Lecture 3

Cezar Ionescu

Unsupervised learning

Unsupervised learning is a form of “learning from experience”. It is related to data mining: identifying patterns in data, and to classification.

Example: Assume we have a sequence of images that fall into `k` classes. They could be images of cats and dogs, or of possibly malfunctioning engines (in both cases we would have `k = 2`), or handwritten digits (in which case `k = 10`). We want the learning program to determine the `k` classes by itself.

- Task: `Task = Image -> {1, 2, ..., k}`
- Experience: `Experience = List(Image)`
- Performance: `perf : (Task, List(Image)) → ℝ`
  - $\texttt{perf(t, [img}_1\texttt{, ..., img}_n\texttt{]) = sum [correct(t, img}_1\texttt{), ..., correct(t, img}_2\texttt{)] / n}$
  - the function `correct : (Task, List(Image)) → {1, 2, ..., k}` is a “hypothetical” function, since we might not know the correct classification (as in data mining).

A typical situation in which this type of learning can be achieved is when the data can be seen as having been generated from `k` “ideal” exemplars, distorted in some way. Unsupervised learning can then be seen as an attempt to reconstruct these exemplars from the data.

The data, and hence the ideal exemplars, usually comes in the form of tuples of fixed length (the **features**):

$$\xi = (\xi_1, ..., \xi_m) \in (\Omega_1, ..., \Omega_m), \Omega_i \subseteq \mathbb{R}$$

We can model the “distortion” of the data by assuming that the data is obtained by perturbing the various features with a “noise”:

$$x = (x_1, ..., x_m) = (\xi_1 + \varepsilon_1, ..., \xi_m + \varepsilon_m)$$

where ε_j is a random variable with mean 0.

Thus, if we only had one exemplar, we could reconstruct it by estimating the mean value of the data.

Statistics

Random variables

Consider the typical experiment of flipping a (potentially biased) coin:

```
Ω = {H, T}
p : Event → [0, 1]
p(H) = α
```

What is the expected value of a coin toss?

The answer is, of course, there isn't one: we expect the coin toss to result either in heads, or tails.

Now consider the situation in which the coin toss is the starting point of a bet: if the coin lands heads, you get `h` $, if tails, you get `t` $. What is the expected value of a coin toss in this situation? Intuitively, this would be the value we get if we take the bet a large number of times, i.e.:

```
h * α + t * (1 - α)
```

Remarks:

- `h` and `t` are *not* possible results of the random experiment, which consists of flipping a coin.
- We can construct a "derived" probability space as follows:
  - `Ω' = {h, t}`
  - `p' : Event' → [0, 1]`
    - `p'(h) = p(H)`
    - `p'(t) = p(T)`
- Even if we consider `Ω'`, it is still the case that the expected value is *not* a possible result of the experiment.

Definition: Given `Ω, Event, p`. A **random variable** is a function `X : Ω → ℝ`.

Probability theory and statistics are largely the study of random variables.

Expected values, standard deviations

Definition: Given `Ω`, `Event`, `p` and a random variable `X`. Assume that `Ω` is finite: `Ω = {ω₁, ω₂, ..., ωₙ}`. The **mean** or **expected value** of `X` is

```
E(X) = X(ω₁) * p(ω₁) + X(ω₂) * p(ω₂) + ... + X(ωₙ) * p(ωₙ)
```

Remark: Consider the following game: a random experiment defined by `Ω, Event, p` is performed many times. We are given a random variable `X : Ω → ℝ`. We attempt, each time, to guess the value of the random variable, and we are rewarded proportionally to the distance between our guess and the actual result. Then the optimal strategy consists in always naming the expected value of the variable. Note that this **only the case if the reward is proportional** to the distance between guess and result! This is typically the case in traditional statistics applications, but **not** in machine learning.

Example: A football player always shoots penalties either to the right of the goalkeeper, or to his left. The goalkeeper can minimise the distance to the actual shots by standing still in the middle. Nevertheless, it might actually be better to dive randomly left or right, and actually catch a ball from time to time.

Exercise: Consider a card game played with a full deck, in which aces have a value of 11, jacks 12, queens 13, kings 14, and all other cards a value of 0. We draw a card at random from the deck. What is its expected value? Define Ω, Event, p and the random variable whose expected value you are computing.

