0 A note on notations

Please read the equations carefully, as the capital sigma Σ and the sum operator $\sum$ look quite similar. The sum operator is larger however, and usually has the bounds displayed underneath and over it.

1 Gaussian Mixtures

If we assume $P(X|y)$ is Gaussian, we can represent the data X as a mixture of several gaussian models. This means each point does not belong to a single cluster, but belongs to multiple clusters with a certain probability:

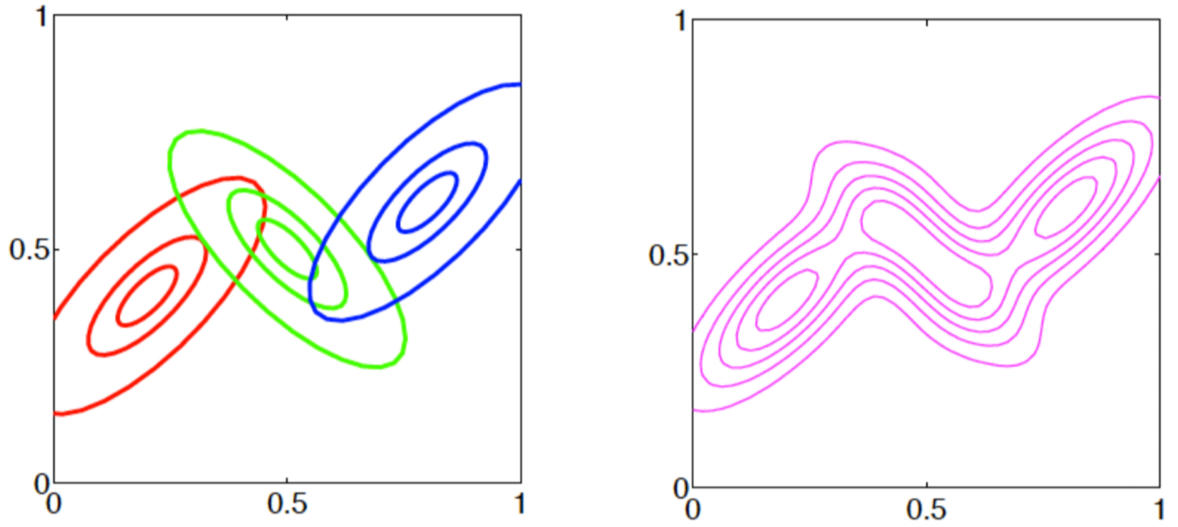

Fig. 1: Left: The 3 different gaussians; Right: The "mixture" of those gaussians

We can write this as a convex-conbination of Gaussian distributions:

$$P(x|\theta) = P(x|\mu, \Sigma, w) = \sum_{i=1}^{c} w_i \mathcal{N}(x; \mu_i, \Sigma_i)$$

where $w_i \geq 0$ and $\sum_i w_i = 1$

1.1 MLE for Gaussian mixtures

We can formulate the previous problem as a optimization problem:

$$L(w_{1:k}, \mu_{1:k}, \Sigma_{1:k}) = -\sum_{i=1}^{n} \log \sum_{j=1}^{k} w_j \mathcal{N}(x_i|\mu_j, \Sigma_j)$$

We'd like to solve argmin $L(w_{1:k}, \mu_{1:k}, \Sigma_{1:k})$, however this is a nonconvex objective and therefore difficult to solve.
We could try to apply (stochastic) gradient descent, however the covariance matrices must remain symmetric positive definite, and this might be difficult to maintain.

1.2 GMM vs. Gaussian Bayes Classifier

The joint distribution $P(z, x) = w_z \mathcal{N}(x|\mu_z, \Sigma_z)$ of (cluster index, features) is identical to the generative model used by the Gaussian Bayes Classifier (GBC).

The main difference between GBCs and GMMs: In the GMMs, the label (cluster) variable z is unobserved. This means that fitting a GMM is the same as training a GBC without labels.

