

# Expectation Maximization

---

November 10, 2020

# Table of contents

1. The k-means algorithm
2. Mixture Models
3. Expectation Maximization (more generally)
4. Some Cybersecurity Applications of EM
  - 4.1 Better malware ground truth
  - 4.2 Behavioral biometrics

# Acknowledgements

This slide deck borrows heavily from an excellent course on statistical ML by Peter Orbanz.

Another important resource was Christopher Bishop's Machine Learning textbook.

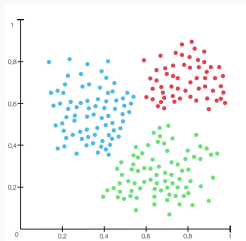
# The k-means algorithm

---

# Clustering

## Problem

- Given: Data  $x_1, \dots, x_n$ .
- Assumption: Each data point belongs to exactly one group or class. These groups are called **clusters**.
- Our task is to find the clusters, given only the data.



## Representation

For  $K$  clusters, we encode assignments to clusters as a vector  $\mathbf{z} \in \{1, \dots, K\}^n$  where:

$$z_i = k \iff x_i \text{ assigned to cluster } k$$

# A very simple clustering algorithm: K-means

## K-means algorithm

- Randomly choose K “cluster centers” (the “means”)  $\mu_1^{(0)}, \dots, \mu_K^{(0)} \in \mathbb{R}^d$
- Iterate until convergence ( $j$  = iteration number):
  1. Assign each  $x_i$  to the closest (in Euclidean distance) mean:

$$z_i^{(j+1)} := \arg \min_{k \in \{1, \dots, K\}} \|x_i - \mu_k^{(j)}\|$$

2. Recompute each  $\mu_k^{(j)}$  as the mean of all points assigned to it.

$$\mu_k^{(j+1)} := \frac{1}{|i : z_i^{(j+1)} = k|} \sum_{i: z_i^{(j+1)} = k} x_i$$

## Convergence Criterion

For example: Terminate when the total change of the means satisfies:

$$\sum_{k=1}^K \|\mu_k^{(j+1)} - \mu_k^{(j)}\| < \tau$$

The threshold value  $\tau$  is set by the user.

# Illustration

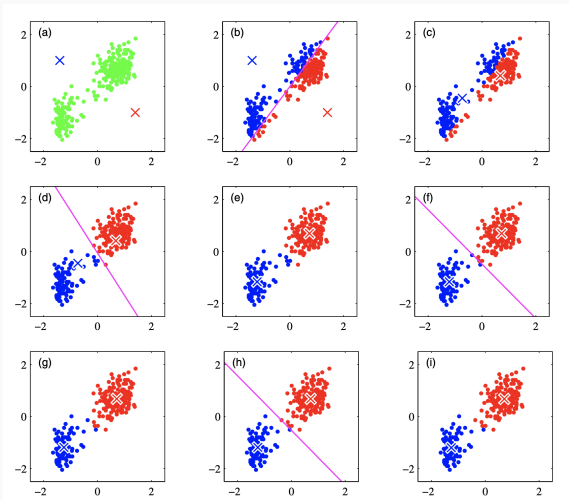


Illustration of the  $K$ -means algorithm using the re-scaled Old Faithful dataset (in green). The initial choices for centers  $\mu_1$  and  $\mu_2$  are shown by the red and blue crosses, respectively.

# Application: Image Segmentation and Clustering

## Image Segmentation

**Image segmentation** is the problem of partitioning an image into “coherent” regions. The problem is not well-posed: Its solution depends on the meaning of “coherent”.

## $K$ -means on images

- Each pixel is treated as a separate point representing  $\{R, G, B\}$  intensities.
- Note: not sophisticated; spatial proximity ignored.

