# Introduction to Data Science - 1MS041

Benny Avelin

Department of Mathematics

HT 2023

# Recall from last time

- We defined a stochastic process as a index family of random variables $X_\alpha$ where $\alpha \in I$ is the index set. The base example is a sequence of i.i.d. random variables, $X_1, \ldots, X_n$, here the index set is $\mathbb{N}$.

# Recall from last time

- We defined a stochastic process as a index family of random variables $X_\alpha$ where $\alpha \in I$ is the index set. The base example is a sequence of i.i.d. random variables, $X_1, \ldots, X_n$, here the index set is $\mathbb{N}$.
- We defined the concept of Markov chain, which is a stochastic process which takes a finite number of states and its dependency on the past is only the previous value, i.e.

$$\mathbb{P}(X_{t+1} = x \mid X_1, \ldots, X_t) = \mathbb{P}(X_{t+1} = x \mid X_t).$$

# Recall from last time

- We defined a homogeneous Markov chain, as one where the transition probabilities does not depend on the index $t$ (or time).

$$\mathbb{P}(X_{t+1} = y \mid X_t = x) = P_{xy}$$

that is the probability of transitioning from state $x$ to state $y$ does not depend on the particular time $t$.

# Recall from last time

- We defined a homogeneous Markov chain, as one where the transition probabilities does not depend on the index $t$ (or time).

$$\mathbb{P}(X_{t+1} = y \mid X_t = x) = P_{xy}$$

  that is the probability of transitioning from state $x$ to state $y$ does not depend on the particular time $t$.
- Since $X_t \in \mathbb{X}$, where $\mathbb{X}$ is called the state space and $\mathbb{X}$ is a finite set. We can define a matrix $P_{xy}$ where $x, y \in \mathbb{X}$. We call this matrix the transition matrix.

# Recall from last time

- We defined a homogeneous Markov chain, as one where the transition probabilities does not depend on the index $t$ (or time).

$$\mathbb{P}(X_{t+1} = y \mid X_t = x) = P_{xy}$$

  that is the probability of transitioning from state $x$ to state $y$ does not depend on the particular time $t$.
- Since $X_t \in \mathbb{X}$, where $\mathbb{X}$ is called the state space and $\mathbb{X}$ is a finite set. We can define a matrix $P_{xy}$ where $x, y \in \mathbb{X}$. We call this matrix the transition matrix.
- We can use the transition matrix to compute how distributions change when the Markov chain progresses. I.e. let us assume that at time $t$ we have a distribution over the states $p_t$, then the distribution at time $t + 1$ is computed as

$$p_{t+1} = p_t P \qquad p_t = p_0 P^t.$$

# Today

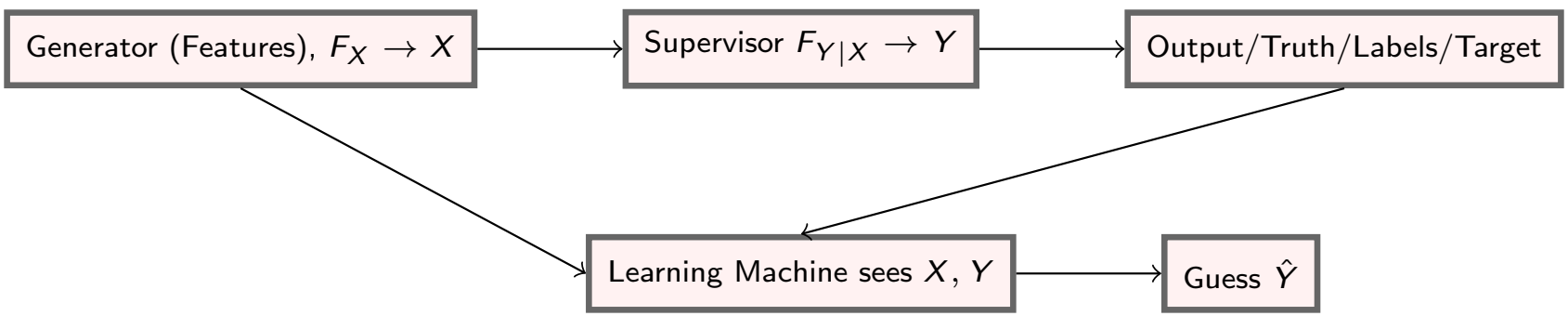
Today we will take a look at the pattern recognition problem.

