EM Algorithm for flat clustering: Gaussian Mixture Models (and Bernoulli Mixture Models)

James G. Shanahan

[1]Church and Duncan Group,

[2]Information School, UC Berkeley

[3]School of Informatics, Computing and Engineering, Indiana University

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Reading material

- **https://en.wikipedia.org/wiki/Latent_variable**

EM Algorithm for flat clustering: References

- **See Section 16.5 Model-based clustering in**
  - http://nlp.stanford.edu/IR-book/pdf/16flat.pdf
  - Bernouilli EM versus GMM EM
  - For K-Means see 16.4 K-means
  - http://cs229.stanford.edu/notes/cs229-notes2.pdf
- **Other references**
  - What is the expectation maximization algorithm?,
    - http://ai.stanford.edu/~chuongdo/papers/em_tutorial.pdf
  - Harvard notes on EM,
  - stackoverflow on EM
- **Advanced**
  - Mixtures of Gaussians and the EM algorithm
    - http://cs229.stanford.edu/notes/cs229-notes7b.pdf
  - The EM Algorithm
    - http://cs229.stanford.edu/notes/cs229-notes8.pdf

Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

Latent Variables (LV) vs observed variables

- **In statistics, latent variables**
  - (from Latin: present participle of lateo (“lie hidden”), as opposed to observable variables),
  - are variables that are not directly observed but are rather inferred (through a mathematical model) from other variables that are observed (directly measured).
- **Latent variable models**
  - Mathematical models that aim to explain observed variables in terms of latent variables are called latent variable models.
- **LV models are used in many disciplines**
  - including psychology, demography, economics, engineering, medicin e, physics, machine learning/artificial intelligence, bioinformatics, natural language processing, econometrics, management and the social sciences.

https://en.wikipedia.org/wiki/Latent_variable

Sometimes LVs are Hidden variables

- **Sometimes LVs are Hidden variables**
  - Sometimes latent variables correspond to aspects of physical reality, which could in principle be measured, but may not be for practical reasons.
  - In this situation, the term hidden variables is commonly used (reflecting the fact that the variables are "really there", but hidden).
- **LVs are abstract concepts**
  - Other times, latent variables correspond to abstract concepts, like categories, behavioral or mental states, or data structures.
  - The terms hypothetical variables or hypothetical constructs may be used in these situations.

LVs can lead to dimensionality reduction

- **LVs can lead to dimensionality reduction**
  - One advantage of using latent variables is that they can serve to reduce the dimensionality of data.
    - A large number of observable variables can be aggregated in a model to represent an underlying concept, making it easier to understand the data.
  - In this sense, they serve a function similar to that of scientific theories.
  - At the same time, latent variables link observable ("sub-symbolic") data in the real world to symbolic data in the modeled world.

Common approaches for estimating parameters of models with unobserved LVs

- EM algorithms
- Hidden Markov models
- Factor analysis
- Principal component analysis
- Partial least squares regression
- Latent semantic analysis and Probabilistic latent semantic analysis
- Word2Vec

Maximum-likelihood estimation

- **Maximum-likelihood estimation (MLE) is a method of estimating the parameters of a statistical model.**
- **When applied to a data set and given a statistical model, maximum-likelihood estimation provides estimates for the model's parameters.**

- **When the model depends on unobserved latent variables**
  - Expectation–maximization (EM) algorithm is an iterative method for finding maximum likelihood, where the model depends on unobserved latent variables.
    - The EM iteration alternates between performing
    - an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters,
    - and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the Estep.
      - These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.

Use EM to estimate the parameters of model with unobserved latent variables

- In statistics, an **expectation maximization** (**EM**) **algorithm** is an iterative method to find maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables.
  - The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the *E* step.
  - These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.
- **Present later**

Gaussian Model: Estimate Parameters via MLE (closed form)

Suppose the following are marks in a course

55.5, 67, 87, 48, 63

```
> mean(55.5, 67, 87, 48, 63)
[1] 55.5
> sd(c(55.5, 67, 87, 48, 63))
[1] 14.72413
```

Marks typically follow a Normal distribution whose density function is

$$N(\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma}(x-\mu)^2}$$

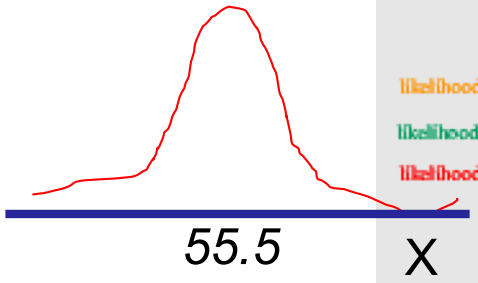
Now, we want to find the best μ,σ such that

$$argmax_{\mu,\sigma} p(Data|\mu,\sigma)$$

$$\hat{\mu}(\mathbf{x}) = \bar{x}$$

$$\hat{\sigma}^2(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})^2.$$

$\hat{\mu}(\mathbf{x})\ \hat{\sigma}^2(\mathbf{x})$ **are MLE for μ, σ²**



$$L(x_1,\ldots,x_N,\underline{\theta}) = \prod_{j=1}^{N}\frac{1}{\sigma\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{x_j-\mu}{\sigma}\right)^2\right]$$

$$= \frac{1}{\sigma^N(2\pi)^{N/2}}\exp\left[-\frac{1}{2\sigma^2}\sum_{j=1}^{N}(x_j-\mu)^2\right]$$

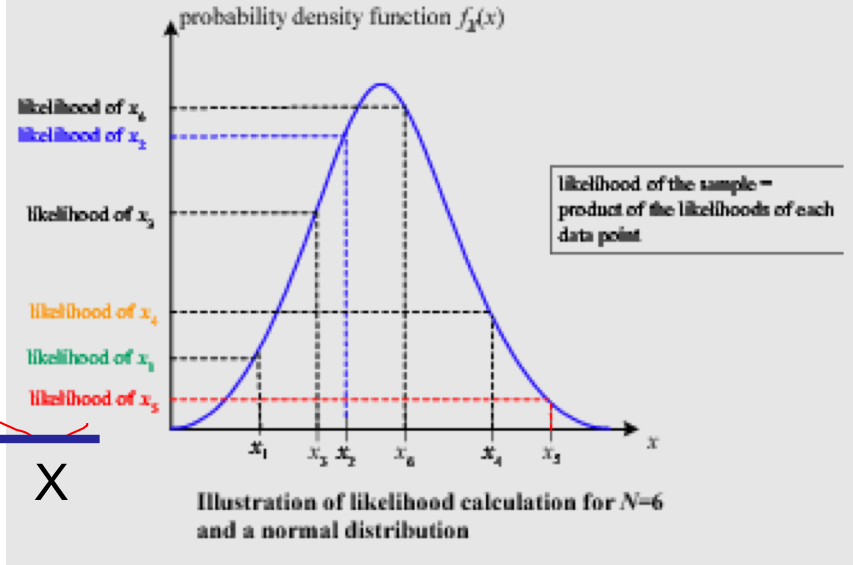
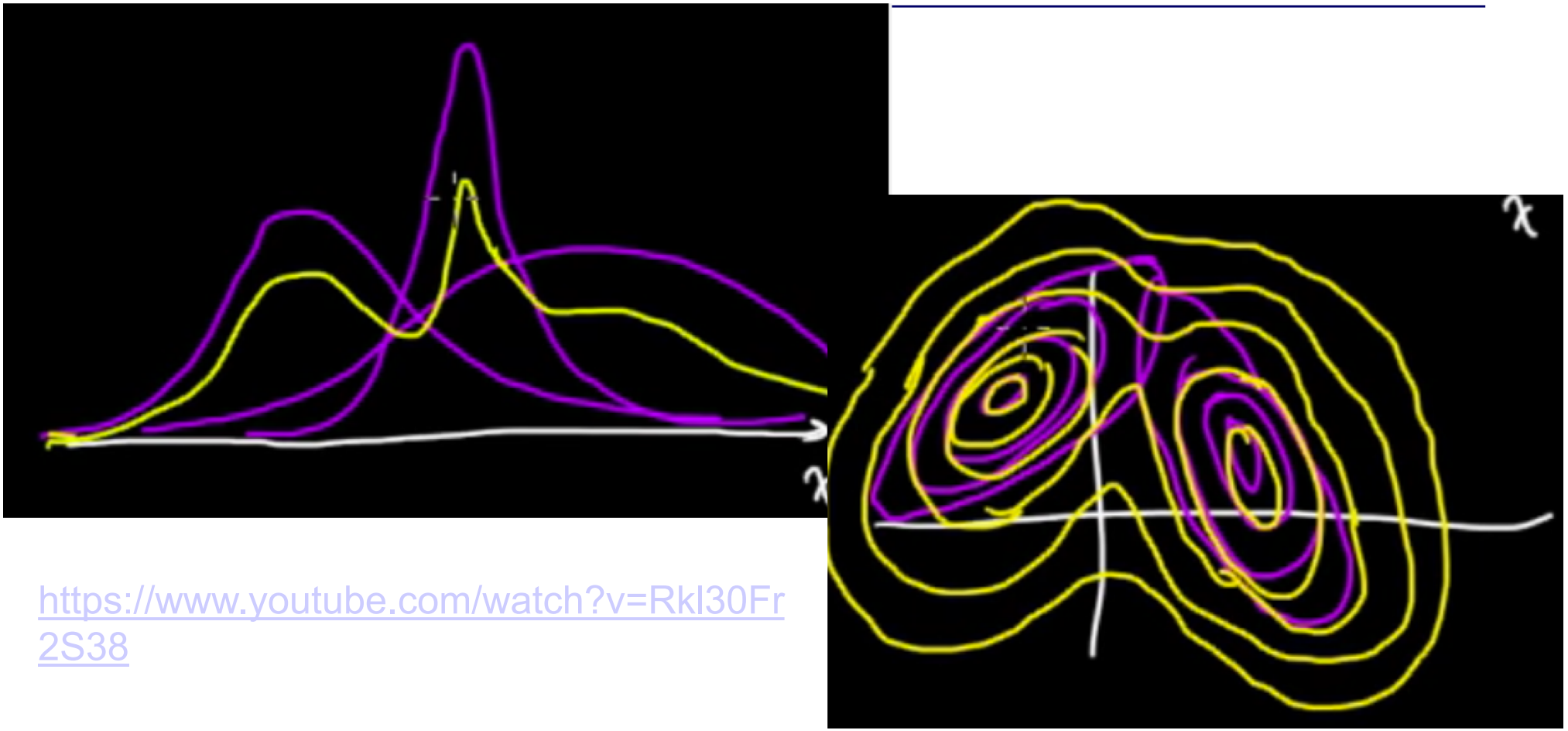


Illustration of likelihood calculation for N=6 and a normal distribution

PDF of a mixture of Gaussians



https://www.youtube.com/watch?v=RkI30Fr2S38

Yellow curve is convex combination of the individual Gaussian

Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

MLE, MAP and Bayes Optimal Classifier

- **In the following we will introduce the general form of the Maximum Likelihood Estimation (MLE) which is a special case of the so called Maximum a Posteriori (MAP) estimation.**
- **MAP in turn is an approximation of the Bayesian prediction (Bayes Optimal Classifier).**

Use Bayes Rule **Posterior** $\propto$ **Likelihood.** **Prior**

P(Hypothesis | Data) $\propto$ **P(Data | Hypothesis)** P(Hypothesis)

P(Data | Hypothesis) can be calculated using the training data;
P(Hypothesis) is uniform so drop in the case of MLE

$MLE_{NaiveBayes}$ = P(Data | Hypothesis) = $\prod_{d=1}^{D}\left(p(C_d|\pi)\prod_{n=1}^{N}p(w_n|\theta_{C_d})\right)$
Choose the model that maximizes **$MLE_{NaiveBayes}$**

MLE, MAP and Bayes Optimal Classifier

- **Imagine you want to learn a model/hypothesis from data, e.g., a Naïve Bayes or even a prediction**
- **We want to estimate P(Hypothesis | Data)**
  - Use Bayes Rule **Posterior** $\propto$ **Likelihood.** **Prior**
    - P(Hypothesis | Data) $\propto$ P(Data | Hypothesis) P(Hypothesis)
- **3 Possible ways to estimate P(Hypothesis | Data)**
  - H_{BO} Ensemble (Bayes optimal model)
  - H_{MAP} Have preferences (MAP): H_{MAP}
  - H_{ML} Assume all possible models are equally likely. (uniform prior; MaxLikelihood):

MLE for Gaussian

Goal

This method is concerned with the parametric modelling of a probability distribution for a random vector $\underline{X}=(X^1,\ldots,X^{n_X})$. The appropriate probability distribution is found by using a sample of data $\{\underline{x}_1,\ldots,\underline{x}_N\}$. Such an approach can be described in two steps as follows:

- Choose a probability distribution (e.g. the Normal distribution, or any other distribution available in OpenTURNS see [standard parametric models]),
- Find the parameter values $\underline{\theta}$ that characterize the probability distribution (e.g. the mean and standard deviation for the Normal distribution) which best describes the sample $\{\underline{x}_1,\ldots,\underline{x}_N\}$.

The maximum likelihood method is used for the second step.

Principle

In the current version of Open TURNS this method is restricted to the case where $n_X = 1$ and continuous probability distributions. Please note therefore that $\underline{X} = X^1 = X$ in the following text. The maximum likelihood estimate (MLE) of $\underline{\theta}$ is defined as the value of $\underline{\theta}$ which maximizes the likelihood function $L(X,\underline{\theta})$:

$$\hat{\underline{\theta}} = \operatorname{argmax} L(X,\underline{\theta})$$

Given that $\{x_1,\ldots,x_N\}$ is a sample of independent identically distributed (i.i.d) observations, $L(x_1,\ldots,x_N,\underline{\theta})$ represents the probability of observing such a sample assuming that they are taken from a probability distribution with parameters $\underline{\theta}$. In concrete terms, the likelihood $L(x_1,\ldots,x_N,\underline{\theta})$ is calculated as follows:

$$L(x_1,\ldots,x_N,\underline{\theta}) = \prod_{j=1}^{N} f_X(x_j;\underline{\theta}) \text{ if the distribution is continuous, with density } f_X(x;\underline{\theta})$$

For example, if we suppose that X is a Gaussian distribution with parameters $\underline{\theta} = \{\mu,\sigma\}$ (i.e. the mean and standard deviation),

$$\begin{aligned} L(x_1,\ldots,x_N,\underline{\theta}) &= \prod_{j=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x_j-\mu}{\sigma}\right)^2\right] \\ &= \frac{1}{\sigma^N(2\pi)^{N/2}} \exp\left[-\frac{1}{2\sigma^2}\sum_{j=1}^{N}(x_j-\mu)^2\right] \end{aligned}$$

The following figure graphically illustrates the maximum likelihood method, in the particular case of a Gaussian probability distribution.

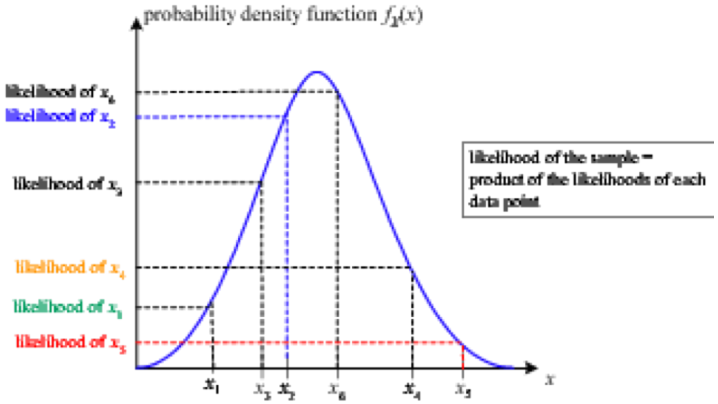


Illustration of likelihood calculation for N=6 and a normal distribution

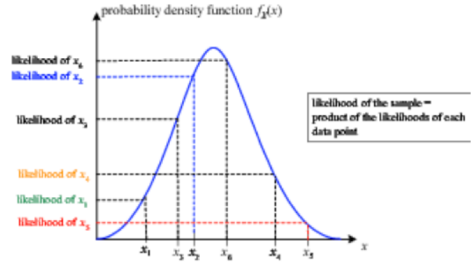


Illustration of likelihood calculation for N=6 and a normal distribution

In general, in order to maximize the likelihood function classical optimisation algorithms (e.g. gradient type) can be used. The Gaussian distribution case is an exception to this, as the maximum likelihood estimators are obtained analytically:

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i,\ \widehat{\sigma^2} = \frac{1}{N}\sum_{i=1}^{N}(x_i-\hat{\mu})^2$$

Maximum Likelihood Model

Compute the optimal hypothesis h_{ML} which is the most probable given the data by maximizing just the likelihood $Pr(d|h_i)$

9.6.3 Maximum Likelihood

A further simplification of the MAP can be made by the assumption that we have the same prior probability $P(h_i)$ for all hypotheses. In applications where a hypothesis is represented by a set of parameters for a given model, we do not have a reason to prefer one single set of parameters. Thus the upper assumption is valid. The formula for the computation of the most probable hypothesis now becomes even more simple.

Definition 9.33 (ML Hypothesis)

$$
\begin{aligned}
h_{ML} &= \arg\max_i P(h_i|d) && (9.23)\\
&= \arg\max_i \alpha P(d|h_i)\, P(h_i) && (9.24)\\
&= \arg\max_i P(d|h_i)\, P(h_i) && (9.25)\\
&= \arg\max_i P(d|h_i) && (9.26)
\end{aligned}
$$

This formula tells us that we can compute the optimal hypothesis h_{ML} which is the most probable one given the data d by maximizing just $P(d|h_i)$. The fact that the maximized quantity $P(d|h_i)$ is the likelihood of the data should explain the term *Maximum Likelihood*.

MAP versus MLE

9.6.2 Maximum a Posteriori

The *Maximum a Posteriori (MAP)* prediction is based only on one hypothesis, not on the sum of all hypotheses as in Bayesian learning. The hypothesis h_{MAP} chosen for prediction is the most probable one, given the data.

