also called **two-tailed test** because we cannot exclude at prior neither of the outcomes, i.e., if A is better than B or vice versa. Formally, no prior knowledge can tell the direction of the difference. If we knew that for B is the worse case, hence it's not possible to do worse than that, then it would be a **one-tailed** test.

12.7.3 T-Test Example

We have two algorithms A and B and we want to know if they perform the same. Moreover, we have a dataset $\mathcal{D}$ and we are going to use k-fold cross validation with $k = 10$. Let's say that the scores we obtain are:

$\mathcal{D}_i$	$S_{\mathcal{D}_i}[L_A(\mathcal{T}_i)]$	$S_{\mathcal{D}_i}[L_A(\mathcal{T}_i)]$
$\mathcal{D}_1$	0.81	0.80
$\mathcal{D}_2$	0.82	0.77
$\mathcal{D}_3$	0.84	0.70
$\mathcal{D}_4$	0.78	0.83
$\mathcal{D}_5$	0.85	0.80
$\mathcal{D}_6$	0.86	0.78
$\mathcal{D}_7$	0.82	0.75
$\mathcal{D}_8$	0.83	0.80
$\mathcal{D}_9$	0.82	0.78
$\mathcal{D}_10$	0.81	0.77

For each of them we can compute the difference between the scores of each fold δ:

$$\widehat{\delta} = \frac{1}{k}\sum_{i=1}^{k}\delta_i = \frac{1}{k}\sum_{i=1}^{k} S_{\mathcal{D}_i}[L_A(\mathcal{T}_i)] - S_{\mathcal{D}_i}[L_B(\mathcal{T}_i)]$$

$\mathcal{D}_i$	$S_{\mathcal{D}_i}[L_A(\mathcal{T}_i)]$	$S_{\mathcal{D}_i}[L_A(\mathcal{T}_i)]$	δ_i
$\mathcal{D}_1$	0.81	0.80	0.01
$\mathcal{D}_2$	0.82	0.77	0.05
$\mathcal{D}_3$	0.84	0.70	0.14
$\mathcal{D}_4$	0.78	0.83	-0.05
$\mathcal{D}_5$	0.85	0.80	0.05
$\mathcal{D}_6$	0.86	0.78	0.08
$\mathcal{D}_7$	0.82	0.75	0.07
$\mathcal{D}_8$	0.83	0.80	0.03
$\mathcal{D}_9$	0.82	0.78	0.04
$\mathcal{D}_10$	0.81	0.77	0.04

We the compute the average error difference:

$$\overline{\delta} = \frac{1}{10}\sum_{i=1}^{10}\delta_i = 0.046$$

We shall compute the unbiased estimate of the standard deviation:

$$\sqrt{\tilde{\text{Var}}[\overline{\delta}]} = \sqrt{\frac{1}{10\times 9}\sum_{i=1}^{10}\left(\delta_i - \overline{\delta}\right)^2} = 0.0154344$$

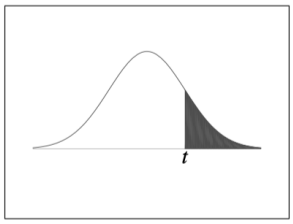
The standardized mean error difference is:

$$\frac{\overline{\delta}}{\sqrt{\tilde{\text{Var}}[\overline{\delta}]}} = \frac{0.046}{0.0154344} = 2.98$$

Since $\alpha = 5\%$ is a good threshold, then we have:

$$t_{k-1,\frac{\alpha}{2}} = t_{9,0.025} = 2.262 < 2.98$$

Since the value is nor smaller than -2.98, nor greater than 2.98, then the null hypothesis is rejected, the classifiers are actually different.



The shaded area is equal to α for $t = t_\alpha$.

df	$t_{.100}$	$t_{.050}$	$t_{.025}$	$t_{.010}$	$t_{.005}$
1	3.078	6.314	12.706	31.821	63.657
2	1.886	2.920	4.303	6.965	9.925
3	1.638	2.353	3.182	4.541	5.841
4	1.533	2.132	2.776	3.747	4.604
5	1.476	2.015	2.571	3.365	4.032
6	1.440	1.943	2.447	3.143	3.707
7	1.415	1.895	2.365	2.998	3.499
8	1.397	1.860	2.306	2.896	3.355
9	1.383	1.833	2.262	2.821	3.250
10	1.372	1.812	2.228	2.764	3.169
11	1.363	1.796	2.201	2.718	3.106
12	1.356	1.782	2.179	2.681	3.055
13	1.350	1.771	2.160	2.650	3.012
14	1.345	1.761	2.145	2.624	2.977
15	1.341	1.753	2.131	2.602	2.947
16	1.337	1.746	2.120	2.583	2.921
17	1.333	1.740	2.110	2.567	2.898
18	1.330	1.734	2.101	2.552	2.878
19	1.328	1.729	2.093	2.539	2.861
20	1.325	1.725	2.086	2.528	2.845
21	1.323	1.721	2.080	2.518	2.831
22	1.321	1.717	2.074	2.508	2.819
23	1.319	1.714	2.069	2.500	2.807
24	1.318	1.711	2.064	2.492	2.797
25	1.316	1.708	2.060	2.485	2.787
26	1.315	1.706	2.056	2.479	2.779
27	1.314	1.703	2.052	2.473	2.771
28	1.313	1.701	2.048	2.467	2.763
29	1.311	1.699	2.045	2.462	2.756
30	1.310	1.697	2.042	2.457	2.750
32	1.309	1.694	2.037	2.449	2.738
34	1.307	1.691	2.032	2.441	2.728
36	1.306	1.688	2.028	2.434	2.719
38	1.304	1.686	2.024	2.429	2.712
∞	1.282	1.645	1.960	2.326	2.576

