# Machine Learning Summaries and Solved Exercises

Nicolò Brandizzi

July 20, 2020

# Contents

# Chapter 1

# Summaries

## 1.1 Intro

ML is producing knowledge from data, learning a function $f : X \to Y$ that takes some input $x \in X$ and outputs $y \in Y$.
Learning $f$ means finding some approximation $f' \approx f$ so that the returned value $y' \simeq y$ is the closes to $y$ as possible.

**Un/Supervised** When we have an output $y$ for each sample $y$ for the dataset $D = \{(x_i, y_i)\}$ we have **supervised** learning. When we do not have $y_i$ then we have **unsupervised**.

**Reinforcement Learning** Having a triple of elements $D : (s_i, a_i, r_i)$ (state, action, reward), RL is the practice of learning a policy $\pi$ in order to maximize the total discounted reward:

$$\pi : argmax(r_0 + \gamma r_1 + \gamma^2 r_2 + ... + \gamma^n r_n)$$

Where $\gamma$ is some discount to give more/less weight to current reward on behalf of the future ones.

**Hypothesis** Given a target function $\phi(x)$ that we want to learn and a set of approximations $H : \{h_1, h_2, ..., h_n\}$ of which $h*$ is the best according to some measure, we have that $h*(x_i)$ is the best approximation of $y_i$.
$h*$ is **consistent** with the dataset $D$ iff:

$$h*(x_i) = \phi(x_i),\ \forall x \in D$$

### 1.1.1 Evaluation and Estimators

**True Error** Given an hypothesis $h$ that approximates a target function $\phi$ the **true error** is the probability that:

$$error_t(h) \equiv P(h(x) \neq \phi(x))$$

Unfortunately it cannot be computed.

**Sample Error** Given an hypothesis $h$ that approximates a target function $\phi$ the **sample error** is the proportion of samples not correctly classified:

$$error_s(h) \equiv \frac{1}{n} \sum_{x \in D} \delta(h(x), \phi(x))$$

Where $\delta(h(x), \phi(x)) = 1$ iff $h(x) \neq \phi(x)$, 0 otherwise.
We have that $accuracy(h) = 1 - error_s(h)$

**Estimation Bias** We have that the bias between true error and sample error is:

$$bias \equiv E[error_s(h)] - error_t(h)$$

**Comparing hypothesis** Having two hypothesis $h_1, h_2$ the comparison can be:

$$d = error_t(h_1) - error_t(h_2)$$
$$\hat{d} = error_s(h_1) - error_s(h_2)$$
$$E[\hat{d}] = d$$

**Overfitting** An hypo $h$ overfits the data if, given another hypo $h'$ we have that:

$$error_s(h) > error_s(h') \quad and \quad error_t(h) < error_t(h')$$

**K-fold Cross Validation** Partition the dataset $D$ into $k$ subsets $D : \{S_1, S_2, ..., S_k\}$. Use one subset $S_i$ as test set and all the other make up the training set $T = S_1 \cup S_2 \cup ... \cup S_{i-1} \cup S_{i+1} \cup ... \cup S_k$.

**Others**

- **Error rate**= $\frac{FN+FP}{TP+TN+FP+FN}$
- **Accuracy**= 1- error rate
- **Recall**= $\frac{TP}{TP+FP}$
- **Precision**=$\frac{TP}{TP+FP}$
- **F1-score**=$\frac{2 \cdot Precision * Recall}{Precision+Recall}$
- **Confusion Matrix**: miss - classification rate between classes $C_j, C_i$.

### 1.1.2 Decision Trees

Given an instance space $X$ coming from a set of attributes a decision tree has:

- an attribute for each internal node test
- a branch for each value of an attribute
- a leaf to which assigns a classification value

A rule is generated for each path to a leaf node.

**Entropy** Having the proportion of positive samples as $p_+$ and the negative ones as $p_- = 1 - p_+$ the entropy measure the impurity of the set of samples $S$:

$$Entropy(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

**Information gain** Information gain measures how well a given attribute separates the training examples according to their target classification. Information gain is measured as reduction in entropy.
$Gain(S, A)$ is the expected reduction in entropy of $S$ caused by knowing the value of attribute $A$:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(S)} \frac{|S_v|}{S} \cdot Entropy(S_v)$$

**ID3** Is an algorithm used to generate a decision tree from a dataset. Hypothesis space search by ID3 is complete (target concept is there), outputs a single hypothesis (cannot determine how many DTs are consistent), no back tracking (local minimal), statistically-based search choices (robust to noisy data), uses al the training examples at each step (not incremental).

**Issues**

- decise depth
- handling continous attributes
- choosing appropriate attribute selection measures
- missing attribute values
- handling attributes with different costs.

**Overfitting and pruning** To avoid tree overfitting an approach is to grow a full tree and then post-prune (replace nodes not important with leafs).
To determine correct tree size, use a separate set of examples (training test and validation set), apply statistical test to estimate accuracy of a tree on data distribution

### 1.1.3 Probability and Bayes

**Posterior** Conditional (or posterior) probability arise after the arrival of some evidence. If I know the outcome of a random variable, how will this affect probability of other random variables?

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

Which also makes up the *chain rule* as:

$$P(a, b) = P(a|b)P(b)$$

## 1.2 Bayes Learning

Basic formulas:

- Same notation from now on $P(A \wedge B) = P(A, B)$
- **Product Rule**: $P(A \wedge B) = P(A|B)P(B) = P(B|A)P(A)$
- **Sum Rule** : $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$
- **Theorem of Probability** (also called marginalization): Given some mutaully exclusive events $A_1, ..., A_n$ where $\sum_{i=1}^{n} P(A_j) = 1$ we have:

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

**Bayes Theorem** Given:

- $P(h)$ the prior probability of the hypothesis $h$.
- $P(D)$ the prior probability of the training data $D$.

We have:

$$p(h|D) = \frac{P(D|h)P(h)}{p(D)}$$

### 1.2.1 Maximum a Posteriori probability

When classifying new data we want to assign it the most probable hypothesis. To do so we can use **Maximum a posteriori** hypothesis $h_{MAP}$:

$$h_{MAP} = \arg\max_{h \in H} P(h|D) = \arg\max_{h \in H} \frac{P(D|h)P(h)}{p(D)} = \arg\max_{h \in H} P(D|h)P(h)$$

where $H$ is the set of all the hypothesis, and from the third equation we have removed the normalizing constant $P(D)$.
Moreover if the prior distribution is uniform, i.e. $P(h_i) = P(h_j),\ \forall i, j \in H$ we can use the **Maximum Likelihood** hypothesis $h_{ML}$ and have:

$$h_{ML} = \arg\max_{h \in H} P(D|h)$$

We can estimate the maximum $h_{MAP}$ by computing $P(h_i|D)$ for every $h_i \in H$ and then get the maximum, **but** $h_{MAP}$ return the most probable *hypothesis*, not the most probable classification, so, given a new instance $x$, $h_{MAP}(x)$ might not the return the correct classification nor the most probable.

**Example** Let's consider three possible hypothesis $h_1, h_2 h_3$ driven from data $D$, where the probability of each hypothesis is:

$$P(h_1|D) = 0.4 \quad P(h_2|D) = 0.3 \quad P(h_3|D) = 0.3$$

as we can see the $h_{MAP} = h_1$, that is the hypothesis is the most probable in the dataset.

Let us now consider a new instance $x$, and the possible classifications by the three hypothesis:

$$h_1(x) = \oplus \quad h_2(x) = \ominus \quad h_3(x) = \ominus$$

It is now obvious that $h_{MAP}(x) = h_1(x) = \oplus$ is not the most probable classification.

### 1.2.2 Bayes Optimal Classifier

To fix the above mentioned problem consider the following:

- $C = \{c_1, c_2, ..., c_n\}$ is a set of possible classes.
- $X = \{x_1, x_2, ..., x_m\}$ is a set of instances of the dataset $D$
- $f : X \to C$ is the target function that maps an instance to a class.
- $P(c_j|x', h_i)$ is the probability of a new instance $x'$ to be classified as the class $c_j$ by a hypothesis $h_i$.
- $P(c_j|x', D)$ the probability that $x'$ belongs to the class $c_j$ conditioned to the entire dataset $D$ (every hypothesis).

We have that:

$$P(c_j|x', D) = \sum_{h_i \in H} P(c_j|x', h_i)P(h_i|D)$$

Moreover this kind of function $f$ is called **Bayes Optimal Classifier** [OB], so the most probable class $c_{OB}$ for a new instance $x'$ would be:

$$c_{OB} = \arg\max_{c_i \in C} \sum_{h_j \in H} P(c_i|x', h_j)P(h_j|D)$$

**Example** Given:

| hypothesis prior | positive class | negative class |
|---|---|---|
| $P(h_1\|D) = 0.4$ | $P(\oplus\|x, h_1) = 0$ | $P(\ominus\|x, h_1) = 1$ |
| $P(h_2\|D) = 0.3$ | $P(\oplus\|x, h_2) = 1$ | $P(\ominus\|x, h_2) = 0$ |
| $P(h_3\|D) = 0.3$ | $P(\oplus\|x, h_3) = 1$ | $P(\ominus\|x, h_3) = 0$ |

We have that:

$$\sum_{h_i \in H} P(\oplus|x', h_i)P(h_i|D) = 0.4$$

$$\sum_{h_i \in H} P(\ominus|x', h_i)P(h_i|D) = 0.6$$

Thus:

$$c_{OB} = \arg\max_{c_i \in C} \sum_{h_j \in H} P(c_i|x', h_j)P(h_j|D) = \ominus$$

## 1.3 Other Distributions

### 1.3.1 Bernoulli

Describes the probability distribution of a binary random variable $X \in \{0, 1\}$:

$$P(X = 1) = \theta \quad P(X = 0) = 1 - \theta$$

The probability of $X$ being a certain value $x$, given $\theta$ is:

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

While the probability of obtaining $k$ times the same result with $n$ trials is:

$$P(X = k, n, \theta) = \binom{n}{k} \theta^k (1 - \theta)^{n-k}$$

**Multi-variate Bernoulli distribution** Having a set of mutually independent binary random variable $X_1, X_2, ..., X_n$ following a Bernoulli distribution, the Multi-variate Bernoulli distribution [MvBd] is the product of all $n$ distributions:

$$\prod_{i=1}^{n} P(X_i = x_i, \theta_i)$$

## 1.4 Naive Bayes Classifier

When we have to deal with a high hypothesis space, the Bayes Optimal Classifier is not practical anymore. A way to avoid computing every hypothesis is using conditional independence.
Let's write a new instance $x'$ as a set of attributes $A = \{a_1, a_2, ..., a_n\}$, so we have that:

$$c_{OB} = P(c_j|x', D) = P(c_j|A, D) = P(c_j|a_1, a_2, ..., a_n, D)$$

Using the Bayes theorem we have:

$$c_{OB} = \arg\max P(c_j|A, D) = \arg\max \frac{P(A|c_j, D)P(c_j|D)}{P(A|D)} = \arg\max P(A|c_j, D)P(c_j|D)$$

In which $P(A|c_j, D)$ is too difficult to compute.
For this reason the Naive classifier assumes that each attribute is independent from the other, effectively transforming $P(a_1, a_2, ..., a_n|c_j, D)$ to $\prod_i P(a_i|c_j, D)$, so that the $f_{NB}$ becomes:

$$c_{NB} = \arg\max_{c_j \in C} P(c_j|D) \prod_i P(a_i|c_j, D)$$

It can happen during estimation that some attributes $a_i$ does not have a prior for a class $c_j$ so that $P(a_i|c_j, D) = 0 \Rightarrow P(c_j|D) \prod_i P(a_i|c_j, D) = 0$. In this case we can set a *virtual prior* to some arbitrary number to avoid having zero.

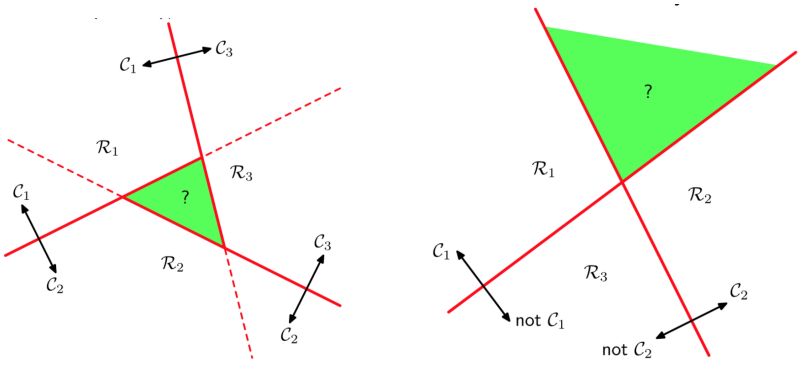
## 1.5 Linear Models

### 1.5.1 Calssification

Some instance are linearly separable if it exists some hyper-plane that splits the space in two regions such that different classes are separated. Such hyper-plane is generated by a function $f$ which maps points in $\mathcal{R}^n$ to $C = \{c_1, c_2, ..., c_m\}$

$$f : y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

The problem with using multiple classes is shown in the following figure.




(a) One Versus One (b) One Versus Rest

Figure 1.1: One Versus Rest

A solution is using a **K-Class Discriminant**:

$$f : y_k(\boldsymbol{x}) = \boldsymbol{w}_k^T \boldsymbol{x} + w_{k0}$$

Where the decision boundary between two classes $C_j, C_k$, is given by:

$$(\boldsymbol{w_j} - \boldsymbol{w_k})^T \boldsymbol{x} + (w_{j0} - w_{k0})$$

which can be written in a more compact notation having:

- $\tilde{\boldsymbol{w_k}} = \binom{w_{k0}}{\boldsymbol{w_k}}$
- $\tilde{\boldsymbol{x}} = \binom{1}{\boldsymbol{x}}$
- $\tilde{\boldsymbol{W}} = (\boldsymbol{w_1}, \boldsymbol{w_2}, ...\boldsymbol{w_k})$

We get:

$$\boldsymbol{y(x)} = \tilde{\boldsymbol{W}}^T \tilde{\boldsymbol{x}}$$

As shown in figure 1.2.

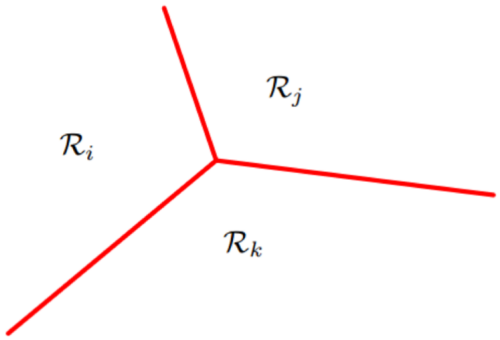



Figure 1.2: K-class discriminant

Our goal is to determine a good $\tilde{\boldsymbol{W}}$, there are various techniques we can use.