Definition 9.32 (MAP Hypothesis)

$$h_{MAP} = \arg\max_i P(h_i|d) \quad (9.20)$$

$$= \arg\max_i \alpha P(d|h_i) P(h_i) \quad (9.21)$$

$$= \arg\max_i P(d|h_i) P(h_i) \quad (9.22)$$

This way the original task of a large (infinite) summation for the Bayesian Learning is replaced by the task of optimization for MAP. MAP is already a reasonable method for a estimation task. It estimates the hypothesis h_{MAP}. But we will see that in most practical cases we can even make a further simplification of the estimation process. This will lead us to the Maximum Likelihood Estimation (MLE).

9.6.3 Maximum Likelihood

A further simplification of the MAP can be made by the assumption that we have the same prior probability $P(h_i)$ for all hypotheses. In applications where a hypothesis is represented by a set of parameters for a given model, we do not have a reason to prefer one single set of parameters. Thus the upper assumption is valid. The formula for the computation of the most probable hypothesis now becomes even more simple.

Using MLE to fit a naive Bayes model

Use Bayes Rule **Posterior** ∝ **Likelihood.** **Prior**

P(Hypothesis | Data) ∝ **P(Data | Hypothesis)** P(Hypothesis)

$MLE_{NaiveBayes}$ = P(Data | Hypothesis) = $\prod_{d=1}^{D}\left(p(C_d|\pi)\prod_{n=1}^{N}p(w_n|\theta_{C_d})\right)$

We find the parameters in the posterior distribution used in class prediction by learning on the labeled training data (in this case, the ham and spam e-mails). We use the maximum likelihood method method in finding parameters that maximize the likelihood of the observed data set. Given data $\{w_{d,1:N}, c_d\}_{d=1}^{D}$, the likelihood under a certain model with parameters $(\theta_{1:C}, \pi)$ is

P(Θ|D) ∝ P(D|Θ) #MLE objective

$$p(\mathcal{D}|\theta_{1:C},\pi) = \prod_{d=1}^{D}\left(p(c_d|\pi)\prod_{n=1}^{N}p(w_n|\theta_{c_d})\right)$$

$$= \prod_{d=1}^{D}\prod_{c=1}^{C}\left(\pi_c\prod_{n=1}^{N}\prod_{v=1}^{V}\theta_{c,v}^{1(w_{d,n}=v)}\right)^{1(c_d=c)}$$

which is the product over each email (D), each category (C), each word in the e-mail (N), and each word in the vocabulary (V). The $1(c_d = c)$ term serves to filter out e-mails that are not under the specific category, and the $1(w_{d,n} = v)$ term serves to filter out the words in the vocabulary that don't appear in the e-mail. Then, taking the logarithm of each side gives

$$\mathcal{L}(\pi,\theta_{1:C};\mathcal{D}) = \sum_{d=1}^{D}\sum_{c=1}^{C}1(c_d=c)\log\pi_c + \sum_{d=1}^{D}\sum_{c=1}^{C}\sum_{n=1}^{N}\sum_{v=1}^{V}1(c_d=c)1(w_{d,n}=v)\log\theta_{c,v}$$

and since the logarithm is a monotonic function, maximizing the log likelihood is the same as maximizing the likelihood of the data. Taking the log allows you to decompose the likelihood into the two separate parts of your model–one term contains only π, and the other term contains only $\theta_{c,v}$. Thus, taking the derivative with respect to one of the parameters eliminates the other one. To maximize the values of π and $\theta_{c,v}$, take the derivative with respect to each and solve for zero. This leads to the maximum likelihood estimates:

$$\hat{\pi}_c = \frac{n_c}{D} \quad (9)$$

MLE for MNB

and

$$\hat{\theta}_{c,v} = \frac{n_{c,v}}{\sum_{v'} n_{c,v'}} \quad (10)$$

where n_c is the number of times you see class c and $n_{c,v}$ is the number of times you see word v in class c.

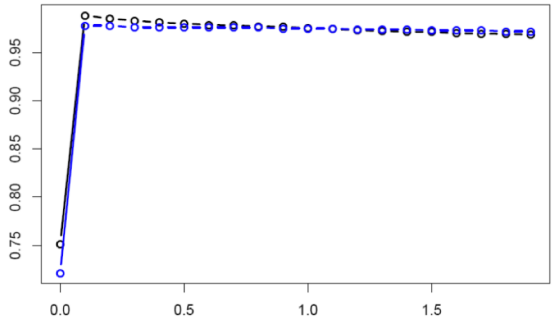
Case Study

We'll consider an example of using the naive Bayes model to do spam filtering. The e-mail data set comes from Enron's subpoenaed e-mails. The training set size is 10,000 e-mails (all labeled as spam or ham) and the test set size is 1,000 e-mails. Preprocessing has removed common words such as "the" from data set. In the graph below, the x-axis represents training size and the y-axis represents accuracy. Note the sensitivity of model performance to training data size. Test accuracy starts off bad, but gets better as training size increases and converges at around 70% accuracy. On the other hand, the training accuracy starts off very high and then drops off. This is due to overfitting, since when you're only training on a few e-mails, the word distribution can perfectly describe the training e-mails.

However, suppose in the above case study, we see a rare word like "peanut" appear in one spam e-mail and none of the ham e-mails. In that case, $\theta_{\text{spam,peanut}}$ will be some small value, but $\theta_{\text{ham,peanut}}$ will be 0. Since the probabilities are multiplied together, this leads to the problem that any e-mail containing the word "peanut" will never be classified as ham. This is clearly a ludicrous conclusion. To resolve that, we use smoothing to make the probability distribution more reasonable by adding some number λ to the per-word class counts. The result is that all words will have non-zero probabilities, while the more frequent words have slighly decreased probability, leading to a smoother overall distribution. Thus, the new equation is

$$\hat{\theta}_{c,w} = \frac{n_{c,w}+\lambda}{\sum_{w'} n_{c,w'} + V\lambda} \quad (11)$$

When $\lambda = 1$, we call it Laplace smoothing. When $\lambda = 0.5$, we call it Jeffrey's smoothing.



The x-axis is λ and the y-axis is the accuracy. We can see that just adding a little bit of smoothing results in a huge increase in accuracy. It seems that the problem of misclassification based on the appearances of rare words is fairly significant.

https://www.dropbox.com/s/rbceyfvsv5yp2yk/0214-Princeton-MLE-for-NaiveBayes.pdf?dl=0

Example: MAP hypothesis

- **In classification problems what we really want to compute is the probability of a class given an example.**
- **Assume we have three hypotheses h1, h2, and h3 (only three possible models in this example;**
  - in general we will have an infinite number of models/hypotheses)
  - with the following posterior probabilities (these posterior probabilities are calculated via the likelihood of the data as presented above):
-
- **P(h1|D) = 0.4, P(h2|D) = 0.3, and P(h3|D) = 0.3.**
-
- **Here *h1* is the MAP hypothesis.**
-

Example: Bayes optimal classification rule

- **P(h1|D) = 0.4, P(h2|D) = 0.3, and P(h3|D) = 0.3.**
- **Next, assume a new example, *X1*, is presented and suppose that the individual classifiers classify the example as follows (i.e., each classifier/hypothesis/model gets to vote in a binary fashion):**
  - P(h1) = + (i.e., $Pr_{h1}(+|X1)=1$ and $Pr_{h1}(-|X1)=0$),
  - P(h2) = -
  - and P(h3) = -.

- **How can we take all hypotheses/votes into account? We can take a weighted combination based on posterior probabilities. This is known as the Bayes optimal classification rule.**

Example: Bayes optimal classification rule

- **Given:**
  - P(h1|D) = 0.4, P(h2|D) = 0.3, and P(h3|D) = 0.3.
  - Inferred: P(h1) = +; P(h2) = - and P(h3) = -.
- **How can we take all hypotheses/votes into account? We can take a weighted combination based on posterior probabilities. This is known as the Bayes optimal classification rule.**
-
- **Let *c* be a possible class, then the Bayes optimal probability is calculated as follows:**
-
- $\mathbf{P(c \mid D) = \Sigma_i \; P(c \mid h_i) \; P (h_i \mid D)}$
  - i.e., a weighted sum over all possible hypothesis h_i.
- **And we want to choose the *class* that maximizes this probability:**
-
- $\mathbf{argmax\ c\ \Sigma i \; P(c \mid h_i) \; P (h_i \mid D)}$
-

Example: Bayes optimal class is -

- **Given:**
  - P(h1|D) = 0.4, P(h2|D) = 0.3, and P(h3|D) = 0.3.
  - Inferred: P(h1) = +; P(h2) = - and P(h3) = -.
- **Let *c* be a possible class, then the Bayes optimal probability is calculated as follows:**
  - $P(c \mid D) = \Sigma_i \; P(c \mid h_i) \; P (h_i \mid D)$
- **And we want to choose the *class* that maximizes this probability:**
- **argmax c $\Sigma i \; P(c \mid h_i) \; P (h_i \mid D)$**
-
- **P(h1|D) = 0.4 P(-|h1) = 0 P(+|h1) = 1**
- **P(h2|D) = 0.3 P(-|h2) = 1 P(+|h2) = 0**
- **P(h3|D) = 0.3 P(-|h3) = 1 P(+|h3) = 0**
-
- **And then**
  - $\Sigma i \; P(+ \mid h_i) \; P (h_i \mid D) = 0.4$
  - $\Sigma i \; P(- \mid h_i) \; P (h_i \mid D) = 0.6$

Maximum likelihood estimation: MLE

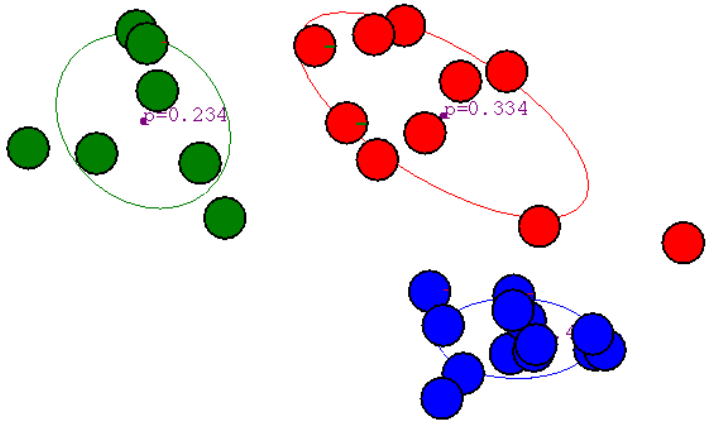
- From the point of view of Bayesian inference, MLE is a special case of maximum a posteriori estimation (MAP) that assumes a uniform prior distribution of the parameters.

- An MLE is the same regardless of whether we maximize the likelihood or the log-likelihood, because log is a strictly increasing function.
-

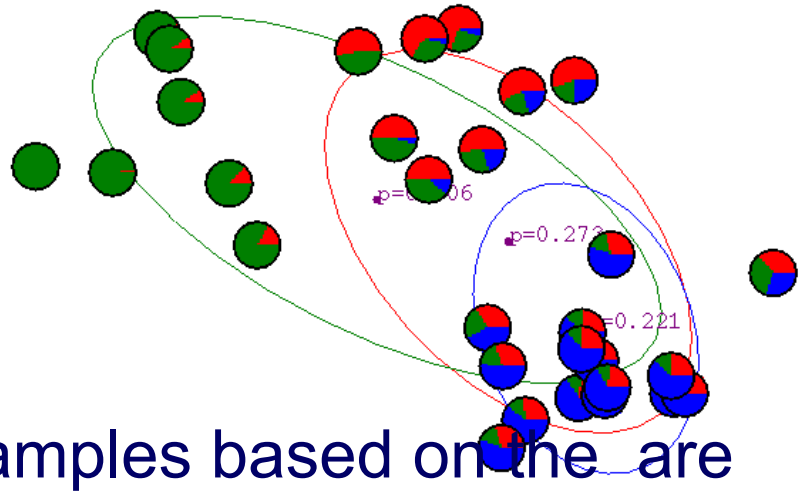
Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

Flat Clustering: Hard versus Soft



- **Hard Clustering**
  - Hard assignment; Kmeans (EM-like)
- **Soft Clustering**
  - This set of assignment probabilities (aka responsibilities) defines a soft clustering.
  - Model-based Clustering using EM and Maximum Likelihood



  - Weighted EM-like
    - EM Centroids are weighted examples based on the are the cluster/class probabilities assignment probabilities (aka responsibilites)

Flat Clustering: Hard vs. soft clustering

- **Hard clustering: Each example(document) belongs to exactly one cluster**
  - More common and easier to do
- **Soft clustering: An example can belong to more than one cluster.**
  - Makes more sense for applications like creating browsable hierarchies
  - You may want to put a "*pair of sneakers*" in two clusters: (i) *sports apparel* and (ii) *shoes*
  - You can only do that with a soft clustering approach.
- **See book IIR Section 16.5, 18**

EM Algorithm: distances vs Distributions

16.5 Model-based clustering

In this section, we describe a generalization of K-means, the EM algorithm. It can be applied to a larger variety of document representations and distributions than K-means.

In K-means, we attempt to find centroids that are good representatives. We can view the set of K centroids as a model that generates the data. Generating a document in this model consists of first picking a centroid at random and then adding some noise. If the noise is normally distributed, this procedure will result in clusters of spherical shape. *Model-based clustering* assumes that the data were generated by a model and tries to recover the original model from the data. The model that we recover from the data then defines clusters and an assignment of documents to clusters.

MODEL-BASED CLUSTERING

A commonly used criterion for estimating the model parameters is maximum likelihood.

Soft Clustering versus Hard Clustering

- Once we have a model Θ, we can compute an assignment probability **$P(d|c_k; \Theta)$ (AKA ($P(d|z_k; \Theta)$))** for each object-cluster pair.
- This set of assignment probabilities defines a soft clustering.
  - An example of a soft assignment is that a document about Chinese cars may have a fraction a membership of 0.5 in each of the two clusters China and automobiles, reflecting the fact that both topics are pertinent
- K-means and the hierarchical algorithms make fairly rigid assumptions about the data.
  - For example, clusters in K-means are assumed to be spheres.

Model-based clustering more flexible

- Model-based clustering provides a framework for incorporating our knowledge about a domain.
- **K** -means and the hierarchical algorithms make fairly rigid assumptions about the data.
  - For example, clusters in K -means are assumed to be spheres.
  - Model-based clustering offers more flexibility.
- The clustering model can be adapted to what we know about the underlying distribution of the data, be it Bernoulli, Gaussian with non-spherical variance (another model that is important in document clustering) or a member of a different family.

Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

Maximum Likelihood Estimation (MLE)

- We have reduced the problem of selecting the best model to that of selecting the best parameter.
- We want to select a parameter p which will maximize the probability that the data was generated from the model with the parameter p plugged-in.
- The parameter p is called the maximum likelihood estimator.
- The maximum of the function can be obtained by setting the derivative of the function ==0 and solving for p.

Two Important Facts

- **If $A_1,\ldots,A_n$ are independent then**

$$P(A_1,\ldots,A_n) = \prod_{i=1}^{n} P(A_i)$$

- **The log function is monotonically increasing.**
  - x · y = Log(x) · Log(y)
  - Therefore if a function f(x) >= 0, achieves a maximum at x1, then log(f(x)) also achieves a maximum at x1.

Example of MLE

$$
\begin{aligned}
L(p) &= P(0,1,1,0,0,1,0,1|p) \\
&= P(0|p)P(1|p)\ldots P(1|p) \\
&= (1-p)p\ldots p \\
&= p^4(1-p)^4
\end{aligned}
$$

- **Now, choose p which maximizes L(p). Instead we will maximize l(p)= LogL(p)**

$$
\begin{aligned}
\ell(p) &= logL(p) = 4log(p) + 4log(1-p) \\
\frac{d\ell(p)}{dp} &= \frac{4}{p} - \frac{4}{1-p} \equiv 0 \\
\rightarrow p &= \frac{1}{2}
\end{aligned}
$$

MLE for binary events: solve

- **Suppose we have the following data**
  - 0,1,1,0,0,1,1,0
- **In this case it is sensible to choose the Bernoulli distribution (B(p)) as the model space.**
- **Now we want to choose the best p, i.e.,**

http://math.arizona.edu/~jwatkins/n-mle.pdf

$$P(X = x) = p^x(1-p)^{1-x}$$

$$\text{argmax}_p P(Data|B(p))$$

Example 2 (Bernoulli trials). *If the experiment consists of n Bernoulli trial with success probability θ, then*

$$\mathbf{L}(\theta|\mathbf{x}) = \theta^{x_1}(1-\theta)^{(1-x_1)} \cdots \theta^{x_n}(1-\theta)^{(1-x_n)} = \theta^{(x_1+\cdots+x_n)}(1-\theta)^{n-(x_1+\cdots+x_n)}.$$

$$\ln \mathbf{L}(\theta|\mathbf{x}) = \ln\theta(\sum_{i=1}^{n} x_i) + \ln(1-\theta)(n - \sum_{i=1}^{n} x_i) = n\bar{x}\ln\theta + n(1-\bar{x})\ln(1-\theta).$$

$$\frac{\partial}{\partial\theta}\ln \mathbf{L}(\theta|\mathbf{x}) = n\left(\frac{\bar{x}}{\theta} - \frac{1-\bar{x}}{1-\theta}\right).$$

This equals zero when $\theta = \bar{x}$. Check that this is a maximum. Thus,

$$\hat{\theta}(\mathbf{x}) = \bar{x}.$$

$\hat{\theta}(\mathbf{x})$ is a maximum likelihood estimator for θ.

$$\hat{\theta}(\mathbf{x}) = \frac{\text{\#Successes}}{\text{\#Trials}}$$

As before, we begin with a sample $X = (X_1, \ldots, X_n)$ of random variables chosen according to one of a family of probabilities P_θ.

In addition, $\mathbf{f}(\mathbf{x}|\theta)$, $\mathbf{x} = (x_1, \ldots, x_n)$ will be used to denote the density function for the data when θ is the true state of nature.

