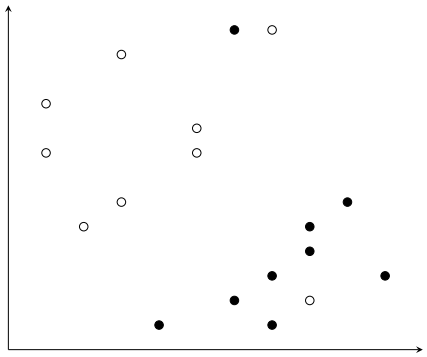
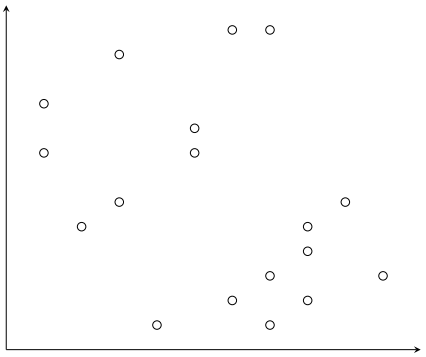
Lecture 10

- Clustering (k-means)
- Mixture of Gaussians
- Jensen's inequality
- General EM algorithm
- EM algorithm for the mixture of Gaussians
- EM algorithm for the mixture of Naive Bayes

In this lecture we start new part of our course that we call **unsupervised learning**. The difference between supervised and unsupervised learning can be explained by the following pictures:



Supervised learning



Unsupervised learning

In unsupervised learning problems no labels are given and job of the algorithm is to discover structure of the data.

1 Clustering (k-means)

The first example of unsupervised learning algorithm is called **clustering**. Clustering algorithm can be applied to identify groups of customers, groups of documents (for example, find groups of related articles on news.google.com), image segmentation and so on. The most popular example of clustering algorithm is **k-means algorithm**. Consider the unlabelled dataset:

$$\{x^{(1)}, \ldots, x^{(m)}\},\ x^{(i)} \in \mathbb{R}^n.$$

The k-means algorithm is listed as follows.

Algorithm 1 K-means algorithm

1: Set the number of clusters k
2: Initialize set of points (cluster centroids) $\mu_1, \dots, \mu_k \in \mathbb{R}^n$
3: **repeat**
4: Set $c^{(i)} = \arg\min_j ||x^{(i)} - \mu_j||$ (assigning point to the cluster j)
5: Update cluster centroids to the means of points from clusters:

$$\mu_j := \frac{\sum_{i=1}^{m} \mathbb{1}\{c^{(i)} = j\} \cdot x^{(i)}}{\sum_{i=1}^{m} \mathbb{1}\{c^{(i)} = j\}}$$

6: **until** convergence
7: **return** $c^{(i)}$

We want to say few words about the convergence of k-means algorithm. It turns out that it is easy to prove that this algorithm converges in some sense. By introducing the distortion function (this function is non-convex)

$$J(c, \mu) = \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2,$$

we can reformulate the steps of k-means algorithm using the coordinate ascent algorithm on J.

Exercise. Prove that this statement is correct.

Non-convexity of the distortion function does not guarantee finding the global minimum during the optimization algorithm. In practice, you should start k-means algorithm several times with different seed, it could help to identify the best set of clusters.

The final remark concerns the number of clusters (first step of the algorithm). What should be the correct number k of clusters. Usually this number is defined manually (there exist some automatic ways to define the number of clusters, in practice, the correct number of clusters is very ambiguous).

1.1 Python implementation

Import necessary libraries:

```
In [1]: import numpy as np
        import pandas as pd
        import random
        import math
        import sklearn.datasets as ds
        import matplotlib.pyplot as plt
```

K-means algorithm is implemented in the `scikit-learn` package:

```
In [2]: from sklearn.cluster import KMeans
```

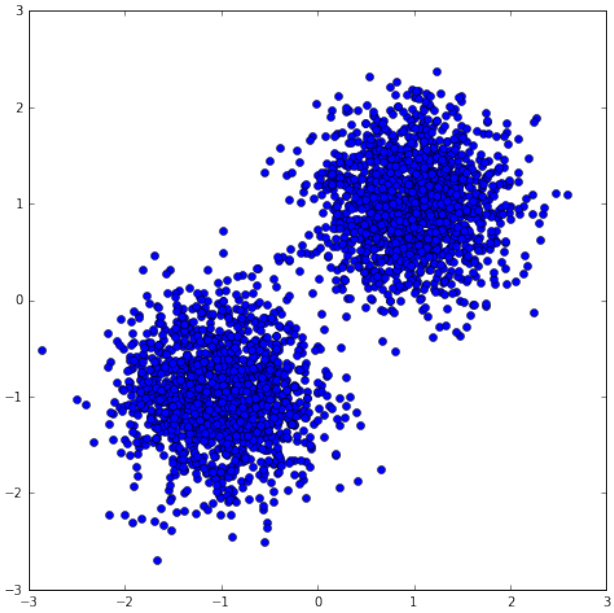
To illustrate how you can train this algorithm we generate some toy data and fit the algorithm on these data.

```
In [3]: from sklearn.datasets.samples_generator import make_blobs
        np.random.seed(0)
        centers = [[1, 1], [-1, -1]]
        n_clusters = len(centers)
        X, labels_true = make_blobs(n_samples=3000,
                                    centers=centers,
                                    cluster_std=0.5)
```