#### 1.5.1.1 Least Squares

Given a dataset $D$ made of instances associated with a one hot vector encoder:

$$\forall x_i \in D : iff\ x_i \in c_k \rightarrow t_i = (0, 0, ..., 1, ...0) : t_i[k] = 1$$

The Least Square aims to minimize the sum-of-square error function

#### 1.5.1.2 Fischer linear discriminant

The main idea is to find projection to a line s.t. samples from different classes are well separated.
Suppose we have two classes and $n$ samples $x_1, ..., x_n$ where $n_1$ samples are from class 1 and $n_2$ are from 2.
Considere a line which direction is given by a vector $v$, so that $v^t x_i$ is the projection of a point $x_i$ (2D) onto the line (1D).
We have that the mean of the points of classes 1 is:

$$\mu_1 = \frac{1}{n_1} \sum_{x_i \in C_1}^{n_1} x_i$$

Same thing for $\mu_2$. While the mean of the points **projected** onto the line is simply:

$$\tilde{\mu_1} = v^t \mu_1$$

So we can use $dist = |\tilde{\mu_1} - \tilde{\mu_2}|$ as a measure of separation, since the bigger the better, but look at Figure 1.4:

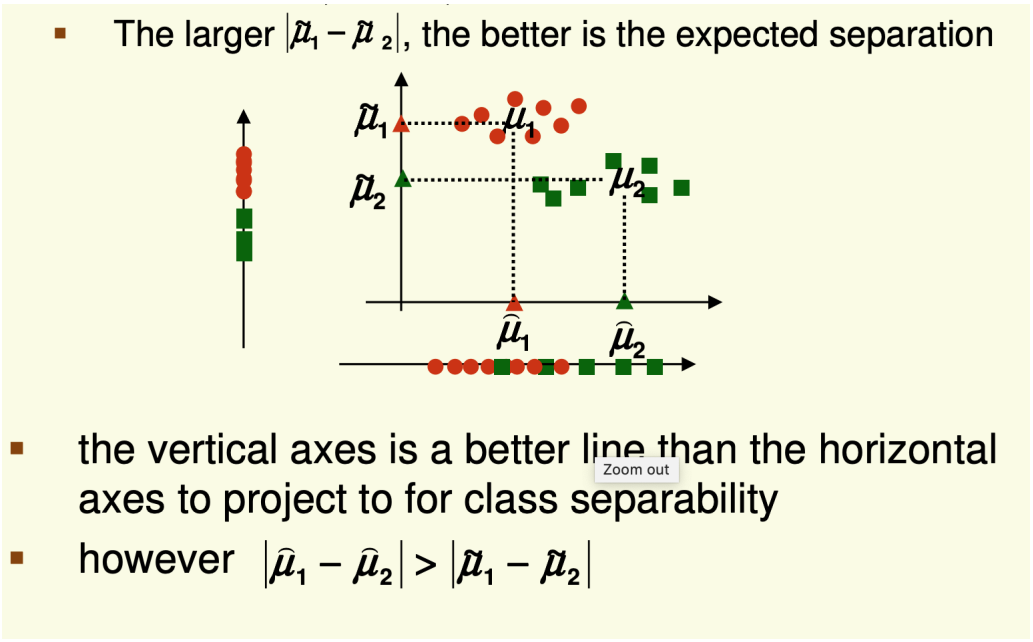



Figure 1.3: Mean proble

The problem is that it does not consider the variance between classes, so we need to normalize *dist* by something proportional to the variance.
Let's define the *scatter* of some samples $z_1, ..., z_n$ as :

$$s = \sum_{i=1}^{n} (x_i - \mu_z)$$

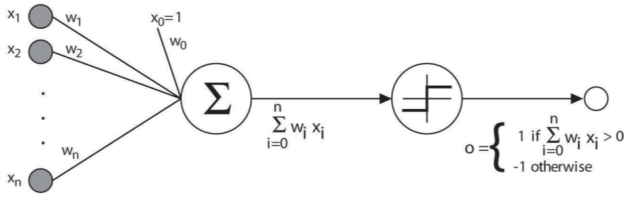
Which is:

$$s_1^2 = \sum_{x_i \in C_1} (x_i - \tilde{\mu_1})^2$$

and the same for $s_2^2$ So we can now normalize the *dist* by the scatter, getting the fisher linear discriminant:

$$J(v) = \frac{(\tilde{\mu_1} - \tilde{\mu_2})^2}{s_1^2 - s_2^2}$$

#### 1.5.1.3 Perceptron




$$o(x_1, \ldots, x_d) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_d x_d > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation (adding $x_0 = 1$):

$$o(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ -1 & \text{otherwise.} \end{cases} = sign(\mathbf{w}^T \mathbf{x})$$

Figure 1.4: Perceptron

The function to minimize is the square error, given the output $o$ and the true label $t$:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} (t_n - o_n)^2 = \frac{1}{2} \sum_{n=1}^{N} (t_n - \boldsymbol{w}^T x_n)^2$$

Since we need to minimize this error we want to move to the direction of the gradient, thus computing the derivative of:

$$\frac{\partial E(\boldsymbol{w})}{\partial w_i} = \sum_{n=1}^{N} (t_n - \boldsymbol{w}^T x_n)(-x_{i,n})$$

Where $-x_{i,n}$ is the value of i-th feature.
so we can update the weight $w_i$ by $w_{i+1} = w_i + \Delta w$, where:

$$\Delta w = -\eta \frac{\partial E(\boldsymbol{w})}{\partial w_i} = \eta \sum_{n=1}^{N} (t_n - \boldsymbol{w}^T x_n) x_{i,n}$$

**Batch mode**: Consider all dataset $D$

$$\Delta w_i = \eta \sum_{(\mathbf{x},t)\in D} (t - o(\mathbf{x}))\, x_i$$

**Mini-Batch mode**: Choose a small subset $S \subset D$

$$\Delta w_i = \eta \sum_{(\mathbf{x},t)\in S} (t - o(\mathbf{x}))\, x_i$$

**Incremental mode**: Choose one sample $(\mathbf{x}, t) \in D$

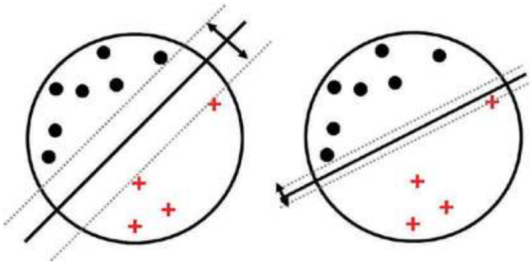
$$\Delta w_i = \eta\, (t - o(\mathbf{x}))\, x_i$$

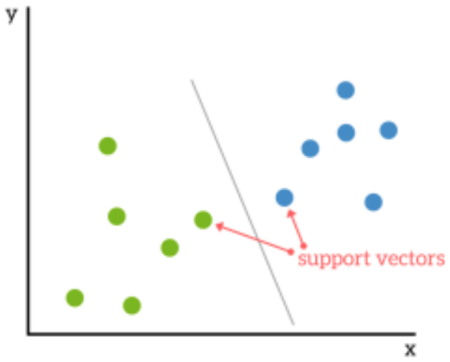
Figure 1.5: Perceptron Algotithm

#### 1.5.1.4 Support Vector Machine [SVM]

SVMs are based on the idea of finding a hyperplane that best divides a dataset into two classes

Support Vector Machines (SVM) for Classification aims at maximum margin providing for better accuracy.






Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. The distance between the hyperplane and the nearest data point from either set is known as the margin. The goal is to choose a hyperplane with the greatest

possible margin between the hyperplane and any point within the training set, where the margin is estimated as:

$$\arg\max_{\boldsymbol{w},w_0} \frac{1}{||\boldsymbol{w}||} \min_{n=1..N}[t_n(\hat{\boldsymbol{w}}^{\boldsymbol{T}}\boldsymbol{x_n} + \hat{\boldsymbol{w}}_{\boldsymbol{0}})]$$

Where the

- $\hat{h} = \hat{\boldsymbol{w}}^{\boldsymbol{T}}\boldsymbol{x_n} + \hat{\boldsymbol{w}}_{\boldsymbol{0}}$ is the hyperplane
- $\frac{1}{||\boldsymbol{w}||} \min_n[\hat{h}]$ is the smallest distance between the hyperplane and $x$.

### 1.5.2 Regression

In this case we want to predict a real value function, our model will be of the kind:

$$y(x; w) = W^T X$$

Where both $W, X$ are vectors of dimension $d$.
There are some cases in which the dataset is non linear, so we can use a non linear function on $x$ of the kind:

$$y(x; w) = W^T \phi(X)$$

The function could be anything (e.g. $\phi(x) = x^2$) but we still have a linearity in the parameters $W$.

**Maximum Likelihood** If our target value $t$ is affected by noise $\eta$:

$$t = y(X; W) + \eta$$

We now have a probability that the target is correct given the regression. If we assume $\eta$ to be gaussian we have:

$$P(t|X, W, \beta) = N(t|y(X; W), \beta^{-1})$$

Where $\beta$ is the inverse of the variance.
The only thing left is to seek that $t \in S$ which maximize this probability:

$$\begin{aligned} P(S|X, W, \beta) = \prod_n N(t_n|w^T\phi(x_n), \beta^{-1}) = \\ \sum_n ln[N(t_n|w^T\phi(x_n), \beta^{-1})] = \\ -\beta\frac{1}{2}\sum_n[t_n - w^T\phi(x_n)]^2 - \frac{N}{2}ln[2\pi\beta^{-1}) \end{aligned} \tag{1.1}$$

Since the second term is a constant we focus on the first one:

$$E_D(w) = \frac{1}{2}\sum_n[t_n - w^T\phi(x_n)]^2$$

Which we want to minimize.

**Regularization** To control overfitting we can set a regularization factor on the parameters of the kind:

$$E_W(w) = \frac{1}{2} W^T W$$

So that the minimization becomes:

$$min[E_D(w) + \lambda E_W(w)]$$

## 1.6 Probabilistic Classification

### 1.6.1 Generative Models

The generative models focus on estimating the probability of a class given a sample $P(c_i|x)$ with the bayes theorem $P(x|c_i)$.
Consider the case of two classes $C_1, C_2$, we have that:

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

with:

$$\alpha = ln\frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}$$

We can write:

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

Which is the sigmoid function.
At this point we can assume that every calss has the same covariance matrix for a certain gaussian distribution:

$$P(x|c_i) = N(x|\mu_i, \Sigma)$$

So that our $\alpha$ becomes:

$$\alpha = w^t x + w_0$$

with:

$$\begin{aligned} w &= \Sigma^{-1}(\mu_1 - \mu_2) \\ w_0 &= -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + ln\frac{P(c_1)}{P(c_2)} \end{aligned} \tag{1.2}$$

For a multiclass problem the equation becomes:

$$P(c_k|x) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

**Maximum Likelihood for K classes** Since we are dealing with the bayes theorem we can use ML for a multiclass problem with K classes. For a class $k$ we have:

$$\begin{aligned} P(c_k) &= \frac{N_k}{N} \\ P(x|c_k) &= N(x|\mu_k, \Sigma) \\ \mu_k &= \frac{1}{N_k}\sum_n t_{n,k}x_n \\ \Sigma &= \sum_k \frac{N_k}{N}S_k \\ S_k &= \frac{1}{N_k}\sum_n t_{n,k}(x_n - \mu_k)(x_n - \mu_k)^T \end{aligned} \tag{1.3}$$

### 1.6.2 Discriminative Models

Differently from the previous one it tries to estimate $P(c_i|x)$ directly from the model.

**Logistic Regression** I tries to compute the likelihood function as:

$$p(t|w) = \prod_n y_n^{t_n}(1-y_n)^{t_n}$$

Which is then optimized using the cross-entropy function:

$$w^* = \arg\min_w E(w), \quad E(w) = -\ln p(t|w)$$

**Iterative re-weighted last squares** The minimization can be solved with the IRLS using:

- $R$ a diagonal matrix where the elements are $R_{nn} = y_n(1-y_n)$
- A hessian matrix $H = X^T RX$

Then the gradient descend step becomes:

$$w \leftarrow w - H^{-1}\nabla E(w)$$

## 1.7 Multiple Learner

The idea is : instead of training a complex learner/model, train many different learners/models and then combine their results.

**Voting** Here the models are trained in parallel on the same dataset $D$ and then the outputs are summed (for regression) or chosen based on the most voted class (for classification).

**Bagging** In here we separate the dataset $D$ into $M$ parts (called bootstraps), then we train a different learner for each part and average the results.

### 1.7.1 Boosting

Contrary to the other methods, the boosting train weak classifiers sequentially. This is because some weak learner at iteration $i$ will output a miss-classified $y_{i,n} \neq t_n$ associated with a point $x_n$ and its weight $w_{i,n}$ . During the next training iteration the learner will find the weight associated with the point $x_n$ greater than it was before $w_{i+1,n} > w_{i,n}$ so that it learns to give more importance to this point and avoid future miss-classifications.

**AdaBoost** AdaBoost works as just described using decision trees with a signle split.

The advantages are:

- fast, simple
- no prior about base learner
- no parameter to tune
- can be combined with any method

The **algorithm** works as follows:

1. having a Dataset $D$ of size $N$ you want to train $M$ learners. Each one will have its own weight array $w_m$ of size $N$ (one value for each data point)
2. Initialize all the weights to $w_m = 1/N$.
3. Start from the first learner until the last one $M$ and do:
4. Train the learner minimizing the weighted error function :

$$J_m = \sum_n w_{m,n} I(e)$$

   Where $e = y_m(x_n) \neq t_n \in \{0, 1\}$ is the correctness of the classification

5. Evaluate the error for this classifier as:

$$\eta_m = \frac{J_m}{w_m}, \quad \alpha_m = ln[\frac{1 - \eta_m}{\eta_m}]$$

   Where $\alpha_m$ will be used later to estimate the grade of correctness

6. Generate the weight for the next learner $m + 1$ as:

$$w_{m+1,n} = w_{m,n} \cdot \exp[\alpha_m I(e)]$$

As can be seen the next weight $w_{m+1,n}$ associate with a specific datapoint $x_n$ will be updated by $\alpha_m$ iff the classification is incorrect $I(e) = 0 \; if \; y_m(x_n) = t_n$.

## 1.8 MDP and RL

In a dynamic system representation we have:

- A set of states $X$
- A set of actions $A$
- a transition function $\delta$ which can be
  - deterministic $\delta : X \times A \to X$
  - non deterministic $\delta : X \times A \to 2^X$
  - stochastic $\delta : P(x_{t+1}|x_t, a_t)$
- A reward function $r$ which can be:
  - deterministic $r : X \times A \to R$
  - non deterministic $r : X \times A \times X \to R$

**Markov decision process** A MDP follows the Markov properties which state:

- Once the current state is known, the evolution of the dynamic system does not depend on the history of states, actions and observations $x_{t+1} = \delta(x_t, a_t)$.
- The current state contains all the information needed to predict the future.
- Future states are conditionally independent of past states and past observations given the current state.
- The knowledge about the current state makes past, present and future observations statistically independent.

**Policy** A policy $\pi$ is some kind of behavior that takes as input some state $x_t$ and chooses an action $a_t$ in order to maximize the reward $r_t$.
The optimality of a reward is defined with respect to maximizing the (expected value of the) cumulative discounted reward:

$$V^\pi(x_1) = E[r_1 + \gamma r_2 + \gamma^2 r_3 + \dots]$$

Where $\gamma \in [0, 1]$ is the discount factor.

