Unsupervised Learning Techniques

By Waad ALMASRI

Unsupervised learning - tasks

- Dimensionality reduction
- Clustering: group similar instances into clusters, used for data analysis, customer segmentation, recommendation systems, search engines, image segmentation, semi-supervised learning, dimensionality reduction, etc.
- Anomaly or outlier detection: learn what “normal” data looks and then use that to detect abnormal instances, called anomalies, used in fraud detection, in quality inspection, in identifying new trends in time series, etc.
- Density estimation: estimate the probability density estimation (PDF) of the random process that generated the dataset, used for anomaly detection (instances located in low-density region are most probably outilers)

Clustering

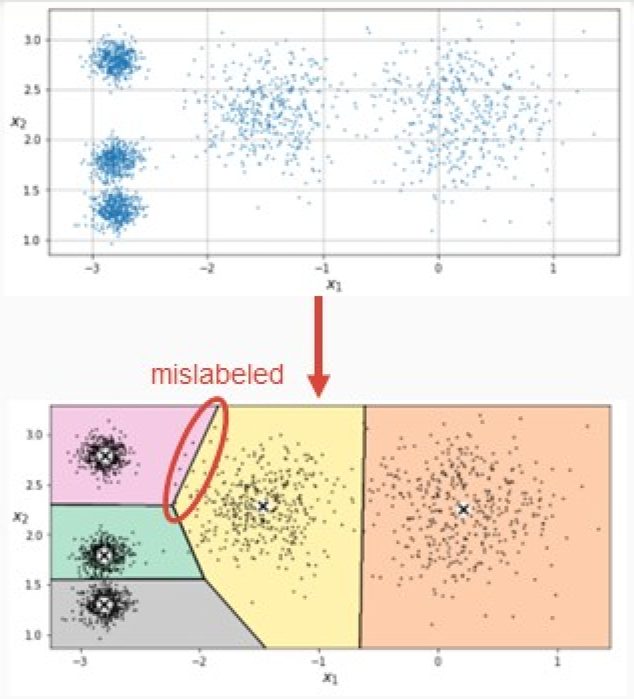
- Grouping instances
- In the context of clustering, an instance's label is the index of the cluster to which the algorithm assigns this instance; it is not to be confused with the class labels in classification, which are used as targets.

K-means - History

- Created by Stuart Lloyd at Bell labs in 1957
- Published in 1982
- In 1965, Edward W. Forgy published a similar algorithm
- → K-means is referred to as the Lloyd-Forgy algorithm

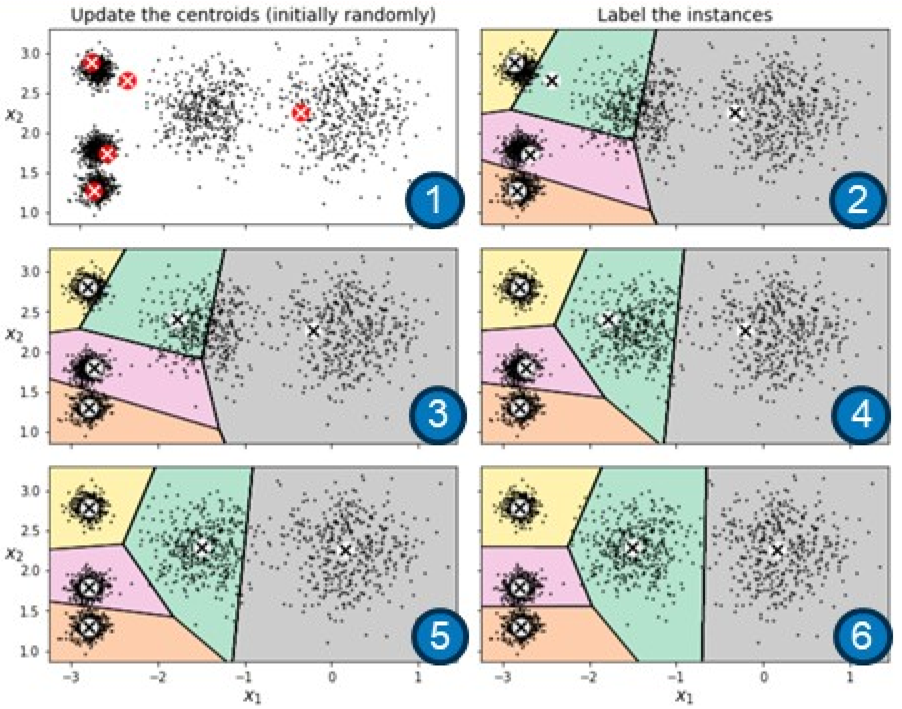
K-means

- Training:
  - Takes as input the number of clusters k that the algorithm must find
  - Finds each cluster's center
  - Assign each data instance to the closest cluster (also called blob)
- Inference: A new instance is assigned it to the cluster whose centroid is closest.
- Sklearn : KMeans
- Go to notebook



K-means - Training

1. Pick k instances randomly from the dataset and set them as the initial centroids
2. For every training instance, compute the distance between the instance and all the centroids, then assign the instance to the cluster of the closest centroid
3. Update the centroid locations using the instances in each cluster
4. Repeat step 2 and 3 until the locations of the centroids stop changing.



Centroid initialization methods

1. We have a prior knowledge of where approximatively the centroids are located: from a visualization or from running a previous clustering algorithm (*init hyperparameter*)
2. We run the algorithm multiple times with different random initializations, and we keep the best solution (*n_init* hyperparameter, sklearn will run the algorithm *n_init* times and will keep the best solution based on the **inertia** performance metric (the lowest inertia).