$K = 2$        $K = 3$        $K = 10$       original

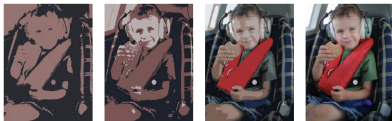


Image Credit: Christopher Bishop

## Utility for image compression

(Lossy) *image compression* is a side effect; one can store only the cluster identity  $k$  and the *code-book vectors*  $\mu_k$ .





K-means demo in python

# K-Means: Gaussian Interpretation

## K Gaussians

Consider the following algorithm:

- Suppose each  $\mu_k$  is the expected value of a Gaussian density  $p(x|\mu_k, \mathbb{I})$  with unit covariance.
- Start with  $K$  randomly chosen means and iterate.
  1. Assign each  $x_i$  to the Gaussian under which it has the highest density.
  2. Given the assignments, fit  $p(x|\mu_k, \mathbb{I})$  by maximum likelihood estimation of  $\mu_k$  from all points assigned to cluster  $k$ .

## Comparison to K-means

- Since the Gaussians are spherical with identity covariance, the density  $p(x|\mu_k, \mathbb{I})$  is largest for the mean  $\mu_j$  which is closest to  $x_i$  in Euclidean distance. (Why?)
- The maximum likelihood estimator of  $\mu_k$  is

$$\mu_k^{(j+1)} := \frac{1}{|i : z_i^{(j+1)} = k|} \sum_{i: z_i^{(j+1)} = k} x_i$$

This is precisely the K-means algorithm!

# What next

- We will discuss a more sophisticated version of  $K$ -means called the *Expectation-Maximization (EM) algorithm*.
- EM gives
  1. A better statistical explanation of what is going on.
  2. A direct generalization to other distributions. We can consider Gaussians with general covariance structure, or other distributions as well.
  3. Better support for the anomaly detection use case.

# Mixture Models

---

# Finite mixture models

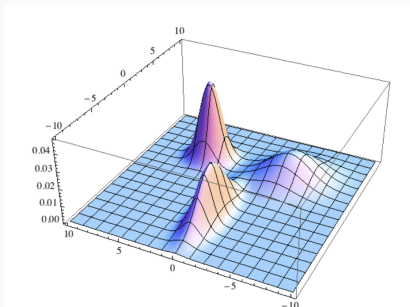
## Finite Mixture Model

A finite mixture model is a distribution with density of the form

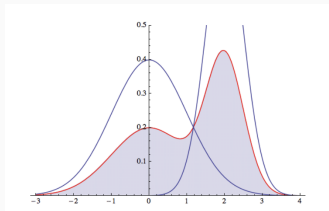
$$\pi(x) = \sum_{k=1}^K c_k p(x | \theta_k)$$

where  $\sum_k c_k = 1$  and  $c_k \geq 0$ .

## Example: Finite mixture of Gaussians



## Mixture of two Gaussians

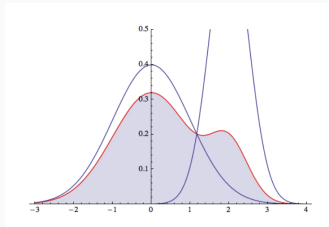


The curve outlined in red is the mixture

$$\pi(x) = 0.5 p(x | 0, 1) + 0.5 p(x | 2, 0.5)$$

where  $p$  is the Gaussian density. The blue curves are the component densities.

## Influence of the weights



Here, the weights  $c_1 = c_2 = 0.5$  above have been changed to  $c_1 = 0.8$  and  $c_2 = 0.2$ . The component distributions are the same as above.

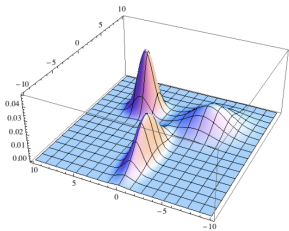
## Sampling from a finite mixture

For a finite mixture with fixed parameters  $c_k$  and  $\theta_k$ , the two-step sampling procedure is:

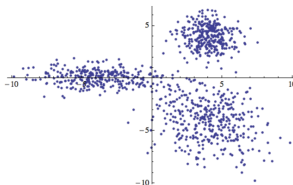
1. Choose a mixture component at random. Each component  $k$  is selected with probability  $c_k$ .
2. Sample  $x_i$  from  $p(x|\theta_k)$ .