2 Hard-EM algorithm

The Hard-EM (Expectation maximization) algorithm can be used to predict the labels we don't have. It works as follows:

Initialize the parameters $\theta^{(0)}$. This has to be done carefully, as it is a non-convex optimization problem. As in previous lectures, z are all latent variables that we optimize over.

For $t = 1, 2, \ldots$ do:

E-step: Predict the most likely class for each datapoint (z_i is the "hidden label" of the i-th datapoint)

$$z_i^{(i)} = \underset{z}{\operatorname{argmax}}\, P(z|x_i, \theta^{(t-1)}) = \underset{z}{\operatorname{argmax}}\, P(z|\theta^{(t-1)}) P(x_i|z, \theta^{(t-1)})$$

This is just taking the current model, apply Bayes' rule such that we can get the most likely label by multiplying the prior probability of that label (cluster) with the likelihood.

After doing that for all datapoints $x_i \in X$, we have complete data:

$$D^{(t)} = \{(x_1, z_1^{(t)}), \ldots, (x_n, z_n^{(t)})\}$$

M-step: (Maximization Step): Compute MLE same way as we did for the Gaussian Bayes classifier:

$$\theta^{(t)} = \underset{\theta}{\operatorname{argmax}}\, P(D^{(t)}|\theta)$$

2.1 Problems with Hard-EM

With the Hard-EM approach, several problems can arise. As an example, take a look at the two pictures below. The data was generated by a single Gaussian (left picture), however the Hard-EM approach generates the fitting (clustering) in the right picture:

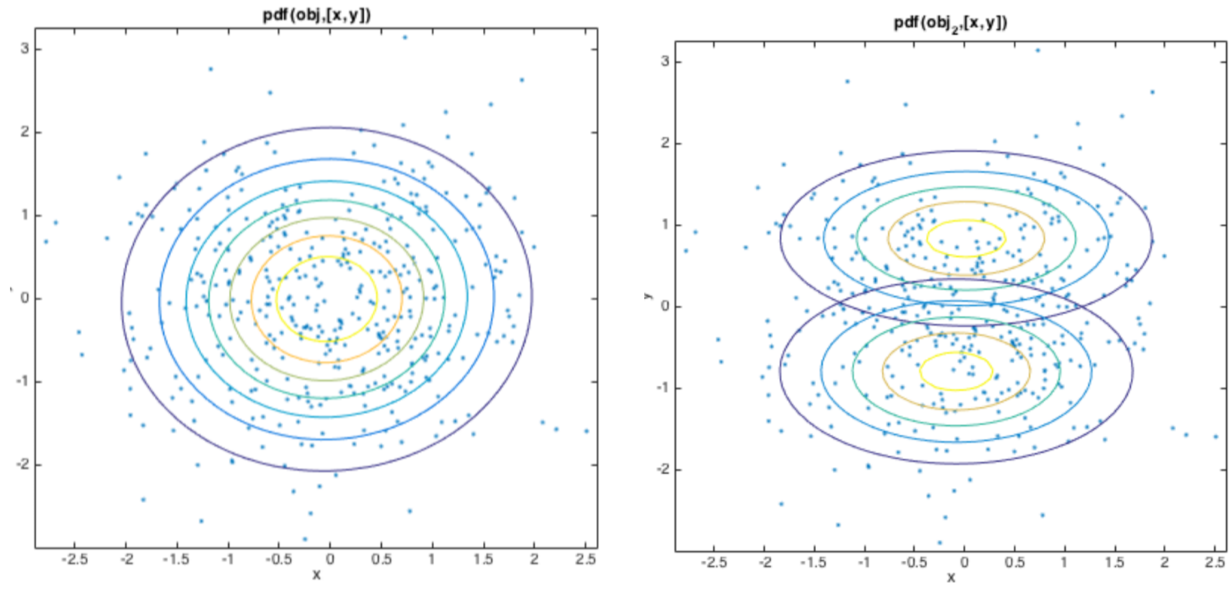

Fig.2: Comparision between the generating Gaussian and the Hard-EM fit

The problem with the Hard-EM approach is that the points are assigned a fixed label, even though the model is uncertain. Intuitively, this tries to extract "too much information" from a single point. This works especially poorly if the clusters are overlapping.