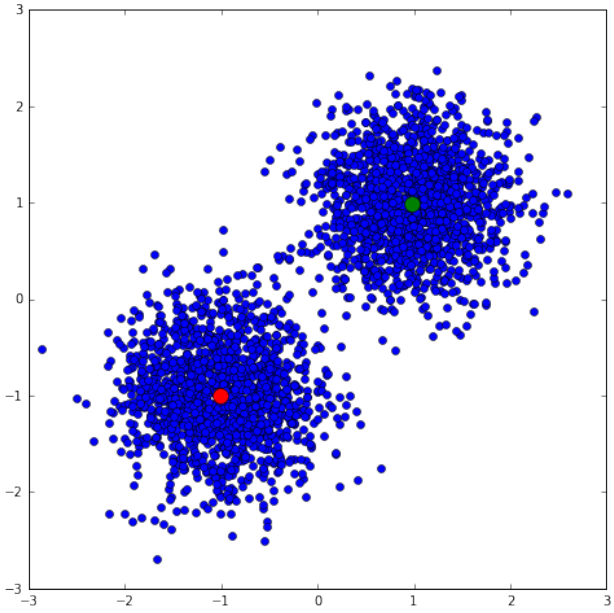
Helper function for data visualization:

```
In [4]: def plot_cluster_data(X, c=[1]*X.shape[0], mu=None):
            fig = plt.figure(figsize=(8, 8))
            ax = fig.add_subplot(1, 1, 1)
            if len(np.unique(c)) == 1:
                ax.plot(X[:,0], X[:,1], 'o')
            else:
                ix = np.where(c==1)
                ax.plot(X[ix,0], X[ix,1], 'o',
                        markerfacecolor='red')
                ax.plot(mu[0,0], mu[0,1], 'o',
                        markerfacecolor='red',
                        markersize=12)
                ix = np.where(c==0)
                ax.plot(X[ix,0], X[ix,1], 'o',
                        markerfacecolor='green')
                ax.plot(mu[1,0], mu[1,1], 'o',
                        markerfacecolor='green',
                        markersize=12)
            if not mu is None:
                ax.plot(mu[0,0], mu[0,1], 'o',
                        markerfacecolor='red',
                        markersize=12)
                ax.plot(mu[1,0], mu[1,1], 'o',
                        markerfacecolor='green',
                        markersize=12)
            plt.show()

In [5]: plot_cluster_data(X)
```




```
In [6]: clst = KMeans(n_clusters=2, random_state=2342)
        clst.fit(X)
        mu = clst.cluster_centers_
        plot_cluster_data(X, mu = mu)
```



Now we show what happens on each step of the algorithm. We will use two additional functions that corresponds to the two steps of the k-means algorithm.

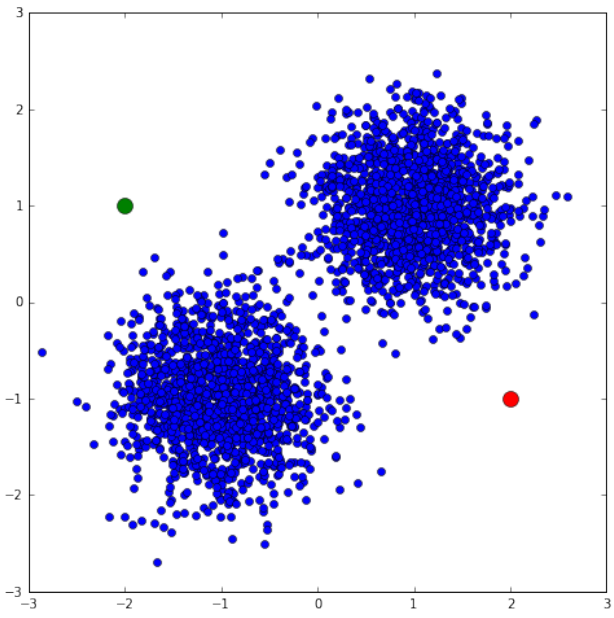
```
In [7]: def update_labels(X, mu):
            c = np.argmax(np.c_[np.sum(np.power(X - mu[0,:], 2), axis=1),
                                np.sum(np.power(X - mu[1,:], 2), axis=1)],
```

```
                        axis=1)
        return c

    def update_cluster_centers(X, c):
        ix = np.where(c==1)
        mu[0,:] = np.mean(X[ix,:], axis=1)
        ix = np.where(c==0)
        mu[1,:] = np.mean(X[ix,:], axis=1)
        return mu
```

First step of the algorithm is to choose the number of clusters k and cluster centers randomly.

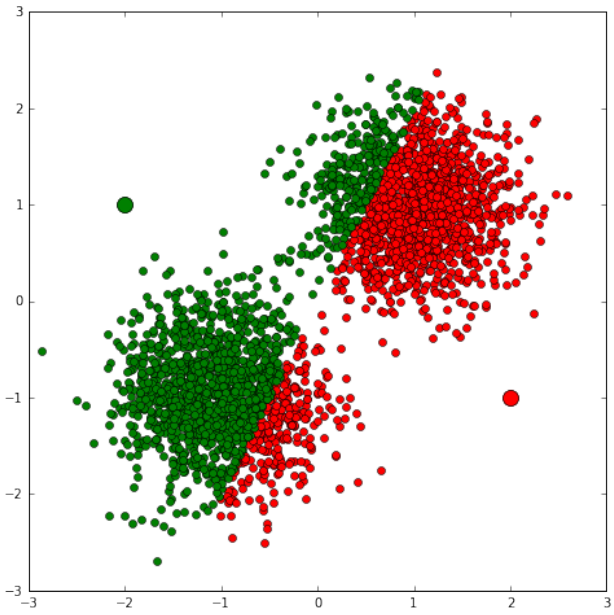
```
In [8]: k = 2
        mu = np.array([[2.0,-1.0], [-2.0,1.0]])
        plot_cluster_data(X, mu=mu)
```



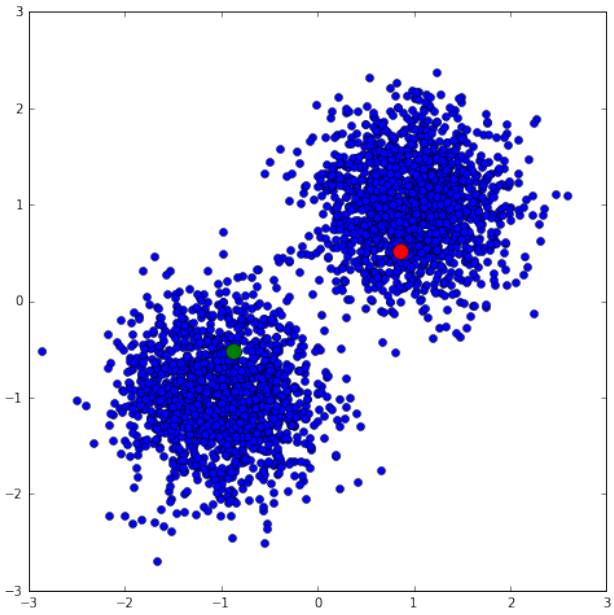
For each point we calculate what is the closest center:

```
In [9]: niter = 2
        for it in range(niter):
            print 'Iteration ' + str(it) + ':'
            c = update_labels(X, mu)
            print '...updating labels:'
            plot_cluster_data(X, c=c, mu=mu)
            print '...updating centers:'
            mu = update_cluster_centers(X, c)
            plot_cluster_data(X, mu=mu)
```