13 07/11/2019

14 Discriminative vs generative

We talked a lot about generative model, that is assuming that there is a distribution governing the data. Moreover we doing generative model, we need to find all the possible weights, which can be pretty difficult, e.g., finding the argument of a website considering its links. In this case, instead of using a generative model, we could use a **discriminative model**, that is a model for which we care only about the boundaries.
Discriminative learnings focuses on directly modeling the discriminant function.
This approach is particularly used in classification since it allows to directly model the decision boundaries instead of inferring them from the modelled data distributions.

PROS	CONS
When data are complex, modelling their distribution (generative learning) can be very difficult	The learned model is less flexible in its usage
If data discrimination is the goal, data distribution modeling is not needed	It does not allow to perform arbitrary inference tasks
Focuses parameters on the desired goal	It is not possible to efficiently generate new data from a certain class

14.1 Linear discriminative model

The idea is to model a linear discriminative function by finding the right multiplicative factors, called **weights**, so that such function is a linear composition of the samples in the dataset:

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

W_0 is called the **bias** or **threshold**.
This is the simplest function, which may mean that it is not always the best one, but in most cases is the best option available.

14.1.1 Linear Binary Classifier

If we'd need to classify only over two possibility, then all we'd need is to have a binary classifier. The easiest thing to do is to consider the sign of the before formula:

$$f(\boldsymbol{x}) = \text{sign}(()\boldsymbol{w}^T \boldsymbol{x} + w_0)$$

So if the result is less than 0, we have a class, if it's positive is the other class, while if is 0 that is the **decision boundary**. This is the set of points for which the discriminant function is 0 and is actually and *hyperplane*.
It follows that the weight vector $\boldsymbol{w}$ is orthogonal to the decision hyperplane:

Proof.

$$\forall \boldsymbol{x}, \boldsymbol{x}' : f(\boldsymbol{x}) = f(\boldsymbol{x}') = 0$$
$$\boldsymbol{w}^T \boldsymbol{x} + w_0 - \boldsymbol{w}^T \boldsymbol{x}' - w_0 = 0$$
$$\boldsymbol{w}^T (\boldsymbol{x} - \boldsymbol{x}') = 0$$

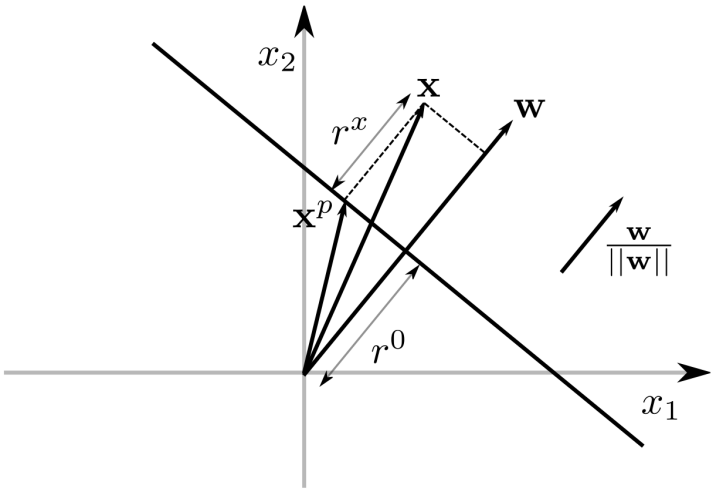
□

The value $f(\boldsymbol{x})$ of the function for a certain point $\boldsymbol{x}$ is called *functional margin* and it can be seen as a confidence in the prediction: if the value is greatly bigger than the boundary, then we can be more confident regarding the decision.
The distance from $\boldsymbol{x}$ to the hyperplane is called *geometric margin*

$$r^x = \frac{f(\boldsymbol{x})}{||\boldsymbol{w}||}$$

It is a normalize version of the functional margin w.r.t. the weights.
The distance from the origin to the hyperplane is:

$$r^0 = \frac{f(\mathbf{0})}{||\boldsymbol{w}||} = \frac{w_0}{||\boldsymbol{w}||}$$



Proof. A point $\boldsymbol{x}$ can be expressed by its projection $\boldsymbol{x}^p$ on the hyperplane H plus its distance to the hyperplane times the unit vector $\boldsymbol{w}/_{||\boldsymbol{w}||}$ in that direction:

$$\boldsymbol{x} = \boldsymbol{x}^p + r^x \frac{\boldsymbol{w}}{||\boldsymbol{w}||}$$

Since the point is on the hyperplane, then we have:

$$f(\boldsymbol{x}^p) = 0$$

Now we know that:

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

By writing *vectx* as the projection we have:

$$\begin{aligned} f(\boldsymbol{x} &= \boldsymbol{w}^T \left(\boldsymbol{x}^p + r^x \tfrac{\boldsymbol{w}}{||\boldsymbol{w}||} \right) + w_0 \\ &= \boldsymbol{w}^T \boldsymbol{x}^p + w_0 + r^x \boldsymbol{w}^T \tfrac{\boldsymbol{w}}{||\boldsymbol{w}||} \end{aligned}$$

