Machine Learning Notes

nickhatzis

August 2022

Figure 1: First PC to run excel

Contents

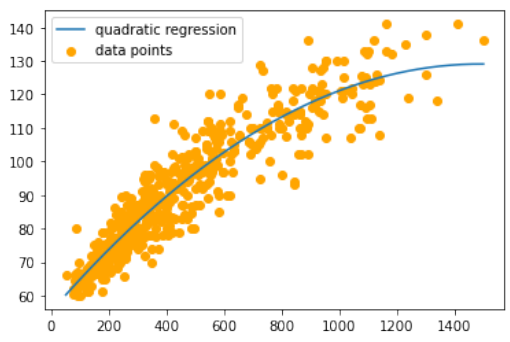
1 ML basics

1.1 Supervised learning

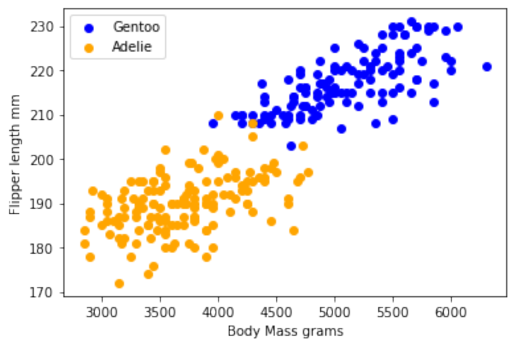
In supervised learning the goal is to learn a mapping from input data to some output space. The training data consist of examples (x,f(x)), where x is what we feed into the model and f is the mapping that ideally the model learns.
Supervised learning can further be classified based on the range of f:

- If f(**X**) is discrete then it is called **classification**
- If f(**X**) is continuous then it is called **regression**

Here **X** denotes the set of all potential inputs.



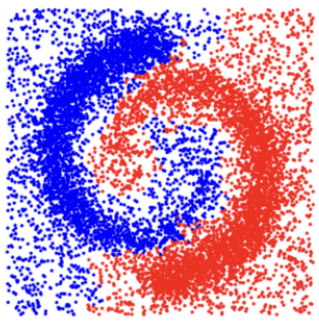
(a) Regression



(b) Classification

1.2 Unsupervised learning

In unsupervised learning the examples come without an output and the task is to find regularities. An example of such regularities is clustering.



(d)

Figure 3: ToMATo cluestering from TDA

1.3 Probably Approximately Correct (PAC) learning

Assume that examples come from some space $\mathbf{X}$ and are sampled from a fixed but **unknown** distribution $\boldsymbol{D}$ on $\mathbf{X}$. Now assume that examples are labeled based on some function f from a **known** concept class $\boldsymbol{C}$

Definition: (($\epsilon - \delta$) PAC learner)
An ($\epsilon - \delta$) PAC learner for a concept class $\boldsymbol{C}$ w.r.t. a distribution $\boldsymbol{D}$ on $\mathbf{X}$, is an algorithm that receives labeled examples $((x_1,f(x_1)),(x_2,f(x_2)),...)$, for some $f \in \boldsymbol{C}$ and $x_i \sim \boldsymbol{D}$, and produces a hypothesis h

such that: $\Pr[\mathrm{err}_{\mathbf{D}}(f,h) \leq \epsilon] \geq 1 - \delta$

Where the randomness is taken over the choice of examples and the algorithm itself.

1.4 VC dimension

For a dataset consisting of N points, there are 2^N binary labelings.
For a concept class $\boldsymbol{C}$ and a finite space $\mathbf{X}$, we say that $\boldsymbol{C}$ shatters $\mathbf{X}$, if for all boolean functions on $\mathbf{X}$ there exists a function in $\boldsymbol{C}$ that simulates that function when restricted on $\mathbf{X}$.

Defintion (VC dimension)
The VC dimension of a concept class $\boldsymbol{C}$ is the maximum dimension of a space that it shatters.