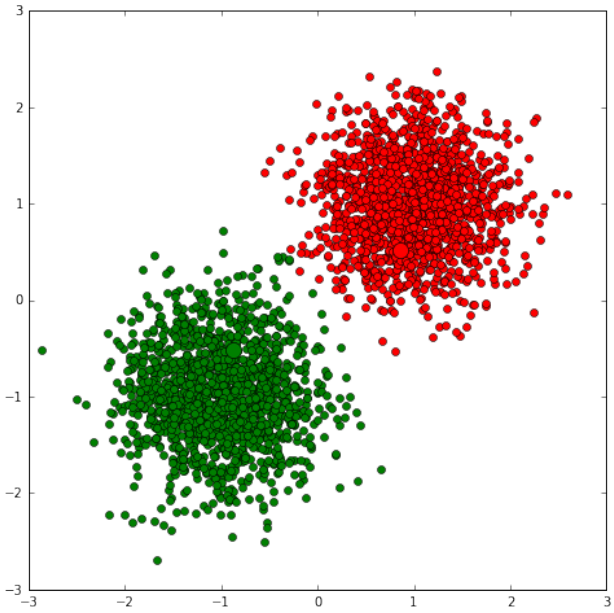
```
Iteration 0:
...updating labels:
```



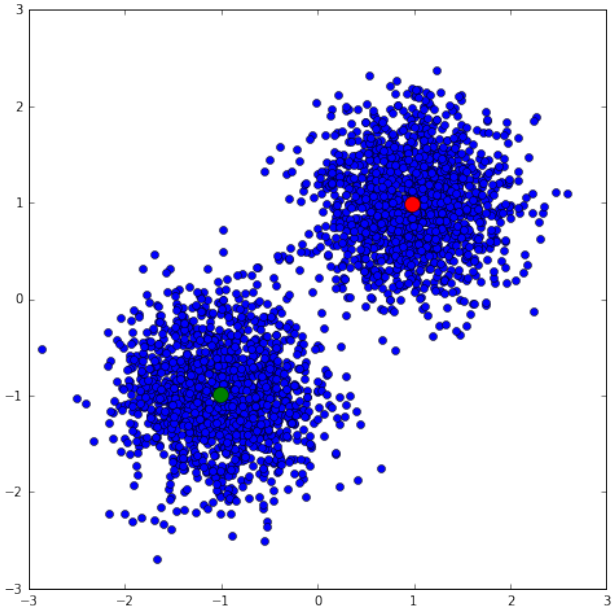

```
...updating centers:
```




```
Iteration 1:
...updating labels:
```

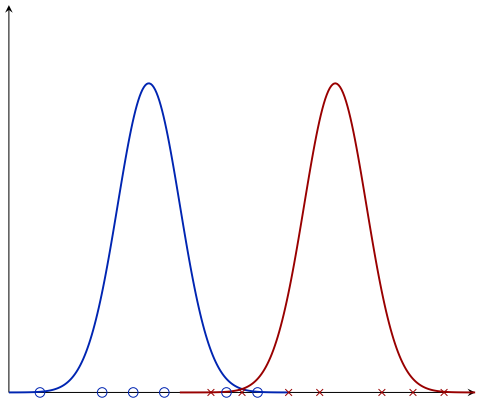


...updating centers:

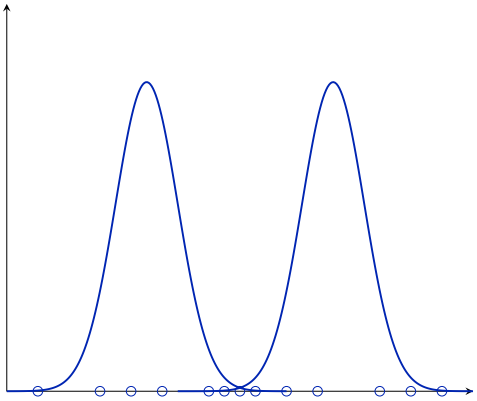


2 Mixture of Gaussians

In some problems we do not need to find the clusters, but to estimate the density. Assuming we have the training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, our aim is to find the probability density function $p(x)$. To understand the idea we compare this problem with Gaussian discriminant analysis algorithm. In the simplest case when $x^{(i)} \in \mathbb{R}$ (training examples are described by one feature) our aim is to build two Gaussian distributions:



Gaussian discriminant analysis



Mixture of Gaussians

The main difference between Gaussian discriminant analysis and mixture of Gaussians is that in the first case we can easily estimate the parameters of our distributions using class labels: for example, if we assume that

$$x^{(i)} \,|\, (y^{(i)} = j) \sim N(\mu_j, \Sigma_j),\ j \in \{0, 1\},$$
$$y^{(i)} \sim \text{Bernoulli}(\varphi),$$

then parameter φ can be calculated as

$$\varphi = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 1\}$$

and maximum likelihood estimation gives the values for μ_j, Σ_j:

$$\mu_j = \frac{\sum\limits_{i=1}^{m} \mathbb{1}\{y^{(i)} = j\} \cdot x^{(i)}}{\sum\limits_{i=1}^{m} \mathbb{1}\{y^{(i)} = j\}},$$

$$\Sigma_0 = \Sigma_1 = \Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}}) \cdot (x^{(i)} - \mu_{y^{(i)}})^T.$$