- We will see the first algorithm developed for this problem the "Perceptron", which eventually became neural networks.
- We will study its limitations and how to overcome them by a concept known as kernelization.

# Supervised learning

## Setup

1. The generator of the data $G$
2. The supervisor $S$
3. The learning machine $LM$.



# Pattern recognition

- Supervisors conditional distribution $F(y|x)$ is discrete, and can take $k$ different values, $y = 0, \ldots, k-1$.

# Pattern recognition

- Supervisors conditional distribution $F(y|x)$ is discrete, and can take $k$ different values, $y = 0, \ldots, k-1$.
- Model space $\mathcal{M} = \{g_\lambda(x) : g_\lambda(x) \in \{0, \ldots, k-1\}\}$.

# Pattern recognition

- Supervisors conditional distribution $F(y|x)$ is discrete, and can take $k$ different values, $y = 0, \ldots, k-1$.
- Model space $\mathcal{M} = \{g_\lambda(x) : g_\lambda(x) \in \{0, \ldots, k-1\}\}$.
- $g_\lambda$ a **decision function**, **decision rule**, **classifier**.

# Pattern recognition

- Supervisors conditional distribution $F(y|x)$ is discrete, and can take $k$ different values, $y = 0, \ldots, k-1$.
- Model space $\mathcal{M} = \{g_\lambda(x) : g_\lambda(x) \in \{0, \ldots, k-1\}\}$.
- $g_\lambda$ a **decision function**, **decision rule**, **classifier**.
- $0-1$ loss

$$L(z, u) = \begin{cases} 0 & \text{if } y = u \\ 1 & \text{if } y \neq u \end{cases}$$

# Pattern recognition

- Supervisors conditional distribution $F(y|x)$ is discrete, and can take $k$ different values, $y = 0, \ldots, k-1$.
- Model space $\mathcal{M} = \{g_\lambda(x) : g_\lambda(x) \in \{0, \ldots, k-1\}\}$.
- $g_\lambda$ a **decision function**, **decision rule**, **classifier**.
- $0-1$ loss

$$L(z, u) = \begin{cases} 0 & \text{if } y = u \\ 1 & \text{if } y \neq u \end{cases}$$

## The pattern recognition problem

Minimize

$$R(\lambda) = \int L(y, g_\lambda(x)) dF(x, y) = \mathbb{E}[L(Y, g_\lambda(X))]$$

where $(X, Y) \sim F(x, y)$, where $g_\lambda \in \mathcal{M}$.

# Example

## From regression to decision function

The regression problem. The goal is to estimate the function $\mathbb{E}[Y \mid X]$, where $Y$ are the labels and $X$ are the features. We assume $f_X$ is a fixed density, that we will not care about and instead assume that $f_{Y|X}$ is part of a parametrized family, i.e. $f_{Y|X} = p_{\alpha^*,X}$ for some $\alpha^*$.

- $p_{\alpha^*,X} = N(\alpha_1 X + \alpha_2, \alpha_3^2)$, Linear regression
- $p_{\alpha^*,X} = \text{Bernoulli}(G(\alpha_1 X + \alpha_2))$,

$$G(x) = \frac{1}{1 + e^{-x}}$$

Logistic regression

## Definition

Let $r(x) = \mathbb{E}[Y \mid X]$, then the **Bayes classification rule** $h^*$ is

$$h^*(x) = \begin{cases} 1 & \text{if } r(x) > 1/2 \\ 0 & \text{otherwise.} \end{cases}$$

## Definition

Let $r(x) = \mathbb{E}[Y \mid X]$, then the **Bayes classification rule** $h^*$ is

$$h^*(x) = \begin{cases} 1 & \text{if } r(x) > 1/2 \\ 0 & \text{otherwise.} \end{cases}$$

Lets say we use Logistic regression and fit the function $G(\hat{\alpha}_1 X + \hat{\alpha}_2)$, now the model is that $p_{\hat{\alpha},X} = \text{Bernoulli}(G(\hat{\alpha}_1 X + \hat{\alpha}_2))$ which is a model of the conditional density, i.e.

$$\hat{r}(x) = \mathbb{E}_{p_{\hat{\alpha},X}}[Y] = G(\hat{\alpha}_1 X + \hat{\alpha}_2)$$

## Definition

Let $r(x) = \mathbb{E}[Y \mid X]$, then the **Bayes classification rule** $h^*$ is

$$h^*(x) = \begin{cases} 1 & \text{if } r(x) > 1/2 \\ 0 & \text{otherwise.} \end{cases}$$