Inertia = the sum of squared distances between the instances and their closest centroids. kmeans.inertia_ , kmeans.score(X) = negative inertia

Centroid initialization methods (2) – k-means++ by David Arthur & Sergei Vassilvitskii (2006)

1. Pick one centroid c_i, randomly sampled from the dataset.
2. For each training instance, compute the squared distance $(D(x_i)^2)$ to the nearest centroid. $D(x_i)$ is the distance between a point x_i and the closest centroid, i.e., we compute the distance between x_i and all the centroids and choose the centroid with the minimum Euclidean distance.
3. Select the next centroid from the data points with a probability $\frac{D(x_i)^2}{\sum_{j=1}^{n} D(x_j)^2}$. The higher the distance $D(x_i)$, the higher the probability. In other words, points that are farther away from existing centroids are more likely to be chosen as the next centroid.
4. Repeat steps 2 & 3 until k centroids are chosen.
5. Then, we use the selected k centroids and run the normal k-means algorithm (slide k-means – Training)

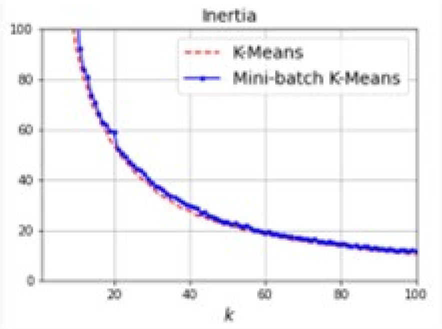
NB: KMeans in sklearn uses this initialization method.

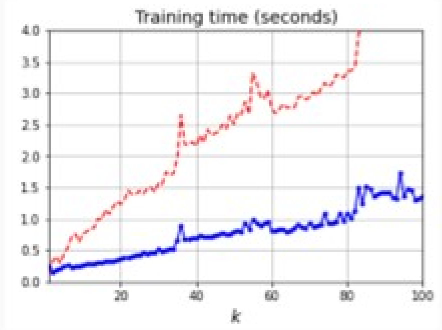
Accelerated K-means & mini-batch K-means

Accelerated kmeans 2003 : On large datasets with many clusters, the algorithm can be accelerated by avoiding many unnecessary distance calculations.
Elkan achieved thy by exploiting the triangle inequality (i.e., a straight line is always the shortest distance between 2 points) and by keeping track of lower and upper bounds for distances between instances & centroids.
However, Elkan's algorithm does not always accelerate the training, and sometimes it can even slow down training significantly; it depends on the dataset. If you want to use it, you should set the hyperparameter algorithm to Elkan.

Mini-batch kmeans: 2010 by David Sculley. Instead of using the full dataset at each iteration, the algorithm is capable of using mini-batches, moving the centroids just slightly at each iteration. This speeds up the algorithm (typically by a factor of 3 to 4) and makes it possible to cluster huge datasets that do not fit in memory.
Sklearn: MiniBatchKMeans

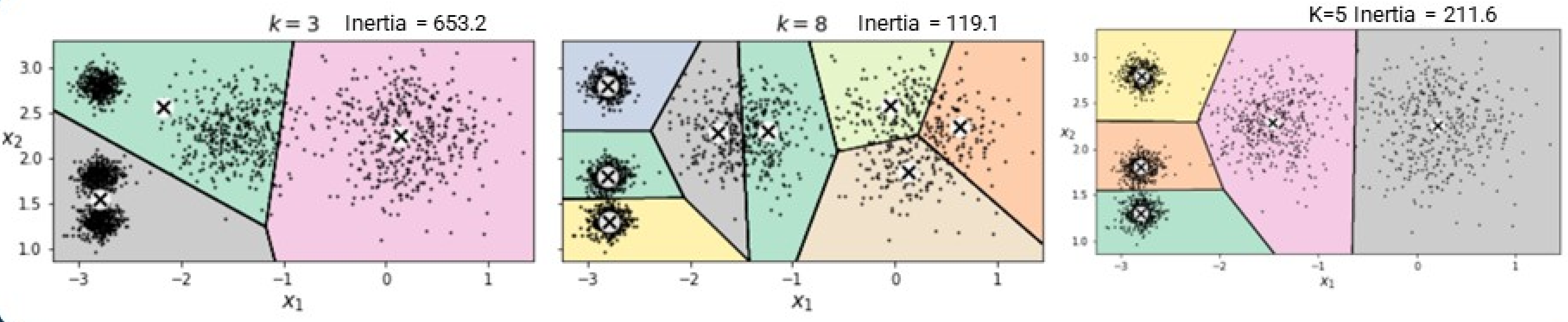
The figure shows that mini batch kmeans is 3,5 times faster and performance is good, similar to kmeans here.





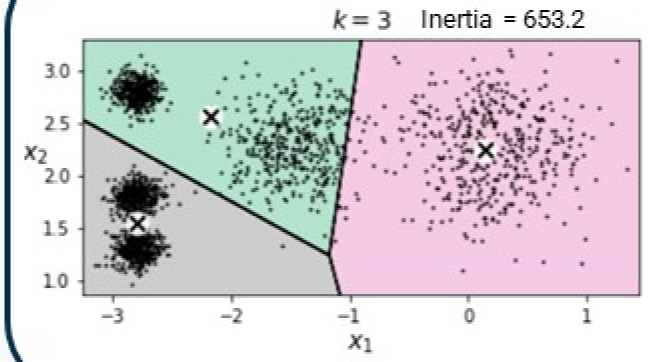
Finding the optimal number of clusters - k

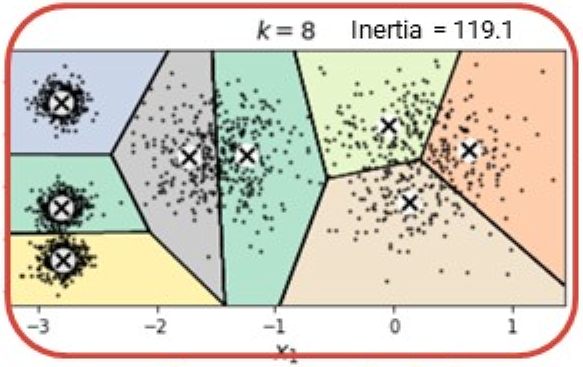
1. Test several models with different k and choose the one with the lowest inertia



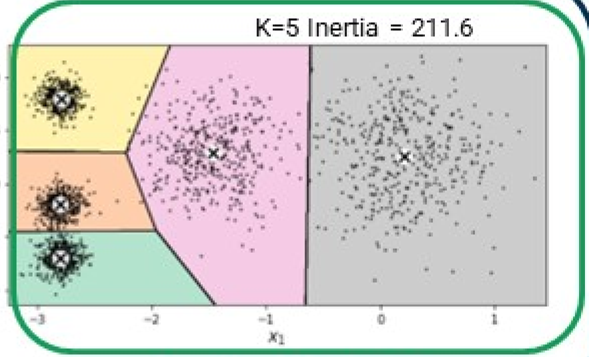
Finding the optimal number of clusters - k