1.5 Model Selection

An **ill-posed-problem** is a problem where the data are not sufficient to find a unique solution. Learning is inherently an ill-posed-problem, so we introduce assumptions to guarantee (up to) a unique solution. The set of assumptions is called as **inductive bias**. One way to introduce inductive bias is assuming a certain hypothesis class H for example polynomial in 1D Regression.
Learning is not possible without inductive bias, but choosing the correct ammount of bias is called **Model selection**, ie. choosing between H. Using our model to predict new data points is called **generalization**. Ideally the complexity of H matches the complexity of the underlying labeling function, however we have two cases:

- If H is less complex than the underlying function is called **underfitting**
- if H is more complex, it is called **overfitting**

There is a triple trade off between:

- The complexity of H
- The number of training data points
- The generalization error

Using a subset of the examples to test the generalization ability is called **cross-validation** and the subset itself is called **validation set**

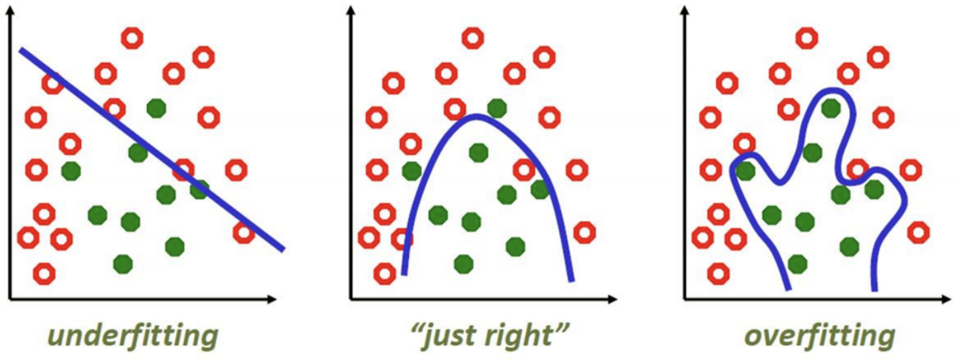


Figure 4: Model selection

2 Regression

2.1 Polynomial Regression

In regression we assume that the examples (x, label(x)) come as follows:

label(x) = f(x) + noise

Where f is an unkown but fixed function. Our goal is to approximate f by our model g (in this case g is a polynomial). Given a training data set $\boldsymbol{X}$ we define the empirical error:

$\mathbf{E}(g|\boldsymbol{X}) = \frac{1}{N}\sum_{i=1}^{N}[\text{label}(x_i) \text{ - } g(x_i)]^2$

Our goal is to minimize the empirical error. Under the assumption that g is linear, then g is completely defined by its coefficients, ie. $g(x) = w_0 + w_1x_1 + ... + w_nx_n$
We take partial derivatives with the respect to $w_n, ..., w_0$ and we choose said coefficients by setting each partial derivative to zero. For higher order polynomials similar approaches are used.

```
from sklearn.linear_model import LinearRegression

X = np array of features
Y = np array of labels

model = LinearRegression().fit(X, Y)
model.coef_, model.intercept_            #gives coefficients, intercept



Alternatively:

import numpy as np

np.polyfit(X, Y, degree = n)    #fits polynomial of degree n of least squares
>>> vector of coefficients

np.poly1d(coefficients)         #produces a polynomial of said coefficients

m = np.poly1d([1, 2, 3])
print(np.poly1d(p))
>>> 1 x^2 + 2 x + 3
m(0)
>>> 3
```

2.2 Logistic regression

Logistic regression , contrary to Polynomial Regression, is a classification model. It can be extended for more than two categories, but we will discuss only 2 here.
Logistic regression operates under the assumption that for an example ($\mathbf{x}$, label($\mathbf{x}$)), label($\mathbf{x}$) = outcome y, follows a sigmoid function of a polynomial.
.

P(label($\mathbf{x}$) = y | $\mathbf{x}$) $= \dfrac{1}{1+e^z}$

Where z($\mathbf{x}$) $= w_0 + w_1x_1 + ... + w_nx_n$

For a training set $\mathbf{X}$, we tune the model by minimizing the Log-loss function:

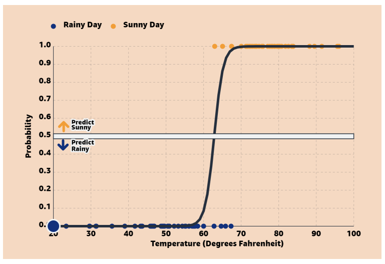


Figure 5: Assumption

Log-loss($\mathbf{X}$) = $\sum_{x \in \mathbf{X}}$ -(label(x)*log(p$_x$(x)) + (1 - label(x))*log(1 - p$_x$(x))