Lets say we use Logistic regression and fit the function $G(\hat{\alpha}_1 X + \hat{\alpha}_2)$, now the model is that $p_{\hat{\alpha},X} = \text{Bernoulli}(G(\hat{\alpha}_1 X + \hat{\alpha}_2))$ which is a model of the conditional density, i.e.

$$\hat{r}(x) = \mathbb{E}_{p_{\hat{\alpha},X}}[Y] = G(\hat{\alpha}_1 X + \hat{\alpha}_2)$$

thus we can construct

$$g_{\hat{\alpha}}(x) = \begin{cases} 1 & \text{if } \hat{r}(x) > 1/2 \\ 0 & \text{otherwise.} \end{cases}$$

## Definition

Let $r(x) = \mathbb{E}[Y \mid X]$, then the **Bayes classification rule** $h^*$ is

$$h^*(x) = \begin{cases} 1 & \text{if } r(x) > 1/2 \\ 0 & \text{otherwise.} \end{cases}$$

Lets say we use Logistic regression and fit the function $G(\hat{\alpha}_1 X + \hat{\alpha}_2)$, now the model is that $p_{\hat{\alpha},X} = \text{Bernoulli}(G(\hat{\alpha}_1 X + \hat{\alpha}_2))$ which is a model of the conditional density, i.e.

$$\hat{r}(x) = \mathbb{E}_{p_{\hat{\alpha},X}}[Y] = G(\hat{\alpha}_1 X + \hat{\alpha}_2)$$

thus we can construct

$$g_{\hat{\alpha}}(x) = \begin{cases} 1 & \text{if } \hat{r}(x) > 1/2 \\ 0 & \text{otherwise.} \end{cases}$$

## Note!!

This is what scikit-learns LogisticRegression predict function does!

# Pattern recognition

## Pattern recognition problem from CS perspective

Suppose we have $n$ training data points $T_n := ((X_i, Y_i))_{i=1}^n$ and are interested in a classification rule $h(X)$ that uses $T_n$ to *predict*, i.e., assign labels to previously unseen data $X$.

# Pattern recognition

**Pattern recognition problem from CS perspective**

Suppose we have $n$ training data points $T_n := ((X_i, Y_i))_{i=1}^n$ and are interested in a classification rule $h(X)$ that uses $T_n$ to *predict*, i.e., assign labels to previously unseen data $X$.

- We want our classification rule $h$, which is typically an algorithm, to perform well on previously unseen data by learning from the training data. This is known as *generalization*.

## Definition

The space $\mathcal{X}$ where $X_i$ belongs to is called the *instance space* or *feature space* and the space $\mathcal{Y}$ where $Y_i$ belongs to is called the *label space*.

## Definition

The space $\mathcal{X}$ where $X_i$ belongs to is called the *instance space* or *feature space* and the space $\mathcal{Y}$ where $Y_i$ belongs to is called the *label space*.

Typically, $\mathcal{X}$ is a subset of $\mathbb{R}^d$ and $\mathcal{Y}$ is binary label space either as $\{0, 1\}$ or $\{-1, 1\}$. For example, $\mathcal{X}$ can be $\{0, 1\}^d$ to indicate the presence or absence of something in the instance space, say a specific set of words in an email if the task is to classify emails with labels 0 and 1 for non-spam or spam.

## Remark

To connect back to our previous terminology, we see that the data space $\mathbb{X} = (\mathcal{X}, \mathcal{Y})$ (we will later write $\mathbb{X} \times \mathbb{Y}$ to avoid $X$ being used for both feature and label) is split into the featue space and the label space. The random variable we are observing is a pair $(X, Y)$ and a collection of $n$ samples is the training dataset.