But $\boldsymbol{w}^T\boldsymbol{x}^p + w_0 = f(\boldsymbol{x}^p)$ which is 0:

$$f(\boldsymbol{x}) = r^x \boldsymbol{w}^T \frac{\boldsymbol{w}}{||\boldsymbol{w}||} = r^x \boldsymbol{w}^T \frac{\boldsymbol{w}}{||\boldsymbol{w}||}$$

The product of the transpose for the vector is the square of the norm, hence:

$$f(\boldsymbol{x}) = r^x ||\boldsymbol{w}||$$

By inverting we have:

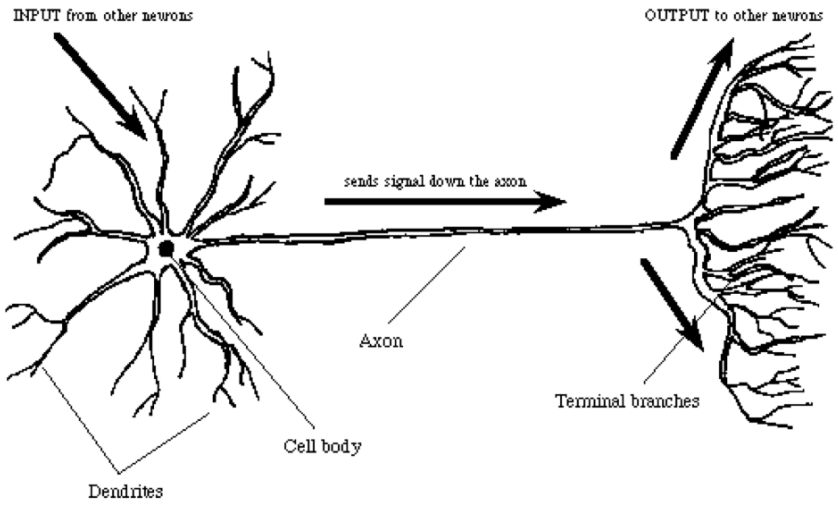
$$r^x = \frac{f(\boldsymbol{x})}{||\boldsymbol{w}||}$$

□

14.2 Perceptron – A Biological Approach

The linear classifier, called **perceptron**, takes inspiration from the brain which is composed of densely interconnected networks of neurons.
Each neuron is made out of:

- *Soma*: a central body containing the nucleus;
- *Dendrites*: they surround the soma and are a set of filaments departing from the body;
- *Axon*: the longest dendrite;
- *Synapses*: connections between dendrites and axons from other neurons.



Electrochemical reactions allows the signals to propagate in the brain via axons, synapses and dendrites.
Each synapse can either *excite* or *inhibit* a neuron potential: once a neuron potential exceeds a certain threshold, a signal is generated and transmitted along the axon.
What it's possible to do is to create a classifier that behaves similarly to how neurons do. This classifier is called **perceptron**.
To compare a perceptron with a neuron we can say that the dendrites are actually the classes present in the inputs, while the synapses are the functions, called **activation functions**, that discriminate between the classes based on a certain threshold.

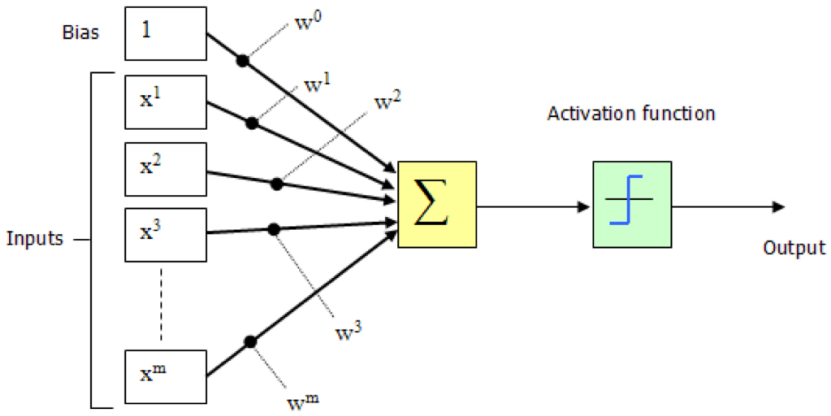


Figure 14.1: The scheme of a perceptron.

Note that a weight is associated to each feature and the results are summed before being sent to the activation function. The simplest formulation for a perceptron is a binary classifier for which the activation function is a sign function:

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T \boldsymbol{x} + w_0) \tag{14.1}$$

In this case the threshold would be 0: if the value is bigger, then we can classify the sample with a class, if the value is smaller, then we can classify the sample with the other class.
A *linear classifier* (perceptron) can *linearly* separate classes that are *linearly separable*, for example boolean functions such as AND, OR, NOT, NAND since in this case two features can be either 1 or 0. For instance, if we consider the OR the result can be either positive or negative and hence easily separable by a line. The same can be said for the AND and the NOT. But if we were to consider the XOR, then dividing by an hyperplane is not possible, hence the XOR is not linear separable function.