Intuitively this model forces p$_x$ to be as close to either 0 or 1, this can be seen on the y-axis of figure 5.
To tune this model we can instead maximize the Log-likelihood function:

Log-likelihood($\mathbf{X}$) = $\sum_{x \in \mathbf{X}}$ (label(x)*log(p$_x$(x)) + (1 - label(x))*log(1 - p$_x$(x))

Which is done by differentiating and setting derivatives to zero.

```
from sklearn.linear_model import LogisticRegression

X = np array of features
Y = np array of labels
model = LogisticRegression().fit(X,Y)

model.predict(test)
>>>predicts array test

model.get_proba(X)
>>> returns a vector of p_x for each x in X

model.get_log_proba(X)
>>> returns a vector of log(p_x) for each x in X
```

3 Decision Trees

Decision trees are a classification algorithm.
A decision tree of depth = n, is a set of binary rules of n levels that at the end give a prediction of the label of a point $\mathbf{x}$. Decision trees are often visualized as graphs.
Decision trees are calculated based on a greedy approach of some appropriate metric. The said appropriate measures are Entorpy (A measure of randomness of the data set) or Gini impurity (a measure of mislabeling a data point if we were to label according to class ratios)
Let P be the probability distribution on the sample classes, with P(belong class i) = p$_i$:

Entropy(P) = -$\sum_i$ p$_i$*log$_2$(p$_i$)

Gini(P) = 1 -$\sum_i p_i^2$

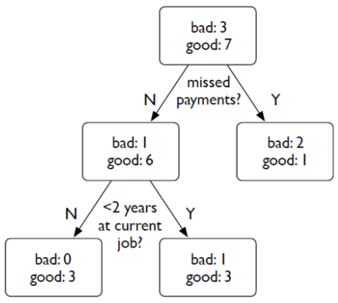
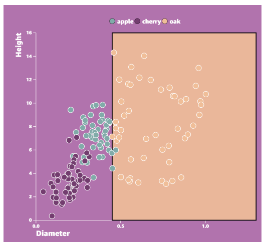
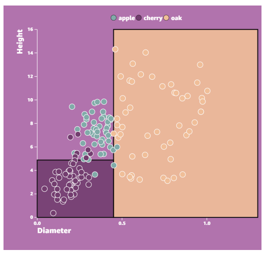


Figure 6: A decision tree

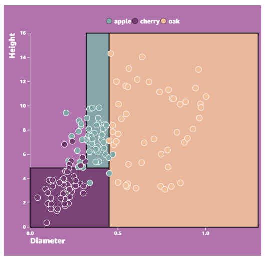
The algorithm performs a search that checks of the greatest change in the metric of choice ie.Metric at parent vertex - the weighted average of the Metric in the children vertices, and assigns a split at that point.



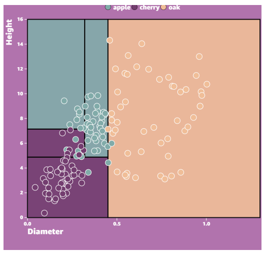
(a) Split 1



(b) Split 2



(c) Split 3



(d) Split 4

```
from sklearn import tree

model =
tree.DecisionTreeClassifier(criterion = k, max_depth = n) #k = 'gini', 'entropy'

X = np array of features
Y = np array of labels
model.fit(X,Y)      #fit data

tree.plot(model)
>>>plots tree as graph

model.predict(vector)
>>>predicts vector
```

4 k Nearest Neighbors

Definition (non-Parametric models)
Non-parametric models are models that make no assumptions on the distribution of the data. Usually this comes from the assumption that the distribution parameters of the data are infinite dimensional.
In contrast parametric models assume a finite number of parameters in the data distribution that they try to optimize or learn

kNN is a non-parametric, supervised classification algorithm. The algorithm requires an input of k (how many neighbors to be checked) and a metric (a way to identify said neighbors, usually Euclidean distance). The algorithm computes the k nearest neighbors for a point we want to classify and decides based on the majority, with equality meaning undecidable or random assignment.

```
from sklearn.neighbors import KNeighborsClassifier

X = np array of data
Y = np array of labels

model = KNeighborsClassifier(n_neighbors=k)

model.fit(X, Y)   #fits model to data

model.predict(vector)
>>>predicts vector
```