Notation: In many situations, the derived probability space is silently substituted for the initial one. That is, we consider

```
Ω = {x(ω₁), ..., x(ωₙ)}
```

and even bypass ωs directly, defining $x_i = x(\omega_i)$:

```
Ω = {x₁, ..., xₙ}
p : Event → [0, 1]
p(A) = sum [p(xᵢ) | xᵢ ∈ A]
```

Definition: Consider $\Omega = \{\omega_1, \omega_2, ..., \omega_n\}$, Event, p, $X : \Omega \to \mathbb{R}$. Let $E(X) = \mu$ The **standard deviation** of X is

```
std_dev(X) = √ ((X(ω₁) - μ)² * p(ω₁) + ... (X(ωₙ) - μ)² * p(ωₙ))
```

The **variance** of X is $\text{std\_dev}(X)^2$

Exercise: Compute the standard deviation of the random variable defined for the card game exercise.

Notation: In the following, when no ambiguities arise, we use μ to denote $E(X)$ and σ for $\text{std\_dev}(X)$

The reason the standard deviation is an interesting measure, is that, no matter what the probability distribution has generated the data, samples are "unlikely" to be "many" standard deviations away from the mean. The following remarkable result formalises this intuition:

Theorem (Chebyshev): Let Ω, Event, p and X as above. Let

```
Farₖ = {ω ∈ Ω | ||X(ω) - μ|| > k * σ}
```

Then $p(\text{Far}_k) \leq 1/k^2$.

Interpretation of Chebyshev's theorem:

Ω, Event, p and X as above. If we draw a random element from Ω, then

- it will fall in the interval `[μ - 2 * σ, μ + 2 * σ]` with probability at least `0.75`
- it will fall in the interval `[μ - 3 * σ, μ + 3 * σ]` with probability at least `0.889`
- it will fall in the interval `[μ - 4 * σ, μ + 4 * σ]` with probability at least `0.938`

Example:

The mean price of houses in a certain neighbourhood is `400000` $, and the standard deviation is `80000` $. Find the price range in which `75%` of the houses will sell. (Bluman, Chapter 3)

Solution: From Chebyshev's theorem, we know that at least `75%` of the houses are within the interval `[μ - 2 * σ, μ + 2 * σ]`. Therefore, the interval is `[240000, 560000]`.

Exercise:

The normal distribution

```
pdf(x) = (1/√(2 * π * σ²)) * exp(- (x - μ)²/(2 * σ²))
```

It is well-known that the normal distribution is ubiquitous. If the data is normally distributed, then we can "tighten" the Chebyshev bounds considerably.

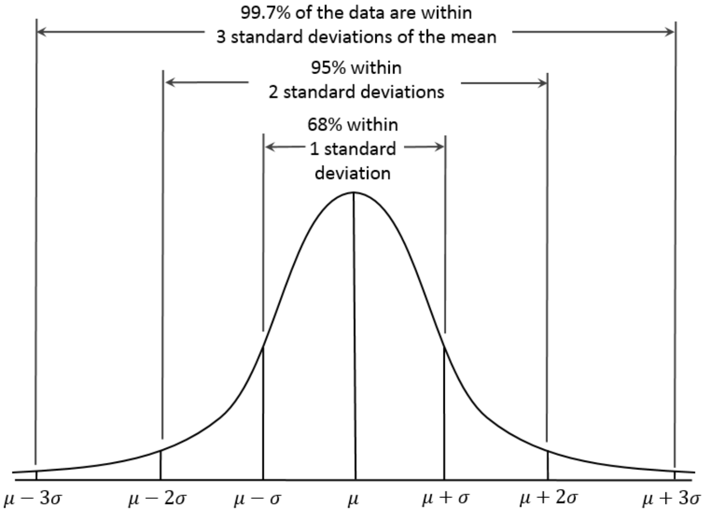


Figure 1: Normal distribution[1]

Example:

Consider the neighbourhood from the previous example, where the mean price of houses is `400000` $, and the standard deviation is `80000` $. Find the price range in which `75%` of the houses will sell, but now assuming that *the prices are normally distributed*.

Solution: We use the Python function `scipy.stats.normal.interval`:

[1] Figure by Dan Kernler - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=36506025

```
import scipy.stats
scipy.stats.norm.interval(0.75, 400000, 80000)
```

We obtain `[307972, 492028]`. From Chebyshev's theorem, we had the much larger interval `[240000, 560000]`.

The central limit theorem

The central limit theorem shows that means of samples are normally distributed around the real mean. Hence, we can use the tighter bounds for estimates.