### 1.8.1 RL

Missing

## 1.9 HMM and POMDW

If the states $x$ of a markov process are non-observable, but instead we have some observation $z$ tied to them we are dealing with a Hidden Markov Model [HMM]. In this case the process is defined by:

- The **observation model**: a probability that a certain state is mapped with a specific observation $b_k(z_t) = P(z_t|x_t)$
- The **transition model**: $A_{ij} = P(x_t|x_{t-1})$
- an initial distribution $\pi_0 = P(x_0)$ which define the probability that the model starts with the state $x_0$

**Chain Rule** Since we are dealing with a model having the markov assumption (current observation only depends on past $N$ observation) we can extend this rule to apply to a series of state/obervation pair:

$$P(x_{0:T}, z_{0:T}) = \prod_{t=1}^{T} p(z_t|x_t) \prod_{t=1}^{T} p(x_t, x_{t-1})$$

**Forward Pass** So now we have a sequence of observation $z_0, z_1, \ldots, z_T = [z]_{t=1}^{T}$ and we would like to know how likely this sequence is $P([z]_{t=1}^{T})$.
This probability requires summing over **all possible hidden state values** at all times:

$$P([z]_{t=1}^{T}) = p(z_0, z_1, \ldots, z_T; x_0, x_1, \ldots, x_T) + p(z_0, z_1, \ldots, z_T; x_1, x_2, \ldots, x_T) + \ldots$$
$$+p(z_0, z_1, \ldots, z_T; x_{T-1}) + p(z_0, z_1, \ldots, z_T; x_T) = \sum_{n=0}^{N} p([z]_{t=1}^{T}, [x_n]_{t=1}^{T}) \tag{1.4}$$

Considering the first term we can apply the chain rule to get :

$$p(z_0, z_1, \ldots, z_T; x_0, x_1, \ldots, x_T) = P(z_{0:T}; x_{0:T}) = \prod_{t=1}^{T} p(z_t|x_t) \prod_{t=1}^{T} p(x_t, x_{t-1})$$

So the previous equation becomes:

$$P([z]_{t=1}^{T}) = \sum_{n=0}^{N} p([z]_{t=1}^{T}, [x_n]_{t=1}^{T} = \sum_{n=0}^{N} P(x_0) \prod_{t=1}^{T} p(z_t|x_t) \prod_{t=1}^{T} p(x_t, x_{t-1})$$

Which is a lot of calculation to do!

But if we set $p([z]_{t=1}^{T}, x = k) = \alpha_T^k$ we can rewrite the above equation as:

$$P([z]_{t=1}^{T}) = \sum_{k} \alpha_T^k$$

Which gives the forward step as:

1. Initialize $\alpha_1^k = P(z_1|x_1 = k)p(x_1 = k)$ for all k

2. for $t = 2, \ldots, T$ estimate

$$a_t^k = p(z_t|x_t = k) \sum_i \alpha_{t-1}^i p(x_t = k|x_{t=1} = i)$$

3. terminate when

$$P([z]_{t=1}^T) = \sum_k \alpha_t^k$$

**Backward step** Now we want to find the probability that the hidden state $x$ at time $t$ was equal to $k$:

$$P(x_t = k|[z]_{t=1}^T) = P(z_1, \ldots, z_t, x_t = k)P(z_{t+1}, \ldots, z_T|x_t = K) = \alpha_t^k \beta_t^k$$

So now we compute the backward algo with:

1. initialize $\beta_T^k = 1$ for all $k$
2. for $t = T - 1, \ldots, t$ do:

$$\beta_t^k = \sum_i P(x_{t+1} = 1|x_t = k)P(z_{t+1}|z_{t+1} = i)\beta_{t+1}^i$$

3. ternimate at:

$$P(x_t = k, [z]_{t=1}^T) = \alpha_t^k \beta_t^k$$

### 1.9.1 POMDP

In a Partially Obervable MDP we cannot see the states, so we can use a belief $b(x)$ which is a probability distribution over the states.
Given a current belief state $b$, an action $a$ and an observation $z'$, we want to compute the next belief state $b'(x')$:

$$b'(x') = P(x'|b, a, z') = \frac{P(z'|x'b, a)P(x'|b, a)}{P(z'|b, a)} = \frac{P(z'|x', a)\sum_{x \in X} P(x'|b, a, x)P(x|b, a)}{P(z'|b, a)}$$

## 1.10 Non-deterministic RL

When the problem is non deterministic we know that the transition function $\delta(x, a)$ is some probability related to the current state and action $P(x_{i+1}|x_i, a_i)$.

To define our value function, we must take the expected values:

$$V^\pi(x) \equiv E(r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} + \dots)$$

When we consider the reward function, $r(x, a)$, to be non deterministic too, then we must include it in the value function which we now call $Q$:

$$Q(x,a) = E[r(x,a)+\gamma V^*(\delta(x,a))] = \cdots = E[r(x,a)]+\gamma \sum_{x_{i+1}} P(x_{i+1}|x_i,a)\cdot \max_{a_{i+1}} Q(x_{i+1}, a_{i+1})$$

Where:

- $E[r(x, a)]$ is the expected reward
- $P(x_{i+1}|x_i, a)$ is the probability introduced above with the transition function
- $\max_{a_{i+1}} Q(x_{i+1}, a_{i+1})$ is finding the action which maximizes the future rewards

Where now the optimal policy shift to:

$$\arg\max_{a \in A} Q(x, a)$$

### 1.10.1 K-Armed bandit problem

The classic (stocastic) version of the k-armed bandit problem sees k slot machines, each one having some gaussian distribution of winning $N(\mu_i, \sigma_i)$ and the goal is to earn the most money.
In this context a RL agent can either:

- Perform x trials on each machine, estimate the mean winning rate and then choose the one with the highest.
- Adopt an $\epsilon$-greedy strategy in which they play at random with prob $\epsilon$ and choose the optimal policy with $1 - \epsilon$

In this case the training rule would be:

$$Q_n(a_i) \leftarrow Q_{n-1}(a_i) + \alpha[r_{t+1}(a_i) - Q_{n-1}(a_i)]$$

Where

- $\alpha = 1/v_{n-1}(x_i, a_i)$
- $v_{n-1}(x_i, a_i)$ is the number of execution of action $a_i$ in state $x_i$, up to time $n - 1$

**Non-deterministic case** But if the probability $\mu_i$ changes during the algorithm then the above solution would not work:

- the first would fail because $\mu_i$ can change after the $x$ trials
- the $\epsilon$-greedy strategy will not reach an optimal value

## 1.11 Dimensionality Reduction

**Latent Variable** Usually, in a dataset with high dimensional data, the difference between the samples can be related to a distortion regarding fewer dimensions than the ones making up the sample itself. For example, in a 2D image undergoing a one degree of freedom rotation the latent variable would be $1 < 2$.

### 1.11.1 PCA

Check out this video for a detailed explanation of PCA.

**Steps** The main goal of PCA is to find a line $u$ which best separates the projected points $x$ of the dataset. To find this line we need to :

1. Find the mean value of all the points $x = \hat{x} = \frac{1}{N}\sum_n x_n$
2. Estimate the total variance as $S = \frac{1}{N}\sum_n (x_n - \hat{x})(x_n - \hat{x})^T$
3. Maximize the projected variance $\max_u [u^T S u]$

**Solution** To find $u$ we derivate wrt $u$:

$$Su = \lambda u \rightarrow u^T S u = \lambda$$

Which means that $u$ must be the eigenvector of $S$ since $\lambda$ is its eigenvalue.
So having the matrix $S$ the easiest thing is to find the eigenvector associated with the largest eigenvalue.

**Error Minimization** With the above solution we can rewrite some sample $x$ of dimension $D$ as $\tilde{x}$ of dimension $\tilde{D} < D$. This will inevitably lead to some error which we can estimate with:

$$J = \frac{1}{N}\sum_n ||x_n - \tilde{x}_n||^2 = \sum_{i=\tilde{D}+1}^{D} u_i^T S u_i = \sum_{i=\tilde{D}+1}^{D} \lambda_i$$

So the error is the sum of all the discarded eigenvalues $\lambda_i$ after $\tilde{D}$.

### 1.11.2 Linear Latent Variable Model

How to estimate PCA efficently?

**Distributions** For this we assume that:

- The relationship between $x$ and its lower dimensional representation $\tilde{x}$ is linear and can be written as:

$$x = W\tilde{x} + \mu$$

- $\tilde{x}$ has a gaussian distribution of the form $P(\tilde{x}) = N(z, 0, I)$

- The relationship between $x$ and $\tilde{x}$ is linear in the Gaussian space:

$$P(x|\tilde{x}) = N(x, W\tilde{x} + \mu, \sigma^2 I)$$

Then we have:

- Marginal Distribution $P(x) = N(x, \mu, C)$
- Posterior Distribution $P(\tilde{x}|X) = N(\tilde{x}, M^{-1}W^T(x - \mu), \sigma^2 M)$
- $C = WW^T + \sigma^2 I$
- $M = W^T W + \sigma^2 I$

**Maximum Likelihood** Since we have $P(x)$ and $P(\tilde{x}|X)$ we can use ML to estimate $W$ which depends on the $u$ as:

$$\arg\max_{W,\mu,\sigma}[lnP(X|W, \mu, \sigma^2)]$$

### 1.11.3 Autoencoders

These are NN with bottolnecks in the middle which learn to reconstruct their input by minimizing a sum-of-squares error.

**Generative** This kind of autoencoders focus on leargning the latent space structure.
VAEs are an example which use an encoder to produce a distribution and a decoder to produce sample from such distribution.
Since the sampling operation is not differentiable they use a reparametrization

**Generative** This kind focus on learning the distribution.
GANs are an example which are basically an inverted CNN trained in an adversarial way

## 1.12 Kernel

Kernels are just a type of similarity measures which can be applied in any circumstance. Since they are a similarity measure they take as input two variables and return a value:

$$k(x_1, x_2) \in R$$

Typically they are simmetric $k(x_1, x_2) = k(x_2, x_1)$ and non negative.

**Gram Matrix** Given a linear model of the form $y(x, w) = w^t x$ we know that the optimal solution is given by :

$$w* = X^T(XX^T + \lambda I)^{-1}t$$

Where $X, t$ are the vector of samples and outputs.
By setting $\alpha = (XX^T + \lambda I)^{-1}t$ we get:

$$w* = X^T\alpha = \sum_n \alpha_n x_n$$

So the linear model becomes:

$$y(x, w) = w*^T x = \sum_n \alpha_n x_n^T x$$

As can be seen the last equation has two inputs $x_n, x$ so we can use a kernel $k(x_1, x_2) = x_1^T x_2$ to get the same result:

$$y(x, w) = w*^T x = \sum_n \alpha_n k(x_n, x)$$

If we now look into $\alpha$ we see that the term $XX^T$ (which we will call $K$) can be also kernelized as:

$$K = \begin{bmatrix} k(x_1, x_1) & \dots & k(x_1, x_N) \\ \vdots & \ddots & \vdots \\ k(x_N, x_1) & \dots & k(x_N, x_N) \end{bmatrix}$$

Which is called the Gram matrix.

**Kernel Trick** The thing we did above, substituting the inner producct $X^T X$ with the kernel $k(x^T, x)$ is called the kernel trick and can be applied to any $x$. But why is it useful?
Having a linear model of the kind :

$$y(x, w) = w*^T x = \sum_n \alpha_n x_n^T x$$

Shows that we actually do not need to know each datapoint but just its inner product $x^T x$. So if our dataset is linearly separable in a higher dimension we do not need to compute the projection of each point $x_i$ into that higher dimension, but we just need to compute the inner product in that dimension. So we can use the kernel trick to get that product and ease the computation.

## 1.13 Artificial Neural Network

### 1.13.1 FeedForward NN

There are no loops in this NN, the final function is a composition of multiple functions $g$ and parameters $\theta$ (one for each layer $m$):

$$f(X, \theta) = g_m(g_{m-1}(\dots(g_1(x, \theta_1)\dots)\theta_{m-1})\theta_m)$$

These many functions can tackle non-convex problems contrary to linear or kernel methods.

**Depth** The depth of a FNN is given by the number of hidden layers. With enough hidden layer you can approximate any Borel measurable function.

**Width** Is the number of units (neurons) in each layer, with enough units you can approximate any function but having a deeper FNN is easier to train.

**Number of Parameter** The number of parameters is estimated in between layers, so having a laryer $i$ with $n$ neurons and a layer $j$ with $m$ neurons the number of parameters are estimates as:

$$|\delta_{ij} = (n + 1)m$$

Where the 1 term is introduced for the bias.

**Output activation Functions** These functions are tied to the output of the FNN, so they are dependent of the problem at hand and shape the cost function. Since they come after the hidden unit output $h_i$, they will show such parameter in their input.
They change together with the problem requirement:

- Regression: we use the identity function: $y = W^t h + b$
- Binary classification: sigmoid is used: $y = \sigma(w^t h + b)$
- Multi-class classification= softmax: $y_i = softmax(\alpha)_i = \frac{e_i^\alpha}{\sum_j e_j^\alpha}$

**Loss Functions** The loss function is the cost function used for training. The value of such functions is simply the natural logarithm $ln$ of the output activation function.
Recall the ML in which we wanted the class which maximized $P(C_i|x, D)$, if we use the same principle here we get the **cross-entropy** loss function:

$$J(\theta) = -ln(P(t|x, \theta))$$

Which changes depending on the problem at hand, In the following cases $\alpha = w^t h + b$

- Regression: then $P(t|x) = N(t|y, \beta^{-1}) \rightarrow J(\theta) = (t - f(x, \theta))^2/2$ so the cost function becomes the mean square error.

- Binary classification: $J(\theta) = softplus((1-2t)\alpha)$ becomes a Bernoulli Distribution
- Multi-class classification= $J(\theta)_1 = -ln\ softmax(\alpha)_1$ becomes a Multinomial distribution.

**Hidden activation functions** In this case the activation function can be diverse for each neuron.

- Rectified linear units (ReLU): $g(\alpha) = max(0, \alpha)$ which is easy to optimize but not differentiable at 0.
- Sigmoid $g(\alpha) = \sigma(\alpha)$ and hyper tan $g(\alpha) = thanh(\alpha)$, both are:
  - Easy to saturate since there is no logarithm at the output
  - slow
  - usefull for RNN and autoencoders

**Backprop** Since the error is estimated at the end of the FNN on the output-layer, but he have multiple parameters $\theta$ which need to be updated, we must propagate the error back trough the network.
For this reason we compute the gradient of the cost function with respect to each parameter using the chain rule which states that, having $z = f(g(x))$ where $y = g(x)$ (similar to a network having one hidden layer) the gradient of z with respect to x is:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

### 1.13.2 Learning Algorithms

Since the backprop is just a method to compute the gradient (not learn) there are various other algo for this purpose.

**Stocastic Gradient Descent** Using a learning rate $\eta$ these method computes the gradient on a subset (minibatch) of samples from the dataset. This gradient is computed with the backprop method and the parameters are updated with the following formula:

$$\theta_i^{(k+1)} = \theta_i^{(k)} - \eta g_i$$

Where $g$ is the value of the gradent with respect to $\theta_i$.

**SGD with momentum** To speed up the training an additional parameter $v$ can be used to increase or decrease the value of the update depending on the training iteration.

### 1.13.3 Regularization

Reduces overfitting

**Parameter norm penalties** You can add a regularizatoin term to the cost function in order to decrease the magnitude of each parameter, so no parameter saturates.

**Dataset Augmentation** Transform the dataset (image distortion, noise adding) in order to have more diverse samples.

**Early stopping** Stop iteration to avoid overfitting when train loss zeros out while test loss increases.

**Parameter sharing** Constrain subset of parameters to be the same. Decrease memory consumption, used principally in CNN to allow translation invariance.

**Dropout** Randomly remove network units with some probability.

## 1.14 CNN

In CNNs there are usually three stages:

1. Convolution: in which the dimension of the input is reduced and the most important features are extracted

2. Activation function: usually non linear, allows the method to apply to non linearly separable datasets

3. Pooling: an averaging of some sort (usually max or average) used to implement invariance to local translations. If applied with a stride $\geq 0$ then it reduces the dimension of the output.

Since the convolution is done with a kernel (which can be seen as a rectangle taking as input some pixels and outputting one) of size $k$ the parameters become $k$ instead of $m \times n$ which is the size of the network. But since the kernel requires the input to have a certain dimension (to be applied to the borders too) we need to pad the images on the borders.