1. Test several models with different k and choose the one with the lowest inertia





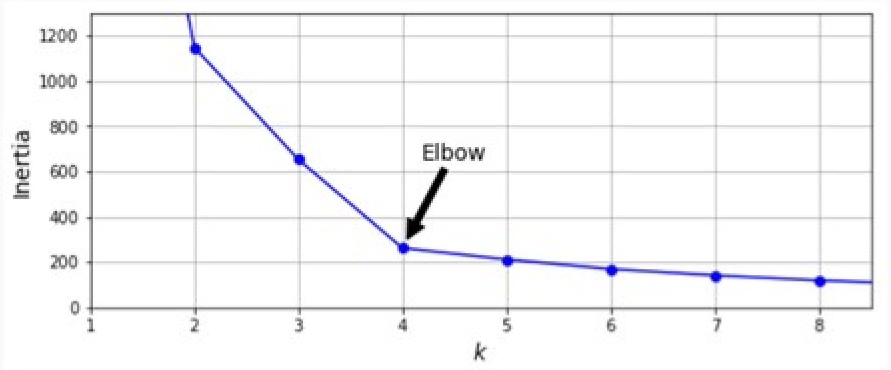
Lowest inertia but worst model



Low inertia but best model

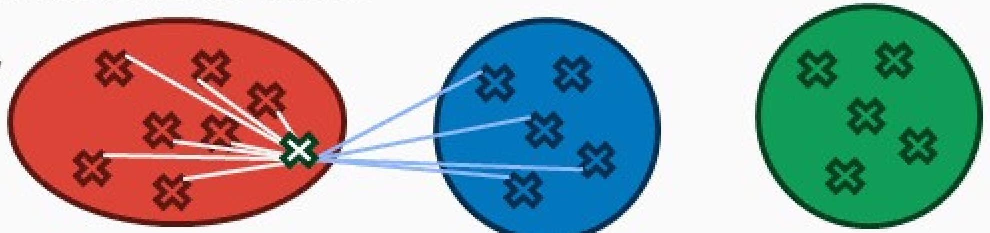
Finding the optimal number of clusters - k

1. ~~Test several models with different k and choose the one with the lowest inertia~~. Inertia decreases, indefinitely, inversely proportional to k. Because the more the clusters, the closer the instances will be to the centroids, the lower the inertia → Inertia is not a good metric to identify the optimal k. If we plot k versus the inertia, i.e., we train m models (here 8) with an input $k = 1 \ldots \ to$ 8 and compute the inertia, we might end up assuming that $k = 4$ is a good model, $k = 5$ and 6 are very similar models, and $k = 8$ is the best model , while the optimal k is 5.



Finding the optimal number of clusters - k - The silhouette score

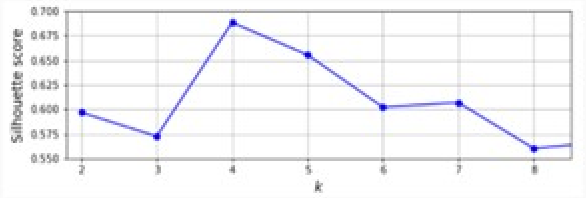
Precise but more computationally expensive approach.
The silhouette score = the mean silhouette coefficient over all the instances.
An instance 's silhouette coefficient is $s_i = \frac{b-a}{\max(a,b)}$ with $-1 \leq s_i \leq 1$
a = the mean distance to the other instances in the same cluster as instance i, called the mean intra-cluster distance
b = the mean nearest-cluster distance, i.e., the mean distance to the instances of the next closest cluster
$s_i = 1$; the instance is well inside its own cluster and far from other clusters
$s_i = -1$; the instance is in the wrong cluster
$s_i = 0$; the instance is close to a cluster boundary
Sklearn: silhouette_score in metrics.


$a = \text{ mean of all the white distances}$

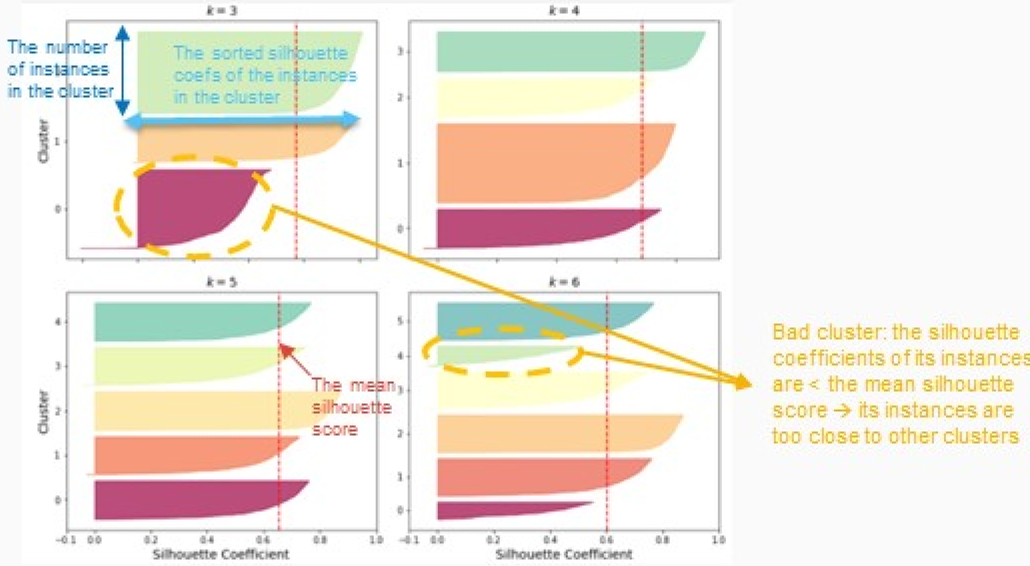
$b = \text{ mean of all the blue distances}$
The closest cluster to the instance in the red cluster is the blue blob

Finding the optimal number of clusters - k - The silhouette score (2)

To find the optimal k, plot the silhouette score versus k.
The figure on the right shows that $k = 4$ is the best choice, $k = 5$ is also a good choice and $k = 6, 7, and\ 8$ is not unlike the inertia technique's results.