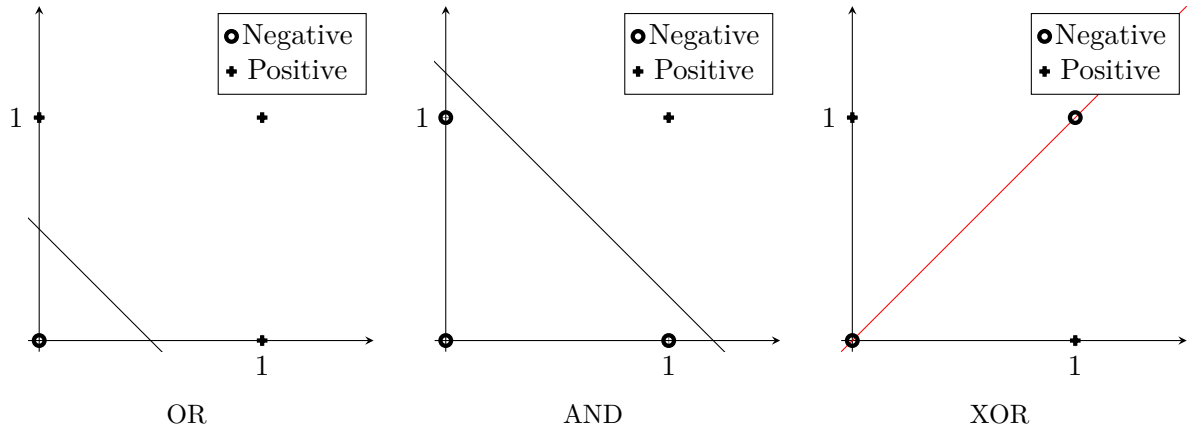
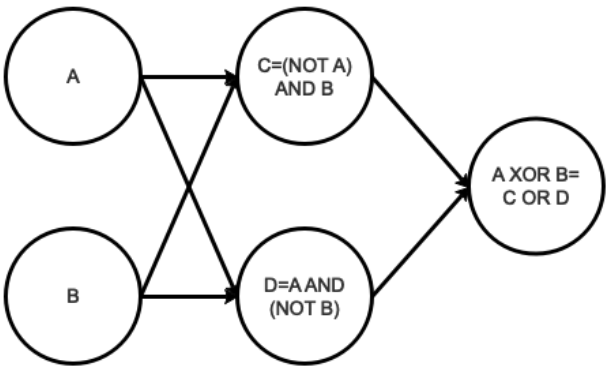


Figure 14.2: It's possible to notice that while both AND and OR are easily linearly separable, the same cannot be said for the XOR since it's not possible to find a line that separate correctly the positives from the negatives.

To solve the XOR we can use a multilevel perceptron. Indeed, we know that

$$A \oplus B = (A \wedge notB) \vee (notA \wedge B)$$

So we could compute the parenthesis separately first and the unite them via an OR.



This could be true for every boolean formula: all of them could be learned by a two layer perceptron since all boolean formulas can be expressed in conjunctive or disjunctive normal formulas. Why then do we need multi strata deep networks instead of using only two? Because the conjunctive, disjunctive, normal formulas are exponentially in term of number of neurones. In the Figure 14.1, it's possible to notice that also a bias is present. Such bias is set to 1 by default and its weight will let it be useful by learning the actual value. Moreover, this value allows us to rewrite Equation 14.1 as augmented vectors:

$$f(\boldsymbol{x}) = \text{sign}(\widehat{\boldsymbol{w}}^T \widehat{\boldsymbol{x}})$$

where:

$$\widehat{\boldsymbol{w}} = \begin{pmatrix} w_0 \\ \boldsymbol{w} \end{pmatrix} \qquad \widehat{\boldsymbol{x}} = \begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}$$

From now one all the formula will include bias and weight without specifying.

14.3 Parameter Learning

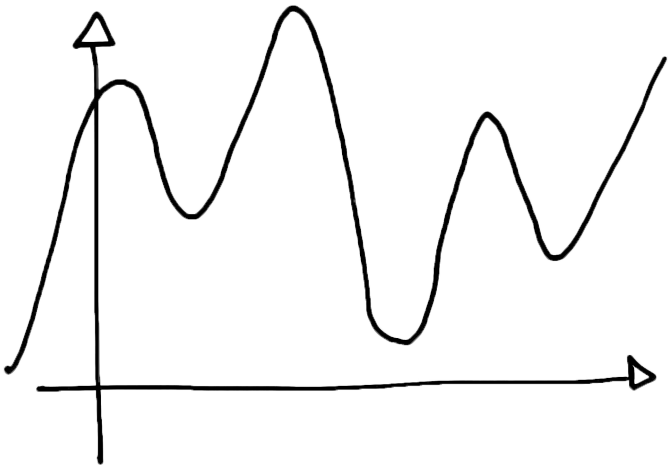
Our goal now is to optimize a function of the parameters in the same way we did for maximum likelihood for probability distributions.
Since the weights are learned from the data $\mathcal{D}$, then an error function, e.g., the loss function l, will do the deed:

$$E(\boldsymbol{w}; \mathcal{D}) = \sum_{(\boldsymbol{x},y)\in\mathcal{D}} l(\boldsymbol{y}, f(\boldsymbol{x}))$$

This though can lead to overfitting since the algorithm could simply memorize all the training examples. This actually is not a problem with linear classifiers, but with more difficult systems, this solution is not sufficient.