5 Naive Bayes

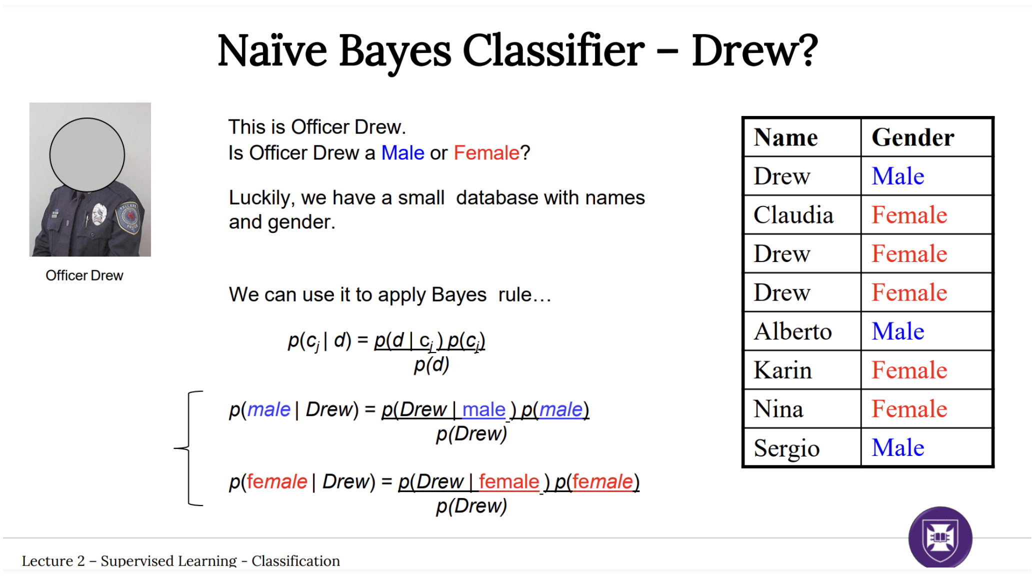
Naive Bayes classifier is a classifier based on Bayes theorem:

$$\mathrm{P}(B|A) = \frac{P(B|A)P(A)}{P(B)}$$

Given a training data set of labeled examples, the classifier works as follows:

- Given a data point $\mathbf{x}$, for each class C_i calculate
- $\dfrac{P(x|C_i) * P(C_i)}{P(x)}$, where the denominator can be ignored as it is constant for all classes
- Find the maximum value over all classes and that is your guess

Worked example:



Naïve Bayes Classifier – Drew?

Officer Drew

This is Officer Drew.
Is Officer Drew a Male or Female?

Luckily, we have a small database with names and gender.

We can use it to apply Bayes rule…

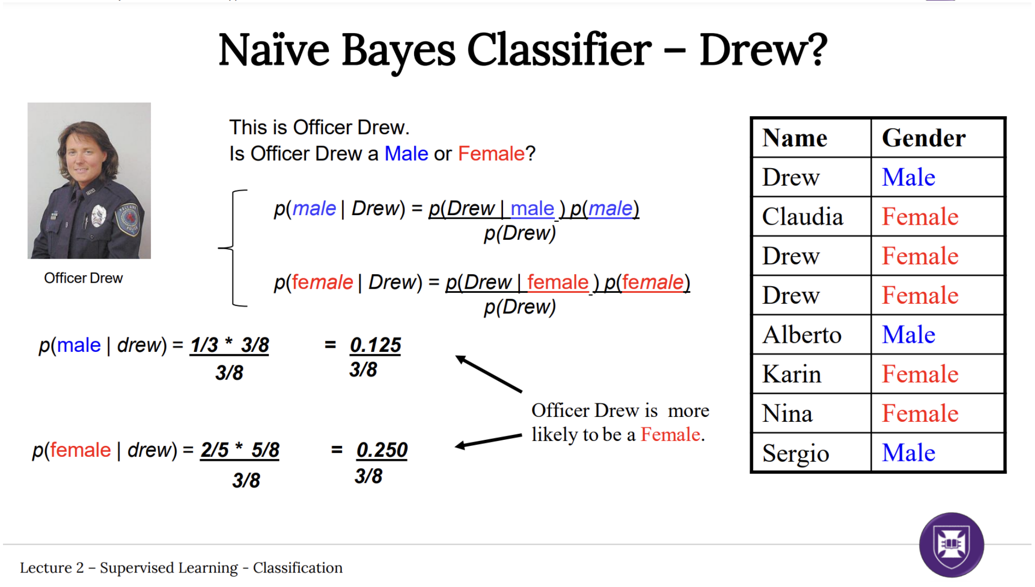
$$p(c_j \mid d) = \frac{p(d \mid c_j)\, p(c_j)}{p(d)}$$