Definition 1. *The* **likelihood function** *is the density function regarded as a function of* θ.

$$\mathbf{L}(\theta|\mathbf{x}) = \mathbf{f}(\mathbf{x}|\theta),\ \theta \in \Theta. \tag{1}$$

The **maximum likelihood estimator (MLE),**

$$\hat{\theta}(\mathbf{x}) = \arg\max_{\theta} \mathbf{L}(\theta|\mathbf{x}). \tag{2}$$

Example 2 (Bernoulli trials)**.** *If the experiment consists of* n *Bernoulli trial with success probability* θ*, then*

$$\mathbf{L}(\theta|\mathbf{x}) = \theta^{x_1}(1-\theta)^{(1-x_1)} \cdots \theta^{x_n}(1-\theta)^{(1-x_n)} = \theta^{(x_1+\cdots+x_n)}(1-\theta)^{n-(x_1+\cdots+x_n)}.$$

$$\ln \mathbf{L}(\theta|\mathbf{x}) = \ln\theta(\sum_{i=1}^{n} x_i) + \ln(1-\theta)(n - \sum_{i=1}^{n} x_i) = n\bar{x}\ln\theta + n(1-\bar{x})\ln(1-\theta).$$

$$\frac{\partial}{\partial\theta}\ln \mathbf{L}(\theta|\mathbf{x}) = n\left(\frac{\bar{x}}{\theta} - \frac{1-\bar{x}}{1-\theta}\right).$$

This equals zero when $\theta = \bar{x}$*. Check that this is a maximum. Thus,*

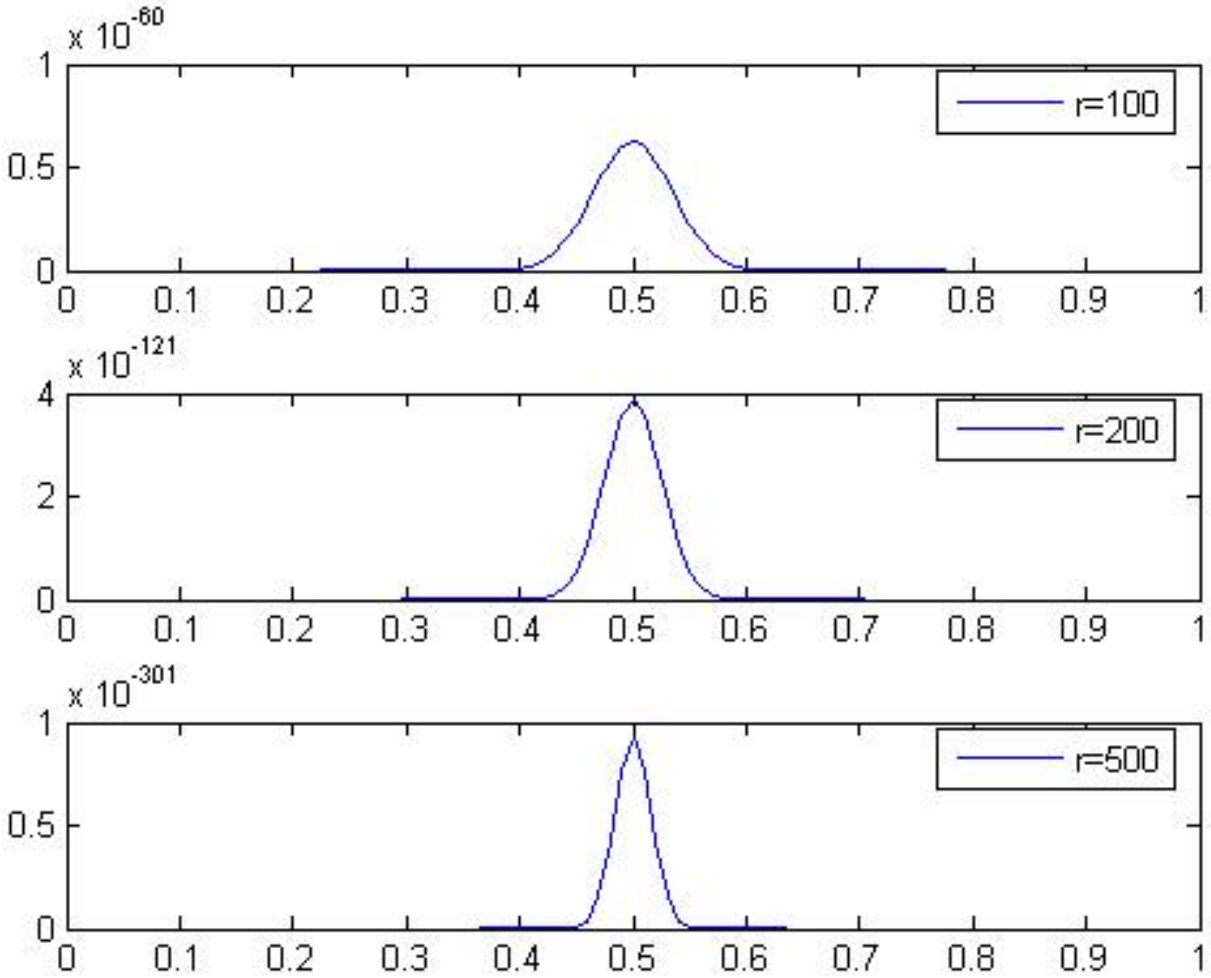
$$\hat{\theta}(\mathbf{x}) = \bar{x}. \qquad \hat{\theta}(\mathbf{x}) = \frac{\text{\#Successes}}{\text{\#Trials}}$$

$\hat{\theta}(\mathbf{x})$ is a maximum likelihood estimator for θ.

Properties of MLE

- **There are several technical properties of the estimator but lets look at the most intuitive one:**
  - As the number of data points increase we become more sure about the parameter p

Properties of MLE



r is the number of data points. As the number of data points increase the confidence of the estimator increases.

Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

Gaussian Model Example

Suppose the following are marks in a course

55.5, 67, 87, 48, 63

Marks typically follow a Normal distribution whose density function is

$$N(\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma}(x-\mu)^2}$$

Now, we want to find the best μ,σ such that

$$argmax_{\mu,\sigma} p(Data|\mu,\sigma)$$

$$\hat{\mu}(\mathbf{x}) = \bar{x}$$

$$\hat{\sigma}^2(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})^2.$$

$\hat{\mu}(\mathbf{x})\ \hat{\sigma}^2(\mathbf{x})$ **are MLE for μ, σ^2**

$$L(x_1,\ldots,x_N,\underline{\theta}) = \prod_{j=1}^{N}\frac{1}{\sigma\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{x_j-\mu}{\sigma}\right)^2\right]$$

$$= \frac{1}{\sigma^N(2\pi)^{N/2}}\exp\left[-\frac{1}{2\sigma^2}\sum_{j=1}^{N}(x_j-\mu)^2\right]$$

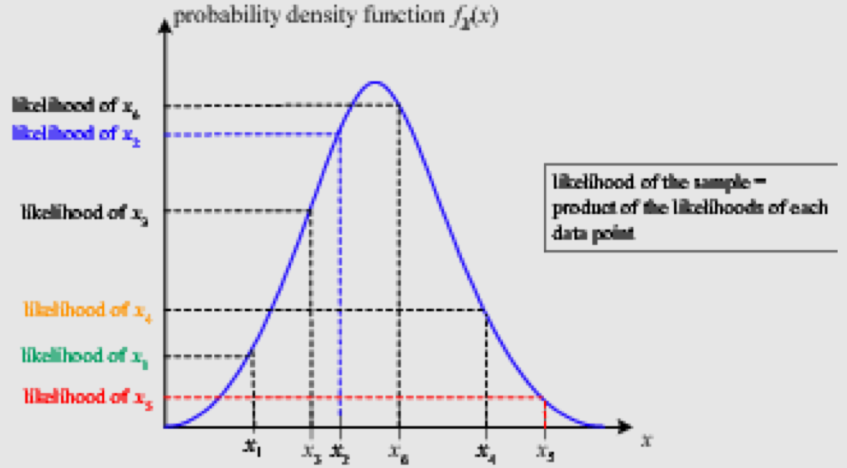


Illustration of likelihood calculation for N=6 and a normal distribution

$$L = f(x_1) f(x_2) \cdots f(x_n)$$

Let us take an example of a Normal distribution which probability density function is given as

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2} \right]$$

MLE of Gaussian

For n sample data x_i, the likelihood function is

$$L = \left(\frac{1}{\sqrt{2\pi}} \right)^n \left(\frac{1}{\sigma} \right)^n \exp[-b] \text{ where } b = \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2$$

Taking the logarithm to the likelihood function gives

$$\ln L = -n \ln\left(\sqrt{2\pi}\right) - n \ln(\sigma) - b$$

Taking the partial derivative with respect to the mean, we have

$$\frac{\partial}{\partial \mu} \ln L = -\frac{\partial b}{\partial \mu} = 0 \rightarrow \sum_{i=1}^{n} (x_i) - n\mu = 0 \rightarrow \boxed{\mu = \tfrac{1}{n} \sum_{i=1}^{n} x_i}$$

Taking the partial derivative with respect to the standard deviation, we have

$$\frac{\partial}{\partial \sigma} \ln L = -\frac{n}{\sigma} - \frac{\partial b}{\partial \sigma} = 0 \rightarrow -\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{i=1}^{n} (x_i - \mu)^2 = 0 \rightarrow \boxed{\sigma^2 = \tfrac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2}$$

Multivariate Normal Distributions

1.1 The multivariate normal distribution

The multivariate normal distribution in n-dimensions, also called the multivariate Gaussian distribution, is parameterized by a **mean vector** $\mu \in \mathbb{R}^n$ and a **covariance matrix** $\Sigma \in \mathbb{R}^{n \times n}$, where $\Sigma \geq 0$ is symmetric and positive semi-definite. Also written "$\mathcal{N}(\mu, \Sigma)$", its density is given by:

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right).$$

In the equation above, "$|\Sigma|$" denotes the determinant of the matrix Σ.

For a random variable X distributed $\mathcal{N}(\mu, \Sigma)$, the mean is (unsurprisingly) given by μ:

$$\mathrm{E}[X] = \int_x x\, p(x; \mu, \Sigma) dx = \mu$$

The **covariance** of a vector-valued random variable Z is defined as $\mathrm{Cov}(Z) = \mathrm{E}[(Z - \mathrm{E}[Z])(Z - \mathrm{E}[Z])^T]$. This generalizes the notion of the variance of a

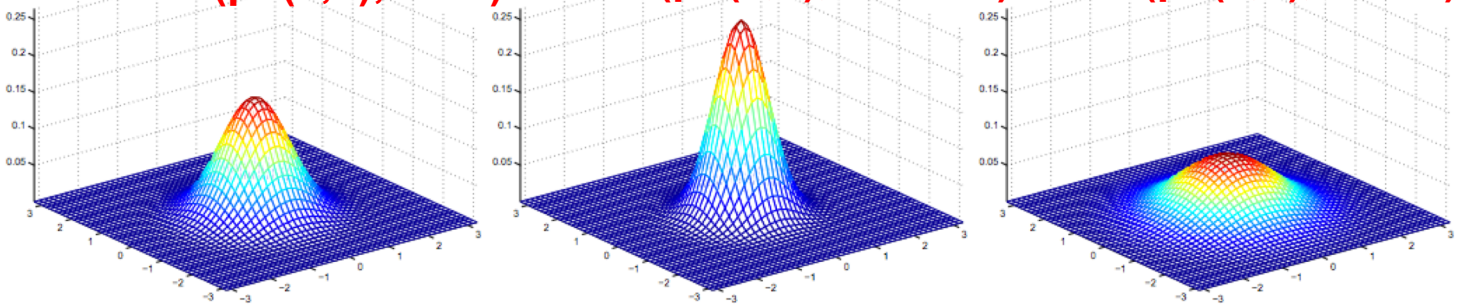
Covariance: E.g., standard normal distribution

real-valued random variable. The covariance can also be defined as $\mathrm{Cov}(Z) = \mathrm{E}[ZZ^T] - (\mathrm{E}[Z])(\mathrm{E}[Z])^T$. (You should be able to prove to yourself that these two definitions are equivalent.) If $X \sim \mathcal{N}(\mu, \Sigma)$, then

$$\mathrm{Cov}(X) = \Sigma.$$

Here're some examples of what the density of a Gaussian distribution looks like:

X ~ N(μ=(0,0), Σ = I) **X ~ N(μ=(0,0), Σ = 0.6I)** **X ~ N(μ=(0,0), Σ = 2I)**

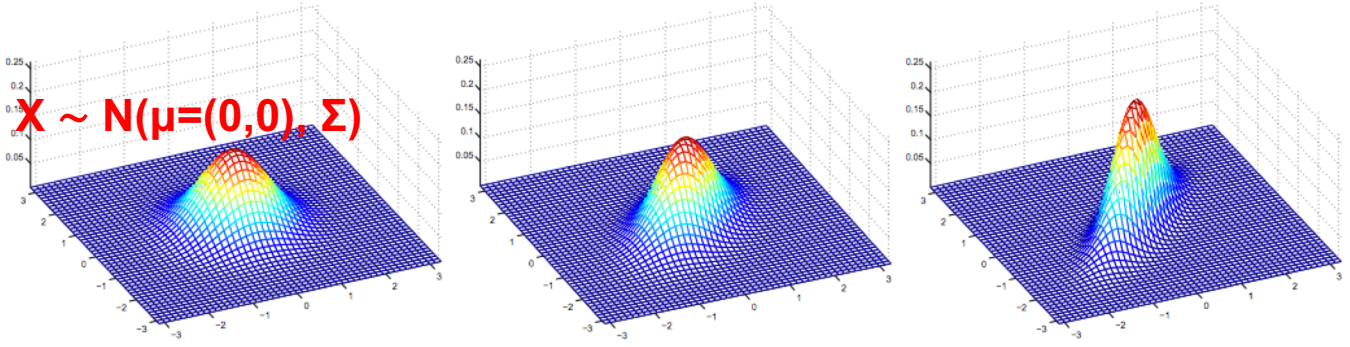


The left-most figure shows a Gaussian with mean zero (that is, the 2x1 zero-vector) and covariance matrix $\Sigma = I$ (the 2x2 identity matrix). A Gaussian with zero mean and identity covariance is also called the **standard normal distribution**. The middle figure shows the density of a Gaussian with zero mean and $\Sigma = 0.6I$; and in the rightmost figure shows one with , $\Sigma = 2I$. We see that as Σ becomes larger, the Gaussian becomes more "spread-out," and as it becomes smaller, the distribution becomes more "compressed."

Std versus varying off-diagonal elements in Σ

Let's look at some more examples.

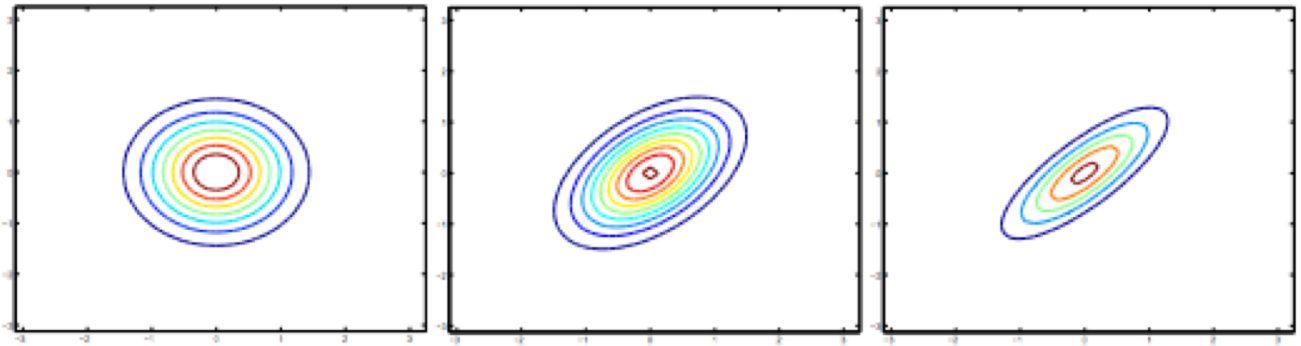
- ..



The figures above show Gaussians with mean 0, and with covariance matrices respectively

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}.$$

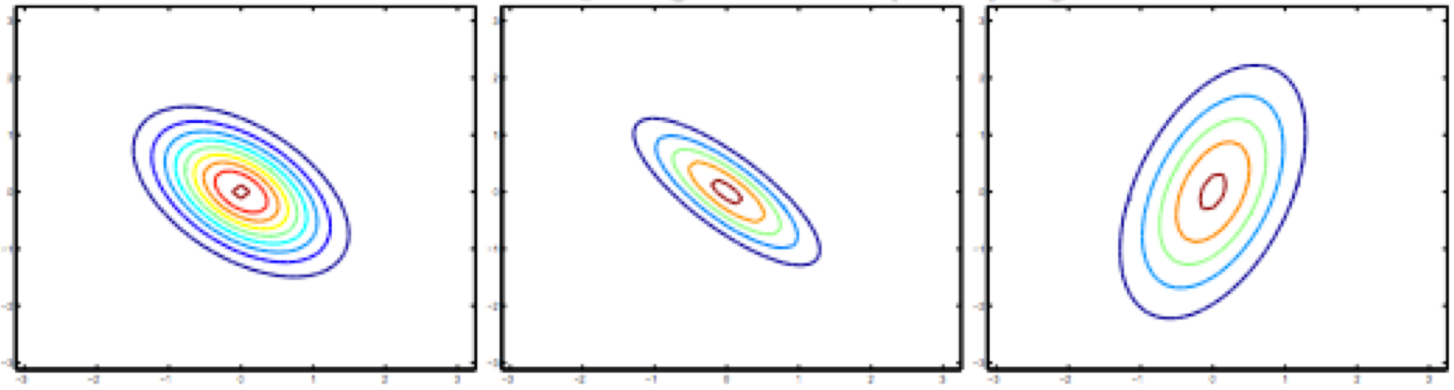
The leftmost figure shows the familiar standard normal distribution, and we see that as we increase the off-diagonal entry in Σ, the density becomes more "compressed" towards the 45° line (given by $x_1 = x_2$). We can see this more clearly when we look at the contours of the same three densities:



varying off-diagonal elements in Σ: bigger, positive versus negative

- ..