3 Soft-EM algorithm

3.1 Posterior probabilities

If we're given a model $P(z\theta)$, $P(x|z,\theta)$. For each point $x \in X$ we can compute a *posterior distribution over cluster membership*. This means inferring the distributions over the hidden variables z:

$$\gamma_j(x) = P(Z = j|x, \sigma, \mu, w) = \frac{w_j P(x|\sigma_j, \mu_j)}{\sum\limits_{l=1}^{K} w_l P(\Sigma_l, \mu_l)}$$

where K is the number of Gaussians we want to model.

In other words: for each point x we can compute the probability of it belonging to cluster j, which we represent by $\gamma_j(x)$.

3.2 The Soft-EM algorithm

As the MLE equations for μ, Σ, w are coupled, they are difficult to solve jointly, and as such the MLE minimization problem is hard to solve.

However, we can use the Soft-EM algorithm to solve this problem iteratively:

While not converged do:

E-step: Calculate the cluster membership weights $\gamma_j^{(t)}(x_i)$ for each point x_i and each cluster j using the estimates of $\mu^{(t-1)}$, $\Sigma^{(t-1)}$ and $w^{(t-1)}$ from the previous iteration:

$$\gamma_j^{(t)}(x_i) = P(Z = j|x_i, \mu^{(t-1)}, \Sigma^{(t-1)}, w^{(t-1)})$$

M-step: Fit the clusters to weighted data points:

$$w_j^{(t)} \leftarrow \frac{1}{n}\sum_{i=1}^{n} \gamma_j^{(t)}(x_i) \qquad \mu_j^{(t)} \leftarrow \frac{\sum\limits_{i=1}^{n} \gamma_j^{(t)}(x_i) x_i}{\sum\limits_{i=1}^{n} \gamma_j^{(t)}(x_i)}$$

$$\Sigma_j^{(t)} \leftarrow \frac{\sum\limits_{i=1}^{n} \gamma_j^{(t)}(x_i)(x_i - \mu_j^{(t)})(x_i - \mu_j^{(t)})^T}{\sum\limits_{i=1}^{n} \gamma_j^{(t)}(x_i)}$$

4 GMM compared to k-Means

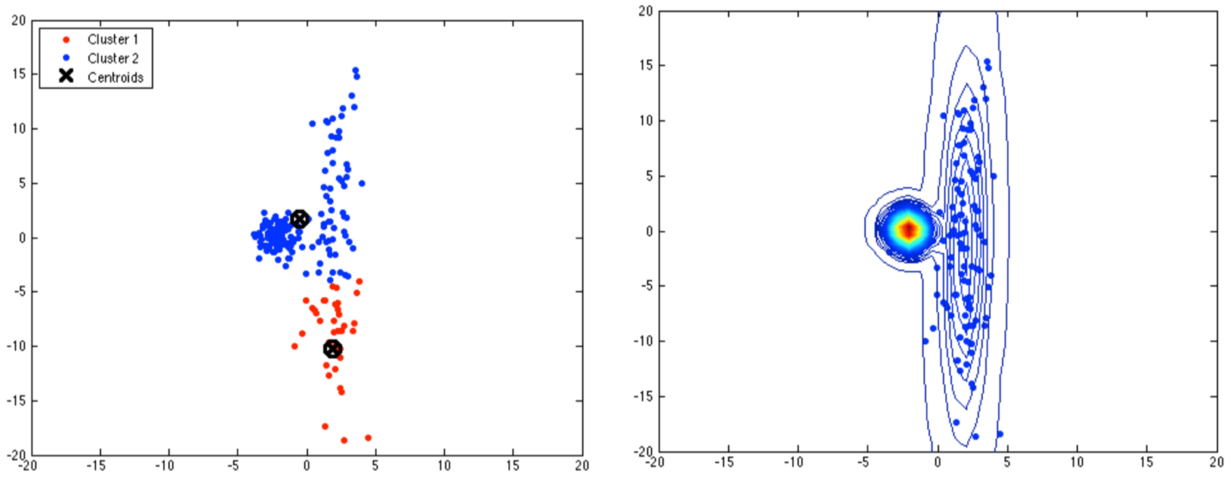

Fig. 3: Comparison between k-Means (left) and GMM (right)

As we can see here, the GMM captures the "small cluster" on the left side of the graph better than k-Means. k-Means groups the points of the small cluster together with the points of the large, stretched cluster.
The GMM on the other hand seperate the small cluster from the large, stretched cluster.