It becomes more complicated for the mixture of Gaussians, because now labels are not given. We introduce a latent (hidden, unobserved) random variable z (this variable plays a role of y in Gaussian discriminant algorithm). By analogy we make the following assumptions:

$$z^{(i)} \sim \ \text{Multinomial}(\varphi),\ \varphi_j \geqslant 0,\ \sum_{j=1}^{k} \varphi_j = 1,$$
$$x^{(i)} \,|\, (z^{(i)} = j) \sim N(\mu_j, \Sigma_j),\ j \in \{1, \ldots, k\},$$

where k is a number of Gaussian distributions in our mixture. For $k = 2$ the random variable $z^{(i)} \sim \text{Bernoulli}(\varphi)$. We can write the log-likelihood

$$l(\varphi, \mu, \Sigma) = \sum_{i=1}^{m} \ln p(x^{(i)}, z^{(i)}; \varphi, \mu, \Sigma).$$

Notice that the probability in the right hand side is a joint distribution of $x^{(i)}$, $z^{(i)}$ that can be calculated as

$$p(x^{(i)}, z^{(i)}) = p(x^{(i)} \mid z^{(i)}) \cdot p(z^{(i)}),$$

which implies

$$l(\varphi, \mu, \Sigma) = \sum_{i=1}^{m} \ln \sum_{j=1}^{k} p(x^{(i)} \mid z^{(i)} = j; \mu, \Sigma) \cdot p(z^{(i)} = j; \varphi).$$

If we know the values of $z^{(i)}$ for all training example then we can maximize this log-likelihood by analogy with Gaussian discriminant analysis. Unfortunately, $z^{(i)}$ and parameters φ are unknown and MLE does not give the result in some closed form, to find the solution we should apply additional step to guess what is the distribution of z. We formulate our problem as

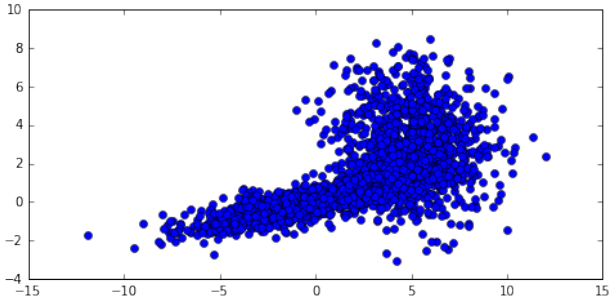
$$\arg\max_{\varphi,\mu,\Sigma} \sum_{i=1}^{m} \ln \sum_{j=1}^{k} p(x^{(i)} \mid z^{(i)} = j; \mu, \Sigma) \cdot p(z^{(i)} = j; \varphi) \tag{1}$$

2.1 Python implementation

As an example we generate data based on two different gaussian distributions

```
In [10]: np.random.seed(0)
         n_samples = 1000
         X1 = 2.0*np.random.randn(n_samples, 2) + np.array([5, 3])
         C = np.array([[0., -0.5], [3.5, .7]])
         X2 = np.dot(np.random.randn(n_samples, 2), C)
         X_train = np.vstack([X1, X2])

         fig = plt.figure(figsize=(8, 8))
         ax = fig.add_subplot(1, 1, 1)
         ax.plot(X_train[:,0], X_train[:,1], 'o')
         plt.gca().set_aspect('equal', adjustable='box')
         plt.show()
```



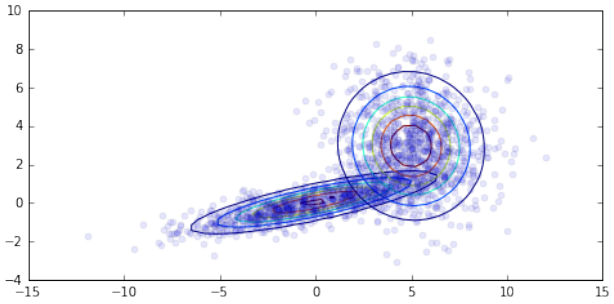
and train mixture of Gaussians model on these data:

```
In [11]: from sklearn.mixture import GMM
```

```
In [12]: np.random.seed(1)
         model = GMM(n_components=2, covariance_type='full')
         model.fit(X_train)

         fig = plt.figure(figsize=(8, 8))
         ax = fig.add_subplot(1, 1, 1)
         ax.plot(X_train[:,0], X_train[:,1], 'o', alpha=.1, ms=5)

         for i in range(2):
             mu = model.means_[i]
             sigma = model.covars_[i]
             sigma_inv = np.linalg.inv(sigma)
             sigma_det = np.linalg.det(sigma)
             x = np.linspace(-15.0, 15.0)
             y = np.linspace(-4.0, 10.0)
             X, Y = np.meshgrid(x, y)
             XX = np.array([X.ravel(), Y.ravel()]).T
             XX = np.dot(np.dot(XX - mu, sigma_inv),
                         np.transpose(XX - mu))
             P = np.exp(-0.5*np.diagonal(XX))/(2*np.pi*sigma_det**0.5)
             P = P.reshape(X.shape)
             CS = plt.contour(X, Y, P)
         plt.gca().set_aspect('equal', adjustable='box')
         plt.show()
```



3 Jensen's inequality

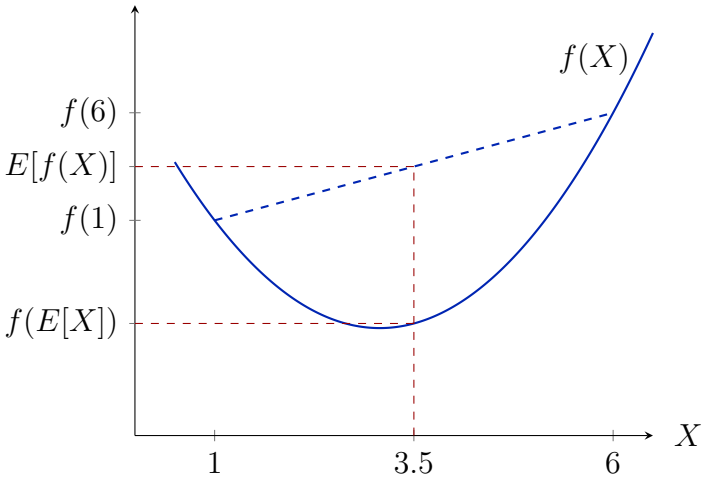
Theorem 1. Let f be a convex function (e.g. $f''(x) \geqslant 0$) and X be a random variable. Then

$$f(E[X]) \leqslant E[f(X)].$$

Example. Consider the random variable X with the density function

$$p(X) = \begin{cases} 0.5, & \text{if } X = 1, \\ 0.5, & \text{if } X = 6. \end{cases}$$