A more informative method: plotting the silhouette diagram (figure on the right), i.e., plotting every instance's silhouette coefficient, sorted by the clusters they are assigned to and by the value of the coefficient.
A diagram consists of k knife-like shapes; one knife shape per cluster.



Pros vs Cons of k-means

- Fast and scalable: The computational complexity is linear regarding the number of instances n, the number of clusters k, and the number of features m only when the data has a clustering structure.

- Need to specify the number of clusters k
- Need to run the algorithm several times to avoid suboptimal solutions
- k-means does not perform well when the clusters have very different diameters because, assigning an instance to a cluster and not to another is dependent of **only** the distance to the centroid.
- Also, k-means does not perform well when the clusters have varying sizes, different densities, or non-spherical shapes.
- If the data has no clustering structure, the complexity of the worst-case scenario can increase exponentially with n. In reality, this situation is rare; k-means is known to be the fastest clustering algorithm.
- While convergence is not an issue with k-means, it may converge to a local optimum because its convergence highly depends on the initialization of the centroids' locations.

K-means - Tip

With k-means, features need to be scaled before running the algorithm. Otherwise, the clusters will be stretched, and k-means will perform poorly. NB: scaling the features does not result in spherical clusters but it helps the algorithm converge to a better solution.

K-means can be used as a dimensionality reduction technique and as a clustering algorithm for image segmentation.

K-means as a dimensionality reduction technique

- Instead of hard clustering (i.e., assigning each instance to a single cluster), K-means can be used to give each instance a score per cluster.
- This score = the distance between the instance and the centroid of every cluster or a similarity score (like gaussian RBF)
- In this case, we use the transform() function instead of the predict()
- If X is a matrix of n rows and m columns (i.e., n = total number of training points and m=number of attributes/independent variables), then the output after transform is a matrix of n rows and K columns such that $x_{ik} =$ the distance between instance i and the centroid of cluster k
- if we have a high dimensional dataset (m=number of features/independent variables >>>) then, we end up with a K-dim dataset with $K \lll m$.
- This transformation can be a very efficient nonlinear dimensionality reduction technique.

K-means for Image segmentation

Image segmentation consists of partitioning an image into distinct segments or regions. There are several variants of image segmentation:

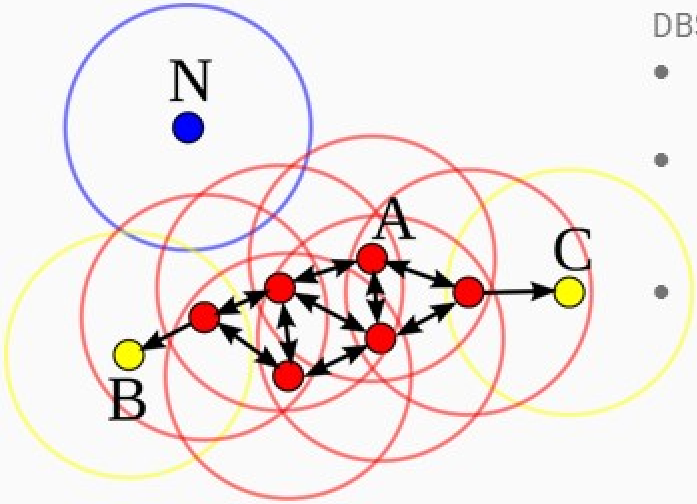
- Color segmentation: pixels of the same colors are merged in the same cluster (segment). For example, satellite images are used to measure deforestation: forests are identified from non-forest areas thanks to the color feature of the satellite image.
- Semantic segmentation: pixels of the same object type are merged in the same segment. For example, object detection in autonomous vehicles uses complex semantic segmentation technique (CNNs, transformers) to identify pedestrians, signalization, route edges, etc.
- Instance segmentation: pixels of the same unique object are merged in the same segment. For example, surveillance system, tracking systems; we need to track one particular subject.

Active Learning - uncertainty sampling

1. The model is trained on the labeled data points collected so far, and this model is used to predict on the unlabeled data points.
2. The data points for which the model is most uncertain (i.e., where the estimated probability is lowest) are given to the expert for labeling.
3. Iterate the process until the performance improvement stops being worth the labeling effort.

Other active learning strategies include labeling the data points that would result in the largest model change or the largest drop in the model's validation error, or the instances that different models disagree on (svm, random forest, etc.)

DBSCAN: Density-Based Spatial Clustering of Applications with Noise



A DBSCAN cluster is defined as a high-density region.
DBSAN works as follows:

- For each instance i, it counts how many instances are located within a small distance epsilon ε from i. This region is the $\varepsilon\text{-}neighborhood$.
- If i has at least $\boldsymbol{min\_samples}$ instances in its $\varepsilon\text{-}neighborhood$ (including itself), then it is considered ***a core instance***. → core instances are located in dense regions. Points A and all red points are core instances.
- All the instances located in the neighborhood of a core instance belong to the same cluster. This neighborhood may include other core instances. ***→ a long sequence of neighboring core instances forms a single cluster.*** Points B & C are not core points but are directly reachable from A → they belong to the same cluster.
- Any instance that is not a core instance and does not have one in its neighborhood is considered an ***anomaly.*** N is neither a core point, nor reachable → it is an anomaly.