The k-Means algorithm with Lloyd's heuristic can be seen as a special case of Hard-EM for GMMs, with:

- Uniform weights over mixture components (distributions)
- Assuming identical, spherical covariance matrices

The k-Means can even be seen as a special case of Soft-EM for GMM with the assumptions from above as well as:

- Additional variances tend to 0

5 Soft-EM vs. Hard-EM

Soft-EM uses "soft assignments" to clusters, meaning it only assigns a probability $\gamma_j(x_i)$ for point x_i belonging to cluster j.
The Hard-EM algorithm on the other hand assigns each point to one, and only one cluster, so points at the "edges" of clusters (where multiple clusters overlap) pose a problem.

In general, Soft-EM will usually result in higher likelihood values because it can deal better with overlapping clusters.

6 Initialization

As with similar problems, we face the question on how to initialize the parameters we wish to optimize over. For GMMs, we can do it as follows:

Weights: Typically use uniform distribution
Means (μ): Randomly initialize (use k-Means++)
Variances (Σ): Initialize as spherical, e.g. according to empirical variance in the data
Number of clusters (k): Generally can use the same techniques as with k-Means. However, with GMMs we can use cross-validation, which usually works quite well. We aim to maximize the log-likelihood on the validation set.

7 GMMs for density estimation

We can not only use GMMs for clustering, but we can also use GMMs to estimate a density over a set of points X and then use any other classification method.
E.g.:

- Model $P(x)$ as Gaussian mixture
- Model $P(y|x)$ using logistic regression, neural networks, etc.

This combines the advantage of accurate predictions / robustness from discriminative models (e.g. Neural Networks) with the ability to detect outliers from generative models.

8 Anomaly detection

We can also use mixture models for detecting anomalies in the data. For this, we can compare the estimated density of a data point against a threshold, and if the density is below that threshold, we call the point an anomaly.

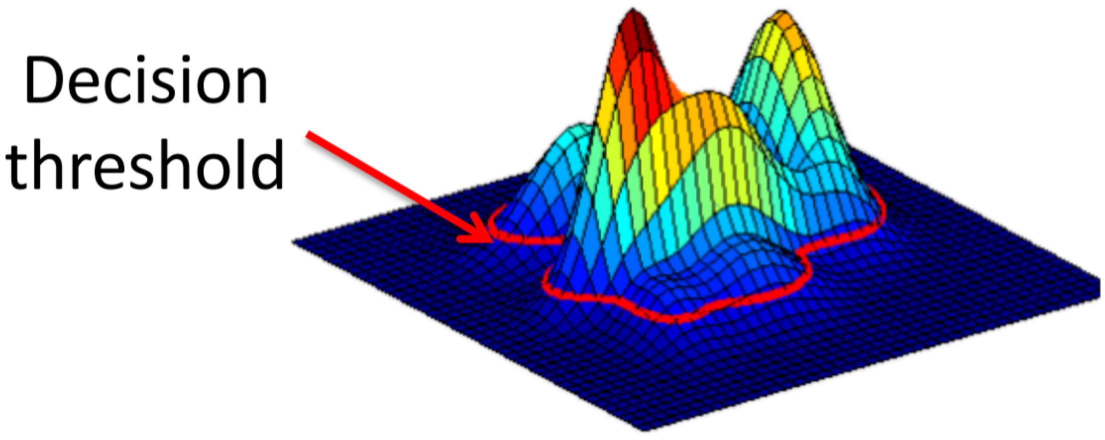

Fig. 4: Visualization of the decision threshold

As an example: In the graphic above, we have a Probability for each point x in the plane. If $P(x)$ is high, we have a peak at that point. Now, for some given threshold λ: If $P(x) < \lambda \Leftrightarrow x$ is an anomaly.