**Note:** We always repeat both steps, i.e. for  $x_{i+1}$ , we choose again choose a (possibly different) component at random.

Note that k-means does not support sampling new points.



Plot of the mixture density.



A sample of size 1000.



## Maximum likelihood for finite mixtures

Writing down the maximum likelihood problem is straightforward:

$$(\hat{\mathbf{c}}, \hat{\boldsymbol{\theta}}) = (\hat{c}_1, \dots, \hat{c}_K, \hat{\theta}_1, \dots, \hat{\theta}_K) = \arg \max_{\mathbf{c}, \boldsymbol{\theta}} \prod_{i=1}^n \left( \sum_{k=1}^K c_k p(x_i | \theta_k) \right)$$

The maximality equation for the logarithmic likelihood is

$$\frac{\delta}{\delta(\mathbf{c}, \boldsymbol{\theta})} \sum_{i=1}^n \log \left( \sum_{k=1}^K c_k p(x_i | \theta_k) \right) = 0$$

The component equation for each  $\theta_k$  is:

$$\sum_{i=1}^n \frac{c_k \frac{\delta}{\delta \theta_k} p(x_i | \theta_k)}{\sum_{k=1}^K c_k p(x_i | \theta_k)} = 0$$

Solving this problem is analytically infeasible (note that we cannot multiply out the denominator, because of the sum over  $i$ ). Even numerical solution is often difficult.

## Cluster assignments

- The mixture assumption implies that each  $x_i$  was generated from one component.
- For each  $x_i$ , we again use an **assignment variable**  $z_i \in \{1, \dots, K\}$  which encodes which cluster  $x_i$  was sampled from.



## Latent Variables

Since we do not know which component each  $x_i$  was generated by, the values of the assignment variables are *unobserved*. Such variables whose values are not observed are called **latent variables** or **hidden variables**.

# Estimation with latent variables

## Latent variables as auxiliary information

If we knew the correct assignments  $z_i$ , we could:

- Estimate each component distribution  $p(x|\theta_k)$  separately, using only the data assigned to cluster  $k$ .
- Estimate the cluster proportions  $c_k$  as  $\hat{c}_k = \frac{\# \text{ points in cluster } k}{n}$

## EM algorithm: Idea

The EM algorithm estimates values of the latent variables to simplify the estimation problem. EM alternates between two steps:

1. Estimate assignments  $z_i$  given current estimates of the parameters  $c_k$  and  $\theta_k$  ("E-step").
2. Estimate parameters  $c_i$  and  $\theta_k$  given current estimates of the assignments ("M-step").

These two steps are iterated repeatedly.

# Representation of Assignments

We re-write the assignments as vectors of length  $K$ :

$$x_i \text{ in cluster } k \quad \text{as} \quad Z_i := \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \leftarrow k\text{th entry}$$

so  $Z_{ik} = 1$  if  $x_i$  in cluster  $k$ , and  $Z_{ik} = 0$  otherwise.

We collect the vectors into a matrix

$$\mathbf{Z} := \begin{bmatrix} Z_{11} & \dots & Z_{1K} \\ \vdots & & \vdots \\ Z_{n1} & \dots & Z_{nK} \end{bmatrix}$$

Note: Rows = observations, columns = clusters

Row sums = 1, column sums = cluster sizes.