Then for the convex function $f(X)$ the Jensen's inequality is illustrated on the picture



Theorem 2. If $f''(x) > 0$ (f is strictly convex), then

$$E[f(X)] = f(E[X]) \text{ if and only if } X = E[X] \text{ with probability 1.}$$

Theorem 3. If $f''(x) \leqslant 0$ (concave downward), then

$$f(E[X]) \geqslant E[f(X)].$$

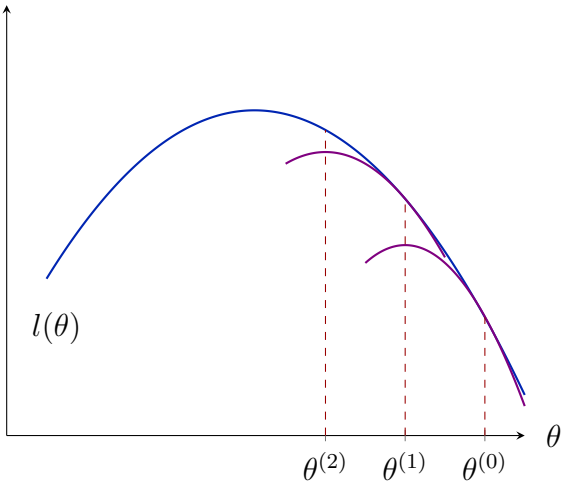
4 General EM algorithm

The algorithm that helps to solve the optimization problem (1) for mixture of Gaussians model is called **EM (expectation-maximization) algorithm**.

The general optimization problem that can be solved using EM algorithm is formulated as follows. Assuming that we have some model for $p(x, z; \theta)$, where x are observed and z are latent, our goal is to maximize log-likelihood with respect to parameters θ:

$$l(\theta) = \sum_{i=1}^{m} \ln p(x^{(i)}; \theta) = \sum_{i=1}^{m} \ln \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta).$$

EM algorithm is an iterative algorithm that constructs lower bound for the log-likelihood on each step and maximizes this lower bound instead of the original log-likelihood (violet curves on the picture are lower bounds):



Now we make this idea more formal. The maximum log-likelihood can be transformed as

$$\begin{aligned}\max_\theta l(\theta) = \max_\theta \sum_{i=1}^m \ln p(x^{(i)};\theta) = \sum_{i=1}^m \ln \sum_{z^{(i)}} p(x^{(i)}, z^{(i)};\theta) = \\ = \sum_{i=1}^m \ln \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})},\end{aligned} \tag{2}$$

where $Q_i(z^{(i)})$ is some unknown probability density function for $z^{(i)}$ with properties $Q_i(z^{(i)}) \geqslant 0$, $\sum\limits_{z^{(i)}} Q_i(z^{(i)}) = 1$. The expression inside the logarithm is the expectation of $\dfrac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})}$ given that $z^{(i)}$ drawn from the distribution Q_i:

$$\sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})} = E_{z^{(i)}\sim Q_i}\left[\frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})}\right] \tag{3}$$

(by definition, if the random variable Z has a density p, then

$$E[g(Z)] = \sum_z p(z)g(z)$$

for any function $g(Z)$).

Then Jensen's inequality

$$\ln E[X] \geqslant E[\ln X]$$

for the concave function $\ln X$ and equation (3) applied to the right hand side of the equation (2) gives

$$\begin{aligned}\sum_{i=1}^m \ln \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})} = \sum_{i=1}^m \ln E_{z^{(i)}\sim Q_i}\left[\frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})}\right] \geqslant \\ \geqslant \sum_{i=1}^m E_{z^{(i)}\sim Q_i}\left[\ln \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})}\right] = \sum_{i=1}^m \sum_{z^{(i)}} Q_i(z^{(i)}) \ln \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})}.\end{aligned}$$

We just showed that

$$l(\theta) \geqslant \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \ln \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})},$$

where the right hand side is a lower bound from the previous picture for the log-likelihood $l(\theta)$. It makes sense to have this lower bound as close to the log-likelihood as possible, which means that we should replace $\geqslant$ by $=$. But the theorem 2 from the previous section implies that we have equation instead of inequality if and only if the random variable $\dfrac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$ is constant, or

$$Q_i(z^{(i)}) \propto p(x^{(i)}, z^{(i)}; \theta).$$

By definition, $\sum\limits_{i=1}^{m} Q_i(z^{(i)}) = 1$, then

$$Q_i(z^{(i)}) = \frac{p(x^{(i)}, z^{(i)}; \theta)}{\sum\limits_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta)} = \frac{p(x^{(i)}, z^{(i)}; \theta)}{p(x^{(i)}; \theta)} = p(z^{(i)} \,|\, x^{(i)}; \theta).$$

Algorithm 2 General EM algorithm

1: **repeat**

2: **E-step**: set

$$Q_i(z^{(i)}) = p(z^{(i)} \,|\, x^{(i)}; \theta)$$

3: **M-step**: optimize the lower bound with respect to θ

$$\theta := \arg\max_{\theta} \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \ln \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

4: **until** convergence

5: **return** θ

By analogy with the k-means algorithm, define the distortion function

$$J(Q, \theta) = \sum_{i=1}^{m} \sum_{z^{i)}} Q_i(z^{i)}) \ln \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{i)})},$$

then $l(\theta) \geqslant J(Q, \theta)$ and EM can be viewed as a coordinate ascent algorithm on J where on the E-step we maximize J with respect to Q and on the M-step we maximize it with respect to θ.

Exercise. Prove that the algorithm converges, i.e. if we have consider $\theta^{(t)}$ and $\theta^{(t+1)}$ from two consequent iterations of the algorithm, then

$$l(\theta^{(t+1)}) \geqslant l(\theta^{(t)}).$$

5 EM algorithm for the mixture of Gaussians

To solve the optimization problem (1) we apply EM algorithm.