Central limit theorem: Ω, `Event`, p and X as above. Consider the following experiment: n elements $e_1, \ldots, e_n$ are drawn independently from Ω and we compute the mean value of X for this sample $\mu_s = (X(e_1) + \ldots + X(e_n))/n$. Then μ_s is a random variable whose probability distribution approaches with increasing n the normal distribution with mean μ and standard deviation $\sigma/\sqrt(n)$.

Homework: The theorem states that μ_s is a random variable. But a random variable is a function defined in the context of a probability space. What is that probability space here? You will need to specify Ω', `Event'`, $p' : \text{Event}' \to [0, 1]$ and define $\mu_s : \Omega' \to \mathbb{R}$ as a function in terms of in terms of the given `Omega`, `Event`, p, and X.

Remarks (quality of approximation):

- If the random variable X is itself normally distributed, then the approximation in the central limit theorem is good even for small sample sizes n.
- If the random variable X is not normally distributed, the approximation requires larger n. In many practical applications, a value $n \geq 30$ will be adequate.

Example:

The Nielsen agency reported that children between 2 and 5 watch an average of 25 hours of TV per week. Assuming a normal distribution with $\sigma = 3$ hours. If 20 children are randomly selected, find the probability that the average TV watching time in a week will be $\mu_s > 26.3$ hours. (Bluman, Chapter 6)

Solution: The sample average μ_s is a random variable that is approximately normally distributed, with mean 25 and standard deviation $3/\sqrt(20)$. We need to find the probability that the value of μ_s is bigger than 26.3. We use the Python function `scipy.stats.norm.cdf` to find the probability that μ_s is smaller or equal to 26.3, and subtract from unity:

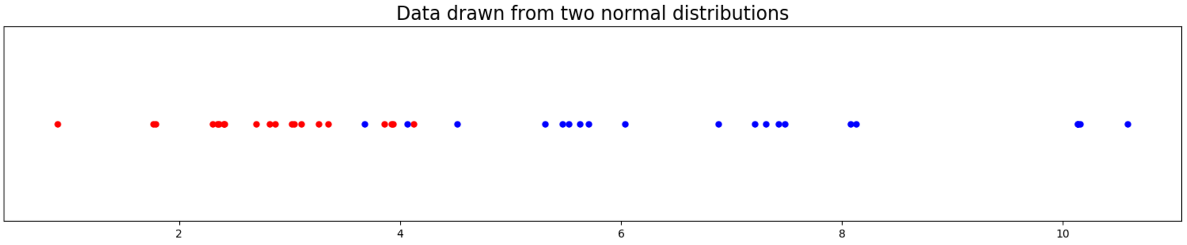
```
import scipy.stats
1 - scipy.stats.norm.cdf(26.3, 25, 3/scipy.sqrt(20))
```

We obtain `0.026316151282870237`, or approximately 2.6%.

EM algorithm

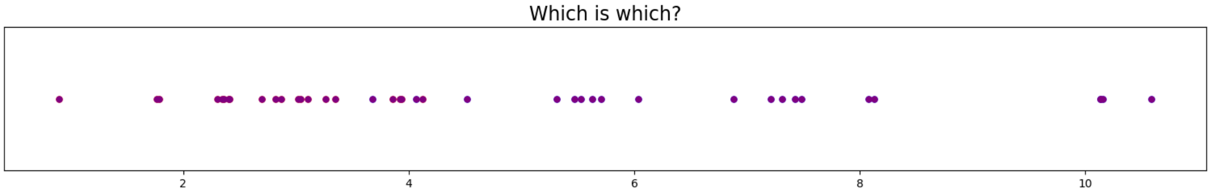
The central limit theorem explains why, when we have a single distorted exemplar, we can recover it as the mean of the data. We now turn to the problem of multiple exemplars.

Example: Consider the following one-dimensional data generated from two normal distributions. The red dots have been generated from `normal(3, 0.8)`, the blue dots from `normal(7, 2)`.



There are `20` dots of each colour. Since they come from normal distributions, the sample means and standard deviations are already very good approximations of the initial ones: `2.8, 0.8` for the red ones, `6.97, 2.0` for the blue ones.

However, if we do not know which points come from which distribution, the situation is rather bleak:



We are looking for $\mu_1, \sigma_1, \mu_2, \sigma_2$. Bayesian learning suggests trying to find the MAP hypothesis.

- hypothesis space: $H = (\mathbb{R}, \mathbb{R}, \mathbb{R}, \mathbb{R})$
- data: $d = (x_1, ..., x_n)$ drawn from the two probability distributions.