Here's one last set of examples generated by varying Σ:



The plots above used, respectively,

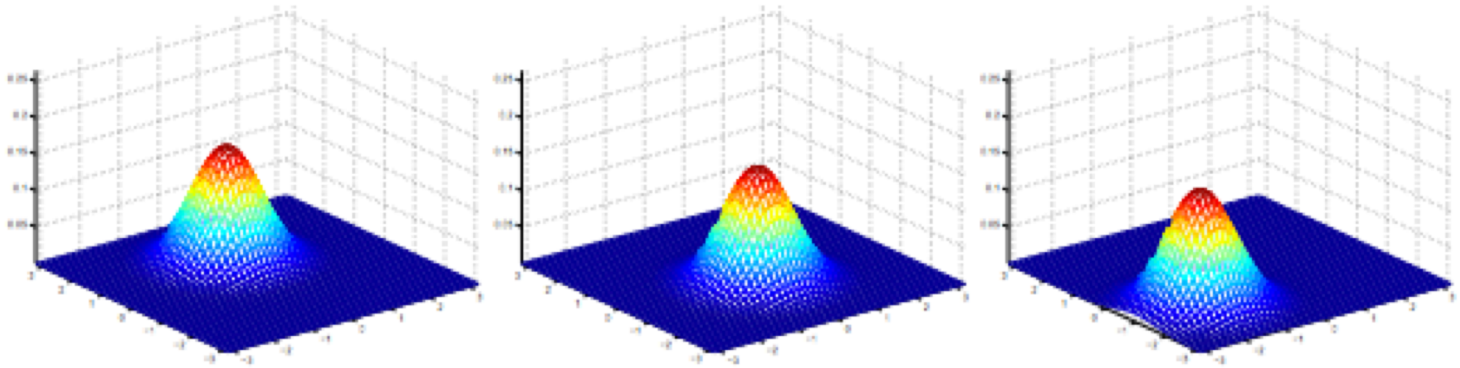
$$\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}.$$

From the leftmost and middle figures, we see that by decreasing the diagonal elements of the covariance matrix, the density now becomes "compressed" again, but in the opposite direction. Lastly, as we vary the parameters, more generally the contours will form ellipses (the rightmost figure showing an example).

Let covariance be identity matrix and vary μ

- Example:

As our last set of examples, fixing $\Sigma = I$, by varying μ, we can also move the mean of the density around.



The figures above were generated using $\Sigma = I$, and respectively

$$\mu = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad \mu = \begin{bmatrix} -0.5 \\ 0 \end{bmatrix}; \quad \mu = \begin{bmatrix} -1 \\ -1.5 \end{bmatrix}.$$

Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

Gaussian Model: Estimate Parameters

Suppose the following are marks in a course

55.5, 67, 87, 48, 63

Marks typically follow a Normal distribution whose density function is

$$N(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2\sigma}(x-\mu)^2}$$

Now, we want to find the best μ,σ such that

$$argmax_{\mu,\sigma} p(Data|\mu, \sigma)$$

$$\hat{\mu}(\mathbf{x}) = \bar{x}$$

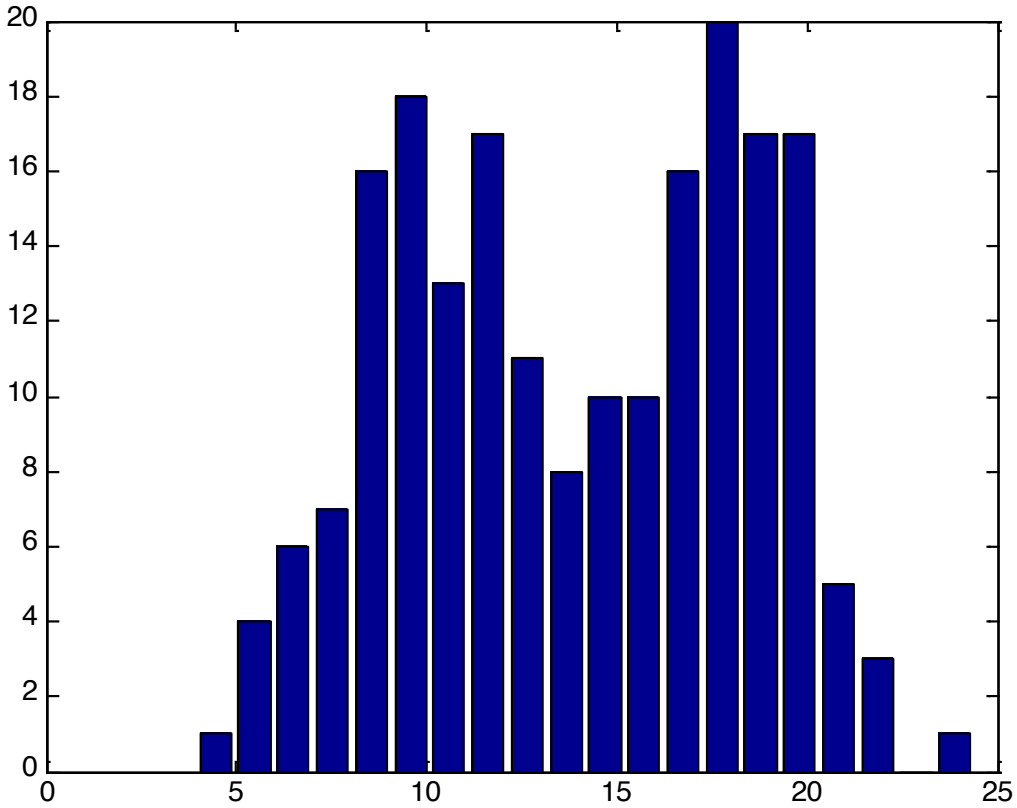
$$\hat{\sigma}^2(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})^2.$$

$\hat{\mu}(\mathbf{x})\ \hat{\sigma}^2(\mathbf{x})$ **are MLE for μ, σ²**

So far..

- **Let us summarize what you have learned so far:**
  - 1. Maximum Likelihood method is useful to find the parameters of a distribution.
  - 2. For GMM, maximum likelihood method using partial derivative is too difficult.
    - We need a numerical solution.

A Mixture Model Problem



- Apparently, the dataset consists of two modes
- How can we automatically identify the two modes?

Motivation

- **Problem:**

  Describe data with Mixture Model(MM)

- **Approach:**
  - Decide on MM, e.g.
    - Gauss distribution
    - Mix of two

$$p(x \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

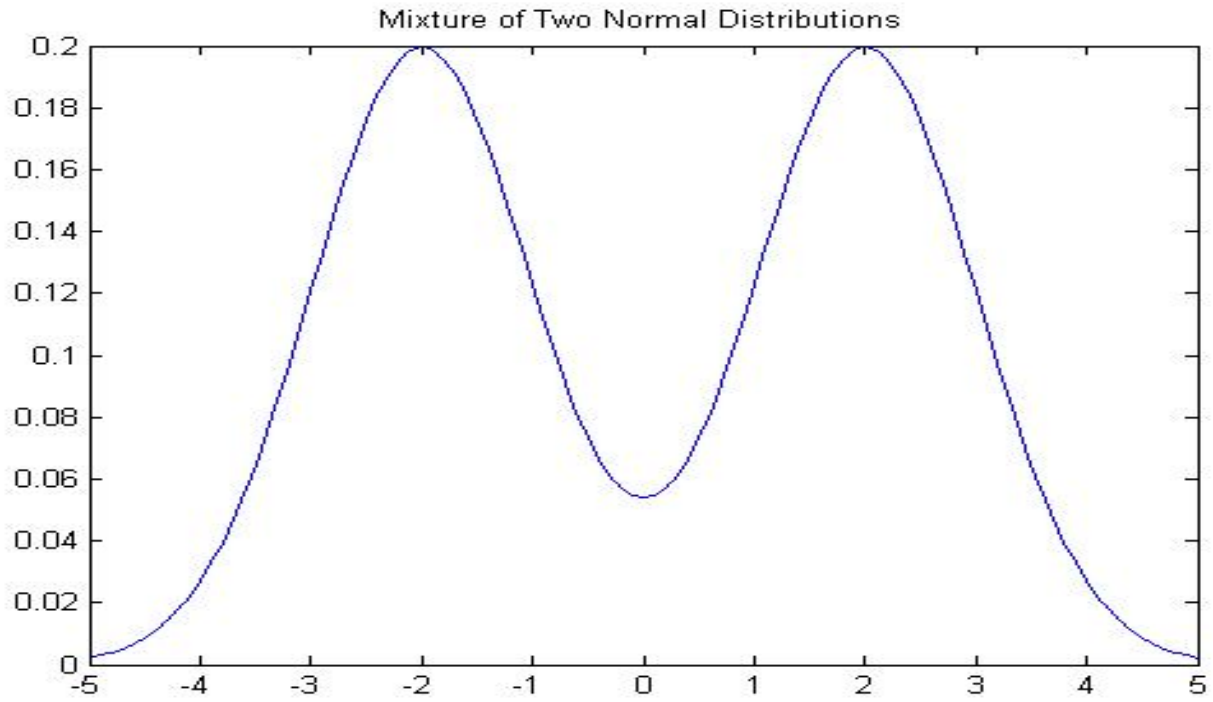
  - Estimate parameters $p(x \mid \Theta) = \alpha_1 p_1(x \mid \mu_1, \sigma_1^2) + \alpha_2 p_2(x \mid \mu_2, \sigma_2^2)$

$$\Theta = (\alpha_1, \alpha_2, \quad \mu_1, \sigma_1^2, \mu_2, \sigma_2^2)$$

Gaussian Mixture Model (GMM)

- **Assume that the dataset is generated by two mixed Gaussian distributions**
  - Gaussian model 1: $\theta_1 = \{\mu_1, \sigma_1; p_1\}$
  - Gaussian model 2: $\theta_2 = \{\mu_2, \sigma_2; p_2\}$
- **If we know the memberships for each bin, estimating the two Gaussian models is easy.**
- **How to estimate the two Gaussian models without knowing the memberships of bins?**

A Mixture Distribution

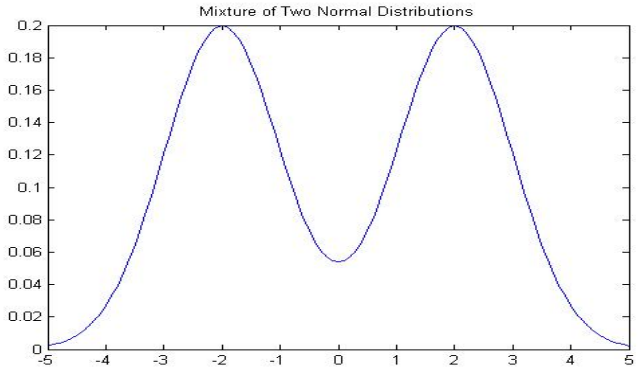


$$f(\pi_1, \pi_2, \mu_1, \mu_2, \sigma_1, \sigma_2) = \pi_1 N(\mu_1, \sigma_1) + \pi_2 N(\mu_2, \sigma_2)$$

Mixture of Gaussian Distributions Examples

- **Suppose we have data about heights of people (in cm)**
  - 185,140,134,150,170
- **Heights follow a normal (log normal) distribution but men on average are taller than women. This suggests a mixture of two distributions**

$$f(\pi_1, \pi_2, \mu_1, \mu_2, \sigma_1, \sigma_2) = \pi_1 N(\mu_1, \sigma_1) + \pi_2 N(\mu_2, \sigma_2)$$



Recap

- Multivariate Gaussian

$$p(\mathbf{x}|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

  - By translation and rotation, it turns into multiplication of **normal distributions**
  - MLE of mean: $\hat{\mu} = \frac{\Sigma_i x_i}{N}$
  - MLE of covariance: $\hat{\Sigma} = \frac{\Sigma_i (x_i - \hat{\mu})(x_i - \hat{\mu})^T}{N}$

MLE for Mixture Distributions

- When we proceed to calculate the MLE for a mixture, the presence of the sum of the distributions prevents a “neat” factorization using the log function.
- A completely new rethink is required to estimate the parameter.
- The new rethink also provides a solution to the clustering problem.

- Expectation-Maximization (EM) algorithm is an iterative procedure to estimate the maximum likelihood of the mixture density distribution.

Missing Data

- We think of clustering as a problem of estimating missing data.
- The missing data are the cluster labels.
- Clustering is only one example of a missing data problem. Several other problems can be formulated as missing data problems.

Types of classification data

Data set is complete.

Input **Classes** **Classification**

	x	y	z		
$\mathbf{x}^{(1)}$		2	0	1	0	…
$\mathbf{x}^{(2)}$		3	0	0	1	…
$\mathbf{x}^{(3)}$		1	1	0	0	…
⋮		⋮	⋮	⋮	⋮	⋮

K-means: Data set is incomplete, but we complete it using a specific cost function.

Missing Labels

	x	y	z		
$\mathbf{x}^{(1)}$		?	?	?	?	…
$\mathbf{x}^{(2)}$		?	?	?	?	…
$\mathbf{x}^{(3)}$		?	?	?	?	…
⋮		⋮	⋮	⋮	⋮	⋮

EM: data set is incomplete, but we complete it using posterior probabilities (a "soft" class membership).

Posterior Probs Pr(C|X)

x	y	z			
$\mathbf{x}^{(1)}$	?	$P(z_1=1\|\mathbf{x}^{(1)},\theta)$	$P(z_2=1\|\mathbf{x}^{(1)},\theta)$	$P(z_3=1\|\mathbf{x}^{(1)},\theta)$	…
$\mathbf{x}^{(2)}$	?	$P(z_1=1\|\mathbf{x}^{(2)},\theta)$	$P(z_2=1\|\mathbf{x}^{(2)},\theta)$	$P(z_3=1\|\mathbf{x}^{(2)},\theta)$	…
$\mathbf{x}^{(3)}$	?	$P(z_1=1\|\mathbf{x}^{(3)},\theta)$	$P(z_2=1\|\mathbf{x}^{(3)},\theta)$	$P(z_3=1\|\mathbf{x}^{(3)},\theta)$	…
⋮		⋮	⋮	⋮	⋮

Missing data problem

- **Let D = {x(1),x(2),…x(n)} be a set of n observations.**
- **Let H = {z(1),z(2),..z(n)} be a set of n values of a hidden variable Z.**
  - z(i) corresponds to x(i)
- **Assume Z is discrete.**

K-means: Data set is incomplete, but we complete it using a specific cost function.

Missing Labels

	x	y	z			
$\mathbf{x}^{(1)}$		?	?	?	?	…
$\mathbf{x}^{(2)}$		?	?	?	?	…
$\mathbf{x}^{(3)}$		?	?	?	?	…
⋮		⋮	⋮	⋮	⋮	⋮

EM: data set is incomplete, but we complete it using posterior probabilities (a "soft" class membership).

Posterior Probs Pr(C|X)

x	y	z			
$\mathbf{x}^{(1)}$	?	$P(z_1=1\|\mathbf{x}^{(1)},\theta)$	$P(z_2=1\|\mathbf{x}^{(1)},\theta)$	$P(z_3=1\|\mathbf{x}^{(1)},\theta)$	…
$\mathbf{x}^{(2)}$	?	$P(z_1=1\|\mathbf{x}^{(2)},\theta)$	$P(z_2=1\|\mathbf{x}^{(2)},\theta)$	$P(z_3=1\|\mathbf{x}^{(2)},\theta)$	…
$\mathbf{x}^{(3)}$	?	$P(z_1=1\|\mathbf{x}^{(3)},\theta)$	$P(z_2=1\|\mathbf{x}^{(3)},\theta)$	$P(z_3=1\|\mathbf{x}^{(3)},\theta)$	…
⋮		⋮	⋮	⋮	⋮

But $z^{(i)}$'s are not known: so use EM

- **However, in our density estimation problem, the $z^{(i)}$'s are not known.**
- **What can we do?**
- **The EM algorithm is an iterative algorithm that has two main steps.**
  - Applied to our problem, in the E-step, it tries to "guess" the values of the $z_{(i)}$'s.
  - In the M-step, it updates the parameters of our model based on our guesses. Since in the M-step we are pretending that the guesses in the first part were correct, the maximization becomes easy. Here's the algorithm:

Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

EM algorithm for GMM/BMM/etc.

- In this section, we describe a generalization of K-means, the EM algorithm.
- It can be applied to a larger variety of domains and distributions than K-means.
- Model-based clustering assumes that the data were generated by a model and tries to recover the original model from the data.
- The model that we recover from the data then defines clusters and an assignment of documents to clusters

- EM algorithm for GMM/BMM/etc.