- **E-step** (guess values of $z^{(i)}$). With the assumptions

$$x^{(i)} \,|\, (z^{(i)} = j) \sim N(\mu_j, \Sigma_j),$$
$$z^{(i)} \sim \text{Multinomial}(\varphi) \Rightarrow p(z^{(i)} = j) = \varphi_j$$

and the Bayes rule we determine the distribution

$$Q_i(z^{(i)} = j) = p(z^{(i)} = j \,|\, x^{(i)}; \varphi, \mu, \Sigma) = \frac{p(x^{(i)} \,|\, z^{(i)} = j) \cdot p(z^{(i)} = j)}{\sum\limits_{l=1}^{k} p(x^{(i)} \,|\, z^{(i)} = l) \cdot p(z^{(i)} = l)}.$$

Denote

$$w_j^{(i)} = Q_i(z^{(i)} = j).$$

- **M-step** (update estimates for the parameters).

$$J(Q, \varphi, \mu, \Sigma) = \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \ln \frac{p(x^{(i)}, z^{(i)}; \varphi, \mu, \Sigma)}{Q_i(z^{(i)})} =$$
$$= \sum_{i=1}^{m} \sum_{z^{(i)}} w_j^{(i)} \ln \frac{p(x^{(i)} \,|\, z^{(i)}; \mu, \Sigma) \cdot p(z^{(i)}; \varphi)}{w_j^{(i)}} =$$
$$= \sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} \ln \frac{\dfrac{1}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \exp\left(-\dfrac{1}{2} (x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j) \right) \cdot \varphi_j}{w_j^{(i)}}.$$

Remember that there is an additional constraint

$$\sum_{j=1}^{k} \varphi_j = 1.$$

To use this constraint we can write down the Lagrangian

$$L = J(Q, \varphi, \mu, \Sigma) - \beta \left(\sum_{j=1}^{k} \varphi_j - 1 \right)$$

and set it derivatives with respect to parameters to zero. Doing this gives the estimates of parameters:

$$\nabla_{\varphi_j} L = 0 \Rightarrow \varphi_j = \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)},$$
$$\nabla_{\mu_j} L = 0 \Rightarrow \mu_j = \frac{\sum\limits_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum\limits_{i=1}^{m} w_j^{(i)}},$$
$$\nabla_{\Sigma_j} L = 0 \Rightarrow \Sigma_j = \frac{\sum\limits_{i=1}^{m} w_j^{(i)} \cdot (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum\limits_{i=1}^{m} w_j^{(i)}}.$$

The algorithm is summarized as follows:

Algorithm 3 EM algorithm for the mixture of Gaussians

1: Set the number of components k
2: Initialize values for the parameters φ_j, μ_j, Σ_j, $j = \{1, \dots, k\}$
3: **repeat**
4: **E-step**: set

$$w_j^{(i)} = \frac{\frac{1}{(2\pi)^{n/2}|\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1}(x^{(i)} - \mu_j)\right) \cdot \varphi_j}{\sum_{l=1}^{k} \frac{1}{(2\pi)^{n/2}|\Sigma_l|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu_l)^T \Sigma_l^{-1}(x^{(i)} - \mu_l)\right) \cdot \varphi_l}$$

5: **M-step**: update the parameters

$$\varphi_j := \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}$$

$$\mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}$$

$$\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} \cdot (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}$$

6: **until** convergence
7: **return** φ_j, μ_j, Σ_j, $j = \{1, \dots, k\}$

We will show how to derive φ_j in the M-step of the algorithm:

$$\nabla_{\varphi_j} L = 0 \Leftrightarrow \sum_{i=1}^{m} \frac{w_j^{(i)}}{\varphi_j} - \beta = 0 \Leftrightarrow \sum_{i=1}^{m} w_j^{(i)} - \beta \cdot \varphi_j = 0 \text{ for all } j.$$

Sum obtained equations by j:

$$\sum_{j=1}^{k} \sum_{i=1}^{m} w_j^{(i)} - \beta \sum_{j=1}^{k} \varphi_j = 0 \Leftrightarrow \sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} - \beta = 0,$$

but $\sum_{j=1}^{k} w_j^{(i)} = 1$, because $w_j^{(i)} = p(z^{(i)} = j \,|\, x^{(i)}; \varphi, \mu, \Sigma)$ are probabilities for multinomial random variable. Finally,

$$\beta = \sum_{i=1}^{m} 1 = m \Rightarrow \sum_{i=1}^{m} w_j^{(i)} - m\varphi_j = 0 \Rightarrow \varphi_j = \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}.$$

We compare Gaussian discriminant analysis and mixture of Gaussians in the following table.

Param.	Gaussian discriminant analysis	Mixture of Gaussians
φ_j	$\frac{1}{m}\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=j\}$	$\frac{1}{m}\sum_{i=1}^{m}w_j^{(i)}$
μ_j	$\frac{\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=j\}\cdot x^{(i)}}{\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=j\}}$	$\frac{\sum_{i=1}^{m}w_j^{(i)}\cdot x^{(i)}}{\sum_{i=1}^{m}w_j^{(i)}}$
Σ_j	$\frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu_{y^{(i)}})\cdot(x^{(i)}-\mu_{y^{(i)}})^T$	$\frac{\sum_{i=1}^{m}w_j^{(i)}\cdot(x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^T}{\sum_{i=1}^{m}w_j^{(i)}}$

We can conclude that in the mixture of Gaussians we replace the percentage of each class by the probability $w_j^{(i)}$.