**Number of Parameters** The number of parameters of a layer $i$ depends on the previous output $i-1$ and the kernel size.
With images of size $k \times v \times d$ the third dimension $d$ is the feature map (in 2D dataset it's considered to be 1), a conv layer $i$ takes as input some data with $x_i$ feature map and outputs $y_i$ feature maps.
Having a kernel of size $n \times m$ the parameters of layer $i$ are :

$$|\theta_i| = (n \times m \times x_i + 1) \times y_i$$

It is trivial to see that the parameters are much less than in the FCN where we have:

$$|\theta_i| = (k+1) \times v$$

Usually $k \gg n$ and $v \gg m$.

**Output Dimension** Here we want to know what is the dimension of the output of a conv layer. For this we need two additional parameters which are the padding $p$ and the stride $s$. The output is of dimension $w_i \times h_i \times d_i$ where $d$ is also called the feature map.
The dimensions depend on the previous ones as follows:

$$\begin{aligned} w_i &= \frac{w_{i-1} - n + 2p}{s} + 1 \\ h_i &= \frac{h_{i-1} - m + 2p}{s} + 1 \\ d &= d_i \end{aligned} \tag{1.5}$$

## 1.15 Unsupervised

This kind of learning is used when a dataset does not have any label $t$, so we need to cluster similar samples $x$ based of some other metrics.

### 1.15.1 Gaussian Mixture Models

One way is to assume the data is made of a mixture (sum) of gaussians , so we let the model maximize the probability of $K$ gaussians.
This model will have three parameters for each $k$:

- a mean $\mu$
- a co/variance $\Sigma$
- a mixing probability $\pi$ that defines how big or small the Gaussian function will be.

Obviously the total mixing probability should sum up to 1 $\sum_k \pi_k = 1$.
Normally, for one Gaussian, we would use ML and differentiate on $\mu, \Sigma$ to get the optimal weight value, but here we are dealing with multiple Gaussians which may render this approach unfeasible.

**Latent variable** So now, for each gaussian $k$, we assocaite a (latent) variable $z_k$ to it and say that if a datapoint $x_n$ is coming from that specific gaussian then $z_k = 1$. So the probability that a data point $x_n$ comes from Gaussian $k$ is:

$$P(z_{n,k} = 1|x_n)$$

It is trivial to see that the bigger the gaussian is (bigger $\pi_k$) the more likely is that a datapoint $x_n$ belongs to it, so:

$$\pi_k = p(z_k = 1)$$

That is the overall probability of observing a point that comes from Gaussian $k$ is actually equivalent to the mixing coefficient for that Gaussian.
If we consider all the statistically independent (latent) variables $Z = z_1, z_2, \ldots, z_K$, the probability would be:

$$p(Z) = \prod_k \pi_k^{z_k}$$

**Joint Distributions** We still wish to find the probability that a point $x_n$ belongs to some gaussian $k$, we can do so with:

$$p(x_n|z) = \prod_k N(x_n|\mu_k, \Sigma_k)^{z_k}$$

By applying the chain rule we know that:

$$p(x_n, z) = p(x_n|z)p(z) \rightarrow p(x_n) = \sum_k p(x_n|z)p(z) = \sum_k \pi_k N(x_n|\mu_k, \Sigma_k)$$

Finally we can find the likelihood as the joint probability of all observations $x_n$, defined by:

$$p(X) = \prod_n p(x_n) = \prod_n \sum_k \pi_k N(x_n|\mu_k, \Sigma_k)$$

**Expectation Maximization** Let's define :

$$\gamma(z_k) \equiv P(z_k = 1|X) = \frac{P(z_k = 1)P(X|z_k = 1)}{P(X)} = \frac{\pi_k N(X; \mu_k, \Sigma_k)}{\sum_j^K \pi_j N(X; \mu_j, \Sigma_j)}$$

We need to determine $\mu_k, \Sigma_k, \pi_k$.
Using ML we get:

- $\mu_k = \frac{1}{N_k} \sum_n \gamma(z_{nk}) x_n$
- $\mu_k = \frac{1}{N_k} \sum_n \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T$
- $\pi_k = \frac{N}{N_k}$
- $N_k = \sum_n \gamma(z_{nk})$

The steps for the algorithm are reported in the next figure.

- Initialize $\pi_k^{(0)}, \boldsymbol{\mu}_k^{(0)}, \boldsymbol{\Sigma}_k^{(0)}$
- Repeat until termination condition $t = 0, \ldots, T$
  - **E step**

$$\gamma(z_{nk})^{(t+1)} = \frac{\pi_k^{(t)} \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Sigma}_k^{(t)})}{\sum_{j=1}^K \pi_j^{(t)} \mathcal{N}(\mathbf{x}_k; \boldsymbol{\mu}_j^{(t)}, \boldsymbol{\Sigma}_j^{(t)})}$$

  - **M step**

$$\boldsymbol{\mu}_k^{(t+1)} = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk})^{(t+1)} \mathbf{x}_n$$

$$\boldsymbol{\Sigma}_k^{(t+1)} = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk})^{(t+1)} (\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)})^T$$

$$\pi_k^{(t+1)} = \frac{N_k}{N}, \quad with \;\; N_k = \sum_{n=1}^N \gamma(z_{nk})^{(t+1)}$$

### 1.15.2 K-means

In this one you first decide the number of clusters $K$. The you start the algorithm:

1. Partition the dataset into $K$ parts at random or use some kind of initialization algorithm
2. For each partition estimate the centroid $c_i = \frac{1}{N_k} \sum_k x_k$
3. For each sample $x$, compute two distances:
   - Distance $d_i$ between $x$ and its current centroid $c_i$
   - Distance $d_j$ between the $x$ and the closest centroid $c_j$.
4. if $d_j < d_i$ assign $x$ to $d_j$
5. repeat until convergence

**Convergence** Convergence is achieved if:

- The number of clusters $K$ is finite
- $d_i$ decreases at each iteration

**Cons**

- K must be determined before hand
- Sensitive to initial condition (local optimum) when a few data available.
- Not robust to outliers. Very far data from the centroid may pull the centroid away from the real one.

# Chapter 2

# Exercises

## 2.1 Bayes

### 2.1.1 Naive Bayes Classifier

1. Describe the *Naive Bayes Classifier* and highlight the approximation made with respect to the Bayes Optimal Classifier.
2. Provide design and implementation choices for solving the following problem through *Naive Bayes Classifier*:

   *Classification of scientific papers in categories according to their main subject. The categories to be considered are: ML (Machine Learning), KR (Knowledge Representation), PL (Planning). Data available for each scientific paper are: title, authors, abstract and publication site (name of the journal and/or of the conference).*

**Naive description and difference** So the optimal one is equal to:

$$c_{ob} = P(c_j|x, D) = \sum_i P(c_j|x, h_i)P(h_i|D)$$

Which requires to estimate all the hypos probabilities.
So this is where the naive classifiers comes into pay saying that the sample $x$ can be seen as a set of attributes $A = \{a_1, a_2, \ldots, a_n\}$ which are **independent** from each other.
It does so using the bayes theorem in the following way:

$$P(c_j|x, D) = P(c_j|A, D) = \frac{P(A|c_j, D)P(c_j|D)}{P(A|D)} = P(A|c_j, D)P(c_j|D) = P(c_j|D)\prod_i P(a_i|c_j, D)$$

So now the classifiers becomes

$$C_{NB} = \arg\max_{c_i \in C} P(c_j|D)\prod_i P(a_i|c_j, D)$$

Since the $P(x|c_i, D)$ factor can be difficult to compute when there are many attributes of $x$, the naive bayes classifier assumes that each attribute is independent, so that.

**Implementation** In Multinomial Naïve Bayes each $p(c)$ is a multinomial distribution, so it is possible to solve the classification problem of documents using Multinomial Naïve Bayes because multinomial distribution works well for data which can easily be turned into counts, such as word counts in a text. Given:

- $P(c_j)$: probability of class $c_j$
- $P(w_i|c_j)$: probability of the word $w_i$ given the class $c_j$.

We must estimate $P(c_j)$ and $P(w_i|c_j)$ using multinomial distribution, then use them to classify a new document. Remove every word in the documents not appearing in the vocabulary V build in the previous phase and then use:

$$V_{NB} = \arg\max_{c_j \in C} P(c_j) \prod_{i=1}^{lenght(d)} P(w_i|c_j, D)$$

### 2.1.2 MAP, ML, optimality

In Bayesian Learning, given a data set $D$ and a hypothesis $h$, we can express the following relationship between the probability distributions (Bayes theorem):

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

In this context:

1. define *Maximum a posteriori* (MAP) hypotheses and *Maximum likelihood* (ML) hypotheses.
2. define the concept of *Bayes Optimal Classifier*
3. discuss about practical applcability of the *Bayes Optimal Classifier*

**Define MAP and ML** Both the MAP and ML try to get the hypo which maximize the probability $p(h|D)$ in following way:

$$MAP = \arg\max_{h \in H} p(h|D) = \arg\max_{h \in H} p(D|h)p(h)$$

When the prior for every hypo is the same then we can drop the $p(h)$ factor:

$$ML = \arg\max_{h \in H} p(h|D) = \arg\max_{h \in H} p(D|h)$$

**Bayes optimal classifier** Since the previous do not take into account the classification given by multiple hypos we use an optimal classifier as:

$$C_B = \arg\max_{c_i \in C} \sum_{h_j \in H} p(c_i|h_j, x)p(h_j|D)$$

That is we get the class which yields the higher sum of two elements: the probability that the element $x$ is classified as $c_i$ by $h_j$ and the probability that $h_j$ in $D$.

**Practical use** The applicability should be for simple tasks such as predicting the gender of a dataset consisting of heights and weights.

### 2.1.3 ML

1. Provide a formal definition of a maximum likelihood (ML) hypothesis
2. Comment the following statement: *in a classification problem, the class returned by the ML hypothesis on a new instance x is always the most probable class.*

**Definition** ML is derived using the bayes theorem when the prior for each hypo is the same $p(h_1) = p(h_2) = ... = p(h_n)$. The ML returns the hypo which maximizes $p(h|D)$ as follows:

$$ML = \arg\max_{h_i \in H} p(D|h)$$

**Comment** The ML returns the most probable hypo which then may or may not correctly classify the element.

### 2.1.4 Ex. Sigma

Consider a dataset containing 3D points belonging to two classes $C_1$ and $C_2$. Each set of points belonging to a class is assumed to be normally distributed, namely $P(\mathbf{x}|C_i) \simeq \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma})$ (same covariance matrix for both the classes).

1. Describe the conditional probability of class $C_1$ considering a data instance $\mathbf{x}$. (Hint: apply the Bayes rule)
2. Given $\alpha = \ln \frac{p(\mathbf{x}|C_1)P(C_1)}{p(\mathbf{x}|C_2)P(C_2)}$, and the sigmoid function $\sigma(\alpha) = \frac{1}{1+\exp(-\alpha)}$, express $P(C_1|\mathbf{x})$ in terms of $\sigma$
3. What are the model parameters? Which is the size of the model (number of independent parameters)?

**point 1** We want to estimate $p(C_1|x)$. Using the bayes rules we have:

$$p(C_1|x) = p(x|C_1)P(C_1)$$

Where $p(x|C_1) = N(x; \mu_1, \Sigma)$

**Express in terms of sigma** To solve this let's start from $\sigma(\alpha)$ and work out way back to $P(C_1|x)$. First let's use:

$$\tau = \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}$$

So we can write

$$\sigma(\alpha) = \frac{1}{1+e^{-\alpha}} = \frac{e^{\alpha}}{1+e^{\alpha}} = \frac{\tau}{1+\tau}$$

Let's focus on the denominator:

$$\tau+1 = \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}+1 = \frac{P(x|C_1)P(C_1)+P(x|C_2)P(C_2)}{P(x|C_2)P(C_2)} = \frac{1}{P(x|C_2)P(C_2)}$$

So we get:

$$\frac{\tau}{1+\tau} = \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)} \cdot P(x|C_2)P(C_2) = P(x|C_1)P(C_1)$$

So we can say that:

$$P(C_1|x) = P(x|C_1)P(C_1) = \sigma(\alpha)$$

**Model Parameters** The model parameters are defined by $N(x;\mu_1,\Sigma)$, so we have 2 means $(\mu_1,\mu_2)$ and 4 others since $\Sigma$ has dimension $2\times 2$ and it's the same for both classes. So we get a total of 6 parameters.

### 2.1.5 Secret String

A secret string $s = b_0b_1b_2b_3$ of 4 bits fulfills the following constraints:

- if $b_0 = 0$, $s$ contains an even number of 0's and 1's;
- if $b_0 = 1$, $s$ contains at least three 1's.

No other prior information about $s$ is available.

1. Define the prior probability distribution $P(s)$ of the hypothesis string $s$.
2. Assuming $b_0 = 0$, define the conditional probability distribution $P(s|b_0 = 0)$ and indicate all maximum a-posteriori hypotheses.
3. Assuming $b_0 = 1$ and $b_1 = 1$, indicate all maximum likelihood hypotheses and compute the likelihood that $b_2 = 1$.

**Prior** By counting the possible configurations we see that they are only 7:

- [0|101]
- [0|011]
- [0|110]
- [1|110]
- [1|101]
- [1|011]
- [1|111]

3 where the first bit is 0 and 4 were it's 1.
So our prior will be

$$P(s) = P(b_0,b_1,b_2,b_3) = \frac{1}{7} = 0.142$$

**Conditional prob** First thing let us call $b_0 = 0$ as $B_0$, we can apply the bayes theorem to get:

$$P(s|B_0) = P(B_0|s)P(s)$$

Now it is trivial to see that the probability of having $B_0$ with $s$ is 3 out of 7. So the value of $P(s|B_0)$ becomes:

$$P(s|B_0) = \frac{3}{7} \cdot \frac{1}{7} = 0.06122$$

The hypotesis are the one involving $B_0$, so :

- $h_1 : [0|101]$
- $h_2 : [0|011]$
- $h_3 : [0|110]$

**Likelihood** Out of the 4 possibilities we had for $b_0 = 1$, now we have just 3:

- $h_1 : [1|110]$
- $h_2 : [1|101]$
- $h_3 : [1|111]$

In this, there are only 2 in which $b_2 = 1$, so

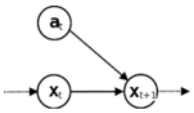
$$P(b_0 = b_1 = b_2 = 1|s) = \frac{2}{3}$$

## 2.2 MDP, RL

### 2.2.1 Definitions

Describe the Markov property of Markovian models representing dynamic systems. Describe the difference between a Markov Decision Process (MDP) and a Hidden Markov Model (HMM). Draw and explain the graphical models of MDP and HMM.

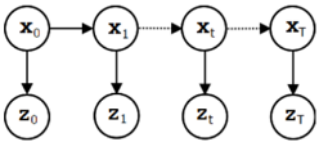
**Markov properties** once the current state is known, the evolution of the dynamic system does not depend on the history of states, actions and observations. The current state contains all the information needed to predict the future. Future states are conditionally independent of past states and past observations given the current state. The knowledge about the current state makes past, present and future observations statistically independent. Markov process has Markov properties.