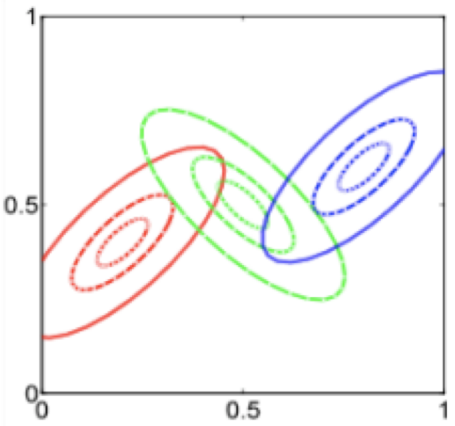
EM for GMM and BMM

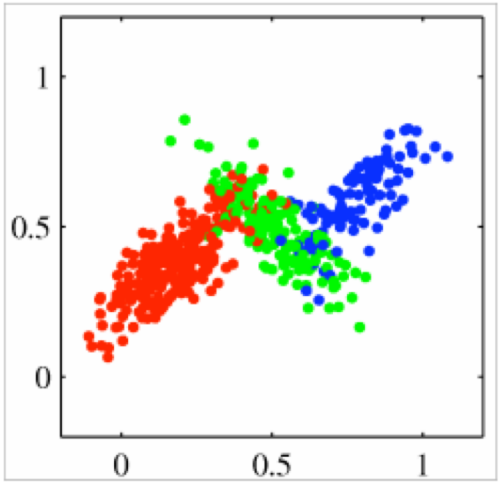
- **A commonly used algorithm for model-based clustering is the Maximization algorithm or EM algorithm.**
- **EM clustering is an iterative algorithm that maximizes L(D|Θ).**
- **EM can be applied to many different types of probabilistic modeling.**
  - We will work with a mixture of Gaussians (Focus on this one)
  - We will work with a mixture of multivariate Bernoulli distributions (included in slides)

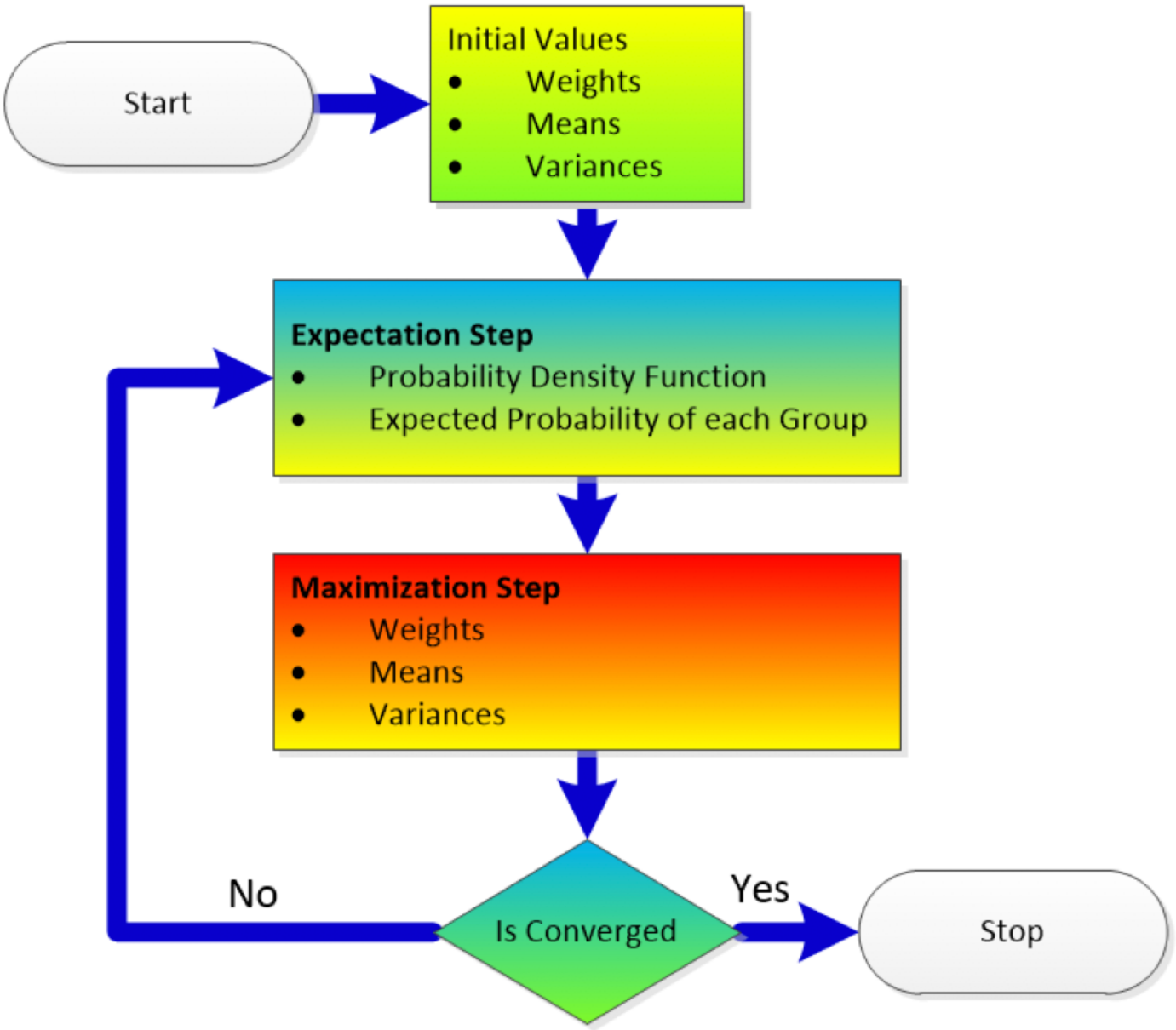
Gaussian Mixture Model

- Probabilistic story: Each cluster is associated with a Gaussian distribution. To generate data, randomly choose a cluster k with probability π_k and sample from its distribution.
- Likelihood $\Pr(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$ where

$$\sum_{k=1}^{K} \pi_k = 1, 0 \leq \pi_k \leq 1.$$






Start
Initial Values
Weights
Means
Variances
Expectation Step
Probability Density Function
Expected Probability of each Group
Maximization Step
Weights
Means
Variances
Is Converged
No
Yes
Stop

MLE Estimates for GMM

If we knew z: class assignments for each example

likelihood problem would have been easy. Specifically, we could then write down the likelihood as

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^{m} \log p(x^{(i)}|z^{(i)}; \mu, \Sigma) + \log p(z^{(i)}; \phi).$$

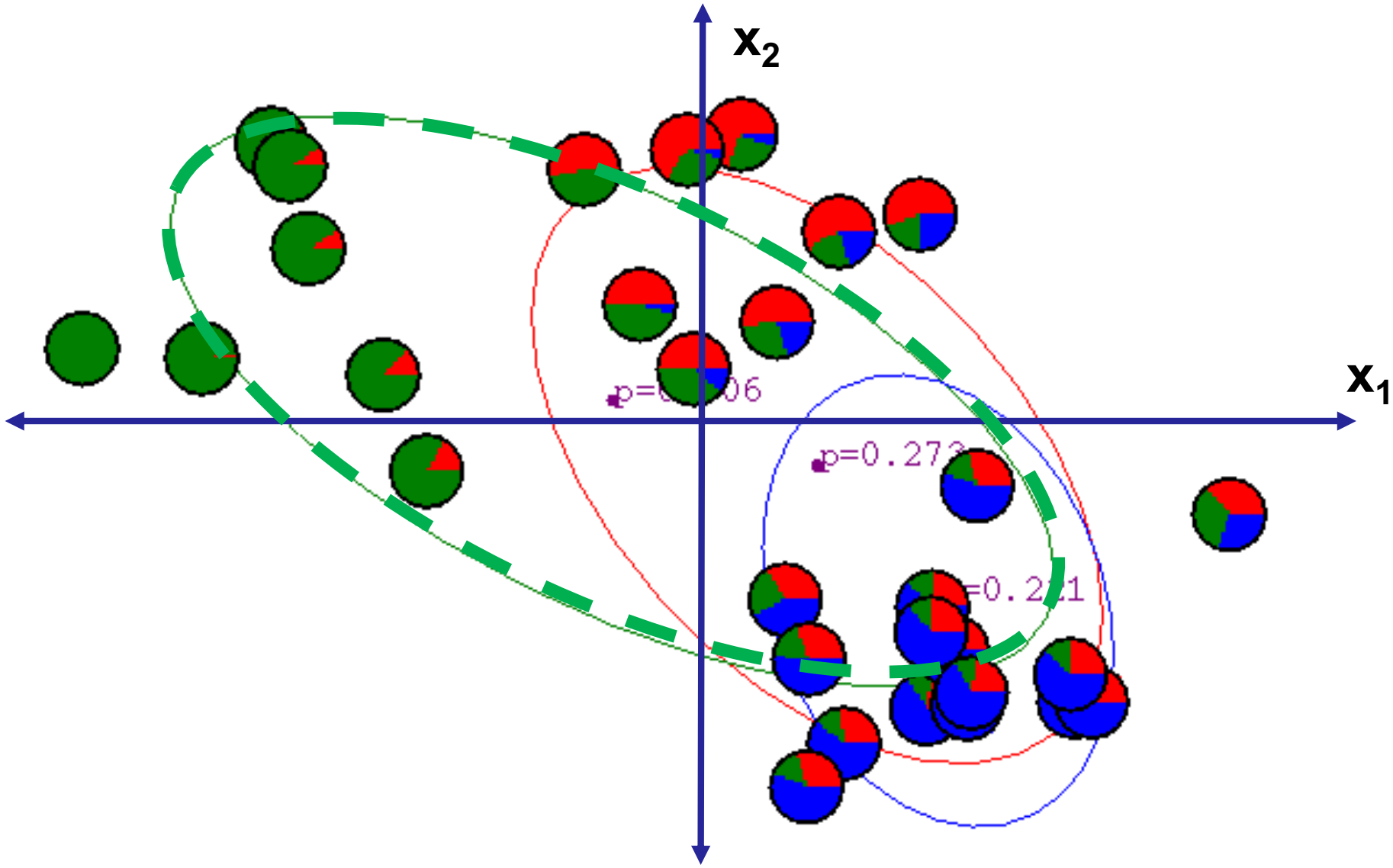
Maximizing this with respect to ϕ, μ and Σ gives the parameters:

$$\begin{aligned}
\phi_j &= \frac{1}{m} \sum_{i=1}^{m} 1\{z^{(i)} = j\}, \\
\mu_j &= \frac{\sum_{i=1}^{m} 1\{z^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{z^{(i)} = j\}}, \\
\Sigma_j &= \frac{\sum_{i=1}^{m} 1\{z^{(i)} = j\} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} 1\{z^{(i)} = j\}}.
\end{aligned}$$

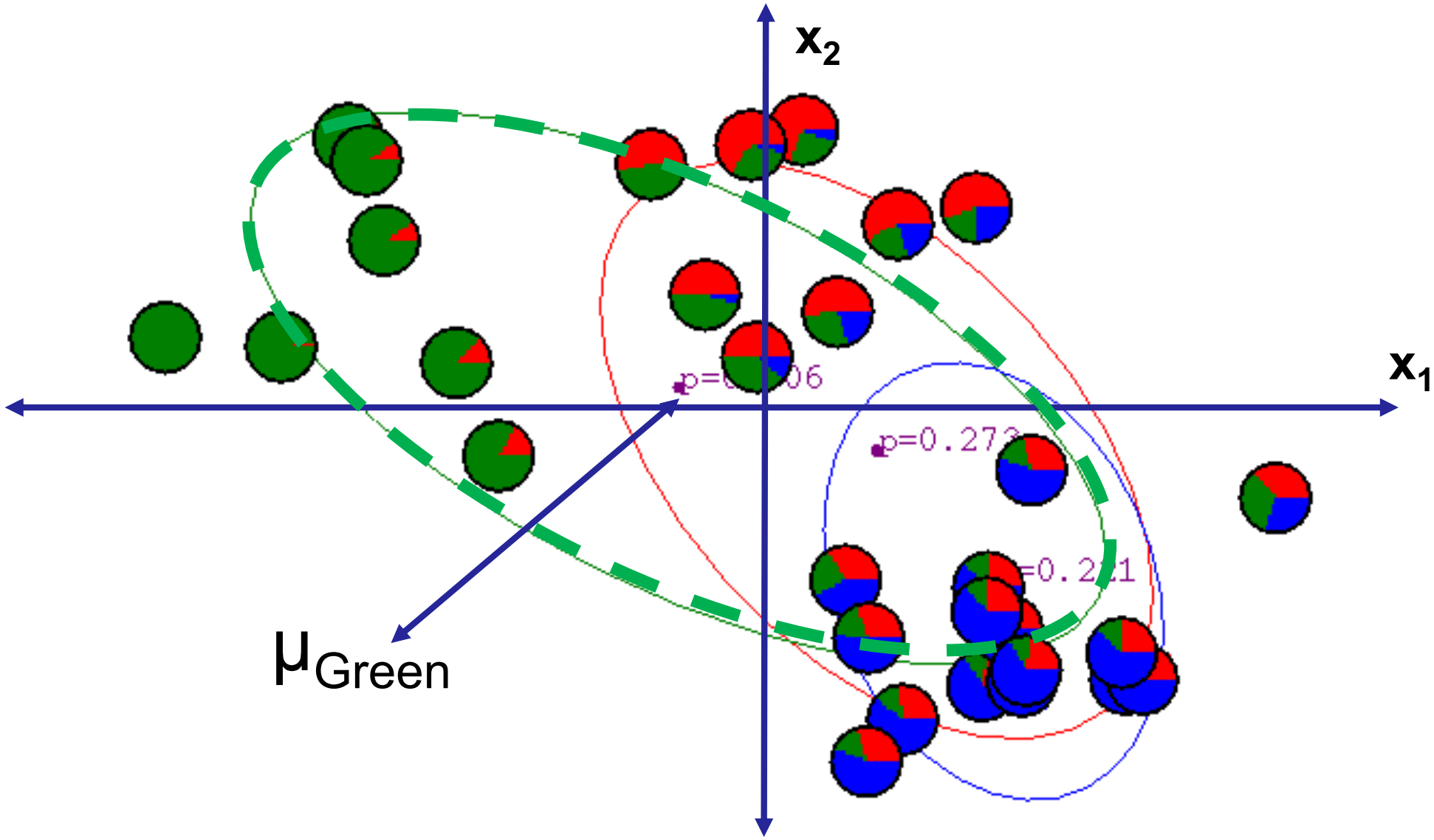
But $z^{(i)}$'s are not known: so use EM

- **However, in our density estimation problem, the $z^{(i)}$'s are not known.**
- **What can we do?**
- **The EM algorithm is an iterative algorithm that has two main steps.**
  - Applied to our problem, in the E-step, it tries to "guess" the values of the $z_{(i)}$'s.
  - In the M-step, it updates the parameters of our model based on our guesses. Since in the M-step we are pretending that the guesses in the first part were correct, the maximization becomes easy. Here's the algorithm:

GMM with 3 clusters: intermediate EM result



GMM with 3 clusters: intermediate EM result



EM for GMM:

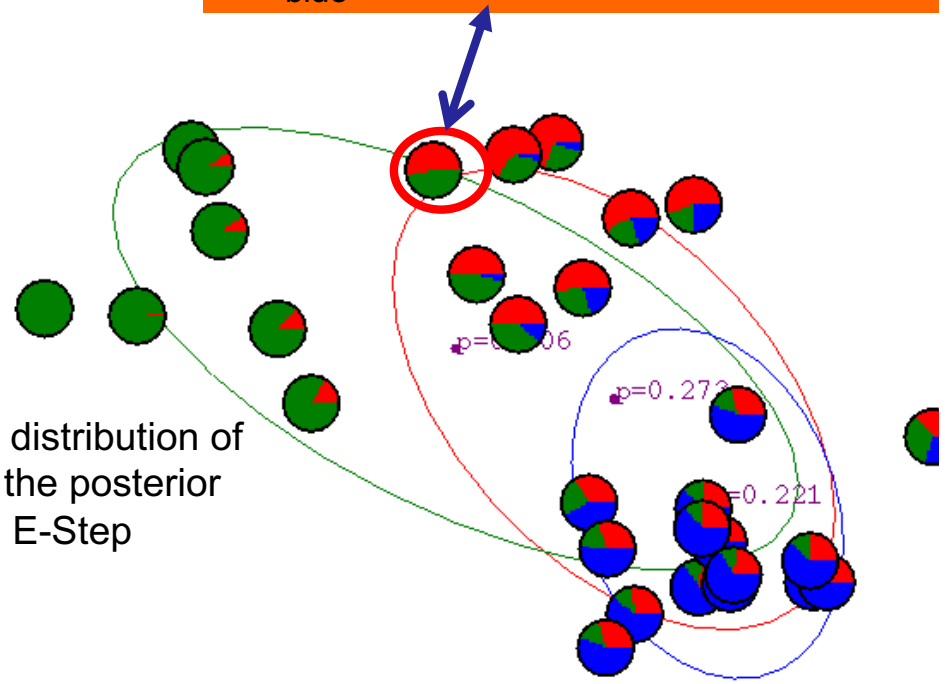
μ, Centroid
Σ, Variance
Φ, Cluster Priors

Repeat until convergence: {

(E-step) For each i, j, set J class; I example index

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

E-Step Pr(Cluster = Red|X) = W_{red}=0.5;
W_{green}=0.5;
W_{blue}=0.0



(M-step) Update the parameters:

phi $$\phi_j := \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)},$$

Compute relative distribution of each class using the posterior probabilities from E-Step