14.3.1 Gradient Descent – Batch Training

Now let's suppose we know the loss function we want to use, we can try to do error minimization by changing the weighs. One way to approach this problem is gradient descend. Let's suppose we have just one parameter and the error function over the parameter is the following:



To minimise it, we start from $\boldsymbol{w} = 0$, then we compute the gradient and move opposite of the gradient (otherwise we'd maximise). Then we compute the new weights as:

$$\boldsymbol{w} = \boldsymbol{w} - \eta \nabla E(\boldsymbol{w}; \mathcal{D})$$

where η is the learning rate and tells the algorithm how much it needs to move on the graph each time.
After recomputing the weights, we repeat the procedure until the gradient is approximately zero. Mind that with this approach we reach just a local minimum. The learning rate η plays an important role in finding the local minimum:

- Too low: means a slow convergence;
- Too high: means jumping around the curve.

There exist also techniques to adaptively modify η. Let:

$$E(\boldsymbol{w}; \mathcal{D}) = \sum_{(\boldsymbol{x}, y) \in \mathcal{D}_E} -y f(\boldsymbol{x})$$

Where $\mathcal{D}_E$ is the set of current training errors for which:

$$y f(\boldsymbol{x}) \leq 0$$

The error is the sum of the functional margins of incorrectly classified examples.
The error gradient is:

$$\begin{aligned} \nabla E(\boldsymbol{w}; \mathcal{D}) &= \nabla \sum_{(\boldsymbol{x}, y) \in \mathcal{D}_E} -y f(\boldsymbol{x}) \\ &= \nabla \sum_{(\boldsymbol{x}, y) \in \mathcal{D}_E} -y(\boldsymbol{w}^T \boldsymbol{x}) \\ &= \sum_{(\boldsymbol{x}, y) \in \mathcal{D}_E} -y\boldsymbol{x} \end{aligned}$$

The amount of update is:

$$-\eta \nabla E(\boldsymbol{w}; \mathcal{D}) = \eta \sum_{(\boldsymbol{x}, y) \in \mathcal{D}_E} -y\boldsymbol{x}$$

Hence the update of the weights will be:

$$\boldsymbol{w} = \boldsymbol{w} + \eta \sum_{(\boldsymbol{x}, y) \in \mathcal{D}_E} -y\boldsymbol{x}$$

This is called batch training because every step we compute the error on the entire training set.

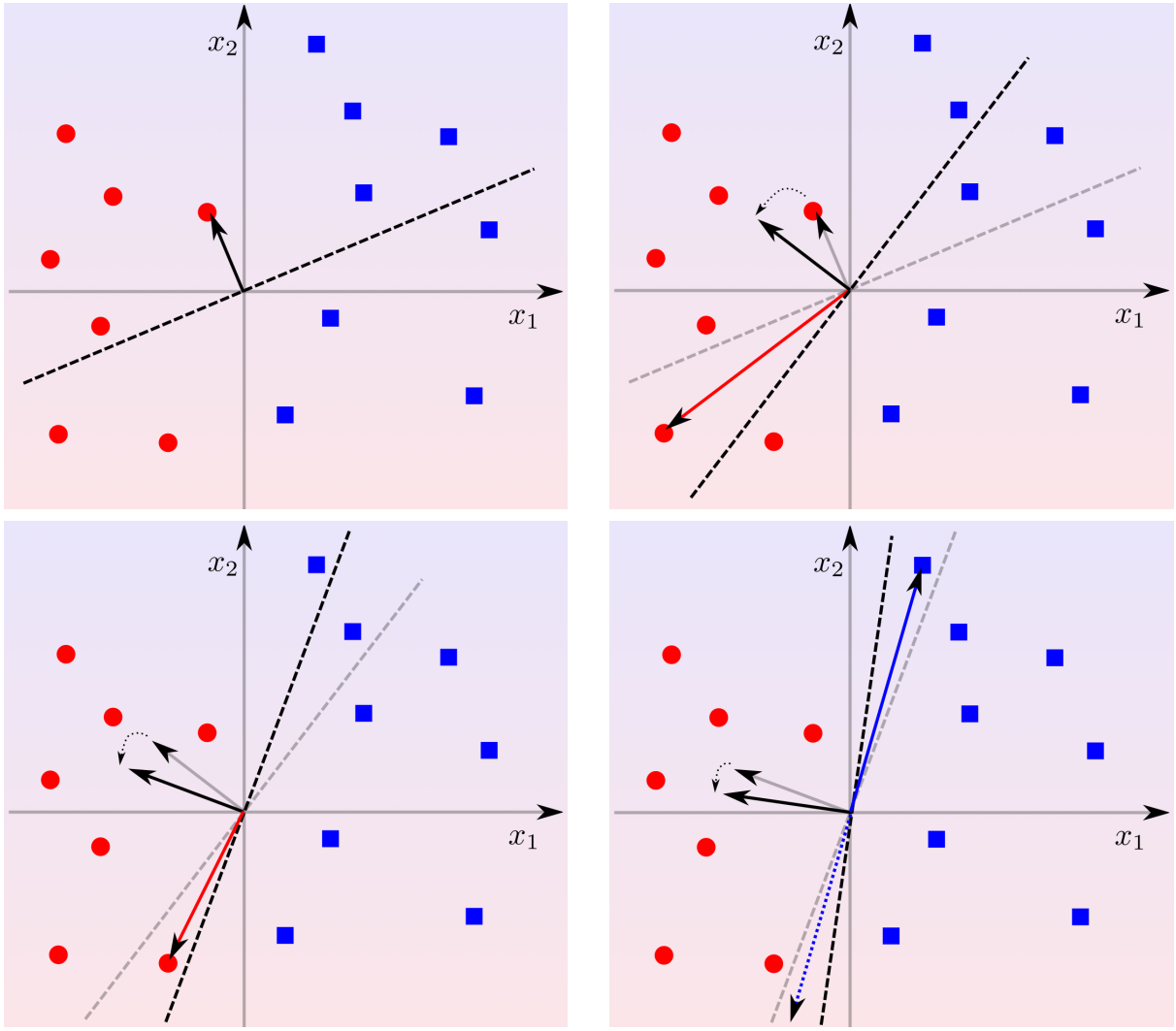
14.3.2 Stochastic Perceptron Training Rule