**MDP** An MDP is for decision making where the states are fully observable, hence there is no need of observations. In presence of non-deterministic or stochastic actions, they can be fully observed after its execution.
Having

- $X$ finite set of states:
- $A$: finite set of actions
- Deterministic $\delta : X \times A \to X$: a function which maps an action-state tuple to the next state.
- Non-deterministic $\delta : X \times A \to 2^X$.
- Stochastic $\delta : P(x'|x, a)$: the probability of having he state $x'$ given previous state $x$ and action $a$.
- Deterministic $r : X \times A \to R$: a function which maps a tuple state-action to a reward.
- Non-deterministic/Stochastic $r : X \times A \times X \to R$

An MDP is made of $MSP = \{X, A, \delta, r\}$




**HMM** In a HMM sate states $x_t$ are discrete and not observable, while the observations $z_t$ can be either discrete or continuous. Finally there is no control over the system.
Given:

- $X$: set of not observable states
- $Z$: set of observations
- $P(x_t|x_{t-1})$: the transition model
- $P(z_t|x_t)$: an observation models which maps the probability of having an observation $z_t$ to a state $x_t$.
- $\pi_0$: an initial distribution
- $\boldsymbol{A} = \{A_{ij}\} = P(x_t = j|x_{t-1} = i)$ a transition matrix
- $b_k(z_t) = P(z_t|x_t = k)$ an observation model.
- $\pi_0 = P(x_0)$: an initial probability.

We have that a HMM is made of $HMM = \{X, Z, \pi_0\}$.

### 2.2.2 Ex. Rome river

A car driver in Rome has to move from one side of the Tiber river to the other very often every day. There are three possible alternative paths passing to three different bridges and the paths are known. The driver wants to minimize the time to reach the target location, but due to traffic conditions, it is not guaranteed that the shortest path is also the quickest way. Moreover, traffic conditions are unpredictable, fully observable and (quasi-)stationary.

1. Describe a complete model for this problem based on MDP, specifying all its elements.
2. Describe how to solve the problem based on Reinforcement Learning and determine the exact training rule to use to learn the best behavior.
3. Discuss the strategy for balancing exploration and exploitation.

**Complete model** An MDP has the four components:

- Set of states $X = \{x_1, x_2\}$: Since the action is crossing the bridge, the states indicate on which part of the bridge the car currently is.
- Set of actions $A = \{b_1, b_2, b_3\}$: contains the three possible bridges to choose.
- A deterministic transition function $\gamma$: which is independent of the action since the car will always cross the bridge to get to the other side
- A reward function $r : X \times A$, which takes as input a state and an action and returns the time.

### 2.2.3 Definition and algo

1. Provide a formal definition of the Reinforcement Learning (RL) problem. Describe formally what are the inputs and the outputs of a RL algorithm.
2. Describe the main steps of a RL algorithm. Provide an abstract pseudo-code of a generic algorithm for RL (e.g., Q-learning).

**Definition** A RL problem includes an agent accomplishing a task according to an MDP $(X, A, \gamma, r)$, for which functions $\gamma$ and $r$ are unknown to the agent. The aim is to find an optimal policy $\pi$ which maximizes the a value function $V$

**Main Steps** Givan an MDP the agent start executing random actions and saves the triple (state, action, reward) on a table. As the training continues the agent is able to map the current state to a saved triple, thus choosing the action which maximizes the reward.

### 2.2.4 K-armed bandit

1. Describe the k-armed bandit problem (also known as One-state MDP).
2. Describe the Reinforcement Learning procedure to compute the optimal policy in the k-armed bandit problem with stochastic behavior and unknown functions.

**Description** The k-armed bandit problem is a problem highlighting the trade-off between exploration and exploitation.
There are K slot machines each one with a probability distribution, and the goal is for a player to maximize the total reward.

**Optimal policy** An optimal policy is given by the Upper-Confidence-Bound:

$$A = \arg\max_a [Q_t(a) + c\sqrt{\frac{2ln(t)}{N_t(a)}}]$$

Where:

- $Q_t(a)$ is the reward associated with pulling the arm $a$, which can also been simply the mean reward given by $a$.
- $c$: coefficient to control tradeoff
- $t$: current iteration
- $N_t(a)$: times arm $a$ was pulled

## 2.3 Linear Regression, Classification

### 2.3.1 Sequential vs Batch

1. Briefly describe the goal of linear regression and define the corresponding model.
2. Given a dataset $\mathcal{D} = \{(\mathbf{x}_1^T, t_1)^T, \ldots, (\mathbf{x}_N^T, t_N)^T\}$ with $\mathbf{x}_n$ the input values and $t_n$ the corresponding target values, explain how the parameters of the model can be estimated either in a batch or in a sequential mode.

**Linear regression** The goal of regression is to predict the value of one or more continuous target variables $Y = \mathcal{R}$ given the value of a D-dimensional vector of input variables $X \subset \mathcal{R}^D$. The simplest linear model for regression is one that involves a linear combination of the input variables:

$$y(x, w) = w_0 + w_1 x_1 + ... + w_D x_D = \boldsymbol{w^T x}$$

Where $x_0 = 1$.

**Batch vs sequential** We have that a target value is given by:

$$t = y(x, w) + \epsilon$$

Where $\epsilon$ is some Gaussian noise. If we assume the samples to be i.i.d. then we can use the maximum likelihood and have:

$$max[P(\{t_1, ..., t_n\}|x_1, ..., x_n, w, \beta)]$$

where $\beta^{-1}$ is the variance of the noise. This approach is used with batch learning in which the entire training set is processed altogether.

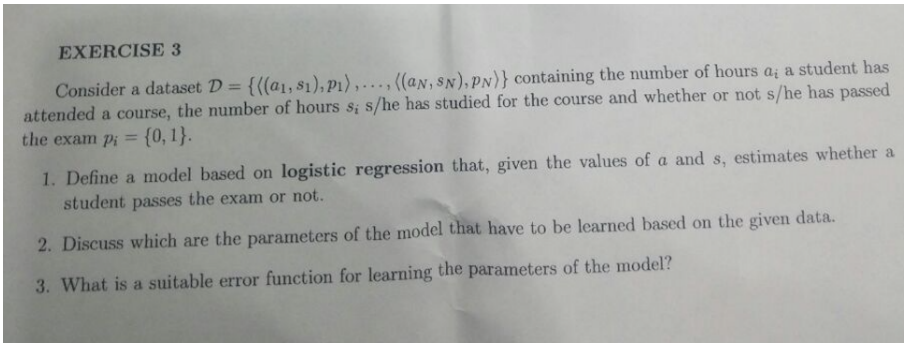
For sequential learning *stochastic gradient descent* can be used to update the model parameters in the following way:

$$w_{n+1} = w_n - \eta \nabla E_n = w_n + [t_n - w_n^T \phi(x_n)]\phi(x_n)$$

Where:

- $\eta$: is the learning rate
- $\phi(x_n)$ : is a non linear transformation of the input vector

### 2.3.2 Logistic Regression



**EXERCISE 3**

Consider a dataset $\mathcal{D} = \{\langle(a_1, s_1), p_1\rangle, \ldots, \langle(a_N, s_N), p_N\rangle\}$ containing the number of hours $a_i$ a student has attended a course, the number of hours $s_i$ s/he has studied for the course and whether or not s/he has passed the exam $p_i = \{0, 1\}$.

1. Define a model based on **logistic regression** that, given the values of $a$ and $s$, estimates whether a student passes the exam or not.
2. Discuss which are the parameters of the model that have to be learned based on the given data.
3. What is a suitable error function for learning the parameters of the model?

**Definition** The Logistic regression is a classification method based on maximum likelihood. The likelihood function is given by:

$$p(t|w) = \prod_n y_n^{t_n}(1 - y_n)^{1-t_n}$$

Where $y_n$ is the predicted class for the n element, that is $\sigma(w^t x_n)$. The prediction is estimated using the non linearity $\sigma(w^t x_n)$.

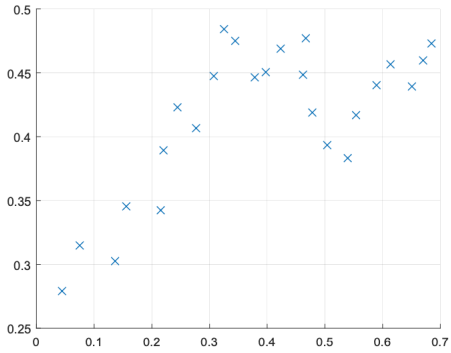
**Parameters** The parameters to be learn are just 3: 2 for the attributes + the bias term.

**Error Function** The cross-entropy error function of the kind:

$$E(w) = -ln(p(t|w))$$

### 2.3.3 Ex. Plotted points

Consider the learning problem of estimating the function $f : \Re \mapsto \Re$ with dataset $D = \{(x_i, y_i)\}$ plotted in the figure below:




1. Describe how to perform regression based on these data using a method of your choice. Specifically, provide a mathematical formulation of the model, highlighting the model parameters.
2. Considering the method you have chosen describe a way to reduce overfitting.
3. Draw a plausible plot of the learned model based on your choices.

**Description** To perform such regression a linear model with a polynomial curve fitting can be used of the form:

$$y = \sum_i w_i x^i$$

In such case the model should have $i \geq 3$, that is $W = w_0, w_1, w_2, w_3$.
The training algorithm can be a simple SDG.

**Overfitting** To reduce overfitting a regularization term can be introduced during the model updates .

### 2.3.4 Classification

Briefly describe a linear classification method and discuss its performance in presence of outliers. Use a graphical example to illustrate the concept.

The goal in classification is to take an input vector $x$ and to assign it to one of $K$ discrete classes $C_k$ where $k = 1, ..., K$. In the most common scenario, the classes are taken to be disjoint, so that each input is assigned to one and only one class. Data sets whose classes can be separated exactly by linear decision surfaces are said to be linearly separable. There are various ways of using target values to represent class labels.
For probabilistic models, the most convenient, in the case of two-class problems, is the binary representation in which there is a single target variable $t \in \{0, 1\}$

such that $t = 1$ represents class $C_1$ and $t = 0$ represents class $C_2$. The value of $t$ can be viewed as representing the probability of belonging to the class $C_1$.
For $K > 2$ it is convenient to use a 1-of-K coding scheme, which is a vector $T = \{t_1, t_2, ..., t_n\}$ of length $K$ such that if the class is $C_j$, for some sample $x_i \in X,\ |X| = n$ , then the value of $t_i$ is zero everywhere except for $t_i^j$ which is one.
Given a dataset in the form $D = \{(x_n, t_n)_{n=1}^N\}$ we want to find a linear discriminant $y(x) = W^T x$, hence we minimize the sum of squares error function:

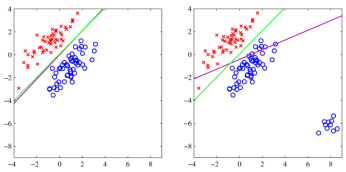
$$E(W) = \frac{1}{2} Tr\{(XW - T)^T (XW - T)\}$$

obtaining:

$$W = (X^T X)^{-1} X^T T$$

$$y(x) = W^T x$$

Unfortunately, least squares solutions lack robustness to outliers and are highly sensitive to them, unlike logistic regression.



The left plot shows data from two classes (red crosses and blue circles) with the decision boundary found by least squares (magenta) and by the logistic regression model (green). The right plot shows the corresponding results obtained when extra data points (outliers) are added at the bottom left of the diagram.

### 2.3.5 Ex. Variable length string

Consider a setting where the input space $I$ is the set of finite strings over the characters $a, b, c, \cdots, z$. Notice that input strings can be of different length.

Given the following dataset $D$:

| **x** | **t** |
|---|---|
| $a$ | 1 |
| $ab$ | 1 |
| $caza$ | 4 |
| $ayka$ | 4 |
| $aabba$ | 9 |
| $aaa$ | 9 |
| $zazaa$ | 9 |
| $accaaca$ | 16 |
| $khaaala$ | 16 |
| $akdfkkatyuakka$ | 16 |
| $jhxaaksrtkaatyuap$ | 25 |

1. Identify the learning problem at hand, in particular the form of the target function, and define a suitable linear model for it.
2. Apply the kernel trick to the model defined above and provide the analytical form of the corresponding error function.
3. Define the solution obtained with your choices for the dataset $D$.

**Definition "Cheat"** If you look closely you can see that $t$ is equal to the number of $a$s in the string squared.
Since we have to keep a linear model of the kind:

$$y = W^T \phi(X)$$

We can define :

$$\phi(x) = \sum_n I(x_n = a)^2$$

To be a function which counts the number of $a$s in the string and squares the result.
While $W = 1$ is a matrix of all ones.

**Definition Classical** Since the previous one is "too easy" and we want to stick to the theoretical approach we can define a k-hot-vector representation $V \in Z_+^{26}$ of the characters were we associate to an index ($a = 1, b = 2, \ldots, z = 26$) the number of character in the string.
Since we want the square of the number of repetition we can simply multiply $V$ for itself to obtain the model:

$$y = \sum_n (w_n^T v_n) v_n = (W^T V^T) V$$

Which can become linear if we use the function $\phi(x) = x^T x$, so we get:

$$y = W^T \phi(V)$$

**Kernel trick** Since we love overkilling this problem, we can even use the kernel trick instead of $\phi(x)$ so to have:

$$y = W_n^T k(V^T, V) = \sum_n \alpha_n k(x_n, x), \quad k(x, y) = x \cdot y$$

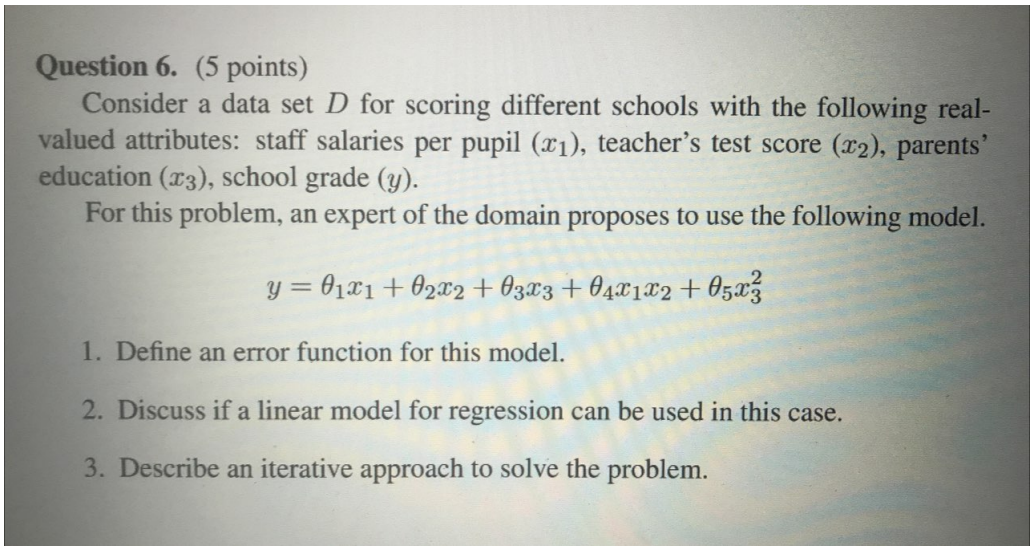
The solution to this stuff will be the same as for every linear kernel:

$$\alpha = (K + \lambda I)^{-1} t$$

Where $K$ is the gram matrix where $k_{i,j}(x, y) = x \cdot y$

**Solution** If we use the previous result and train the model it will just result in $W = [1, 0, 0, \ldots, 0]$, that is the only repetition that matters is a.

### 2.3.6 Ex. school scoring



**Question 6.** (5 points)

Consider a data set $D$ for scoring different schools with the following real-valued attributes: staff salaries per pupil ($x_1$), teacher's test score ($x_2$), parents' education ($x_3$), school grade ($y$).

For this problem, an expert of the domain proposes to use the following model.

$$y = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_1 x_2 + \theta_5 x_3^2$$

1. Define an error function for this model.
2. Discuss if a linear model for regression can be used in this case.
3. Describe an iterative approach to solve the problem.