## Hard vs. soft assignments

- The vectors  $Z_i$  are “hard assignments” with values in  $\{0, 1\}$  (as in  $k$ -means)
- EM computes “soft assignments”  $r_{ik}$  with values in  $[0, 1]$ .<sup>1</sup>
- The vectors  $Z_i$  are the latent variables in the EM algorithm. The  $r_{ik}$  are their current estimates

## Assignment probabilities

The soft assignments are computed as

$$r_{ik} = \frac{c_k p(x_i | \theta_k)}{\sum_{l=1}^K c_l p(x_i | \theta_l)}$$

They can be interpreted as

$$r_{ik} := \mathbb{E}[Z_{ik} | x_i, \mathbf{c}, \boldsymbol{\theta}] = \Pr\{x_i \text{ generated by component } k | \mathbf{c}, \boldsymbol{\theta}\}$$

---

<sup>1</sup> Once the algorithm terminates, each point could still be assigned to a cluster by setting

$$z_i = \arg \max_k r_{ik}$$

# M-Step (1)

## Objective

The M-step re-estimates  $\mathbf{c}$  and  $\boldsymbol{\theta}$ . In principle, we use maximum likelihood within each cluster, but we have to combine it with the use of weights  $r_{ik}$  instead of  $Z_{ik}$

## Cluster sizes

If we knew which points belong to which cluster, we could estimate the cluster proportions  $c_k$  by counting points:

$$\hat{c}_k = \frac{\# \text{ points in cluster } k}{n} = \frac{\sum_{i=1}^n Z_{ik}}{n}$$

Since we do not know  $Z_{ik}$ , we substitute our current best guess, which is the expectations  $r_{ik}$

$$\hat{c}_k = \frac{\sum_{i=1}^n r_{ik}}{n}$$

## M-Step (2)

### Gaussian special case

The estimation of the component parameters  $\theta$  depends on which distribution we choose for  $p$ . For now, we assume a Gaussian.

### Component parameters

We use maximum likelihood to estimate  $\theta = (\mu, \Sigma)$ . We can write the MLE of  $\mu_k$  as

$$\hat{\mu}_k = \frac{1}{\# \text{ points in cluster } k} \sum_{i: x_i \text{ in } k} x_i = \frac{\sum_{i=1}^n z_{ik} x_i}{\sum_{i=1}^n z_{ik}}$$

By substituting current best guesses ( $= r_{ik}$ ) again, we get

$$\hat{\mu}_k = \frac{\sum_{i=1}^n r_{ik} x_i}{\sum_{i=1}^n r_{ik}}$$

For the covariance matrices:

$$\hat{\Sigma}_k = \frac{\sum_{i=1}^n r_{ik} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T}{\sum_{i=1}^n r_{ik}}$$

# Notation Summary

## Assignment probabilities

$$\mathbf{r} := \begin{bmatrix} r_{11} & \dots & r_{1K} \\ \vdots & & \vdots \\ r_{n1} & \dots & r_{nK} \end{bmatrix} = \mathbb{E} \left[ \begin{bmatrix} Z_{11} & \dots & Z_{1K} \\ \vdots & & \vdots \\ Z_{n1} & \dots & Z_{nK} \end{bmatrix} \right] = \begin{bmatrix} \mathbb{E}[Z_{11}] & \dots & \mathbb{E}[Z_{1K}] \\ \vdots & & \vdots \\ \mathbb{E}[Z_{n1}] & \dots & \mathbb{E}[Z_{nK}] \end{bmatrix}$$

## Mixture parameters

$\tau = (\mathbf{c}, \boldsymbol{\theta})$ ,  $\mathbf{c}$  = cluster proportions     $\boldsymbol{\theta}$  = component parameters

## Iterations

$\boldsymbol{\theta}^{(j)}, \mathbf{r}^{(j)}, \dots$  = values in  $j$ th iteration



# Summary: EM for Gaussian Mixture

## Gaussian special case

$\theta = (\mu, \Sigma)$  (mean & covariance)  $p(x | \theta) = p(x | \mu, \Sigma)$  (Gaussian density)