DBSCAN – Pros and cons

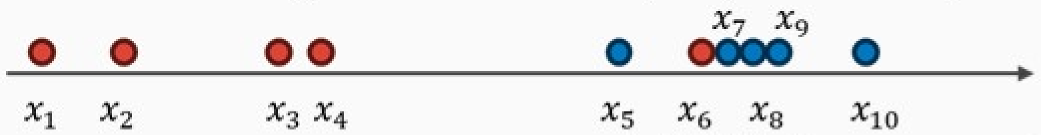
- Pros:
  - DBSCAN is simple and powerful algorithm with 2 hyperparameters (ε and min_samples) to identify clusters of any shape.
  - It is robust to outliers.
- Cons:
  - DBSCAN struggles to capture all the clusters when the density varies significantly across the clusters or when there is no enough low-density region around some clusters.
  - It does not scale well to large datasets; its complexity is $\Theta(n^2m)$; n = total number of training instances and m = total number of features

Gaussian Mixture Model (GMM)

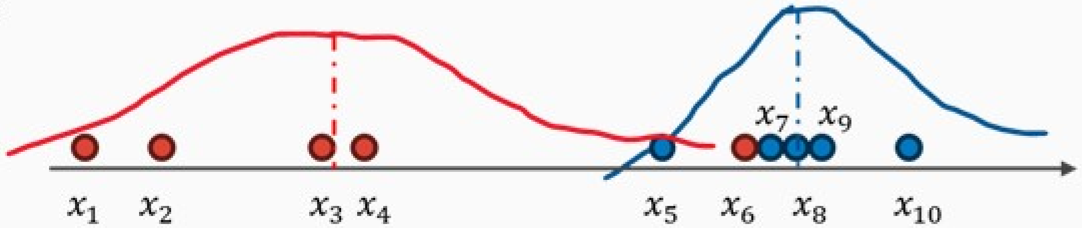
- Probabilistically-grounded model doing soft clustering
- Each cluster is a generative model (Gaussian or Multinomial)
- With unknown parameters (mean, covariance)

GMM in 1D

- Let us suppose we have the following observations $x_1, x_2, \dots, x_n$
- We know that the data came from 2 gaussian sources a & b $(K = 2)$ with unknown parameters (mean and variance)

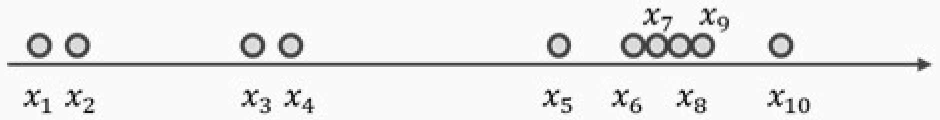


- To estimate (μ_a, σ_a) and (μ_b, σ_b), we simply compute: $\mu_a = \frac{x_1+x_2+x_3+x_4+x_6}{n_a}, n_a = 5$ and $\sigma_a^2 = \frac{(x_1-\mu_a)^2+(x_2-\mu_a)^2+(x_3-\mu_a)^2+(x_4-\mu_a)^2+(x_6-\mu_a)^2}{n_a}$ $and\ \mu_b = \frac{x_5+x_7+x_8+x_9+x_{10}}{n_b}, n_b = 5$ and $\sigma_b^2 = \frac{(x_5-\mu_b)^2+(x_7-\mu_b)^2+(x_8-\mu_b)^2+(x_9-\mu_b)^2+(x_{10}-\mu_b)^2}{n_b}$
- Result:

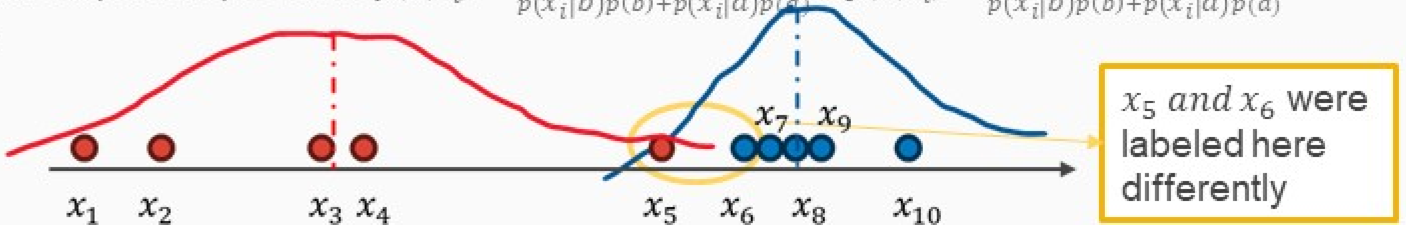


GMM in 1D (2)

- Let us suppose we have the following observations $x_1, x_2, \dots, x_n$
- This time we do not know the source of the observations, but we know the sources' parameters (mean and variance)



- In this case, we can guess whether a data point x_i is more likely to belong to the gaussian distribution a or b by computing the probability $p(x_i|a)$ and $p(x_i|b)$
- $p(x_i|a) = \frac{1}{\sqrt{2\pi\sigma_a^2}} \exp(-\frac{(x_i-\mu_a)^2}{2\sigma_a^2})$ and $p(x_i|b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp(-\frac{(x_i-\mu_b)^2}{2\sigma_b^2})$
- And then compute the posteriors $p(b|x_i) = \frac{p(x_i|b)p(b)}{p(x_i|b)p(b)+p(x_i|a)p(a)}$ & $p(a|x_i) = \frac{p(x_i|a)p(a)}{p(x_i|b)p(b)+p(x_i|a)p(a)}$
- Result:



x_7 x_9
x_1 x_2
x_3 x_4
x_5 x_6 x_8 x_{10}
x_5 and x_6 were labeled here differently

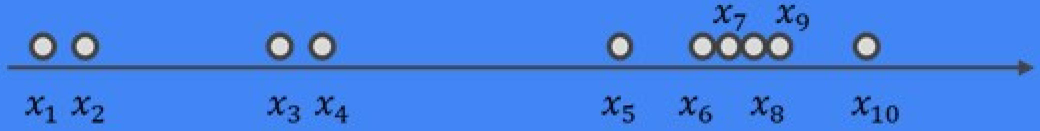
GMM (3)

Chicken Egg problem: In reality, we do not know the sources of each data points (no labelled data points), nor we know the parameters of the sources, and we need one to estimate the other.
So how do we solve this?
Solution = Expectation Maximization Algorithm (EM):