In this version, each time we have a training error we make a gradient step: if the sample was correctly classified then ok, otherwise we move with the gradient and recompute the weights. The algorithm works as follows:

- Initialize the weights randomly;
- Iterate until all examples are correctly classified:
  - For each incorrectly classified training example $(\boldsymbol{x}, y)$ update the weight vector:

$$\boldsymbol{w} = \boldsymbol{w} + \eta y \boldsymbol{x}$$

This is called stochastic perceptron and is usually pretty fast. In some cases it allows to avoid local minima since each time the gradient function is a little different, hence the algorithm is guided by various gradients for each training examples.
What it basically does is to rotate the separating hyperplane until all the errors disappear.



14.4 Regression with Perceptrons – Exact Solution

With linear predictors we can also do regression other the classification.
Let's say we have n examples $\boldsymbol{x}_i$ and that each example has $|\boldsymbol{x}_i|$ features, formally let $X \in \mathbb{R}^n \times \mathbb{R}^d$

be the input matrix, $X = [\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n]^T$ for $n = |\mathcal{D}|$ and $d = |\boldsymbol{x}|$.
Let $\boldsymbol{y} \in \mathbb{R}^n$ be the output training matrix (i.e., y_i is output for example $\boldsymbol{x}_i$. The desired output is an n-dimensional vector of real values.
Given that the regression learning function could be state as a set of linear equations:

$$X\boldsymbol{w} = \boldsymbol{y}$$

The perfect regression would be:

$$\forall i, y_i = \boldsymbol{w}^T \boldsymbol{x}_i$$

$$\boldsymbol{w} = X^{-1}\boldsymbol{y}$$

But this doesn't work since X is not revertible in many cases. Moreover, the system of equations is usually overdetermined, that is there are more equations than unknowns. This means that usually there is no exact solution: we should forget about having an exact solution, but rather trying to minimize the error.

14.4.1 Mean Squares Error

We shall resort to loss minimization. The standard loss for regression is the mean squared error:

$$E(\boldsymbol{w}; \mathcal{D}) = \sum_{(\boldsymbol{x},y)\in\mathcal{D}} (y - f(\boldsymbol{x}))^2 = (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w})$$

For this we also have a closed form and it can be solved by gradient descent, which make it usable also for classification loss.
The closed form can be computed by setting the gradient descent to 0:

$$\begin{aligned}
\nabla E(\boldsymbol{w}; \mathcal{D}) &= \nabla(\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) \\
&= 2(\boldsymbol{y} - X\boldsymbol{w})^T(-X) = 0 \\
&= -2\boldsymbol{y}^T X + 2\boldsymbol{w}^T x^T X = 0 \\
\boldsymbol{w}^T X^T X &= \boldsymbol{y}^T X \\
X^T X\boldsymbol{w} &= X^T\boldsymbol{y} \\
\boldsymbol{w} &= (X^T X)^{-1} X^T \boldsymbol{y}
\end{aligned}$$

The part $(X^T X)^{-1} X^T$ is called *left-inverse*: if X is square and nonsingular, inverse and left-inverse coincide and the MSE solution corresponds to the exact one.
The left-inverse exists provided $(X^T X) \in \mathbb{R}^{d\times d}$ is full rank, that is the features are linearly independent. If not we could simply remove the redundant ones and have a left-inverse.
This said, the solution via gradient descent is usually the most efficient. We descend for each weight w_i so that we can quantify the next weight variation. The following operations are meant to be executed on the full vector of the weights.

$$\begin{aligned}
\tfrac{\delta E}{\delta \boldsymbol{w}_i} &= \tfrac{\delta}{\delta \boldsymbol{w}_i} \tfrac{1}{2} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} (y - f(\boldsymbol{x}))^2 \\
&= \tfrac{1}{2} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \frac{\delta}{\delta \boldsymbol{w}_i} (y - f(\boldsymbol{x}))^2 \\
&= \tfrac{1}{2} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} 2(y - f(\boldsymbol{x})) \frac{\delta}{\delta \boldsymbol{w}_i} (y - \boldsymbol{w}^T \boldsymbol{x}) \\
&= \sum_{(\boldsymbol{x},y)\in\mathcal{D}} (y - f(\boldsymbol{x}))(x_i)
\end{aligned}$$