## Algorithm

The EM algorithm for a finite mixture of Gaussians looks like this

1. **Initialize:** Choose (e.g., random) values  $c_k^{(0)}$  and  $\theta_k^{(0)}$ .
2. **E-Step:** Recompute the assignment weight matrix as

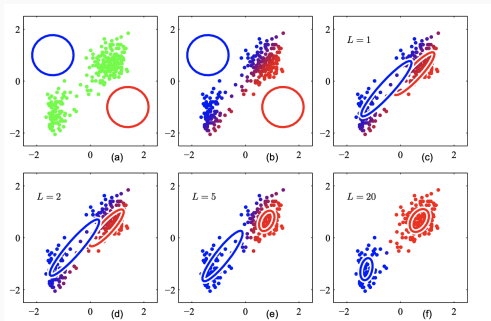
$$r_{ik}^{(j+1)} = \frac{c_k^{(j)} p(x_i | \theta_k^{(j)})}{\sum_{l=1}^K c_l^{(j)} p(x_i | \theta_l^{(j)})}$$

3. **M-Step:** Recompute the proportions  $c_k$  and parameters  $\theta = (\mu, \Sigma)$  as

$$\mu_k^{(j+1)} = \frac{\sum_{i=1}^n r_{ik}^{(j+1)} x_i}{\sum_{i=1}^n r_{ik}^{(j+1)}} \quad \text{and} \quad \Sigma_k^{(j+1)} = \frac{\sum_{i=1}^n r_{ik}^{(j+1)} (x_i - \mu_k^{(j+1)})(x_i - \mu_k^{(j+1)})^T}{\sum_{i=1}^n r_{ik}^{(j+1)}}$$

The E-Step and M-Step are repeated alternately until convergence criterion (e.g. threshold) is satisfied.

## EM for a mixture of two Gaussians



The algorithm fits both the mean and the covariance parameter.

# Two group activities

1. Implementational
2. API Usage / Conceptual

# GMMs as Universal Approximators

*A Gaussian mixture model is a universal approximator of densities, in the sense that any smooth density can be approximated with any specific nonzero amount of error by a Gaussian mixture model with enough components.*

Ian Goodfellow et al. 2016

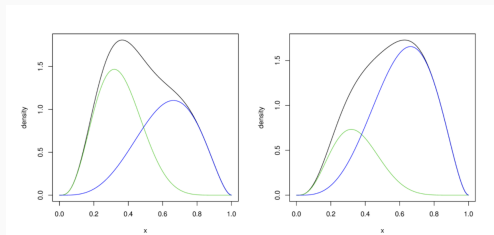
More formally, if  $\mathcal{P}(\mathbb{R}^d)$  is the set of probability Borel measures on  $\mathbb{R}^d$  (with its Euclidean topology), then “Gaussian mixtures” (a.k.a. convex combinations of Gaussian measures) are dense in  $\mathcal{P}(\mathbb{R}^d)$  for the weak\* topology.

**Q:** So why not use GMM's for everything?

# Other Mixture Models

The mixture components do not need to be Gaussian in order for the model to be well-posed, and for EM to estimate the parameters.

## Example: mixture of two betas

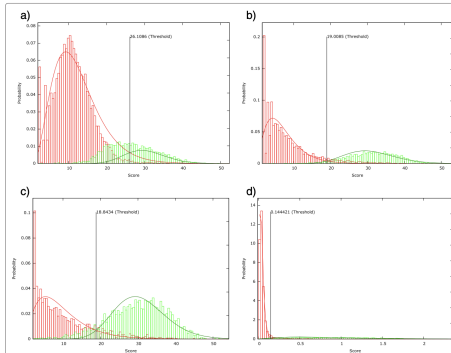


Shown are two beta mixture models, each with two components. The models have fixed components but differ in their mixture weights:  $\mathbf{c} = (.25, .75)$  vs.  $\mathbf{c} = (.75, .25)$ .