Choosing the right value for the threshold λ is challenging if we have no examples of anomalies.

If we have some anomalies, we can choose the right value for λ by:

- Varying the threshold, which trades false-positives and false-negatives (and set the value of λ according to which one is "less bad" (e.g. spam detection, cancer detection, etc.)
- Can use the precision-recall curves / ROC curves as evaluation criterium (e.g. maximize the F1 score)

With this, we can optimize the threshold (e.g. via cross-validation).

9 Semi-supervised learning with GMMs

We had so far:

Supervised Learning: Data consists of (feature, label)-pairs
Unsupervised Learning: Data consists of feature vectors only (unlabeled data)

Often it's possible to obtain a large amount of unlabeled data, but labeled data is expensive (e.g. spam needs to be classified manually). This is where **Semi-supervised learning** is useful: Semi-supervised learning learns from the (large amount of) unlabeled data and the (small amount of) labeled data.

If we recall the MLE for GMMs, we have:

$$w_j^* = \frac{1}{n}\sum_{i=1}^{n}\gamma_j(x_i) \qquad \mu_j^* = \frac{\sum\limits_{i=1}^{n}\gamma_j(x_i)x_i}{\sum\limits_{i=1}^{n}\gamma_j(x_i)} \qquad \Sigma_j^* = \frac{\sum\limits_{i=1}^{n}\gamma_j(x_i)(x_i-\mu_j)(x_i-\mu_j)^T}{\sum\limits_{i=1}^{n}\gamma_j(x_i)}$$

and then

$$\gamma_j(x) = P(z=j|x,\Sigma^*,\mu^*,w_*)$$

Now, in Semi-supervised learning, for datapoints x that have a label y, it must hold that:

$$\gamma_j(x_i) = [j = y_i]$$

In other words, this means that the GMM must assign all datapoints x which already have a label y the correct label.

With this, we now can formulate a EM for Semi-supervised learning:

While not converged do:

E-step: For all *unlabeled* points:

$$\gamma_j^{(t)}(x_i) = P(Z=j|x_i,\mu^{(t-1)},\Sigma^{(t-1)},w^{(t-1)})$$

For each *labeled* point x_i with label y_i:

$$\gamma^{(t)}(x_i) = [j = y_i]$$

For unlabeled points, it's the same as the Soft-EM, and for labeled points we just set the probability of the point x_i belonging to label y_j to 1, and 0 for all other clusters.

M-step: Fit clusters to weighted data points:

$$w_j^{(t)} \leftarrow \frac{1}{n}\sum_{i=1}^{n}\gamma_j^{(t)}(x_i) \qquad \mu_j^{(t)} \leftarrow \frac{\sum\limits_{i=1}^{n}\gamma_j^{(t)}(x_i)x_i}{\sum\limits_{i=1}^{n}\gamma_j^{(t)}(x_i)}$$

$$\Sigma_j^{(t)} \leftarrow \frac{\sum\limits_{i=1}^{n}\gamma_j^{(t)}(x_i)(x_i-\mu_j^{(t)})(x_i-\mu_j^{(t)})^T}{\sum\limits_{i=1}^{n}\gamma_j^{(t)}(x_i)}$$

This is the same as for the Soft-EM.

Basically, the only difference between the EM for Semi-supervised learning and the Soft-EM is that γ_j is calculated differently for labeled points in the Semi-supervised EM. All other steps are exactly the same.

10 Sources

- Introduction to Machine Learning Class at ETH (Prof A. Krause) (https://las.inf.ethz.ch/teaching/introml-s19)

10.1 Images

1. Introduction to Machine Learning, Lecture on the 15.05.2019, Page 5
2. Introduction to Machine Learning, Lecture on the 15.05.2019, Page 11 & 12
3. Introduction to Machine Learning, Lecture on the 15.05.2019, Page 24 & 25
4. Introduction to Machine Learning, Lecture on the 21.05.2019, Page 14