$$p(male \mid Drew) = \frac{p(Drew \mid male)\, p(male)}{p(Drew)}$$

$$p(female \mid Drew) = \frac{p(Drew \mid female)\, p(female)}{p(Drew)}$$

Name	Gender
Drew	Male
Claudia	Female
Drew	Female
Drew	Female
Alberto	Male
Karin	Female
Nina	Female
Sergio	Male

Lecture 2 – Supervised Learning - Classification



Naïve Bayes Classifier – Drew?

Officer Drew

This is Officer Drew.
Is Officer Drew a Male or Female?

$$p(male \mid Drew) = \frac{p(Drew \mid male)\, p(male)}{p(Drew)}$$

$$p(female \mid Drew) = \frac{p(Drew \mid female)\, p(female)}{p(Drew)}$$

$$p(male \mid drew) = \frac{1/3 * 3/8}{3/8} = \frac{0.125}{3/8}$$

$$p(female \mid drew) = \frac{2/5 * 5/8}{3/8} = \frac{0.250}{3/8}$$

Officer Drew is more likely to be a Female.

Name	Gender
Drew	Male
Claudia	Female
Drew	Female
Drew	Female
Alberto	Male
Karin	Female
Nina	Female
Sergio	Male

Lecture 2 – Supervised Learning - Classification

5.1 Multiple features and Laplacian smoothing

Naive Bayes operates under the assumption that in case of multiple features we have independence.
For example:

$$P(A \cap B|C) = P(A|C) * P(B|C)$$

This can cause the unwelcome side effect of $P(A|C) = 0$ for some attributes. In this case (or as a prevention measure) we introduce Laplacian smoothing of parameter α:

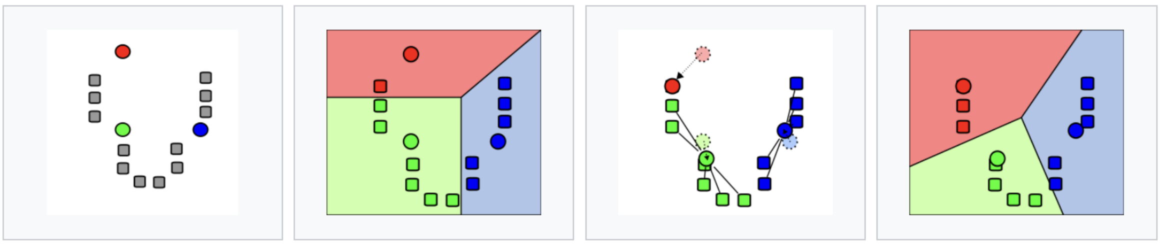
- Instead of $P(A|C) = \dfrac{|A \cap C|}{|C|}$
- We compute $P(A|C) = \dfrac{|A \cap C| + \alpha}{|C| + \alpha|P|}$, where P is the unique mutually exclusive partition that contains A. For example if A = 'has blue eyes', then P might be {'has blue eyes','has brown eyes'}

6 kMeans

kMeans is a non-parametric, unsupervised clustering method that operates as follows:

- We decide on a metric and some value of k
- We randomly set k class centers on the space we are working
- We compute ownership of points to class centers. A point 'belongs' to a class centers if the distance is minimum over all class centers
- We move the class centers to the mean of their ownership points and repeat from the last step
- When convergence is reached, ie. class centers didn't move, we stop the algorithm

Demonstration of the standard algorithm



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Figure 9: Run of the algorithm

```
from sklearn.cluster import KMeans

X = np array of data

 model = KMeans(n_clusters = k)     #choose k here
 model.fit(X)                       #Runs the algorithm on X

 model.labels_
 >>>Returns the labels of points in X

 model.predict(vector of points)
 >>> predicts vector of points
```

7 Hierarchical clustering

This is a non-parametric, unsupervised clustering technique. The output of the process is a dendrogram, a tree showcasing at which distance clusters in the dataset merge.