**Error Function** We can define the error function to be the canonical one for a regression problem , that is least square:

$$E = \min \frac{1}{2} \sum_n (t_n - w^T x_n)^2$$

**Linearity** Out goal is to obtain a linearity in the weight space, that is we want each weight $w_n$ to be multiply with a linear attribute. This is achievable introducing a the functions

- $\phi_4(x, y) = x \times y$
- $\phi_5(x) = x^2$

This vector can be now introduced in the model as:

$$y(X) = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 \phi_4(x_1, x_2) + \theta_5 \phi_5(x_3)$$

**Iterative algo** An iterative approach can be the SGD, where you update the weights value with:

$$w_i \leftarrow w_{i-1} - \eta \nabla E = w_{i-1} - \eta[t_n - w^T \phi(x)]\phi(x)$$

## 2.4 K-NN

### 2.4.1 Algo and example

1. Provide the main steps of classification based on K-nearest neighbors (K-NN).
2. Draw an example in 2D demonstrating the application of the 3-NN algorithm for the classification of 3 points given a dataset consisting of points from 4 different classes.

Notes: You can choose how the points of the 4 classes are distributed. Use a different symbol for each class (e.g. use (*,x,+,-) for the classes and (o) for the points to be classified).

**Main steps** Classification with K-NN, having a target function $f : X \to C$ and a dataset $D = \{(x_i, y_i)_{i=1}^N\}$ is :

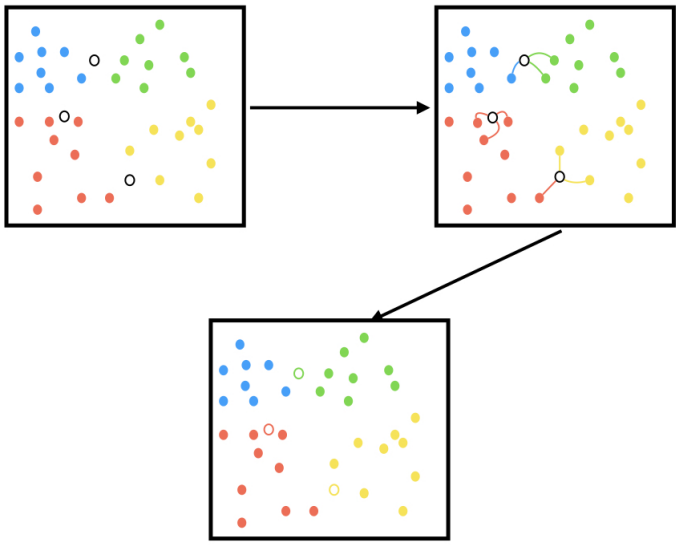
- Find K nearest neighbors of the new instance $x'$
- assign $x'$ to the most common label among the majority of the neighbors.

We can estimate the likelihood of $x'$ belonging to the class $c$ as:

$$p(c|x', D, K) = \frac{1}{K} \sum_{i \in N_k(x', D)} \mathcal{I}(y_i = c)$$

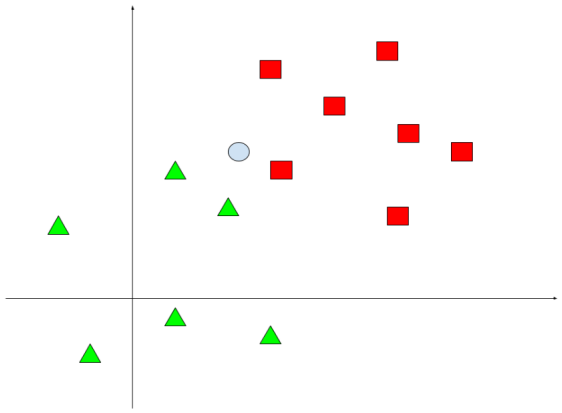
with:

- $N_k(x', D)$ is the set of K nearest points to $x'$
- $\mathcal{I}(y_i = c)$ is one if the label $y_i$ is equal to $c$, zero otherwise



### 2.4.2 Algo and exercise

1. Describe the K-nearest neighbors (K-NN) algorithm for classification.
2. Given the dataset below for the two classes $\{square, triangle\}$, determine the answers of K-NN for the query point indicated with symbol o for K=1, K=3, and K=5. Motivate your answer, showing (with a graphical drawing) which instances contribute to the solution.



**Description** The K-NN algorithm is used in classification problems as follows:

- Set a number of neighbors K
- Select an element $x$ and count the closer $k$ neighbors.

- set $x$ to be the most common class among the neighbors.

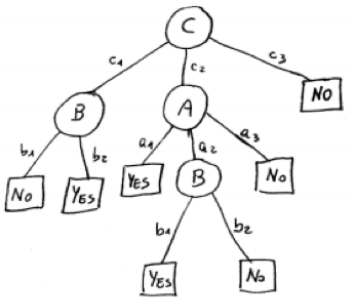
**Exercise** Again, you just need to check which points are closer to the circle.

- K=1 : the square is the closer, since we count just the first neighbors we can set the class to be a square.
- K=3: the closer 3 elements are 2 triangles and 1 square, so the class is triangle.
- K=5: just count, 3 squares and 2 triangles, so square.

## 2.5 Tree

### 2.5.1 Tree classification

Given a classification problem for the function $f : A \times B \times C \to \{+, -\}$, with $A = \{a_1, a_2, a_3\}, B = \{b_1, b_2\}, C = \{c_1, c_2, c_3\}$ and the following decision tree $T$ that is the result of a learning algorithm on a given data set:




1. Provide a rule based representation of the tree $T$.
2. Determine if the tree $T$ is consistent with the following set of samples $S \equiv \{s_1 = \langle a_1, b_1, c_1, No\rangle, s_2 = \langle a_2, b_1, c_2, Yes\rangle, s_3 = \langle a_1, b_2, c_3, Yes\rangle, s_4 = \langle a_2, b_2, c_2, Yes\rangle\}$. Show all the passages needed to get to the answer.

**Rules**

1. $c_1 \wedge b_1 \Rightarrow No$
2. $c_1 \wedge b_2 \Rightarrow Yes$
3. $c_2 \wedge a_1 \Rightarrow Yes$
4. $c_2 \wedge a_2 \wedge b_1 \Rightarrow Yes$
5. $c_2 \wedge a_2 \wedge b_2 \Rightarrow No$
6. $c_2 \wedge a_3 \Rightarrow No$
7. $c_3 \Rightarrow No$

**Consistency**

- $s_1$ is consistent because of 1
- $s_2$ is consistent because of 4
- $s_3$ is not consistent because of 7
- $s_4$ is not consistent because of 5

### 2.5.2 Tree overfitting

1. Provide a formal definition of overfitting.
2. Discuss the problem of overfitting in learning with Decision Trees and illustrate possible solutions to it.

**Definition** The definition of overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably

**Overfitting in Trees** To avoid overfitting we can stop growing the tree when data split are not statistically significant or grow a full tree, then post-prune

### 2.5.3 Tree exercise

---

The following data have been collected and we want to learn the general concept *Acceptable*, by using Decision Tree Learning.

| House | Furniture | Nr rooms | New kitchen | Acceptable |
|---|---|---|---|---|
| 1 | No | 3 | Yes | Yes |
| 2 | Yes | 3 | No | No |
| 3 | No | 4 | No | Yes |
| 4 | No | 3 | No | No |
| 5 | Yes | 4 | No | Yes |

1. Formalize the learning problem: decribe exactly the target function to learn and the dataset.
2. Describe qualitatively how attributes are chosen when building a Decision Tree.
3. Simulate the execution of ID3 algorithm on the data set above and generate the corresponding output tree.

Note: point 3 can be answered even if point 2 is not properly addressed, by using any invented method (or invented numbers) for the selection of the variables.

**Formalization** The dataset consists of 5 samples each having 3 attributes. 2 attributes are boolean while the last one (number of rooms) is categorical, even tho there are only two possible alternatives so it can be considered boolean too. The target function is of the kind $f : B^3 \to B$ where $B = \{0, 1\}$.

**Attributes** The best attribute is decided trough the minimization of entropy, that is the attribute with the max info gain is chosen.

1. Create a *Root* node for the tree
2. **if** all *Examples* are positive, **then** return the node *Root* with label $+$
3. **if** all *Examples* are negative, **then** return the node *Root* with label $-$
4. **if** *Attributes* is empty, **then** return the node *Root* with label = most common value of *Target_attribute* in *Examples*
5. Otherwise ...

**Simulation short** By observing the ID3 algorithm and the data you can easily tell how the tree is supposed to be.
By using rules 2,3,4 you can assest that:

- When there are 4 rooms the answer is always yes. So nr rooms should be the root of the node since it can discern between positive and negative.
- When there are 3 rooms having a kitchen or not is proportional to the house being acceptable. So we can use this feature as the next node and we are done since we described all the possible outcomes.

**Simulation long** Here we do not apply the immediate rules and use the minimization of entropy.

First we need to select the root node using the minimization of the entropy. There are a total of 3 positive and 2 negative in the acceptable col, so $S = [3+, 2-]$.
We first estimate the total entropy as:

$$Ent(S) = -p_+ log_2 p_+ - p_- log_2 p_- = -\frac{2}{5} log_2 \frac{2}{5} - \frac{3}{5} log_2 \frac{3}{5} = 0.97095059$$

Then we estimate the info gain for each attribute as:

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

For the furniture attribute we have :

- $S_+ = [1+, 1-]$: for each positive value of furniture how many positive/negative acceptable?
- $S_- = [2+, 1-]$: for each negative value of furniture how many positive/negative acceptable?

So now we can focus no the second term:

$$-\sum_{v \in Values(Fur)} \frac{|S_v|}{|S|} Ent(S_v) = -\frac{|S_+|}{|S|} Ent(S_+) - \frac{|S_-|}{|S|} Ent(S_-) = \\ -\frac{2}{5}(-\frac{1}{2} \cdot log_2 \frac{1}{2} - \frac{1}{2} \cdot log_2 \frac{1}{2}) - \frac{3}{5}(-\frac{2}{3} \cdot log_2 \frac{2}{3} - \frac{1}{3} \cdot log_2 \frac{1}{3}) = \\ -\frac{2}{5} \cdot 0.928 - -\frac{3}{5} \cdot 0.993 = -0.967 \tag{2.1}$$

So the furniture gain is $Gain(S, Furniture) = Ent(S) - 0.967 = 0.039$

Same thing can be done with nRooms:

- $S_3 = [1+, 2-]$.
- $S_4 = [2+, 0-]$.

$Gain(S, nRoom) = 0.0234$ And newKitchen:

- $S_+ = [2+, 0-]$.
- $S_- = [1+, 2-]$.

$Gain(S, newKitchen) = 0.0124$

The gain values are made up since It took to much time to to id properly, but the formulas are correct.

So we choose the attribute with the most info gain as our root one, Furniture. For the next attributes you just keep doing the same procedures until you come up with. From the online tool the tree is like this:


## 2.6 Evaluation

### 2.6.1 Confusion matrix

1. Provide the definition of *Confusion matrix* for a multi-class classification problem.
2. Provide a numerical example of a confusion matrix for a 3-classes classification problem with a balanced data set including 100 samples for each class. Show the confusion matrix in two formats: with absolute values and with the corresponding percentage values.
3. Compute the accuracy of the classifier for the numerical example provided above.

Hint: use simple numerical values, so that you do not need to make complex calculations.

**Definition** A confusion matrix is a table layout that allows visualization of the performance of an algorithm. Each row of the matrix represents the instances in

a predicted class while each column represents the instances in an actual class (or vice versa).

| | C1 | C2 | C3 |
|---|---|---|---|
| C1 | 60 | 15 | 15 |
| C2 | 6 | 75 | 12 |
| C3 | 6 | 21 | 90 |

Table 2.1: Absolute value

| | C1 | C2 | C3 |
|---|---|---|---|
| C1 | 20 | 5 | 5 |
| C2 | 2 | 25 | 4 |
| C3 | 2 | 21 | 30 |

Table 2.2: Percentage value

**Example** The absolute table is reported in 2.1, counting the elements you find that they sum up to 300.
The percentage table is reported in 2.2, which is the same as the absolute divided by 300.

**Accuracy** The accuracy is simply the sum of the elements on the diagonal of the percentage table. So for this example 75%

### 2.6.2 K-fold

1. Describe with pseudo-code the K-Fold Cross Validation method to estimate the accuracy of a learning algorithm $L$ on a dataset $D$.
2. Describe how the method can be extended to comparing two different learning algorithms $L_A$, $L_B$.

**PseudoCode** The pseudocode for the kfold is the following:

- Choose k, and split the dataset D into k parts
- For every part, perform the training (with some algorithm L) onto k-1 parts and evaluate on the last one
- average the error on k runs

**Comparing** You can extend the method in the following way:

- Choose k, and split the dataset D into k parts
- for every part, train both learning methods on k-1 splits, and evaluate on the last one

- define the error as the difference between the two errors, e.g. $e = e(L_A) - e(L_B)$
- average the result
- if $e < 0 \rightarrow e(L_B) > e(L_A)$ so $L_A$ is better (smaller error)

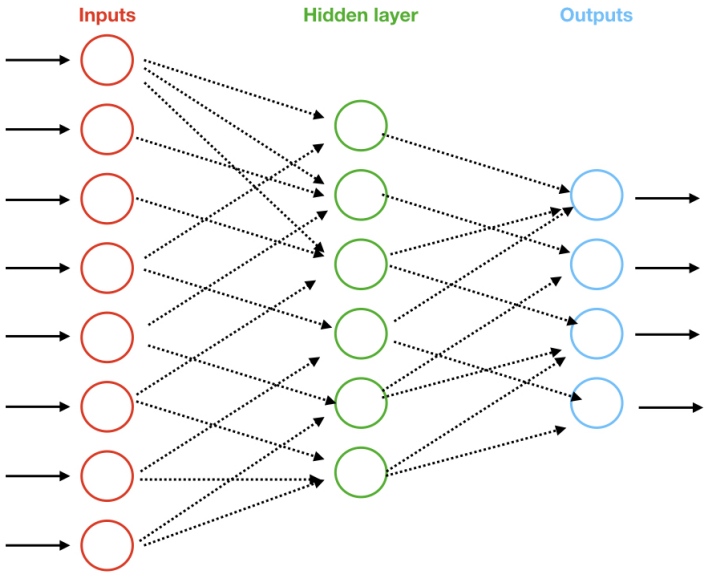
## 2.7 ANN

### 2.7.1 Size, Backprop, Overfitting

Consider a regression problem for the target function $f : \Re^8 \rightarrow \Re^4$. Design a solution based on Artificial Neural Network for this problem: draw a layout of a suitable ANN for this problem and discuss the choices.

1. Determine the size of the ANN model (i.e., the number of unknown parameters).
2. Is Backpropagation algorithm affected by local minima? If so, how can we avoid or attenuate it?
3. Is Backpropagation algorithm affected by overfitting? If so, how can we avoid or attenuate it?

**Architecture** The architecture is the following




**Size of ANN** Since the network is a fully connected with $\mathcal{R}^8 \rightarrow \mathcal{R}^6 \rightarrow \mathcal{R}^4$ the parameters in between layers $i$ and $j$ are estimated with:

$$|\theta_{i,j}| = (size_i + 1) \times size_j$$

So for our network:

$$|\theta_{1,2}| + |\theta_{2,3}| = (8 + 1) \times 6 + (6 + 1) \times 4 = 82$$

.

**Back-Propagation** Backpropagation algorithm is used to propagate gradient computation from the cost through the whole network. The goal is to compute the gradient of the cost function w.r.t. the parameters $\nabla_\theta J(\theta)$ in a simple and efficient way.