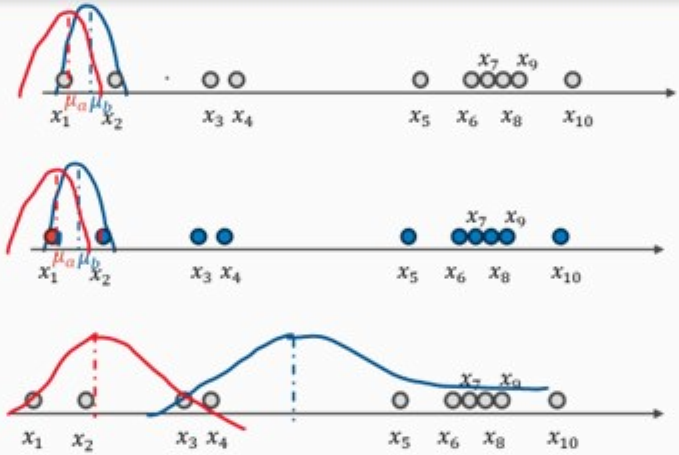
1. Starts with 2 randomly placed gaussians of random parameters (μ_a, σ_a) and (μ_b, σ_b)
2. For each point x_i, compute $p(x_i|a)$ & $p(x_i|b)$ & the posteriors $p(a|x_i)$ & $p(b|x_i)$
3. Adjust (μ_a, σ_a) and (μ_b, σ_b) to fit the points assigned to them using the computed probabilities from the previous step
4. Repeat steps 2 and 3 until convergence.

NB: GMM is called a soft clustering algorithm because it computes probabilities of points belonging to a cluster and does not assign a point to a particular cluster i.e., probabilities are never quantized as 1 and 0, which means in the calculation of the new parameters, every point has an influence on all the clusters, unlike k-means that assigns a point to a particular cluster and will only influence this particular cluster and has no effect on the others.

GMM in 1D (4)



x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}

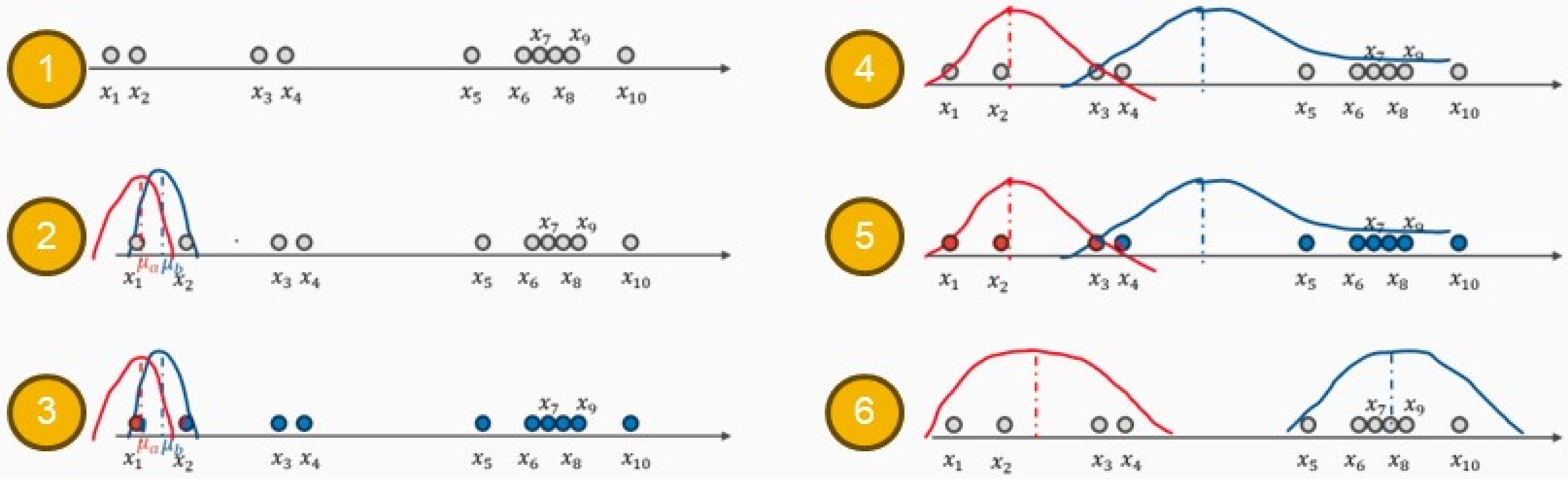


μ_a μ_b
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}

1. Start by placing 2 gaussians of random params in a random position on the line
2. For each point x_i, compute $p(x_i|a)$ & $p(x_i|b)$ and the posteriors $p(a|x_i)$ & $p(b|x_i)$. For example, here, $p(x1|a) \gg p(x_1|b), p(x_2|b) \gg p(x_2|a)$, etc. Thus, $a_1 = p(a|x_1) = \frac{p(x_1|a)p(a)}{p(x_1|b)p(b)+p(x_1|a)p(a)}$ & $b_1 = p(b|x_1) = \frac{p(x_1|b)p(b)}{p(x_1|b)p(b)+p(x_1|a)p(a)}$. The priors $p(a)$ and $p(b)$ are assumed constant = 0.5. a_i & b_i could be perceived as the confidence that point $x_i \,\epsilon\, Gaussian\ a\ or\ b$; on the figure, it is the coloring. *NB: here since there is only 2 gaussians* → $b_i = 1 - a_i$
3. Adjusting (μ_a, σ_a) and (μ_b, σ_b): $\mu_a = \frac{a_1x_1+a_2x_2+\ldots+a_{10}x_{10}}{a+a+\ldots+a_{10}}$ and $\sigma_a = \frac{a_1(x_1-\mu_a)^2+a_2(x_2-\mu_a)^2+\ldots+a_{10}(x_{10}-\mu_a)^2}{a_1+a_2+\ldots+a_{10}}$; similarly for (μ_b, σ_b). Soft clustering thanks to the posteriors a_i and b_i.
4. Repeat until convergence

NB: we could estimate the priors $p(a) = \frac{a_1+a_2+\ldots+a_n}{n}$ & $p(b) = \frac{b_1+b_2+\ldots+b_n}{n}$ with here $n = 10$. The priors estimate the proportion of the points the a or b distribution describes; these priors are also called weights of the clusters. Practically, computing the priors hinders the convergence of the GMM; in this example here, the blue gaussian would have grabbed all points and no points left to the red distribution if we have used the computed priors on the second iteration.