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

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

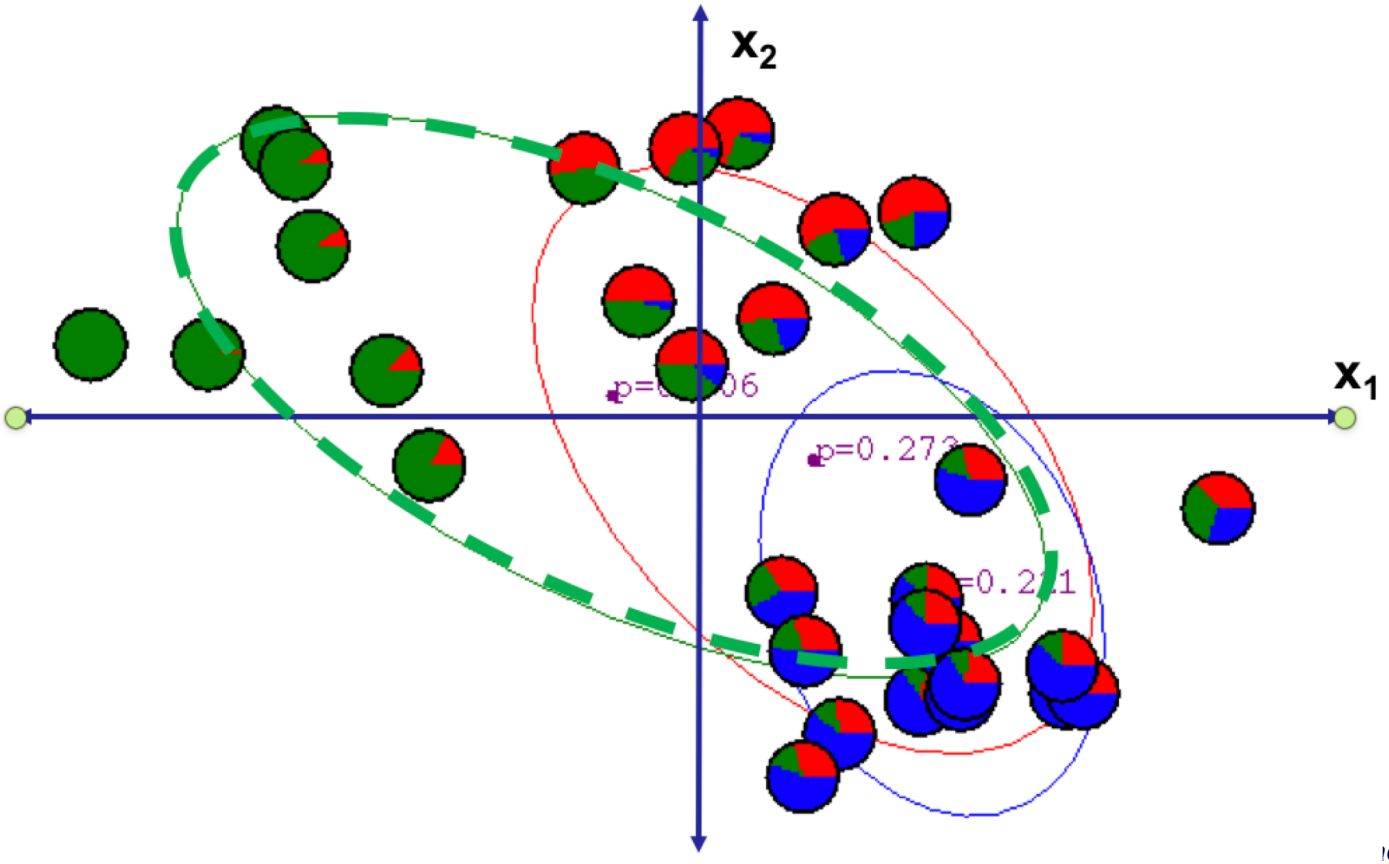
EM for GMM: E-Step

Repeat until convergence: {

(E-step) For each i, j, set

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

Think Cluster posteriors Pr(C|X)
View as weights



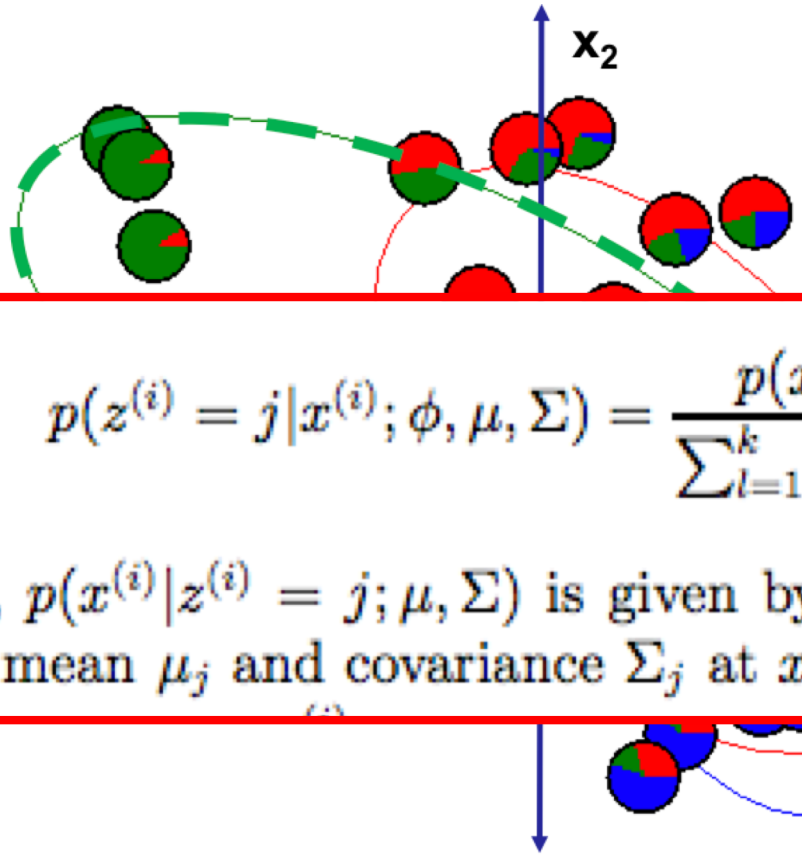
EM for GMM: E-Step

Repeat until convergence: {

(E-step) For each i, j, set

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

Think Cluster posteriors Pr(C|X)
View as weights



Estimate cluster/class assignments (responsibities) or weights
Use the current model to estimate the class assignments

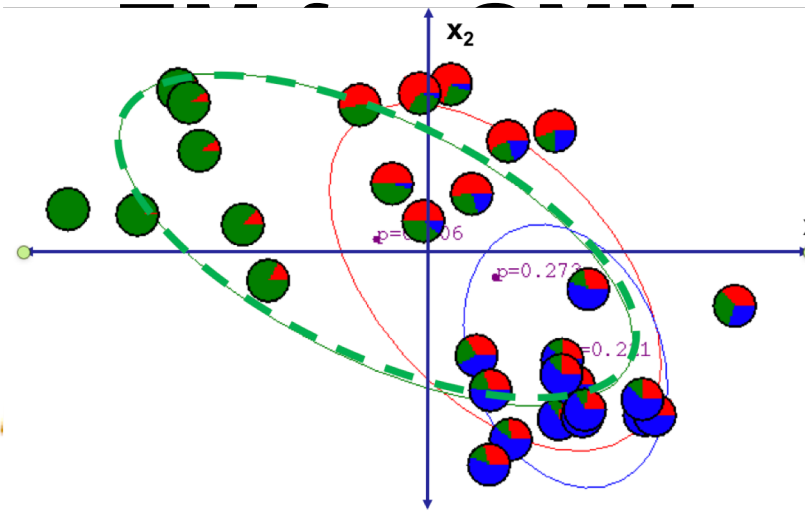
$$p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$

Here, $p(x^{(i)} | z^{(i)} = j; \mu, \Sigma)$ is given by evaluating the density of a Gaussian with mean μ_j and covariance Σ_j at $x^{(i)}$; $p(z^{(i)} = j; \phi)$ is given by ϕ_j, and so

Repeat until convergence: {

(E-step) For each i, j, set

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi,$$



(M-step) Update the parameters:

Total up the proportion of each class

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

Eq1

Kmeans

Weighted avg of X_1 and X_2

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

Eq 2 $\mu_j = \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{m}$

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

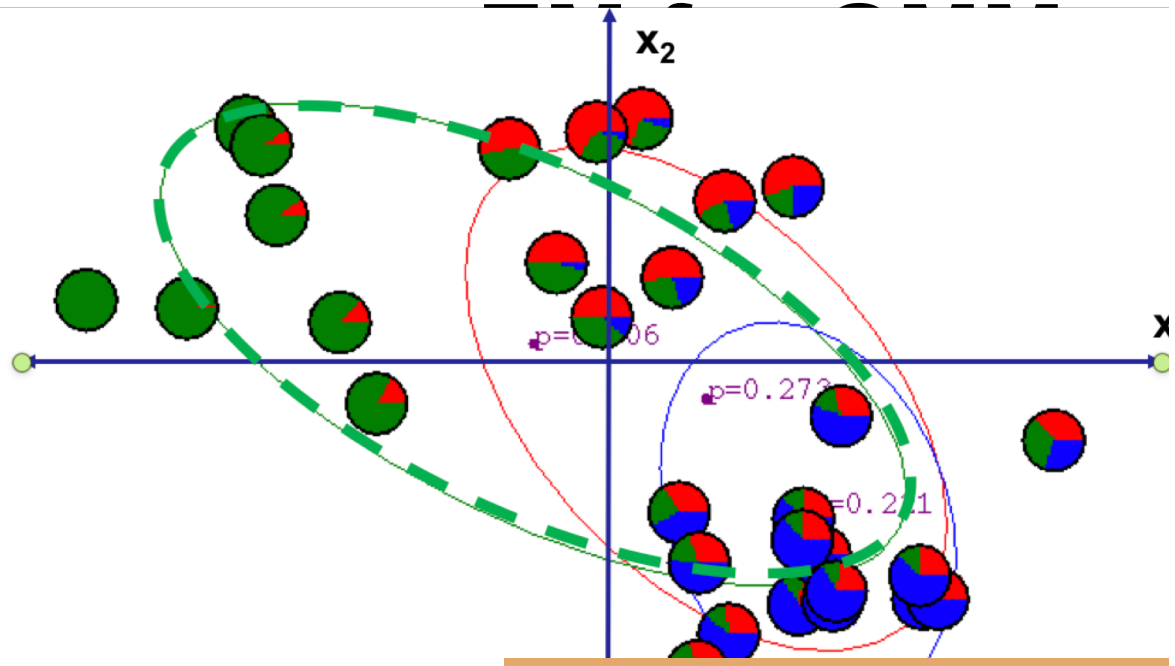
Eq3

Repeat until convergence: {

(E-step) For each i, j, set

$$w_j^{(i)} := p$$

(M-step) Update the paramet



Estimate class assignments (responsibities) or weights
Use the current model to estimate the class assignment

Total up the proportion of each class

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

Prior$_k$

Weighted avg of X_1 and X_2

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

Centroid$_k$

Centroid is just a weighted sum of soft assigned examples

Covariance$_k$

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

Eq3

Weighted covariance

Class prior is based on the sum of the partial weights

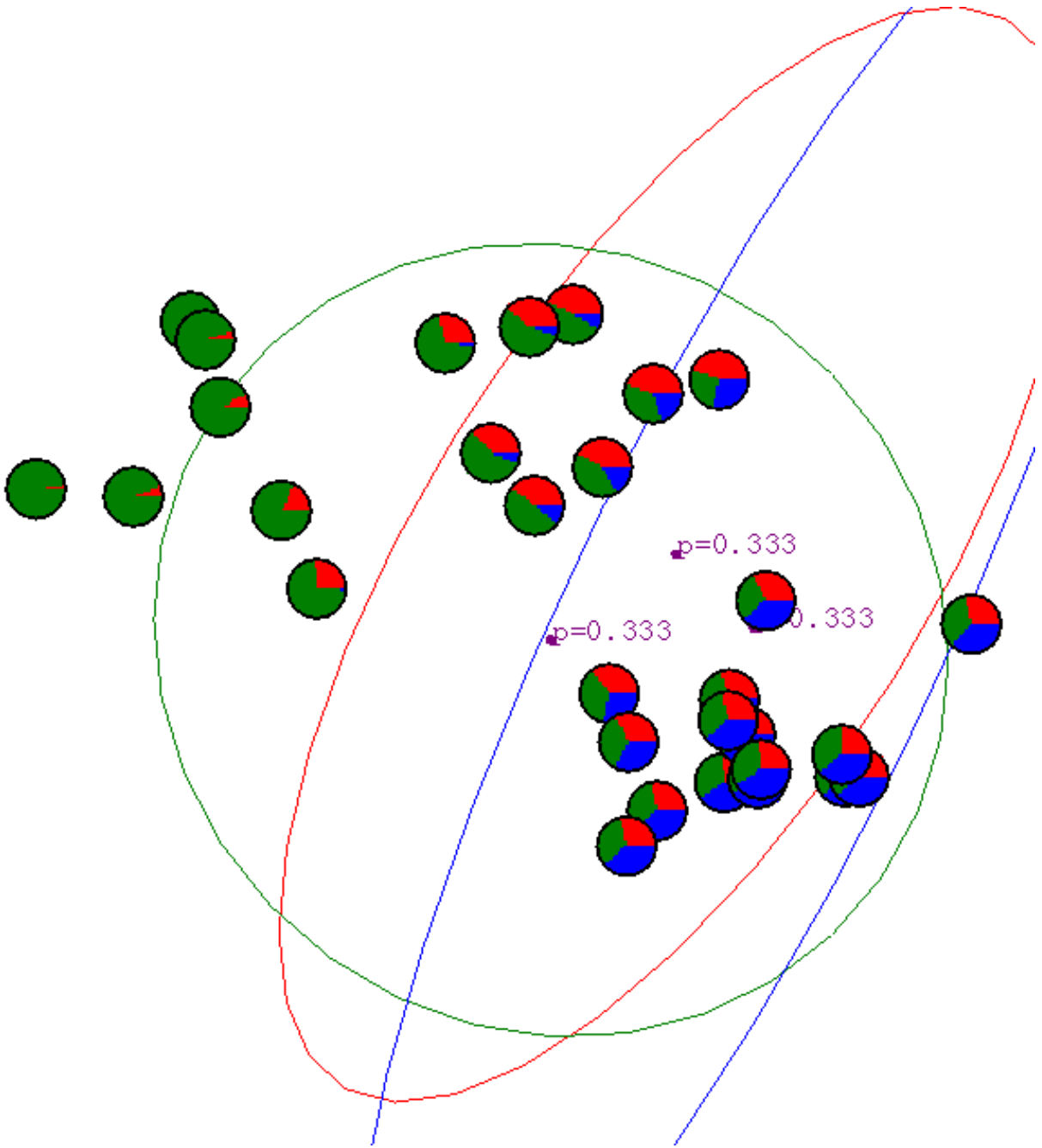
Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

Animation: EM algorithm for a GMM

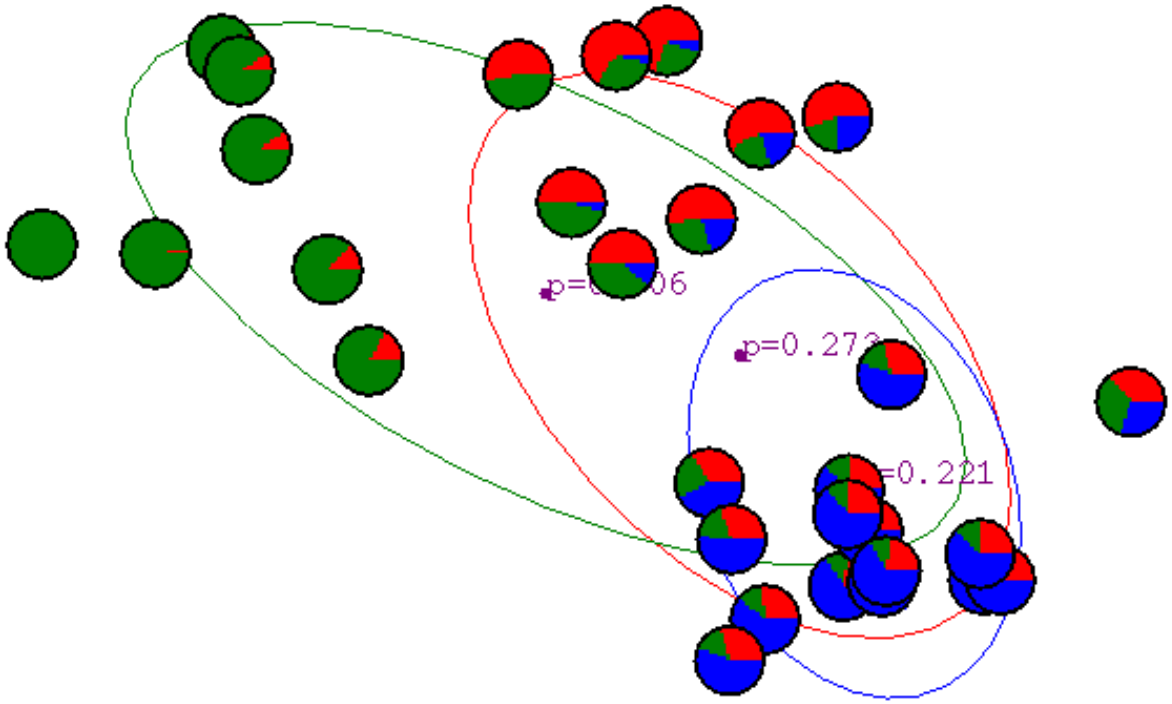
- **Running the EM algorithm to derive the Gaussian Mixture Model (GMM) from a set of training data**

Gaussian Mixture Example: Start

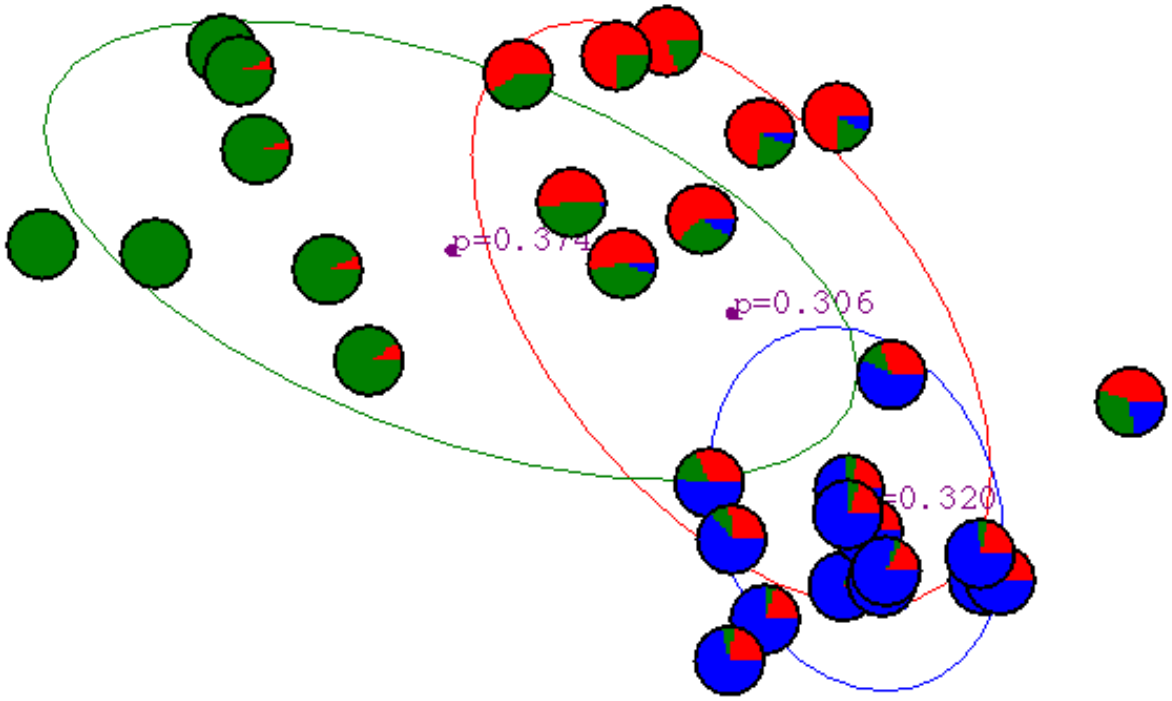


Based on slides from Rong Jin, Andrew Blake, Bill Freeman, Ya Chang and Mathias Kölsch and Andrew W. Moore