We want $h \in H$ which is $\text{argmax } p(h \mid d)$.

If we have a uniform prior on H, then this is equivalent to $\text{argmax } p(d \mid h)$, i.e., we are looking for the ML hypothesis.

How can we compute $p(d \mid h)$? Since the data was generated from independent draws, we have

$$p(d \mid h)$$
$$=$$
$$p(x_1, ..., x_n \mid h)$$

```
=
   p(x₁ | h) * ... * p(xₙ | h)
```

We can see that, for large `n`, this quantity will be vanishingly small. In order to help with the computations, it is customary to maximise the *log-likelihood* instead:

```
   ln p(d | h)
=
   ln (p(x₁ | h) * ... * p(xₙ | h))
=
   ln p(x₁ | h) + ... + ln p(xₙ | h)
```

If we know that `xᵢ` has been drawn from distribution `j`, we can compute

```
   p(xᵢ | h) = pdf(xᵢ, μⱼ, σⱼ) = (1/√(2 * π * σⱼ²)) * exp(- (x - μⱼ)²/(2 * σⱼ²))
```

However, we do not know which distribution `xᵢ` was chosen from.

The EM algorithm deals with this situation by considering the missing information as "hidden data". That is, we take

- data: `d = ((x₁, z₁₁, z₁₂) ..., (xₙ, zₙ₁, zₙ₂))`, where `zᵢⱼ ∈ {0, 1}` is `1`, if `xᵢ` has been drawn from distribution `j`, and `0` otherwise.

Remark: This encoding of the choice of distribution is called "one-hot encoding". It has the advantage that the likelihood can be expressed as

```
   p(xᵢ, zᵢ₁, zᵢ₂ | h) = (1/√(2 * π * σ₁²)) * exp(- zᵢ₁ * (x - μ₁)²/(2 * σ₁²)) + (1/√(2 * π
```

Since the data itself now has an unknown part, it becomes a random variable. We can no longer reasonably want to maximise `ln p(d | h)`, however, we can ask for maximising

```
E (ln p(d | h))
```

This turns out to be relatively easy:

```
   E (ln p(d | h))
=
   E (ln p(x₁, z₁₁, z₁₂ | h) + ... + ln p(xₙ, zₙ₁, zₙ₂ | h))
=
   E (ln p(x₁, z₁₁, z₁₂ | h) + ... + E (ln p(xₙ, zₙ₁, zₙ₂ | h)
```

But

```
   E (ln p(xᵢ, zᵢ₁, zᵢ₂ | h)
=
   E (ln ((1/√(2 * π * σ₁²)) * exp(- zᵢ₁ * (xᵢ - μ₁)²/(2 * σ₁²)) + (1/√(2 * π * σ₂²)) * exp(
=
   E (ln 1/√(2 * π * σ₁²) - zᵢ₁ * (xᵢ - μ₁)²/(2 * σ₁²)) + ln 1/√(2 * π * σ₂²) - zᵢ₂ * (xᵢ -
=
   (ln 1/√(2 * π * σ₁²) + ln 1/√(2 * π * σ₂²) - E(zᵢ₁ * (xᵢ - μ₁)²/(2 * σ₁²)) + ln 1/√(2 * π
```

$$=$$

$$(\ln 1/\sqrt{(2 * \pi * \sigma_1^2)} + \ln 1/\sqrt{(2 * \pi * \sigma_2^2)} - E(z_{i1}) * (x_i - \mu_1)^2/(2 * \sigma_1^2) + \ln 1/\sqrt{(2 * \pi$$

We have reduced the computation of $E\ (\ln p(d \mid h))$ to that of $E(z_{i1})$ and $E(z_{i2})$. But these are easy: they are just the probability that x_i comes from the red (respectively blue) distribution, given the hypothesis h:

$$E(z_{i1}) = p(x_i \mid \mu_1, \sigma_1) / (p(x_i \mid \mu_1, \sigma_1) + p(x_i \mid \mu_2, \sigma_2)$$

The second crucial idea of the EM algorithm is to maximise $E\ (\ln p(d \mid h))$ *iteratively*. We consider an initial h, and then compute the corresponding value of $E\ (\ln p(d \mid h))$. This will involve finding out $E(z_{ij})$. We then assume that the values of the hidden variables are equal to the expected values. This gives us “complete” data, for which we can maximise the log-likelihood (not just the expected log-likelihood). We thus obtain a new h, and we iterate.