**Backward and Forward pass** The forward pass computes values from inputs to output and the backward pass then performs backpropagation which starts at the end and recursively applies the chain rule to compute the gradients all the way to the inputs of the circuit. The gradients can be thought of as flowing backwards through the circuit.

**Stochastic Gradient Descend** Stochastic Gradient Descent updates the weight parameters after evaluation of the cost function after each sample. That is, rather than summing up the cost function results for all the sample then taking the mean, SGD updates the weights after every training sample is analyzed.

**Back-Propagation algorithm**

- Requires:
  - $W^i,\ i \in \{1, ...., l\}$: weight matrices
  - $b^i,\ i \in \{1, ...., l\}$: bias matrices
  - $x$: input value
  - $t$: target value
- Forward step
  - $h^0 = x$
  - *for k in range(1,l):*
  - $a^k = b^k + W^k h^{k-1}$
  - $h^k = f(\alpha^k)$
  - *end for*
- Backward step
  - $g \leftarrow \nabla_Y J = \nabla_y L(t, y)$
  - *for k=l,l-1,...1 do*
  - $g \leftarrow \nabla_{\alpha^k} J = g \odot f'(\alpha^k)$: propagate gradient to pre-non linearity activations
  - $\nabla_{b^k} J = g$
  - $\nabla_{W^k} J = g(h^{k-1})^T$
  - $g \leftarrow \nabla_{h^{k-1}} J = (W^k)^T g$ propagate gradients to the next lower level hidden layer
  - *end for loop*

### 2.7.2 Deep FNN

Consider the problem of finding a function which describes how the salary of a person (in hundreds of euros) depens on his/her age (in years), the months in higher education and average grades in higher education. A dataset in the form $\mathcal{D} = \{(\mathbf{x}_1^T, t_1), \ldots, (\mathbf{x}_N^T, t_N)\}$ is provided, with $\mathbf{x} \in \mathbb{R}^3$ denoting the input values and $t$ the target values (salary).

Assuming that one tries to identify this function with a deep feed-forward network:

1. Explain how the problem is formalized by writing the parametric form of the function to be learned highlighting the parameters $\boldsymbol{\theta}$.
2. Explain what are suitable choices for the activation functions of the hidden and output units of the network.
3. Explain what is a suitable choice for the loss function used for training the network and write the corresponding mathematical expression.

**Formalization** It is trivial to see that we need to FNN to perform a regression. Our output will have the form:

$$y = w_{out}^T \times h + b$$

Where:

- $w_{out}$: are the weights associated with the output
- $h$ is the last hidden layer's output. It can also be seen as a concatenation of multiple functions.
- $b$ is the bias term

Since the requirements are for the FNN to be deep, but no depth is specified, we can use one hidden layer of the kind:

$$h = \phi(x^T W + c_i)$$

Where:

- $\phi$ is a non linearity of some kind (called activation function). We can take it to be $\phi(a) = \max(0, a)$
- $W$ is the weight matrix associated with the hidden unit(s).
- $c_i$ is another bias term

So the final form for the model would be:

$$y(x) = w_{out}^T \times \phi(x^T W + c_i) + b$$

**Activation functions** Since this is a regression problem the output activation function will be the identity function which is simply:

$$y = W^T h + b$$

For the hidden units we can use ReLu:

$$g(\alpha) = max(0, \alpha)$$

**Cost function** The cost function will be the ML:

$$J(\theta) = -ln(P(t|x, \theta))$$

### 2.7.3 BackProp, SGD

1. Describe the role of the following notions related to parameter estimation of an artificial neural network:
   - backpropagation
   - forward and backward pass
   - Stochastic Gradient Descent
2. Provide the main steps of the backpropagation algorithm.

**BackProp** The backpropagation is used to compute the gradient from the cost to the whole network. That is, when the network has classified some input $x$ as $y \neq t$, the gradient is computed.

**Forward and backward pass** The backprop algorithm first execute a forward pass in which the hidden units estimate their output bu multiplying input per weight. Then the gradient of the error is computed and passed backward to adjust the weights.

**SGD** The SGD is a training algo which uses backprop: it takes a sample of elements, compute the gradient estimates and updates the parameters.

**Steps of backprop** The forward steps all weights are unchanged:

1. for every layer multiply input per weight
2. pass the result into the activation function
3. at the end, the outputs units use the weights from the hidden units to compute the output
4. the output passes into the activation function

Once the output is computed, the error can be estimated.
The backprop :

1. for each output unit, multiply the error's gradient into the precedent hidden unit
2. update the weights accordingly

### 2.7.4 Dimension

**EXERCISE 2**

Consider a two-layers ANN which receives in input vectors $\mathbf{x}$ of dimension 128 and produces output vectors $\mathbf{y}$ of dimension 10. The hidden layer of the ANN is composed of 50 units which use the ReLU activation function. The output units use a linear activation function.

- The weight matrices of the hidden and output layers are denoted $W_1$ and $W_2$. Provide the dimensions of the weight matrices $W_1$ and $W_2$
- Provide the formula explicitly stating how the values of $\mathbf{y}$ are computed given an input vector $\mathbf{x}$ in terms of the weight matrices and the activation functions (you can ignore the bias terms).

**Dimension** Since we are not caring about the number of parameters, but rather about the dimension of $W$ (the difference is the lack of a bias term), we can use the formula:

$$W = Y_i \times X_{i+1}$$

Which states that the dimension is equal to the product of:

- dimension of the output $Y_i$ of the previous layer $i$
- with the dimension of the input $X_{i+1}$ of the next layer

With this we get:

$$W_1 = 128 * 50 = 6400$$

While the output layer has:

$$W_2 = 50 * 10 = 500$$

**Formula** The output of the hidden neurons are given by:

$$a_i = f(x * w_i)$$

Where $f$ is the activation function.
The final outputs is computed as follows:

$$y = \sum_i f_{out}(a_i)$$

### 2.7.5 Perceptron

**EXERCISE 3**

1. Describe the perceptron model for classification.
2. Describe the perceptron training algorithm.
3. Discuss convergence properties of perceptón training algorithm.

**Classification** Given a set of inputs $X = x_1, x_2, \ldots, x_n$ and a set of weights $W = w_1, w_2, \ldots, w_n$ the perceptron classifies an input with:

$$\sigma(W \cdot X)$$

where the function $\sigma$ is 1 if $W \cdot X > 0$ and 0 elsewhere.

**Training** The training comes by updating the weights as follows:

$$w_i = w_i + \eta \sum_n (t_n - w^t x) x_{i,n}$$

Where:

- $\eta$ is the learning rate
- $t_n$ is the true class of sample $x_n$

**Convergence** The perception training can converge only if the data is linearly separable and $\eta$ is sufficiently small.

## 2.8 SVM

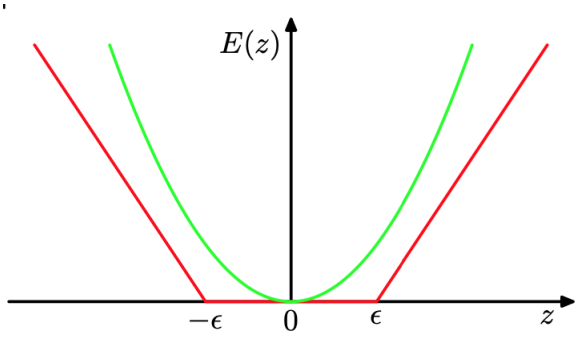
### 2.8.1 Slack variables

Consider the following energy-like function defining Support Vector Machine regression:

$$J(\mathbf{w}, C) = C \sum_{i=1}^{N} L_\epsilon(t_i, y_i) + \frac{1}{2}\|\mathbf{w}\|^2,$$

with $y_i$, $t_i$ target and predicted values, respectively, and $L_\epsilon(t,y) = \begin{cases} 0 & \text{if } |t-y| < \epsilon \\ |t-y| - \epsilon & \textit{otherwise} \end{cases}$ the $\epsilon$-insensitive error function.

1. Plot the $\epsilon$-insensitive error function and explain what is the difficulty in minimizing $J$.
2. To overcome this difficulty slack variables $\xi^+$ and $\xi^-$ are introduced. Explain (qualitatively) the role of the slack variables.



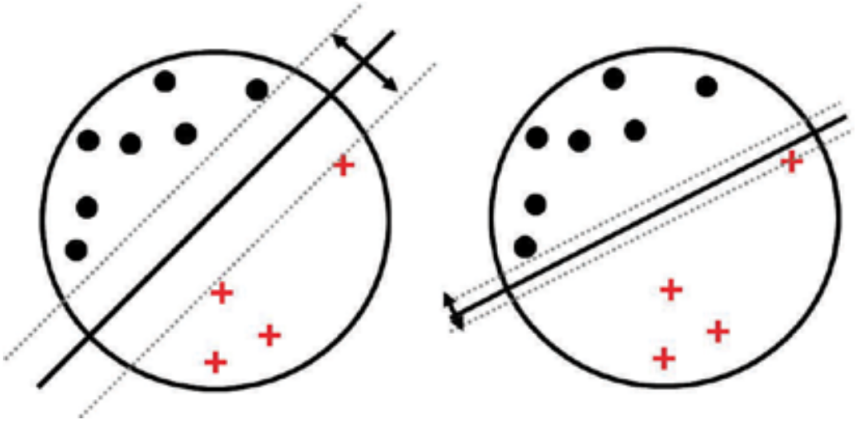

**Plot** The insensitive error function is reported in the above figure.
The difficulty comes from the non differenciability of the $E$ (red) function.

**Slack Variables** The slack variables are introduced to relax the constraint from hard (no points closer than $\eta$ to the splitting plane) to soft (some points are allowed).

### 2.8.2 Maximal Margins

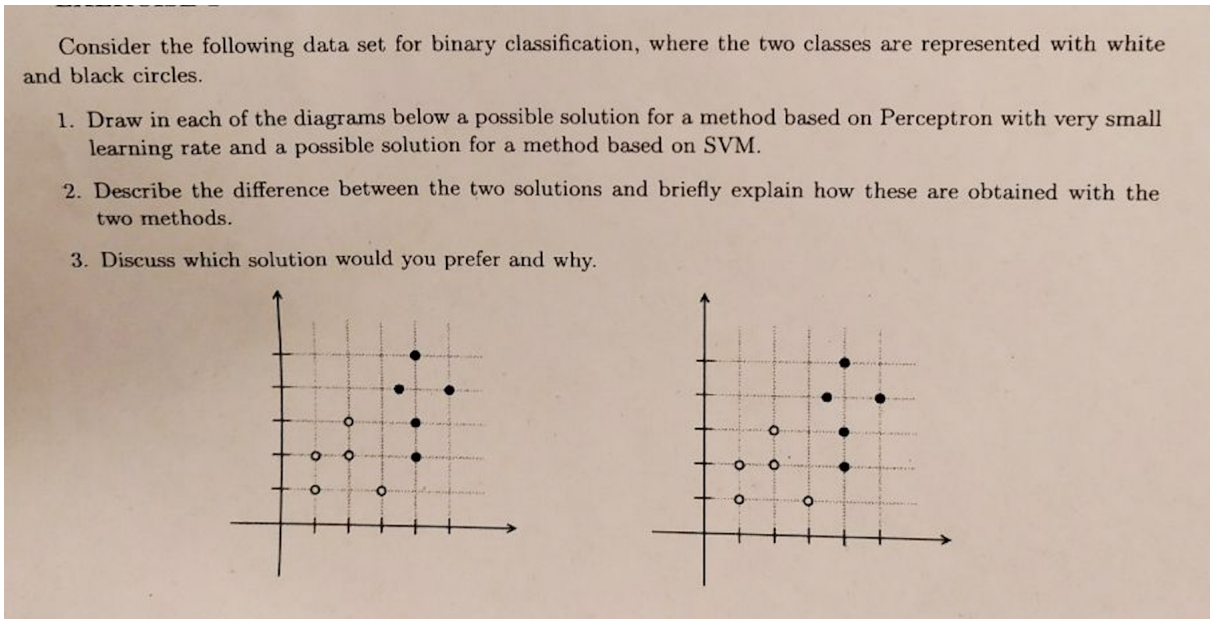
1. Describe the principle of maximal margin used by SVM classifiers. Illustrate the concept with a geometric example.
2. Draw a linearly separable dataset for binary classification of 2D samples. Draw two solutions (i.e., two separation lines): one corresponding to the maximum margin, the other one can be any other solution.
3. Discuss why the maximum margin solution is preferred for the classification problem.

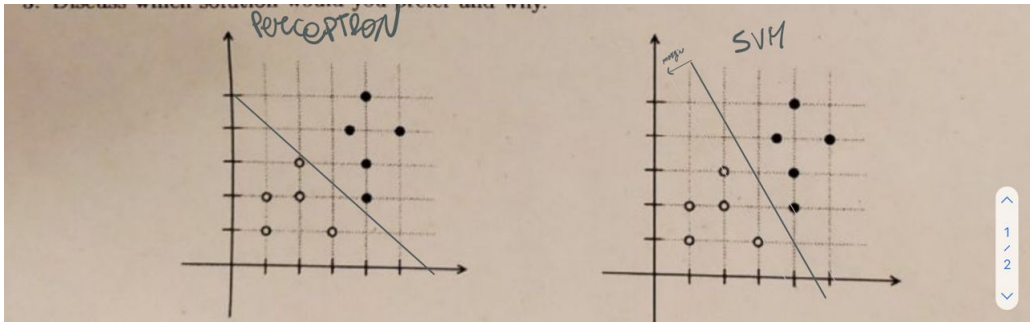
**Description** Having a dataset of two linearly separable classes, we can use a line to split the space into two parts.
Rather than a line we have a family of lines constrain into a small space.
The question is which line is the best one?
Maximize margins between the line and the support vector to find the optimal one.



**Maximum margin for classification** Optimization problem for determining w (dimension $|X|$) transformed in an optimization problem for determining a (dimension $|D|$).
Efficient when $|X| < |D|$ (most of ai will be zero). Very useful when $|X|$ is large or infinite.

### 2.8.3 SVM vs Perceptron




**Plot**

**Describe** svm is better because it finds the best hyperlplane that maximize the the distances between two classes, so it provides a better accuracy, and it takes care about just a few points( points closer to the margin).

The svm works even for non linearly separable dataset, using kernel trick. perceptron, instead, finds a lines that divides two classes, but its not optimal.

## 2.9 CNN

### 2.9.1 Dimensions

Consider a CNN with the following structure for its first two layers:

**conv1** $5 \times 5$ kernel and 64 feature maps with padding 2 and stride 1

**relu1** acting on 'conv1'

**pool1** $2 \times 2$ max pooling with stride 2 acting on 'relu1'

**conv2** $3 \times 3$ kernel and 128 feature maps with padding 0 and stride 2

**relu2** acting on 'conv2'

**pool2** $2 \times 2$ max pooling with stride 4 acting on 'relu2'

1. For input images of dimension $1242 \times 378 \times 3$ compute the dimensions of the volume on the output of each layer and explain how it is computed.
2. Describe what is the number of parameters of each layer.

**Dimensions** Given:

- $s$: the stride
- $p$: the padding
- $w_i$: the output width (first dimension)
- $h_i$: the output height (second dimension)
- $d_i$: the output color channel (third dimension)
- $w_i \times h_i \times d_i$: output dimension of layer i
- $n_i \times m_i$: the dimension of the kernel for conv $i$
- $\theta_i$: total number of parameters for layer i.
- $y_i$: output feature map of layer i.
- $x_i$: input feature map of layer i.