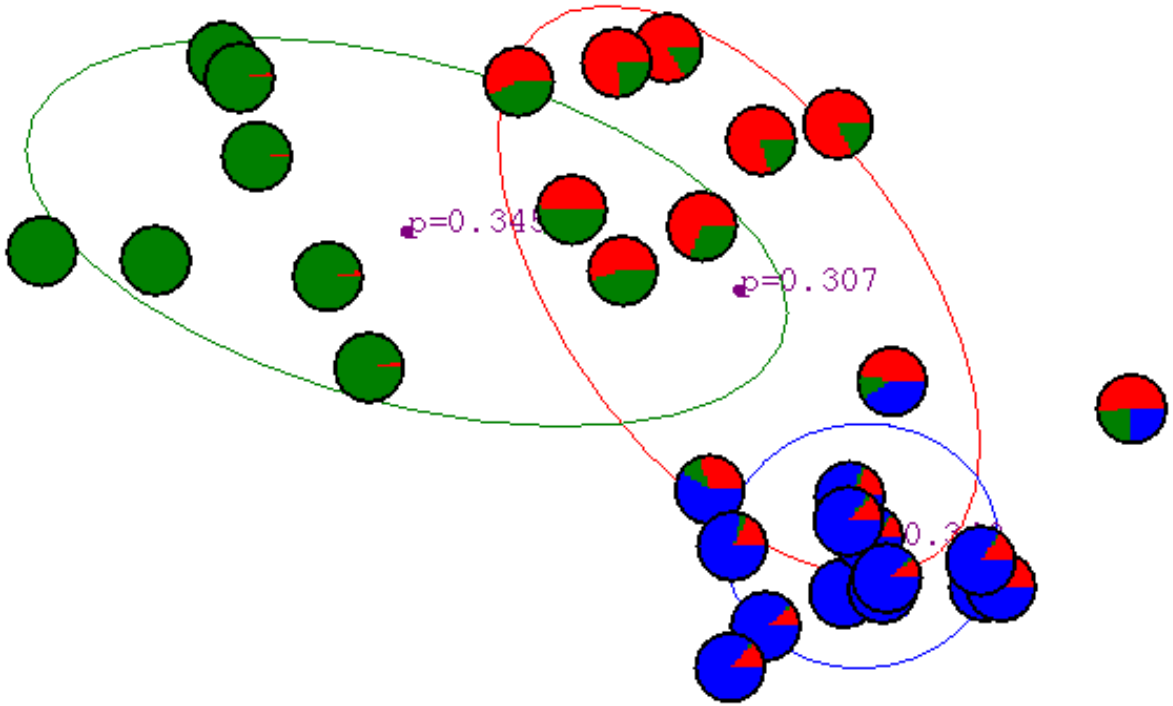
After first iteration



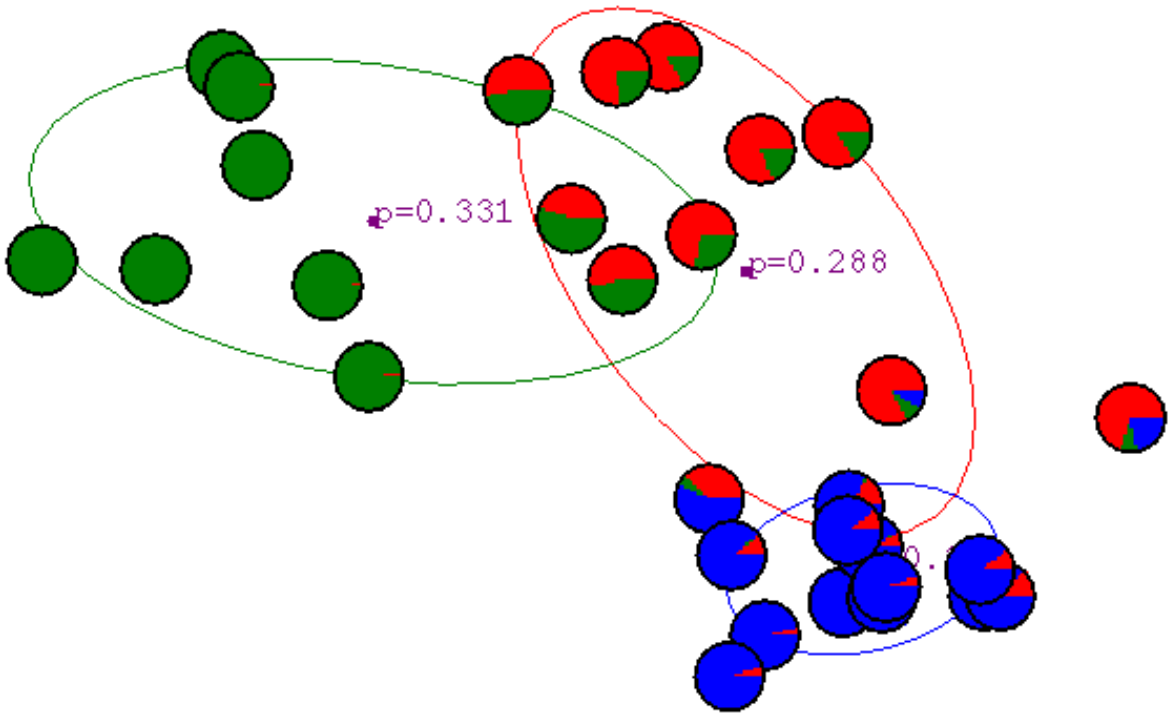
After 2nd iteration



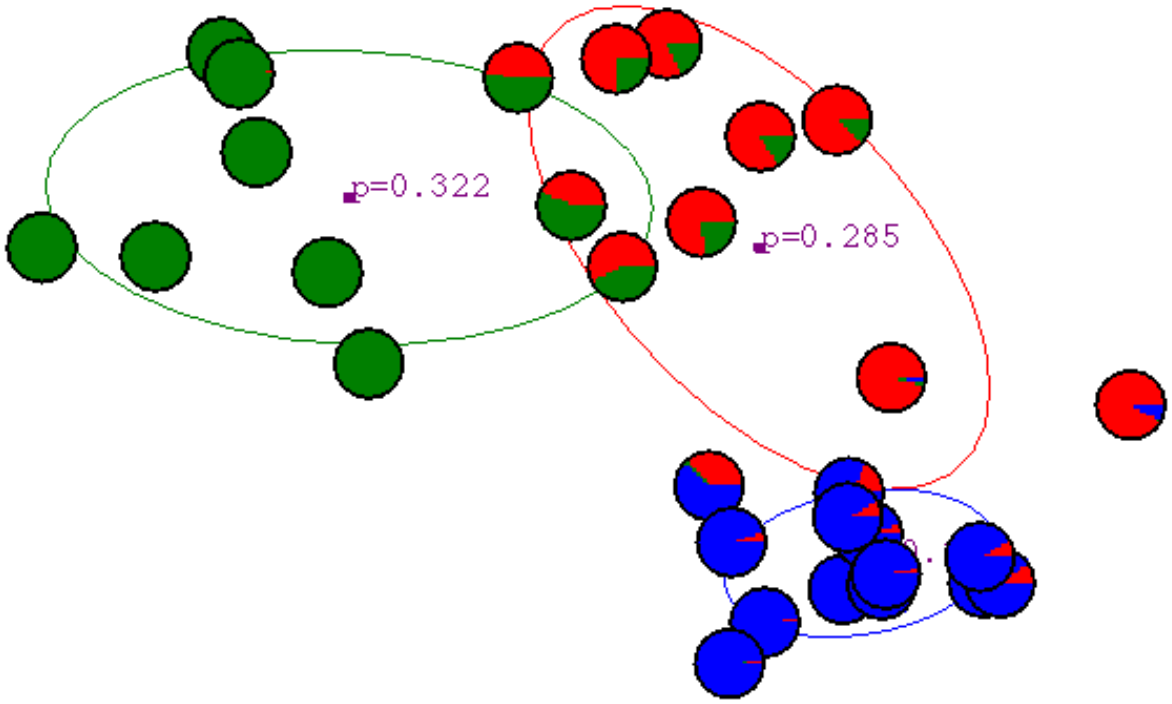
After 3rd iteration



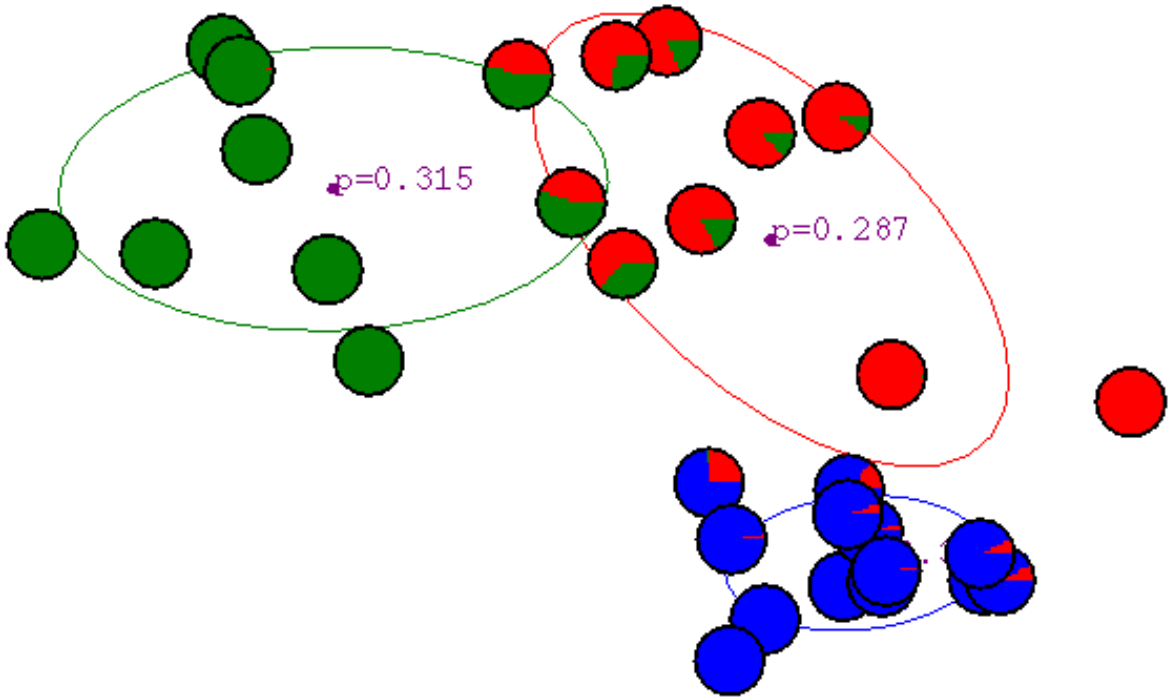
After 4th iteration



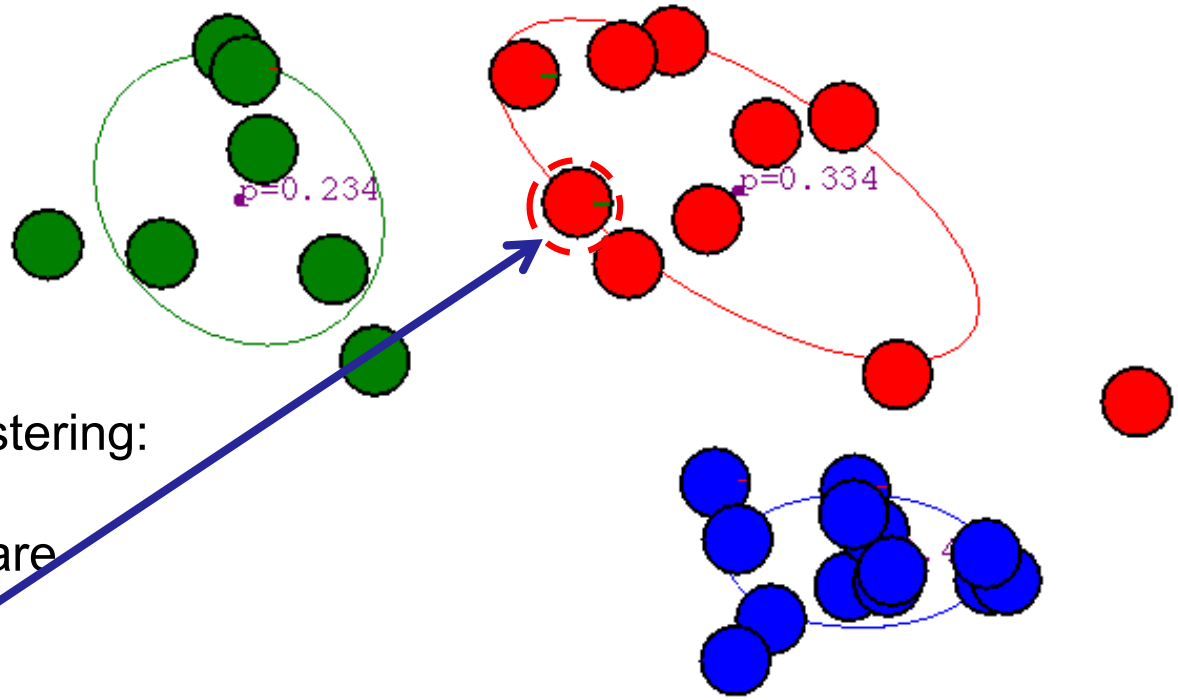
After 5th iteration



After 6th iteration



After 20th iteration



After 20 iterations of clustering: EM seems to yield hard assignments. But there are examples here with soft assignments. E.g.,

Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

EM-GMM in Python

- https://github.com/mcdickenson/em-gaussian
- **See next few slides also**

em-gaussian

Python code for Expectation-Maximization estimate of Gaussian mixture model

Final parameters for example:

	lambda	mu1	mu2	sig1	sig2
0	0.495	4.852624	0.085936	[1.73146140597, 0]	[1.58951132132, 0]
1	0.505	-0.006998	4.992721	[0, 1.11931804165]	[0, 1.91666943891]

Create some artificial data

```
import numpy as np
import pandas as pd
import random as rand
import matplotlib.pyplot as plt
from scipy.stats import norm
from sys import maxint

### Setup
# set random seed
rand.seed(42)

# 2 clusters
# not that both covariance matrices are diagonal
mu1 = [0, 5]
sig1 = [ [2, 0], [0, 3] ]

mu2 = [5, 0]
sig2 = [ [4, 0], [0, 1] ]

# generate samples
x1, y1 = np.random.multivariate_normal(mu1, sig1, 100).T
x2, y2 = np.random.multivariate_normal(mu2, sig2, 100).T

xs = np.concatenate((x1, x2))
ys = np.concatenate((y1, y2))
labels = ([1] * 100) + ([2] * 100)

data = {'x': xs, 'y': ys, 'label': labels}
df = pd.DataFrame(data=data)

# inspect the data
df.head()
df.tail()

fig = plt.figure()
plt.scatter(data['x'], data['y'], 24, c=data['label'])
fig.savefig("true-values.png")
```

Create some artificial data

```python
import numpy as np
import pandas as pd
import random as rand
import matplotlib.pyplot as plt
from scipy.stats import norm
from sys import maxint

### Setup
# set random seed
rand.seed(42)

# 2 clusters
# not that both covariance matrices are diagonal
mu1 = [0, 5]
sig1 = [ [2, 0], [0, 3] ]

mu2 = [5, 0]
sig2 = [ [4, 0], [0, 1] ]

# generate samples
x1, y1 = np.random.multivariate_normal(mu1, sig1, 100).T
x2, y2 = np.random.multivariate_normal(mu2, sig2, 100).T

xs = np.concatenate((x1, x2))
ys = np.concatenate((y1, y2))
labels = ([1] * 100) + ([2] * 100)

data = {'x': xs, 'y': ys, 'label': labels}
df = pd.DataFrame(data=data)

# inspect the data
df.head()
df.tail()

fig = plt.figure()
plt.scatter(data['x'], data['y'], 24, c=data['label'])
fig.savefig("true-values.png")
```

$$p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$

E-Step

Here, $p(x^{(i)} | z^{(i)} = j; \mu, \Sigma)$ is given by evaluating the density of a Gaussian with mean μ_j and covariance Σ_j at $x^{(i)}$; $p(z^{(i)} = j; \phi)$ is given by ϕ_j, and so

```
49  # probability that a point came from a Guassian with given parameters
50  # note that the covariance must be diagonal for this to work
51  def prob(val, mu, sig, lam):
52    p = lam
53    for i in range(len(val)):
54      p *= norm.pdf(val[i], mu[i], sig[i][i])
55    return p
56
57
58  # assign every data point to its most likely cluster
59  def expectation(dataFrame, parameters):
60    for i in range(dataFrame.shape[0]):
61      x = dataFrame['x'][i]
62      y = dataFrame['y'][i]
63      p_cluster1 = prob([x, y], list(parameters['mu1']), list(parameters['sig1']), parameters['lambda'][0] )
64      p_cluster2 = prob([x, y], list(parameters['mu2']), list(parameters['sig2']), parameters['lambda'][1] )
65      if p_cluster1 > p_cluster2:
66        dataFrame['label'][i] = 1
67      else:
68        dataFrame['label'][i] = 2
69    return dataFrame
70
```

M-Step

(M-step) Update the parameters:

$$\phi_j := \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)}, \qquad \text{Prior}_k$$

$$\mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}, \qquad \text{Centroid}_k$$