5.1 Python implementation

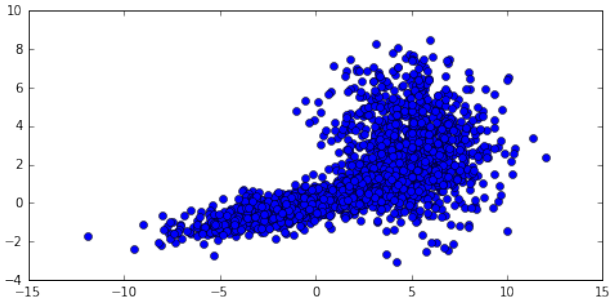
We implement EM algorithm for the mixture of Gaussians from scratch. First, we define the functions for E-step and M-step. Notice that we have tried to use some built-in `numpy` functions to avoid loops.

```
In [13]: def Estep(mu, sigma, phi):
             # calculate determinants of sigma's
             det_sigma = np.array([[np.linalg.det(sigma[i])]
                                   for i in range(k)])
             # calculate inverse matrices for sigma's
             inv_sigma = np.array([np.linalg.inv(sigma[i])
                                   for i in range(k)]).reshape(sigma.shape)
             # calculate Q(z) = p(x|z)*p(z)/p(x)
             pxz = np.array([
                     np.exp(
                         -0.5*np.diagonal(
                             np.dot(
                                 np.dot(X_train - mu[i,:], inv_sigma[i]),
                                 np.transpose(X_train - mu[i,:])
                             )
                         )
                     )/((2.0*np.pi)**(n/2.0)*det_sigma[i,0]**0.5)*phi[i,0]
                     for i in range(k)]).T
             pz = pxz/np.sum(pxz, axis=1).reshape((-1, 1))
             return pz
```

```
        def Mstep(pz):
            pz_sum = np.sum(pz, axis=0).reshape((-1,1))
            # update parameters
            phi_new = pz_sum/m
            mu_new = np.transpose(np.dot(X_train.T, pz)/pz_sum.T)
            sigma_new = np.array([
                    np.dot(np.array([
                                np.outer(X_train[j,:] - mu_new[i,:],
                                         X_train[j,:] - mu_new[i,:])
                                for j in range(m)]).reshape((m, -1)).T,
                           pz[:,i]).reshape((n,n))/pz_sum[i,0]
                    for i in range(k)]).reshape((k,n,n))
            return mu_new, sigma_new, phi_new
```

We use the data from the previous section (generated as a mixture of two gaussian distributions):

```
In [14]: fig = plt.figure(figsize=(8, 8))
         ax = fig.add_subplot(1, 1, 1)
         ax.plot(X_train[:,0], X_train[:,1], 'o')
         plt.gca().set_aspect('equal', adjustable='box')
         plt.show()
```



Define the variables and parameters for our EM algorithm:

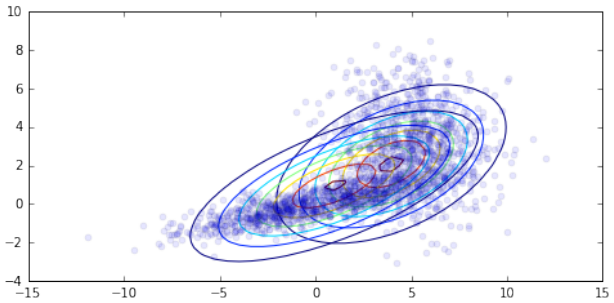
```
In [15]: # number of components
         k = 2
         # number of features
         n = X_train.shape[1]
         # number of training examples
         m = X_train.shape[0]
         # number of iterations
         niter = 10
         # initial values of phi
         phi = np.array([1.0/k]*k).reshape((k,-1))
         # initial values for mu and sigma
```

```
mu = []
sigma = []
np.random.seed(234)
for cl in range(k):
    mu.append(np.mean(X_train[np.random.choice(m, m/2),:], axis=0))
    sigma.append(np.identity(n))
mu = np.array(mu).reshape((k, n))
sigma = np.array(sigma).reshape((k, n, n))
```

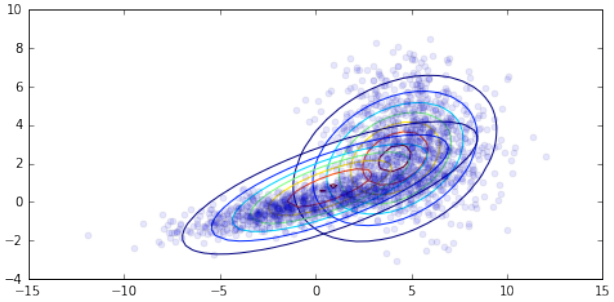
We train the mixture of Gaussians model and plot the fitted distributions after each iteration:

```
In [16]: for nit in range(niter):
             print 'Iteration ' + str(nit+1) + ':'
             pz = Estep(mu, sigma, phi)
             mu, sigma, phi = Mstep(pz)
             fig = plt.figure(figsize=(8, 8))
             ax = fig.add_subplot(1, 1, 1)
             ax.plot(X_train[:,0], X_train[:,1], 'o', alpha=.1, ms=5)
             for i in range(k):
                 mu_i = mu[i,:]
                 sigma_i = sigma[i]
                 sigma_i_inv = np.linalg.inv(sigma_i)
                 x = np.linspace(-15.0, 15.0)
                 y = np.linspace(-4.0, 10.0)
                 X, Y = np.meshgrid(x, y)
                 XX = np.array([X.ravel(), Y.ravel()]).T
                 P = np.exp(-0.5*np.diagonal(
                         np.dot(
                             np.dot(XX - mu_i, sigma_i_inv),
                             np.transpose(XX - mu_i)
                         )))/(2*np.pi*np.linalg.det(sigma_i)**0.5)
                 P = P.reshape(X.shape)
                 CS = plt.contour(X, Y, P)
             plt.gca().set_aspect('equal', adjustable='box')
             plt.show()
```

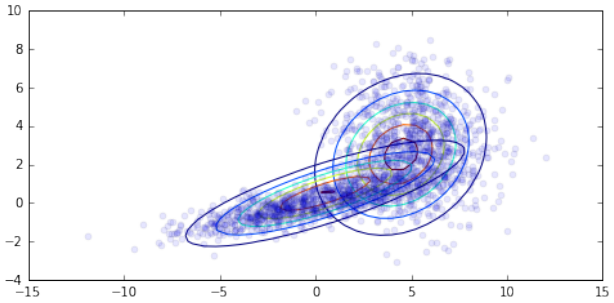
```
Iteration 1:
```



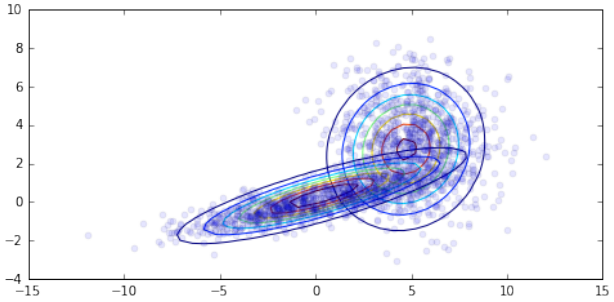
Iteration 2:



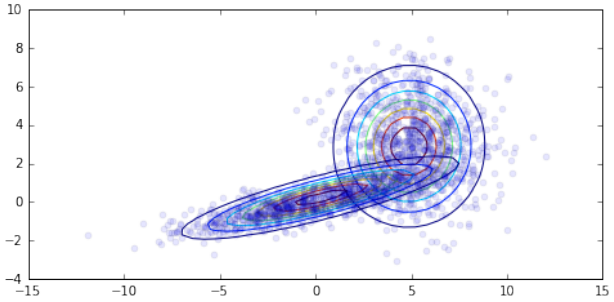
Iteration 3:



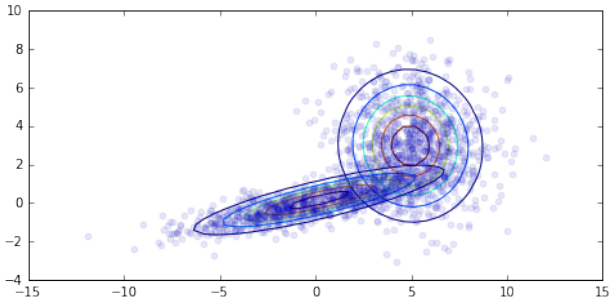
Iteration 4:



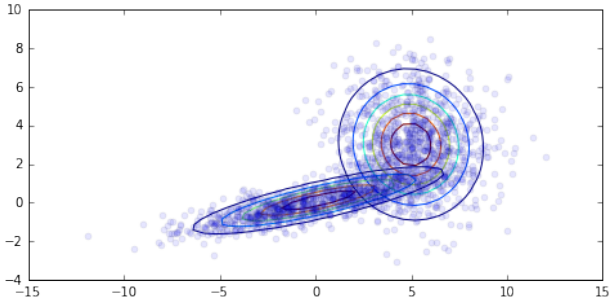
Iteration 5:



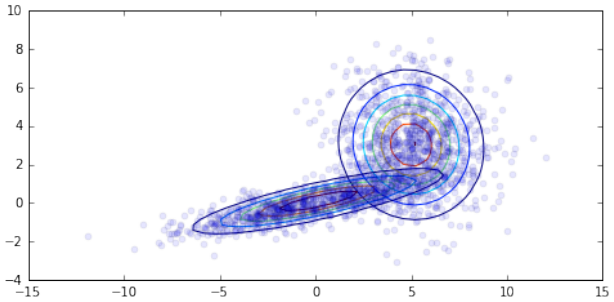
Iteration 6:



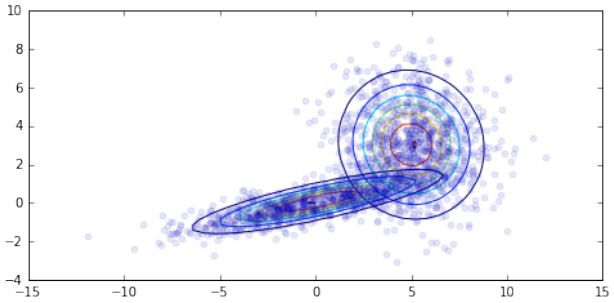
Iteration 7:



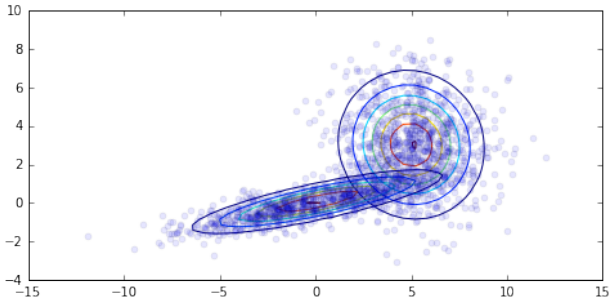
Iteration 8:



Iteration 9:



Iteration 10:



6 EM algorithm for the mixture of Naive Bayes

In the previous lectures we talked about Naive Bayes model applied to the text classification problem. Assuming we have a dataset of m documents: $\{x^{(1)}, \ldots, x^{(m)}\}$, $x^{(i)} \in \mathbb{R}^n$, and $x_j^{(i)} = \mathbb{1}\{\text{word } j \text{ appears in the document } i\}$. It means that each document in the database can be represented by the features vector

$$x^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} a \\ as \\ advert \\ and \\ \vdots \\ claim \\ \vdots \\ light \\ \vdots \\ zip \end{matrix}$$

In this section we apply EM algorithm to build a mixture of Naive Bayes models. As before instead of class labels $y^{(i)}$ we introduce a latent random variable $z^{(i)} \in \{0, 1\}$, which means that we assume two clusters for the text dataset and

$$z^{(i)} \sim \text{Bernoulli}(\varphi).$$

We also make Naive Bayes assumption

$$p(x^{(i)} \mid z^{(i)}) = \prod_{j=1}^{n} p(x_j^{(i)} \mid z^{(i)}),$$

or more specifically,

$$p(x_j^{(i)} = 1 \mid z^{(i)} = 0) = \varphi_{j \mid z=0},$$
$$p(x_j^{(i)} = 1 \mid z^{(i)} = 1) = \varphi_{j \mid z=1}.$$

EM algorithm contains two steps:

- **E-step**.

$$w^{(i)} = p(z^{(i)} = 1 \,|\, x^{(i)}; \varphi_{j\,|\,z}, \varphi)$$

- **M-step**.

$$\varphi_{j\,|\,z=1} = \frac{\sum_{i=1}^{m} w^{(i)} \mathbb{1}\{x_j^{(i)} = 1\}}{\sum_{i=1}^{m} w^{(i)}}$$

$$\varphi_{j\,|\,z=0} = \frac{\sum_{i=1}^{m} (1 - w^{(i)}) \mathbb{1}\{x_j^{(i)} = 1\}}{\sum_{i=1}^{m} (1 - w^{(i)})}$$

$$\varphi = \frac{\sum_{i=1}^{m} w^{(i)}}{m}$$

The final remark is that if we apply this algorithm $w^{(i)}$'s will be close either to 0 or 1.