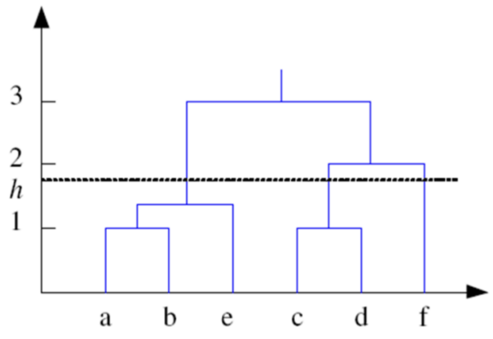


Figure 10: Dendrogram, from the Greek word for tree

We choose a metric and assume that at distance zero, each datapoint is a cluster of itself. Assume ϵ denotes the distance of how far we are looking at a given step. We assign an edge between two points if their distance ϵ and we merge two clusters if their average distance $< \epsilon$ (other rules can be used here as well, example min or max distance between cluster points).
We can also proceed in two directions, we can start from the N individual data points and start connecting clusters (agglomerative method, bottom - up), or start from a distance when we have a single cluster and start dividing until we reach N clusters (divisive method, top - down).
Finally we choose a height level (h in Figure 10), split horizontally and decide the number of clusters based on the number of intersections. The clusters themselves can be retreived by an ϵNeighborhood graph on the data set for example.

```
from sklearn.cluster import AgglomerativeClustering


X= np array of data

model = AgglomerativeClustering(n_clusters=n, affinity= metric)
# metric can be 'euclidean', 'l1', 'l2', 'manhattan', 'cosine', or 'precomputed'

model.fit(X)    #fits data
model.labels_
>>> vector of data cluster labels
```

8 Gaussian mixture models

Definition: (Semiparametric models)
Is a model that assumes that the data come from distinct groups, and each group can be described by a parametric model. However that data do not come with labels to indicate which group is which.

8.1 Refresher on probability

Remember that for a mutually exclusive partition of events A_i ie.:

- $\cup_i A_i = \Omega$
- $A_j \cap A_i = \emptyset$, for all $i \neq j$

The following holds:
$P(x) = P(x \cap \Omega) = P(x \cap (\cup_i A_i)) = P(\cup_i (A_i \cap x)) = \sum_i P(A_i \cap x)$

Where the last equality holds directly from properties of measures (for the 2 people that have done measure theory)

8.2 Setup assumptions

In Gaussian mixture models, we assume that data comes from K classes (hyperparameter) and each class is normally distributed. Following the previous section we have that the probability to see a data point:

$P(x) = \sum_i^K P(x \cap C_i) = \sum_i^K P(x|C_i)P(C_i)$, with $P(x|C_i) \sim N(\text{mean}_i,\text{variance}_i)$

Assuming the data is iid we will want to estimate $\{P(C_i),\text{mean}_i,\text{variance}_i\}$ for all i, from the data. If we had labels on the data, then this task would be trivial $P(C_i)$ would be approximated by its proportion, while the mean and the variance are approximated by sample mean and variance.

8.3 Expectation Maximization Algorithm (EM Algorithm)

Let Φ be the set of parameters to be estimated, then, by previous section, the log likelihood function:

$L(\Phi|X) = \sum_i p(x_i|\Phi)) = \sum_i log \sum_{j=1}^K P(x_i|C_j)P(C_j)$

This function cannot be maximized by analytic methods so we will use an iterative method. More precisely the problem has two random variables, one is X the data sample, two is Z which denotes the class labels of the data points and is hidden. In particular we introduce $L_c(\Phi|X,Z)$ the complete likelihood function on both RVs. Since Z is hidden, we cannot work with L_c directly, but we can work with its expectation. Let Φ_l denote our parameter guess at iteration step l, then for the next step:

Expectation step: $Q(\Phi|\Phi_l) = E[L_c|X]$, expectation over Φ_l
Maximization step: $\Phi_{l+1} = \text{argmax}_\Phi Q(\Phi|\Phi_l)$

The algorithm works based on a proof that an increase in the expectation step means an increase in the incomplete likelihood $L(\Phi|X)$. This algorithm is initialized by kMeans until we have some estimates for the class mean centers

For more, go read the book

```
from sklearn.mixture import GaussianMixture

X = np array of data

model =GaussianMixture(n_components = n).fit(X)

model.means_
>>>returns means

model.predict(X)
>>>returns predicted labels
```