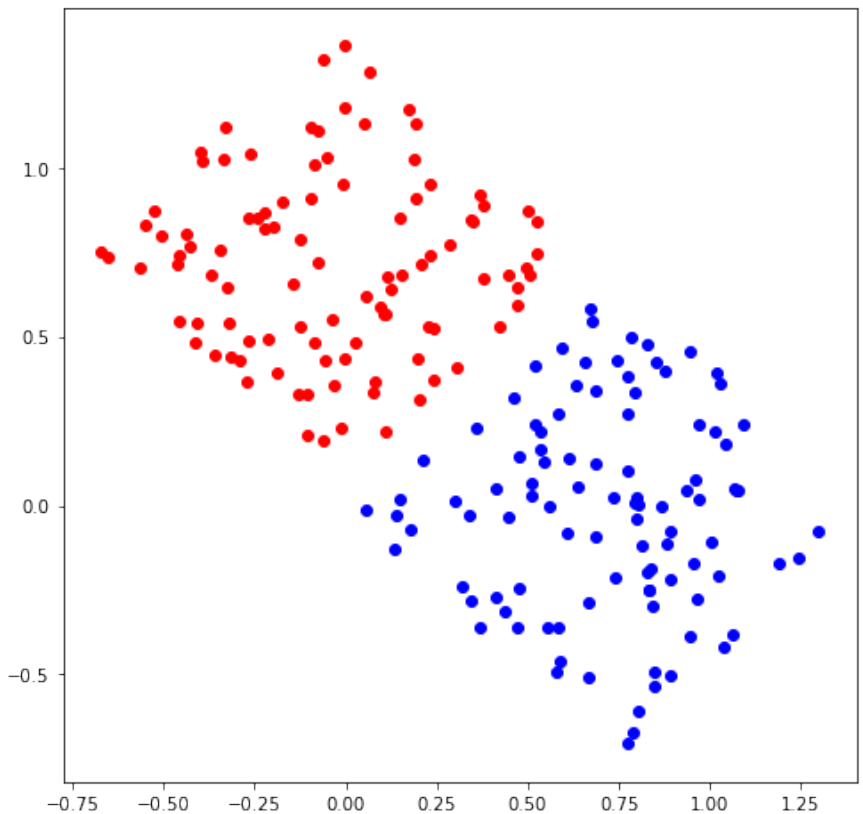
Let us say that we are trying to device a classification rule based on instance space $\mathcal{X} = \mathbb{R}^d$ and label space $\mathcal{Y} = \{-1, 1\}$.

Let us say that we are trying to device a classification rule based on instance space $\mathcal{X} = \mathbb{R}^d$ and label space $\mathcal{Y} = \{-1, 1\}$.

- Simple rules consists of finding a hyperplane that separate out the data, this is called a **linear separator**.

Let us say that we are trying to device a classification rule based on instance space $\mathcal{X} = \mathbb{R}^d$ and label space $\mathcal{Y} = \{-1, 1\}$.

- Simple rules consists of finding a hyperplane that separate out the data, this is called a **linear separator**.



# How do we find such a separator?

- In two or three dimensions it is easy by just looking at the image

# How do we find such a separator?

- In two or three dimensions it is easy by just looking at the image
- In higher dimension, i.e. when the number of features is high, we have to find an automated way of doing this!

# The Perceptron algorithm

- The perceptron is a linear classification rule

# The Perceptron algorithm

- The perceptron is a linear classification rule
- The first training algorithm was devised by Frank Rosenblatt at Cornell Aeronautical Laboratory in 1957.

# The Perceptron algorithm

- The perceptron is a linear classification rule
- The first training algorithm was devised by Frank Rosenblatt at Cornell Aeronautical Laboratory in 1957.
- The goal was to build a machine that could recognize hand written letters.

# Lets explain the algorithm

## The dimension trick

We start by increasing the dimension, i.e. lets say we have the features $X = (x_1, \ldots, x_k)$, i.e. we have $k$ dimensions (features), then we can create $\tilde{X} = (x_1, \ldots, x_k, 1)$ which is now in $k + 1$ dimensions. The point is that we can write

$$X \cdot w + c = 0 \iff \tilde{X} \cdot \tilde{w} = 0$$

where $\tilde{w} = (w_1, \ldots, w_k, c)$.

# Lets explain the algorithm

## The dimension trick

We start by increasing the dimension, i.e. lets say we have the features $X = (x_1, \ldots, x_k)$, i.e. we have $k$ dimensions (features), then we can create $\tilde{X} = (x_1, \ldots, x_k, 1)$ which is now in $k+1$ dimensions. The point is that we can write

$$X \cdot w + c = 0 \iff \tilde{X} \cdot \tilde{w} = 0$$

where $\tilde{w} = (w_1, \ldots, w_k, c)$.

## Warning

We will from now on assume that we have done **the dimension trick** but we go back to the notation $X$ and $w$.