We have that the following hold for a CNN layer:

$$\begin{aligned} w_i &= \frac{w_{i-1} - n + 2p}{s} + 1 \\ h_i &= \frac{h_{i-1} - m + 2p}{s} + 1 \\ d &= y_i \\ \theta_i &= (n_i \cdot m_i \cdot x_i + 1) \cdot y_i \end{aligned} \tag{2.2}$$

For a fully connected layer:

$$\theta_i = (w_i + 1) \cdot h_i$$

So, using the above equation we have:

- Input $1242 \times 378 \times 3$
- conv1 :

$$\begin{aligned} w_{c1} &= \frac{1242 - 5 + 2 \cdot 2}{1} + 1 = 1242 \\ h_{c1} &= \frac{378 - 5 + 2 \cdot 2}{1} + 1 = 378 \\ d_{c1} &= 64 \\ \theta_{c1} &= (5 \cdot 5 \cdot 3 + 1) \cdot 64 = 4864 \end{aligned} \tag{2.3}$$

- relu1: no change
- pool1 :

$$\begin{aligned} w_{p1} &= \frac{1242 - 2 + 2 \cdot 0}{2} + 1 = 621 \\ h_{p1} &= \frac{378 - 2 + 2 \cdot 0}{2} + 1 = 189 \\ d_{p1} &= 64 \end{aligned} \tag{2.4}$$

- conv2 :

$$\begin{aligned} w_{c2} &= \frac{621 - 3 + 2 \cdot 0}{2} + 1 = 310 \\ h_{c2} &= \frac{189 - 3 + 2 \cdot 0}{2} + 1 = 94 \\ d_{c2} &= 128 \\ \theta_{c2} &= (3 \cdot 3 \cdot 64 + 1) \cdot 128 = 73856 \end{aligned} \tag{2.5}$$

- relu2: no change
- pool2 :

$$\begin{aligned} w_{p2} &= \frac{310 - 2 + 2 \cdot 0}{4} + 1 = 78 \\ h_{p2} &= \frac{94 - 2 + 2 \cdot 0}{4} + 1 = 24 \\ d_{p2} &= 128 \end{aligned} \tag{2.6}$$

So the output is of dimensions $78 \times 24 \times 128$ and the total number of weights is:

$$\theta = \theta_{c1} + \theta_{c2} = 78720$$

### 2.9.2 Overfitting

Describe two different methods to overcome overfitting in Convolutional Neural Networks (CNN).

The usual approach can be taken to prevent overfitting:

- Data augmentation: especially for image, you can rotate them, introduce noise or change colors
- Dropout: drop neuron who's connection are not often used
- L1/L2 regularization.

### 2.9.3 Dimension and Design

Consider that the output of layer $l$ of a CNN is the set of feature maps $M$ with size $256 \times 256 \times 64$

1. What is the size of the feature maps $N$ when max-pooling with a $2 \times 2$ kernel and stride 2 is applied on $M$?
2. Design a convolutional layer which, when applied on $M$, produces feature maps with the same size as $N$. Describe all the relevant parameters of the layer you have designed.
3. What happens if the non-linear activation functions of the hidden layers of the CNN are replaced with linear functions? Is the effective depth of the network affected and how?

**Max-pool size** For this we can use the usual formula:

$$w_i = \frac{w_{i-1} - n + 2p}{s} + 1$$

Which translates in:

$$\begin{aligned} w_{p1} &= \frac{256 - 2 + 2 \cdot 0}{2} + 1 = 128 \\ h_{p1} &= \frac{256 - 2 + 2 \cdot 0}{2} + 1 = 128 \\ d_{p1} &= N = 64 \end{aligned} \tag{2.7}$$

**Design** The conv layer can be of any size as long as the feature map is $N$, for example we can have a $2 \times 2$ kernel with $p = 0$ padding and stride $s = 2$ (same as the pool layer). Such layer will have as output: Which translates in:

$$\begin{aligned} w_{c1} &= \frac{256 - 2 + 2 \cdot 0}{2} + 1 = 128 \\ h_{c1} &= \frac{256 - 2 + 2 \cdot 0}{2} + 1 = 128 \\ d_{c1} &= N \end{aligned} \tag{2.8}$$

In such case the number of parameters would be:

$$|\theta| = (2 \times 2 \times 64 + 1) \times 64 = 16448$$

**Depth** The output of the hidden activation function influences the depth (or feature space) of the network in terms of values. ????

### 2.9.4 Ensamble

Consider $N$ convolutional neural networks trained to classify images of cats and dogs with a corresponding confidence value (output of sigmoid activation function)

- Describe a way to combine the predictions of the CNNs in order to get a single more accurate prediction.
- Assume $N = 3$, class '0' represents dogs, class '1' cats and for a given image the three CNN outputs are: $(0.912, 0.432, 0.444)$. Apply the method described above to classify the image using the predictions of the three CNNs.

**Ensamble** A simple yet efficient way to combine the predictions is just to sum them up and get the mean value with:

$$y(x, CNN) = \frac{1}{N} \sum_n y(x, CNN_n)$$

**Solution** The method would follow the above equation as:

$$\frac{1}{N} \sum_n y(x, CNN_n) = \frac{1}{3}(0.912 + 0.432 + 0.444) = 0.596$$

So that the image would be classified as a cat (class 1)

## 2.10 PCA

### 2.10.1 Ex. Rotating 3

**EXERCISE B1**

Consider the following data $\mathbf{x}_1, \ldots, \mathbf{x}_N$ where the intrinsic dimensions are described in terms of a 2D translation and rotation (3 parameters) and the set of principal components $\mathbf{u}_1, \ldots, \mathbf{u}_M$ recovered from this data.

- How can these points be expressed in the basis defined by the principal components? Provide the relative formula.
- Is PCA able to recover a 3 dimensional space that fully describes the data (apart from noise)? Explain your answer.

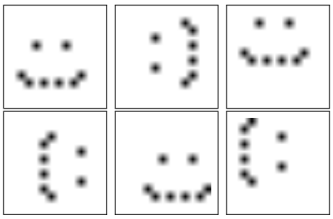
**Formula** The points $x$ can be expressed in the basis of the PC $\hat{x}$ with a simple projection on the direction $u_i$:

$$\hat{x}_n = \sum_i (x_n^T u_i) u_i$$

**Data description** Yes because even if in theory there are as many dimension as there are pixels in the image (assuming black/white), in practice you can fully recover every image using 3 dimensions: x, y of the center and an angle of rotation because images are all the same, so you need no extra info about the image itself, only about its position

### 2.10.2 Ex. Smile Face

Consider the binary (black & white) images below defined on a $12 \times 12$ grid:



1. Explain what is the dimensionality of the data space and what is the intrinsic dimensionality of the given data.
2. Suppose you apply PCA on the data $\mathbf{x}_1, \ldots, \mathbf{x}_6$ and find that the data can be fully described using $M$ principal components, namely $\mathbf{u}_1, \ldots, \mathbf{u}_M$. Describe how the original data can be written in the space defined by these $M$ principal components.
3. Is $M$ going to be equal to the number of intrinsic dimensions? Explain.

**Dimension** Since there are $12 \times 12 = 144$ pixels that can assume the value $B = \{0, 1\}$, the data dimensionality is $B^{144}$. On the other hand, the intrinsic dimensionality is just $D = 3$ two for the coordinates and one for the angle rotation.

**Data projection** Since we already have the principal components we can just apply the projection formula:

$$\hat{x}_n = \sum_i (x_n^T u_i) u_i$$

**Explanation** M should be at least equal to the number of intrinsic dimension but could also increase if there is a high enough gain in accuracy.

## 2.11 Unsupervised

### 2.11.1 Gaussian Mixture models GMM

Given an unsupervised dataset $D = \{\mathbf{x}_n\}$

1. Define the Gassian Mixture Model (GMM) and describe the parameters of the model.
2. Draw an example of a 2D data set (i.e., $D \subset \Re^2$) generated by a GMM with $K = 3$, qualitatively showing in the picture also the parameters of the model.
3. Determine the size of the model (i.e., number of independent parameters) for the dataset illustrated above.

**Definition** A gaussian mixture model has the form:

$$p(x) = \prod_k \pi_k N(x, \mu_k, \Sigma_k)$$

The parameters are given by $\mu_k, \Sigma_k, \pi_k$

**Model size** Since $K = 3$ the parameters are:

- 3 means $\mu_1, \mu_2, \mu_3$
- 3 mixture intensities $\pi_1, \pi_2, \pi_3$
- 3 covariance matrices $\Sigma_1, \Sigma_2, \Sigma_3$. Notice that these matrices are $3 \times 3$ each, so, in terms of parameters, they count as 36 elements.

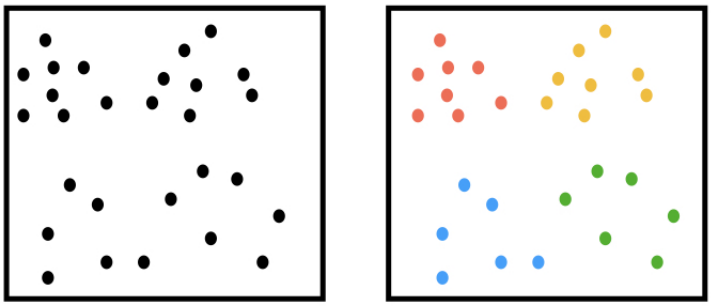
The total size of the model is 9.

### 2.11.2 K-means

1. Define with a precise formal definition the unsupervised learning problem.
2. Provide a full example of unsupervised learning problem (i.e., a specific invented data set), possibly in a graphical form.
3. Describe a solution to the defined problem based on K-Means, providing examples of execution of some steps of the algorithm and a reasonable solution.

**Definition** Unsupervised learning is a branch of machine learning that learns from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning identifies commonalities in the data and reacts based on the presence or absence of such commonalities in each new piece of data. Alternatives include supervised learning and reinforcement learning.

**Example** Using K-Means with number of clusters equal to 4, we get



**K-Means** Steps for k-means:

1. Set k
2. partition data randomly and define centroids
3. For each sample estimate its distance from the current centroid. If the distance is more than another centroid switch.
4. repeat previous step until convergence

### 2.11.3 Unsupervised vs supervised

Machine learning problems can be categorized in supervised and unsupervised. Explain the difference between them providing a precise formal definition (not only explanatory text) in terms of input and output of the two categories of problems.

Supervised learning is typically done in the context of:

- **Classification**: when we want to map input to output labels.
- **Regression**: when we want to map input to a continuous output.

In both regression and classification, the goal is to find specific relationships or structure in the input data that allow us to effectively produce correct output data.

Unsupervised learning uses techniques such as clustering, representation learning and density estimation. In all of these cases, we wish to learn the inherent structure of our input data without using explicitly-provided labels. Since no labels are provided, when analyzing the output there is no specific way to compare model performance in most unsupervised learning methods.

On the other hand, supervised learning intends to infer a conditional probability distribution $p_X(x|y)$ conditioned on the label $y$ of input data; unsupervised learning intends to infer an a priori probability distribution $P_X(x)$ . Compared to supervised learning where training data is labeled with the appropriate classifications, models using unsupervised learning must learn relationships between elements in a data set and classify the raw data without "help."

## 2.12 Probabilistic Models

### 2.12.1 Generative vs Discriminative

Given a dataset $D$ for a classification problem with classes $\{C_1, \ldots, C_n\}$.

1. Describe the difference between generative and discriminative probabilistic models for classification.
2. Draw a 2D dataset for a binary classification problem and show (also in a graphical form) a possible solution using a probabilistic generative model.

**Difference** the difference is that the generative model estimates $P(C_i|x)$ with the bayes theorem while the discriminative uses the model directly.

### 2.12.2 Ex. Binary classification

Consider a dataset $D$ for the binary classification problem $f : \Re^3 \mapsto \{A, B\}$.

1. Describe a probabilistic generative model for such a classification problem, assuming Gaussian distributions.
2. Identify the parameters of the model and determine the size of the model (i.e., the number of independent parameters).

**Definition** Since we are dealing with a 3-dimensional input we can assume that each input comes from a different gaussian distribution, thus $K = 3$. Moreover each gaussian is of the kind:

$$\pi_k N(\mu_k, \Sigma_k)$$

Now let's set a latent variable $z_k$ for each gaussian, we get :

$$P(X) = \prod_n \sum_k \pi_k N(x_n|\mu_n, \Sigma_n)$$

**Parameters** The parameters for each gaussian are 3:

- $\mu_k$, the mean
- $\Sigma_k$, the covariance matrix
- $\pi_k$, the mixing probability

So the total number of params is 9.

## 2.13 Other

### 2.13.1 Boosting

1. Provide the main features about boosting.
2. Write the error function whose minimization leads to a formulation equivalent to the AdaBoost algorithm.

**Boosting** In supervised learning, boosting converts weak learners to strong ones. A weak learner is defined to be a classifier that is only slightly correlated with the true classification (it can label examples better than random guessing). In contrast, a strong learner is a classifier that is arbitrarily well-correlated with the true classification.

**AdaBoost** No prior knowledge about base learner is required, no parameters to tune (except for M), can be combined with any method to find base learners and theoretical guarantees given enough data and base learners with moderate accuracy.
The error function to minimize is:

$$J_m = \sum_{n=1}^{N} w_n^{(m)} \mathcal{I}(y_m(x_n) \neq t_n)$$

Given

- $\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} \mathcal{I}(y_m(x_n) \neq t_n}{\sum_{n=1}^{N} w_n^{(m)}}$
- $\alpha = ln(\frac{1-\epsilon_m}{\epsilon_m})$

The output of the linear classifier is:

$$Y_m(x) = sign(\sum_{m=1}^{M} \alpha_m y_m(x))$$

### 2.13.2 Gram, Kernel

Given input values $\mathbf{x}_i$ and the corresponding target values $t_i$ with $i = 1, \ldots, N$, the solution of regularized linear regression can be written as:

$$y(\mathbf{x}) = \sum_{i}^{N} \alpha_i \mathbf{x}_i^T \mathbf{x},$$

with $\boldsymbol{\alpha} = (XX^T + \lambda I)^{-1}\mathbf{t}$, $X = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T$ and $\lambda$ the regularization weight.

Considering a kernel function $k(\mathbf{x}, \mathbf{x}')$:

1. Provide a definition of the Gram matrix.
2. Explain how a kernelized version for regression can be obtained based on the equations provided above.

**Gram Matrix** Given some kernel $k(x_i, x_j)$, the Gram Matrix $K = \boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}}$ is a $N \times N$ symmetric matrix of the form:

$$K_{nm} = k(x_n, x_m)$$

**Kernalized regression** Considering a linear kernel $k(x, x') = x^T x'$ we can rewrite the model as:

$$y(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x)$$

having:

$$\alpha = (K + \lambda I)^{-1} t$$

### 2.13.3 AutoEncoder

Describe what is the architecture of an autoencoder and its purpose.

**Description** The architecture of an autoencoder is similar to the one of a cnn, but flipped. That is it takes as input a random vector and returns an image. It is usually trained in an adversarial fashion to keep the supervision cycle closed. It's purpose is to encode a sparse representation (e.g. words) into a small vector (dimensionallity reduction)

### 2.13.4 Multiple Learners

Assume you have 4 image classifiers with medium-good classification accuracy.

1. Describe an ensemble method for achieving higher classification accuracy by combining such classifiers.
2. Are there any specific properties that each classifier has to have to achieve higher accuracy? If the answer is positive, explain which these properties are.

**Method** A method to achieve such results we can use the voting method in which we train each classifier in parallel on the trainset. For the prediction we

take the most predicted class (for classification) or the sum of the predictions (for regression).

**Proprieties** A base propriety for each classifier is that they have to achieve a sufficient accuracy on their own. Implementing multi learners methods where the indiviual classifiers do not achieve a more than 10% accuracy is not advised.