GMM in 1D (5)


1
2
3
4
5
6
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}
μ_a μ_b

GMM in D>1

- Data with D attributes, from K Gaussian sources
- Iteratively estimate the params (μ, σ):
  - Start with random K Gaussians (random positioning, i.e., ransom params)
  - Compute the probabilities $p(\vec{x_i}|c) = \frac{1}{\sqrt{2\pi|\Sigma_c|}} \exp(-\frac{1}{2}(\vec{x_i} - \vec{\mu_c})^T \Sigma_c^{-1}(\vec{x_i} - \vec{\mu_c}))$, then the posteriors (weights) $p(c|\vec{x_i}) = \frac{p\left(\vec{x_i}\middle|c\right)p(c)}{\sum_{c'=1}^{K} p\left(\vec{x_i}\middle|c'\right)p(c')}$; $\mathrm{c} = 1, 2, \ldots \mathrm{K}$
  - Compute the priors $p(c) = \frac{1}{n}\sum_{i=1}^{n} p(c|\vec{x_i})$; Prior = the % of instances coming from source c
  - Adjust the expected value of attribute j from source c: $\mu_{c,j} = \sum_{i=1}^{n}\left(\frac{p(c|\vec{x_i})}{np(c)}\right) x_{i,j}$ $with\ x_{i,j}$= value of the jth attribute in the ith data point and $p(c|\vec{x_i})$= the weight of x_i's association with Gaussian c; $\mathrm{j} = 1,2, \ldots, \mathrm{d}$ & $\mathrm{c} = 1,2, \ldots, \mathrm{K}$
  - Adjust the covariance between attributes j & n (how much correlated are attributes j & n in source/cluster c) $(\Sigma_c)_{j,n} = \sum_{i=1}^{n}\left(\frac{p(c|\vec{x_i})}{np(c)}\right)(x_{i,j} - \mu_{c,j})(x_{i,n} - \mu_{c,n})$; Covariances precise the size, shape, and orientation of the cluster.

GMM – How to pick K?

- GMMs are probabilistic models → GMMs converge by maximizing the log likelihood of the data under the fitted mixture model
- $L = \log\big(p(x_1, x_2, \dots, x_n)\big) = \sum_{i=1}^{n} \log(\sum_{c=1}^{K} p(x_i|c)p(c))$
- **Solution 1**: choose K that maximizes L

GMM – How to pick K?

- GMMs are probabilistic models → GMMs converge by maximizing the log likelihood of the data under the fitted mixture model
- $L = \log\big(p(x_1, x_2, \dots, x_n)\big) = \sum_{i=1}^{n} \log(\sum_{c=1}^{K} p(x_i|c)p(c))$
- **Solution 1**: choose K that maximizes L:
    - It is an ill-posed problem: L will continue to grow as long as we increase K → The best solution is when $K = n$, i.e., each point has its own source → bad

GMM – How to pick K?

- GMMs are probabilistic models → GMMs converge by maximizing the log likelihood of the data under the fitted mixture model
- $L = \log\big(p(x_1, x_2, \dots, x_n)\big) = \sum_{i=1}^{n} \log(\sum_{c=1}^{K} p(x_i|c)p(c))$
- **Solution 1**: choose K that maximizes L:
  - It is an ill-posed problem: L will continue to grow as long as we increase K → The best solution is when $K = n$, i.e., each point has its own source → bad
- **Solution 2**: split the data points to train & test sets. Then, for each K, fit the params of the train set and compute the L of the validation set

GMM – How to pick K?

- GMMs are probabilistic models → GMMs converge by maximizing the log likelihood of the data under the fitted mixture model
- $L = \log\big(p(x_1, x_2, \dots, x_n)\big) = \sum_{i=1}^{n} \log(\sum_{c=1}^{K} p(x_i|c)p(c))$
- **Solution 1**: choose K that maximizes L:
    - It is an ill-posed problem: L will continue to grow as long as we increase K → The best solution is when $K = n$, i.e., each point has its own source → bad
- **Solution 2**: split the data points to train & test sets. Then, for each K, fit the params of the train set and compute the L of the validation set
    - Sometimes, we still end up with the best K being n

GMM – How to pick K?

- GMMs are probabilistic models → GMMs converge by maximizing the log likelihood of the data under the fitted mixture model
- $L = \log\big(p(x_1, x_2, \dots, x_n)\big) = \sum_{i=1}^{n} \log(\sum_{c=1}^{K} p(x_i|c)\,p(c))$
- **Solution 1**: choose K that maximizes L:
  - It is an ill-posed problem: L will continue to grow as long as we increase K → The best solution is when $K = n$, i.e., each point has its own source → bad
- **Solution 2**: split the data points to train & test sets. Then, for each K, fit the params of the train set and compute the L of the validation set
  - Sometimes, we still end up with the best K being n
- **Solution 3**: Occam's razor; pick the simplest of all the model that fit
  - Bayes Information Criterion (BIC): $\max_p \{2L - p\log(n)\}$ with L = log likelihood, how well does the model fit the data and p = the number of params learned by the model (the complexity of the model)
  - Akaike Information Criterion (AIC): $\min_p \{2p - 2L\}$
  - Both methods try to balance the likelihood against the complexity of the model
  - NB: with $D - \dim$ and K Gaussians, p = (D mean + $\frac{D\times(D-1)}{2}$ variance)$\times$ K + $(K-1)$ priors
-

GMM in practice

- EM can sometimes converge to poor solutions→ set *n_init* hyperparameter to >1
- When the number of dimensions is >>, or the number of clusters is >>, or the number of data points is <<, EM struggles to converge to optimal solutions. → limit the number of params that the algorithm has to learn by limiting the range of shapes & orientations clusters can have (i.e., imposing constraints on the covariance matrices). This is done by setting the *covariance_type* hyperparameter to either spherical, diagonal, or tied.
    - Spherical = clusters are spherical but can have different diameters (i.e., different variances).
    - Diagonal (diag) = clusters are ellipsoidal but can have any size. However, the ellipsoids' axes must be parallel to the coordinates' axes (i.e., covariance matrices must be diagonal.
    - Tied: clusters are ellipsoidal of the sale size & orientation (all clusters share the same covariance matrix).
    - NB: by default, the *covariance_type* is set to full, which means any cluster can take any shape, size, and orientation
- Instead of manually searching for the best K, we could use the Bayesian Gaussian mixture and set the n_components to a value higher than the optimal number of clusters (per our intuition) and the algorithm will eliminate the unnecessary clusters.
- With GMM, selecting the optimal K cannot be done using inertia or silhouette, like with Kmeans, because these metrics are not reliable when the clusters are not spherical or have different sizes.