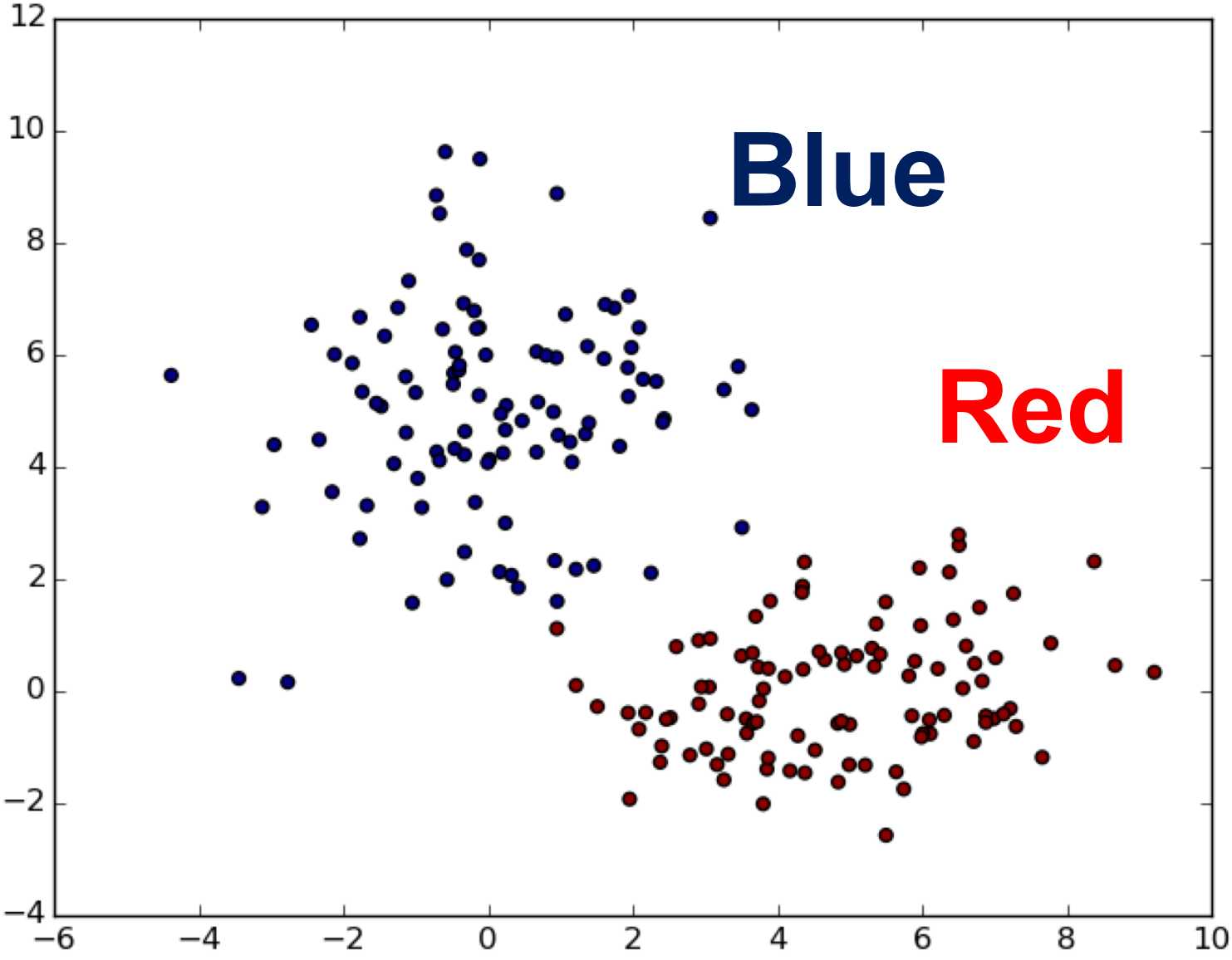
$$\text{Covariance}_k \quad \Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}$$

```
71
72  # update estimates of lambda, mu and sigma
73  def maximization(dataFrame, parameters):
74    points_assigned_to_cluster1 = dataFrame[dataFrame['label'] == 1]
75    points_assigned_to_cluster2 = dataFrame[dataFrame['label'] == 2]
76    percent_assigned_to_cluster1 = len(points_assigned_to_cluster1) / float(len(dataFrame))
77    percent_assigned_to_cluster2 = 1 - percent_assigned_to_cluster1
78    parameters['lambda'] = [percent_assigned_to_cluster1, percent_assigned_to_cluster2 ]
79    parameters['mu1'] = [points_assigned_to_cluster1['x'].mean(), points_assigned_to_cluster1['y'].mean()]
80    parameters['mu2'] = [points_assigned_to_cluster2['x'].mean(), points_assigned_to_cluster2['y'].mean()]
81    parameters['sig1'] = [ [points_assigned_to_cluster1['x'].std(), 0 ], [ 0, points_assigned_to_cluster1['y'].std() ] ]
82    parameters['sig2'] = [ [points_assigned_to_cluster2['x'].std(), 0 ], [ 0, points_assigned_to_cluster2['y'].std() ] ]
83    return parameters
```

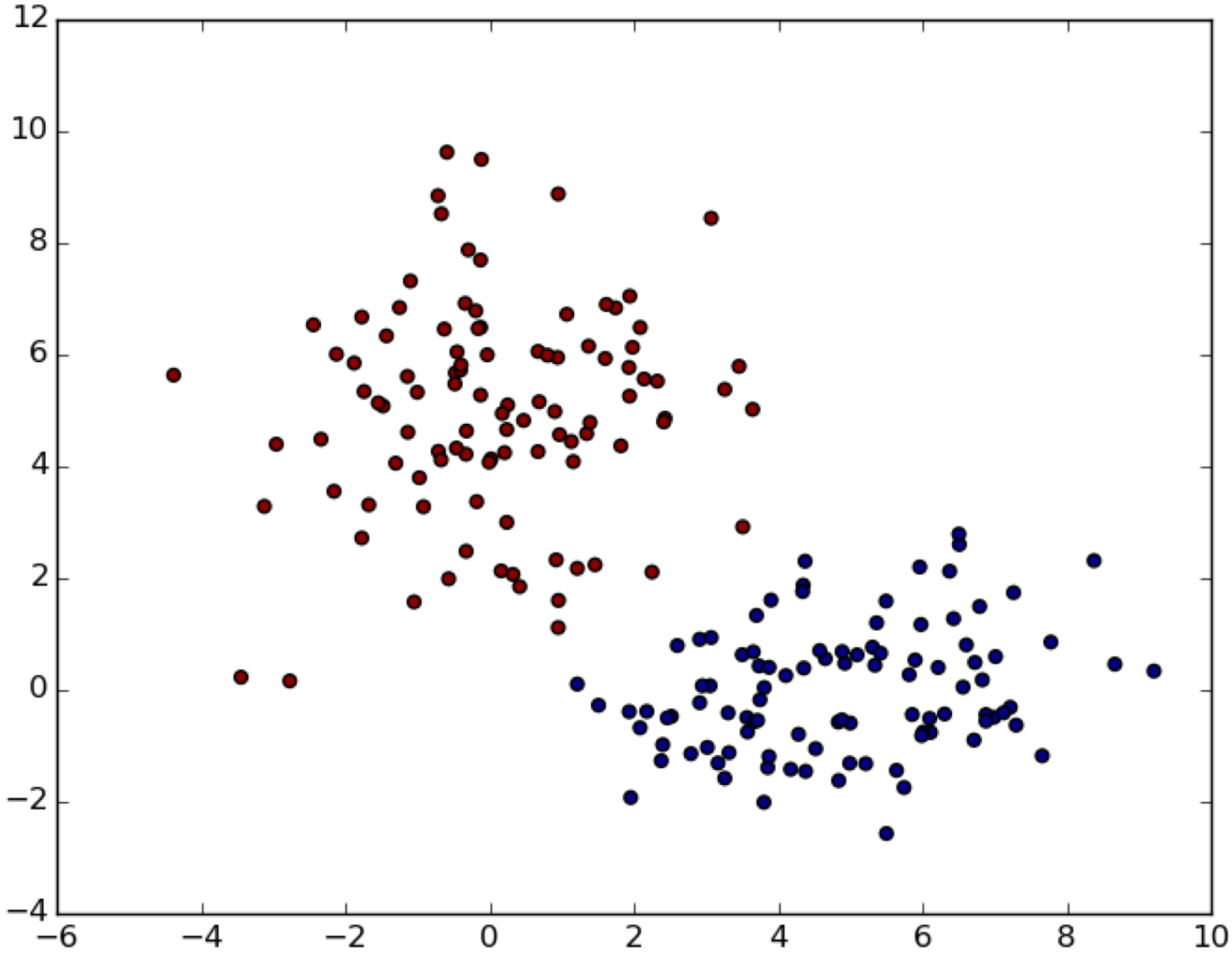
Centroid$_k$

```
85  # get the distance between points
86  # used for determining if params have converged
87  def distance(old_params, new_params):
88    dist = 0
89    for param in ['mu1', 'mu2']:
90      for i in range(len(old_params)):
91        dist += (old_params[param][i] - new_params[param][i]) ** 2
92    return dist ** 0.5
93
94  # loop until parameters converge
95  shift = maxint
96  epsilon = 0.01
97  iters = 0
98  df_copy = df.copy()
99  # randomly assign points to their initial clusters
100 df_copy['label'] = map(lambda x: x+1, np.random.choice(2, len(df)))
101 params = pd.DataFrame(guess)
102
103 while shift > epsilon:
104   iters += 1
105   # E-step
106   updated_labels = expectation(df_copy.copy(), params)
107
108   # M-step
109   updated_parameters = maximization(updated_labels, params.copy())
110
111   # see if our estimates of mu have changed
112   # could incorporate all params, or overall log-likelihood
113   shift = distance(params, updated_parameters)
114
115   # logging
116   print("iteration {}, shift {}".format(iters, shift))
117
118   # update labels and params for the next iteration
119   df_copy = updated_labels
120   params = updated_parameters
121
122   fig = plt.figure()
123   plt.scatter(df_copy['x'], df_copy['y'], 24, c=df_copy['label'])
124   fig.savefig("iteration{}.png".format(iters))
```

True Data

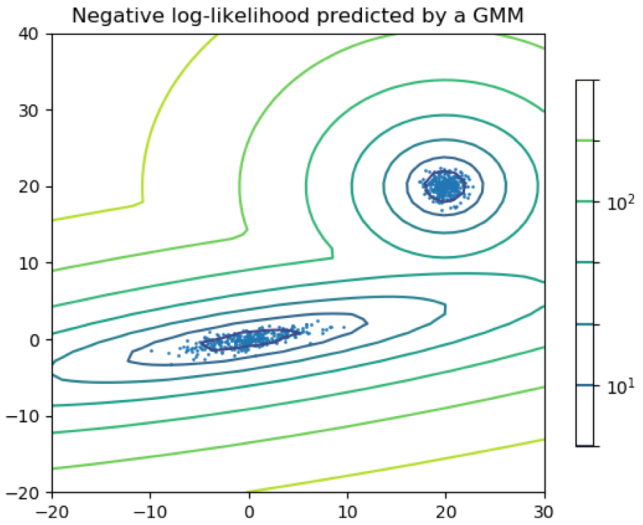


After 6 iterations of EM-GMM



2.1. Gaussian mixture models

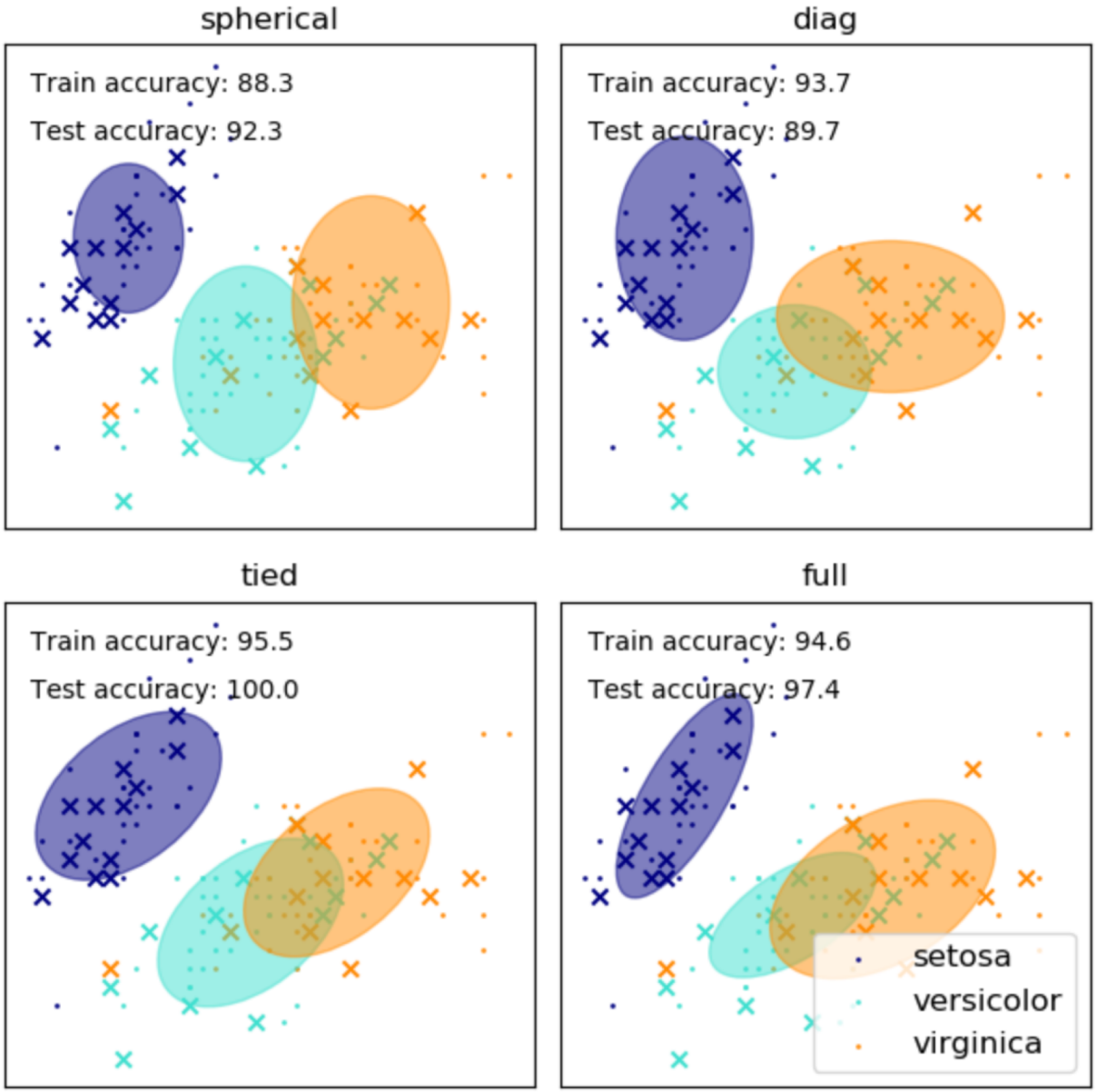
`sklearn.mixture` is a package which enables one to learn Gaussian Mixture Models (diagonal, spherical, tied and full covariance matrices supported), sample them, and estimate them from data. Facilities to help determine the appropriate number of components are also provided.



Two-component Gaussian mixture model: *data points, and equi-probability surfaces of the model.*

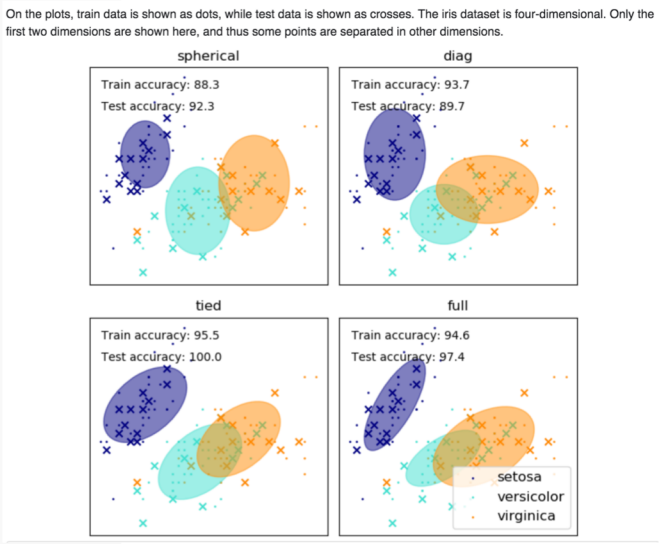
Iris 4D (but 2Dims are shown here)

On the plots, train data is shown as dots, while test data is shown as crosses. The iris dataset is four-dimensional. Only the first two dimensions are shown here, and thus some points are separated in other dimensions.



4 types of GMM via EM

```
X_train = iris.data[train_index]
y_train = iris.target[train_index]
X_test = iris.data[test_index]
y_test = iris.target[test_index]

n_classes = len(np.unique(y_train))

# Try GMMs using different types of covariances.
estimators = dict((cov_type, GaussianMixture(n_components=n_classes,
                   covariance_type=cov_type, max_iter=20, random_state=0))
                  for cov_type in ['spherical', 'diag', 'tied', 'full'])

n_estimators = len(estimators)

plt.figure(figsize=(3 * n_estimators // 2, 6))
plt.subplots_adjust(bottom=.01, top=0.95, hspace=.15, wspace=.05,
                    left=.01, right=.99)


for index, (name, estimator) in enumerate(estimators.items()):
    # Since we have class labels for the training data, we can
    # initialize the GMM parameters in a supervised manner.
    estimator.means_init = np.array([X_train[y_train == i].mean(axis=0)
                                     for i in range(n_classes)])

    # Train the other parameters using the EM algorithm.
    estimator.fit(X_train)

    h = plt.subplot(2, n_estimators // 2, index + 1)
    make_ellipses(estimator, h)

    for n, color in enumerate(colors):
        data = iris.data[iris.target == n]
        plt.scatter(data[:, 0], data[:, 1], s=0.8, color=color,
                    label=iris.target_names[n])
    # Plot the test data with crosses
    for n, color in enumerate(colors):
        data = X_test[y_test == n]
        plt.scatter(data[:, 0], data[:, 1], marker='x', color=color)

    y_train_pred = estimator.predict(X_train)
    train_accuracy = np.mean(y_train_pred.ravel() == y_train.ravel()) * 100
    plt.text(0.05, 0.9, 'Train accuracy: %.1f' % train_accuracy,
             transform=h.transAxes)

    y_test_pred = estimator.predict(X_test)
    test_accuracy = np.mean(y_test_pred.ravel() == y_test.ravel()) * 100
    plt.text(0.05, 0.8, 'Test accuracy: %.1f' % test_accuracy,
             transform=h.transAxes)
```



Outline

- **Introduction**
- **Latent Variables, MLE, MAP and Bayes Optimal Classifier**
- **Flat Clustering (as opposed to hierarchical)**
  - Soft versus Hard
- **Maximum Likelihood Estimation (MLE)**
  - Bernoulli distributions
  - Gaussian Distributions
- **MLE for Gaussian Mixture Models**
- **EM for Gaussian Mixture Models**
  - EM Algorithm for GMMs
  - Animation: EM algorithm for a GMM
  - Notebook for EM Algorithm for GMMs
- **Summary**

End of lecture