# The perceptron algorithm

Let us say that we are trying to device a classification rule based on instance space $\mathcal{X} = \mathbb{R}^d$ and label space $\mathcal{Y} = \{-1, 1\}$.

# The perceptron algorithm

Let us say that we are trying to device a classification rule based on instance space $\mathcal{X} = \mathbb{R}^d$ and label space $\mathcal{Y} = \{-1, 1\}$.

In the following we can take a training dataset $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. For the perceptron the goal is to find

$$(w \cdot x_i) y_i > 0, \qquad i = 1, \ldots, n.$$

# The perceptron algorithm

Let us say that we are trying to device a classification rule based on instance space $\mathcal{X} = \mathbb{R}^d$ and label space $\mathcal{Y} = \{-1, 1\}$.

In the following we can take a training dataset $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. For the perceptron the goal is to find

$$(w \cdot x_i) y_i > 0, \qquad i = 1, \ldots, n.$$

**The perceptron algorithm**

1. $w = 0$
2. while there exists $x_i$ with $x_i y_i \cdot w \leq 0$, update $w := w + x_i y_i$

# Convergence is guaranteed if linearly separable

**Theorem**

*If there exists* $w^*$ *such that* $w^* \cdot x_i y_i \geq 1$ *for all* $i$*. Then the perceptron algorithm finds a* $w$ *satisfying* $w \cdot x_i y_i \geq 0$ *for all* $i$ *in at most* $r^2 |w^*|^2$ *updates, where* $r = \max_i |x_i|$.

# Convergence is guaranteed if linearly separable

**Theorem**

*If there exists $w^*$ such that $w^* \cdot x_i y_i \geq 1$ for all $i$. Then the perceptron algorithm finds a $w$ satisfying $w \cdot x_i y_i \geq 0$ for all $i$ in at most $r^2 |w^*|^2$ updates, where $r = \max_i |x_i|$.*

**Warning**

If the set is not linearly separable then the algorithm will not converge, in fact it will go all over the place.

# Kernelization

## The idea

If we cannot separate our data, then perhaps we can find a function $\phi$ which maps $\mathbb{R}^{k+1} \to \mathbb{R}^m$ where $m > k+1$, such that our data becomes linearly separable.

# Kernelization

## The idea

If we cannot separate our data, then perhaps we can find a function $\phi$ which maps $\mathbb{R}^{k+1} \to \mathbb{R}^m$ where $m > k+1$, such that our data becomes linearly separable.

So if we transform the $x \to \phi(x)$ for some good transformation $\phi$ then our perceptron will try to solve

$$w \cdot \phi(x_i) l_i > 0$$

## The perceptron algorithm

1. $w = 0$
2. while there exists $x_i$ with $x_i y_i \cdot w \leq 0$, update $w := w + x_i y_i$

The weight after $n$ steps will have the form

$$w = \sum_{i=1}^{n} c_i \phi(x_i)$$

for numbers $c_i$. The perceptron algorithm becomes just addition and subtraction of certain $c_i$'s by 1.
Furthermore

$$w \cdot \phi(x_i) = \sum_{j=1}^{n} c_j \phi(x_j) \cdot \phi(x_i) = \sum_{j=1}^{n} c_i k_{ij}$$

where $k_{ij} = \phi(x_i) \cdot \phi(x_j)$.

## The kernelized perceptron algorithm

1. $c = 0$
2. While there exists an $i$ such that $(Kc)_i y_i \leq 0$ update

$$c_i := c_i + y_i.$$

# Kernel trick

**Issue**

If $\phi(x_i)$ is high dimensional there is a big computational cost to computing it.

What if we had a function $k(x, y)$ that could be written as

$$k(x, y) = \phi(x) \cdot \phi(y)$$

for some $\phi$ and $k$ is easier to compute, then our life would be simpler. Also, what if we are given a function $k(x, y)$ and we would like to know if it is a "kernel function".

# Kernel functions

**Definition**

We call a function $k(x, y) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ a kernel function if there is a mapping $\phi : \mathbb{R}^d \to \mathbb{R}^m$ (for some $m$) such that $k(x, y) = \phi(x) \cdot \phi(y)$.

- $k(x, y) = (\gamma x \cdot y + r)^k$, $(k \in \mathbb{N})$ polynomial
- $k(x, y) = x \cdot y$, linear
- $k(x, y) = e^{-\gamma|x-y|}$, called Radial Basis Function
- $k(x, y) = \tanh(\gamma x \cdot y + r)$, sigmoidal