GMM in practice (2)

- In practice GMMs are widely used as a pre-processing step to transform data before a follow-up task (e.g., classification). In this case, choosing the best K should be done with respect to the performance of the follow-up model (e.g., classifier) and not to get the max BIC or min AIC.
- GMM for anomaly detection: any instance located in a low-density region can be considered an anomaly; a threshold should be defined.
- If you notice that we end up with too many false positives (good data points considered as anomaly) → lower the threshold
- Conversely, if too many false negatives (bad data points considered normal) → increase the threshold
- → precision/recall trade-off
- NB: GMM tries to fit all data including outliers → if many data points are outliers, the model will learn that this is a normality, which is not the goal. → If this happens, fit the model once and use it to remove outliers. Once this is done, we refit the model on the good data.

GMM – computational complexity

The computational complexity of GMMs depend on the number of instances n, the number of dimensions m, the number of clusters K, and the number of constraints on the covariance matrices.

If the covariance type is spherical → $\Theta(knm)$.

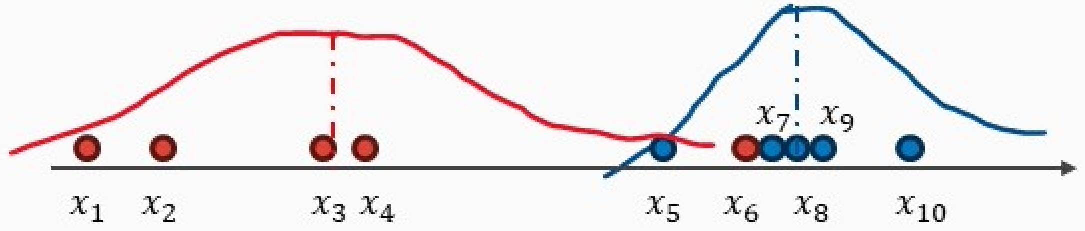
If the covariance type is full or tied → $\Theta(knm^2 + km^3)$ → does not scale well to large number of features ($m \gg$).

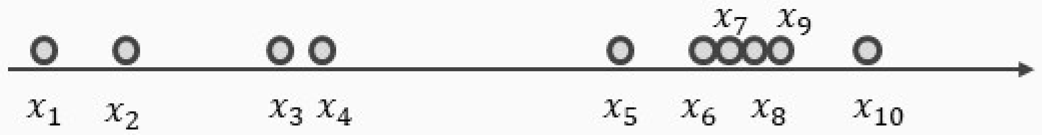
References

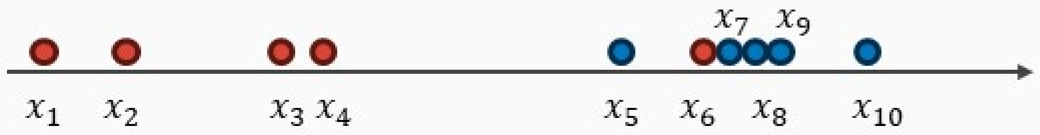
Géron, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc.", 2022.

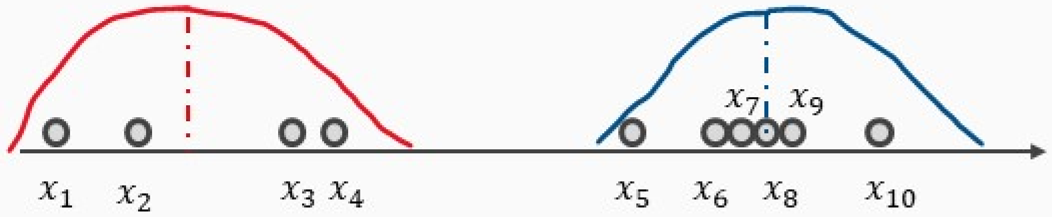
Gaussian Mixture Model (GMM)

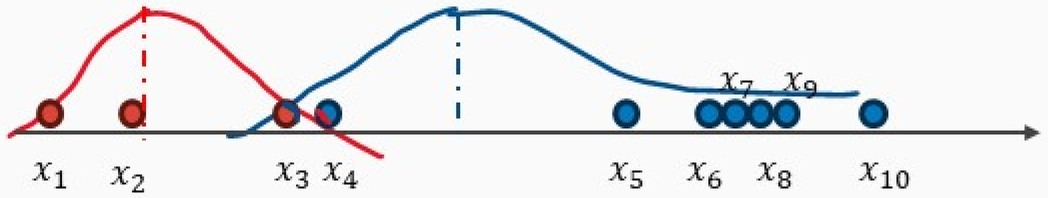
- It is probabilistic model assuming all instances were generated from a mixture of several Gaussian distributions whose parameters are unknown.
- All instances generated from a single Gaussian distribution form a cluster that typically looks like an ellipsoid.
- Each cluster can have a different ellipsoidal shape, size, density, and orientation.
- There are several variants, the simplest one consists of knowing in advance the number k of Gaussian distribution.
- The dataset X is assumed to be generated through the following probas process

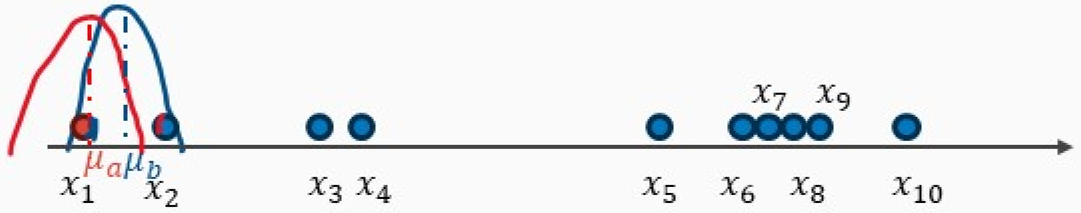

x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}


x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}


x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}


x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}


x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}


μ_a
μ_b
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}