14.5 Multiclass Classification

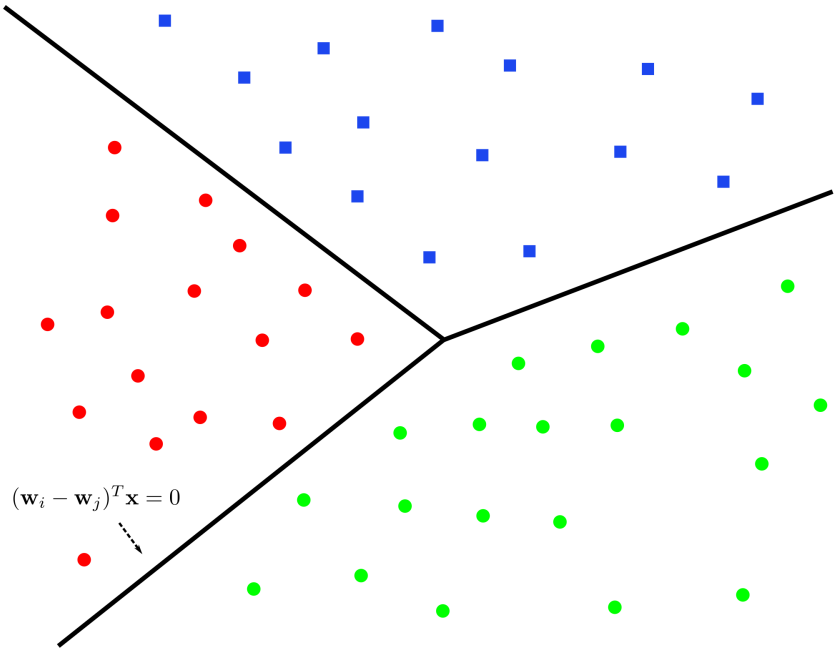
We'll briefly see two possible approaches.

14.5.1 Multiclass Classification – One-VS-All

In this case the idea is to learn a binary classifier for each class: or the example belongs to said class, or it does not an belongs to another class.
The idea is to predict a new example in the class with maximum functional margin.
The decision boundaries for which $f_i(\boldsymbol{x}) = f_j(\boldsymbol{x})$ are pieces of hyperplanes:

$$\boldsymbol{w}_i^T \boldsymbol{x} = \boldsymbol{w}_j^T \boldsymbol{x}$$

$$(\boldsymbol{w}_i - \boldsymbol{w}_j)^T \boldsymbol{x} = 0$$



14.5.2 Multiclass Classification – All-Pairs

In this case the classes are paired and one binary classifier is associated to each pair:

- Positive examples are from class A;
- Negative examples are from class B.

The new example is predicted as belonging to a class when it wins the largest number of pairwise classifications.

14.6 Generative Linear Classifiers

In Gaussian distributions, the boundaries are obtained when the covariance is shared among classes: $\sigma_i = \sigma$. We could use Naive Bayes as a classifier:

$$f_i(\boldsymbol{x}) = P(\boldsymbol{x}|y_i)P(y_i) = \prod_{j=1}^{|\boldsymbol{x}|} \prod_{k=1}^{K} \theta_{ky_i}^{z_k(x[j])} \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$$

$$f_i(\boldsymbol{x}) = \prod_{k=1}^{K} \theta_{ky_i}^{N_{k\boldsymbol{x}}} \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$$

Where $N_{k\boldsymbol{x}}$ is the number of times that feature k appears in $\boldsymbol{x}$.
By applying the logarithm we have:

$$\log(f_i(\boldsymbol{x})) = \underbrace{\sum_{k=1}^{K} N_{k\boldsymbol{x}} \log(\theta_{ky_i})}_{\boldsymbol{w}^T \boldsymbol{x}'} + \underbrace{\log\left(\frac{|\mathcal{D}_i|}{|\mathcal{D}}\right)}_{w_0}$$

Where:

- $\boldsymbol{x}' = [N_{1\boldsymbol{x}} \dots N_{k\boldsymbol{x}}]^T$
- $\boldsymbol{w} = [\log \theta_{1y_i} \dots \log \theta_{ky_i}]^T$

Naive Bayes then can be seen as a *log-linear* model.

15 20/11/2019

16 Support Vector Machines

Support Vector Machines (SVM) are linear classifiers selecting a hyperplane such that the separation maximize the separation margin between classes.
The solution actually depends only on a small subset of training examples, called **support vectors**.
It also present a sound generalization theory and it can be easily extended to nonlinear separation with *kernel machines*.

16.1 Maximum Margin Classifier

Given a training set $\mathcal{D}$, a classifier *confidence* margin is the minimal confidence margin (for predicting a true label) among training examples:

$$\rho = \min_{(\boldsymbol{x},y)\in\mathcal{D}} yf(\boldsymbol{x})$$

Instead a classifier *geometric margin* is:

$$\frac{\rho}{||\boldsymbol{w}||} = \min_{(\boldsymbol{x},y)\in\mathcal{D}} \frac{yf(\boldsymbol{x})}{||\boldsymbol{x}||}$$

There is an infinite number of equivalent formulation for the same hyperplane. Indeed if:

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = 0$$

Then:

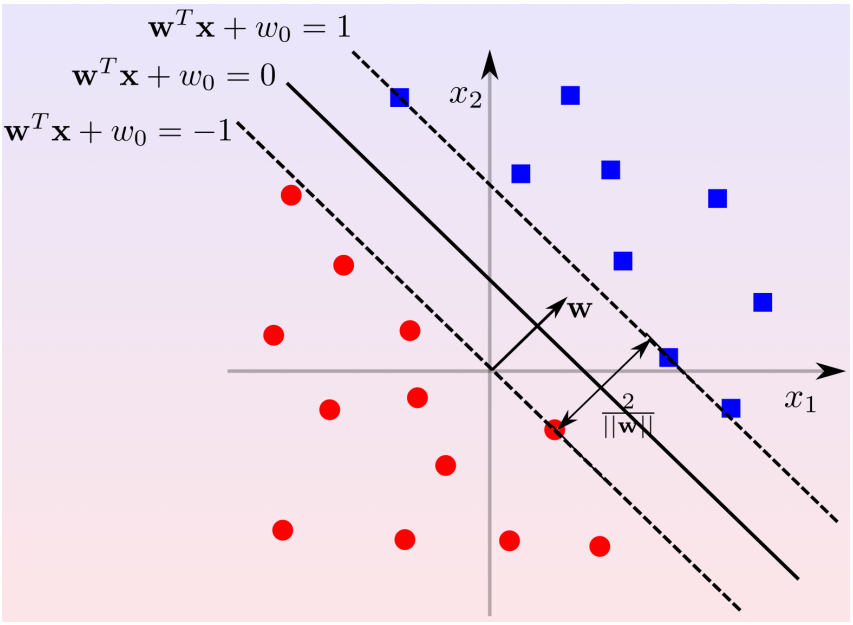
$$\alpha(\boldsymbol{w}^T\boldsymbol{x} + w_0) = 0 \quad \forall\alpha \neq 0$$

The **canonical hyperplane** is the one having confidence margin equal to 1:

$$\rho = \min_{(\boldsymbol{x},y)\in\mathcal{D}} yf(\boldsymbol{x}) = 1$$

and its geometric margin is:

$$\frac{\rho}{||\boldsymbol{w}||} = \frac{1}{||\boldsymbol{w}||}$$



16.2 Hard Margin SVM

Theorem 16.1: Margin Error Bound

Consider the set of decision functions $f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x})$ with $||\boldsymbol{w}|| \leq \wedge$ and $||\boldsymbol{x}|| \leq R$, for some $R, \wedge > 0$. Moreover, let $\rho > 0$ and v denote the fraction of training examples with margin smaller than $myfrac\rho||\boldsymbol{w}||$, referred to as the **margin error**.
For all distributions P generating the data, with probability at least $1-\delta$ over the drawing of the m training patterns, and for any $\rho > 0$ and $\delta \in (0, 1)$, the probability that a test pattern drawn from P will be misclassified is bound from above by:

$$v + \sqrt{\frac{c}{m}\left(\frac{R^2\wedge^2}{\rho^2}\ln^2 m + \ln\left(\frac{1}{\delta}\right)\right)} \tag{16.1}$$

where c is a universal constant.

What the theorem says is that the probability of test error depends on (among other components):

- The number of margin errors v, that is examples with margin smaller than $\rho/||\boldsymbol{w}||$;
- The number of training examples, indeed the error depends on $\ln^2 m/m$;
- The size of the margin, the error depends on $1/\rho^2$.

Notice that if we are considering the canonical hyperplane, then $\rho = 1$ and the problem corresponds to minimizing $||\boldsymbol{w}||$:

$$\min_{\boldsymbol{w},w_0} \frac{1}{2}||\boldsymbol{w}||^2 \quad \text{subject to:} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) \geq 1 \quad \forall(\boldsymbol{x}_i, y_i) \in \mathcal{D}$$

The constraint guarantees that all points are correctly classified.
Notice that this minimization corresponds to maximizing the (squared) margin.

16.2.1 Karush-Kuhn-Tucker Approach (KKT)

This approach starts from the fact that a constrained optimization problem can be addessed by converting it into an unconstrained problem with the same solution.
Let's have a constrained optimization problem in the form:

$$\min_z f(z) \quad \text{subject to:} \quad g_i(z) \geq 0 \forall i$$

Now we can introduce a non-negative variable $\alpha_i \geq 0$, called Langrange multiplier, for each constraint and rewrite the optimization problem in the form of a Langrangian:

$$\min_z \max_{\alpha \geq 0} f(z) - \sum_i \alpha_i g_i(z)$$

The optimal solutions z^* for this problem are the same as the optimal solutions for the original (constrained) problem:

- If for a given z' at least one constraint is not satisfied, i.e., $g_i(z') < 0$ for some i, then maximizing over α_i leads to an infinite value;
- If all constraints are satisfied, i.e., $g_i(z') \geq 0 \forall i$, then the maximization over the $\boldsymbol{\alpha}$ will set all elements of the summation to zero, so that z' is a solution of $\min_z f(z)$.

The KKT approach can be used to solve the hard margin SVM constraint by rewriting the problem with a Langrangian as:

$$L(\boldsymbol{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{m} \alpha_i \Big(y_i(\boldsymbol{w}^T \boldsymbol{x}_i + w_0) - 1 \Big)$$

where $m = |\mathcal{D}|$. The Langrangian is minimized w.r.t. $\boldsymbol{w}, w_0$ and maximized w.r.t. α_i. The solution is a saddle point.
Let's then first minimize over $\boldsymbol{w}$ and w_0:

$$\frac{\partial}{\partial \boldsymbol{w}} L(\boldsymbol{w}, w_0, \boldsymbol{\alpha}) = 0$$

$$\frac{\partial}{\partial \boldsymbol{w}} \left(\frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{m} \alpha_i \Big(y_i(\boldsymbol{w}^T \boldsymbol{x}_i + w_0) - 1 \Big) \right)$$