**Q:** Under what conditions might one use a mixture of betas rather than a mixture of Gaussians?

# Other Mixture Models

## Example: mixture of two Gammas



Shown are four gamma mixture models.

Landesfeind, M., & Meinicke, P. (2014). Predicting the functional repertoire of an organism from unassembled RNA-seq data. *BMC genomics*, 15(1), 1003.

## Algorithm

- **E-step:** Recompute the assignment matrix  $r_{ik}^{(j)}$  as

$$r_{ik}^{(j+1)} = \frac{c_k^{(j)} p(x_i | \theta_k^{(j)})}{\sum_{l=1}^K c_l^{(j)} p(x_i | \theta_l^{(j)})}$$

- **M-step:** Recompute  $(c, \theta)$  as

$$(c^{(j+1)}, \theta^{(j+1)}) = \arg \max_{c, \theta} \left[ \sum_{ik} r_{ik}^{(j+1)} \log(c_k p(x_i | \theta_k)) \right]$$

## Convenient special case

If the MLE of  $p(x | \theta)$  is of the form  $\hat{\theta}_{\text{ML}} = \frac{1}{n} \sum_{i=1}^n s(x_i)$  for some function  $s$  the M-step computes the “weighted maximum likelihood estimate”:

$$c_k^{(j+1)} = \frac{\sum_{i=1}^n r_{ik}^{(j+1)}}{n} \quad \text{and} \quad \theta_k^{(j+1)} = \frac{\sum_{i=1}^n r_{ik}^{(j+1)} s(x_i)}{\sum_{i=1}^n r_{ik}^{(j+1)}}$$

This is true for any distribution in the exponential family.<sup>2</sup>

<sup>2</sup> Note that the ML problem, and therefore the M-step, need not have closed form solution; e.g. click [here](#) for the Gamma distribution.

## Expectation Maximization (more generally)

---



# Latent variable models

- A *parametric statistical model* may posit observed random variables ( $\mathbf{x}$ ), parameters ( $\theta$ ), and latent random variables ( $\mathbf{z}$ ).
- The distinguishing feature between latent variables  $\mathbf{z}$  and parameters  $\theta$  is that the dimensionality of  $\mathbf{z}$  increases with the size of the data set, whereas the dimensionality of  $\theta$  does not.<sup>3</sup>
- Note that some presentations refer to  $\mathbf{z}$  as local hidden variables and  $\theta$  as global hidden variables.

---

<sup>3</sup>A more technical definition can be provided via conditional independence. For example, when there is one latent variable per observation, a latent variable satisfies  $p(\mathbf{x}_n, \mathbf{z}_n \mid \mathbf{x}_{-n}, \mathbf{z}_{-n}, \theta) = p(\mathbf{x}_n, \mathbf{z}_n \mid \theta)$ , where the  $-n$  subscript refers to the set of variables besides the  $n$ th. In other words, the  $n$ th observation and  $n$ th latent variable is independent of all other observations and latent variables, given the model parameters.

# Frequentist estimation for latent variable models

- Latent variable models provide a **complete data likelihood**  $p(\mathbf{x}, \mathbf{z} \mid \boldsymbol{\theta})$ , where  $\mathbf{z}$  is unobserved. The model often factorizes as

$$p(\mathbf{x}, \mathbf{z} \mid \boldsymbol{\theta}) = \prod_{i=1}^n p(x_i, z_i \mid \boldsymbol{\theta})$$

- For frequentist latent variable models, the inferential goal is to compute  $\boldsymbol{\theta}_{\text{ML}}$ , the maximum likelihood value of the parameter.
- Since  $\mathbf{z}$  is not observed, so one seeks to find

$$\boldsymbol{\theta}_{\text{ML}} := \operatorname{argmax}_{\boldsymbol{\theta}} p(\mathbf{x} \mid \boldsymbol{\theta}) = \operatorname{argmax}_{\boldsymbol{\theta}} \int p(\mathbf{x}, \mathbf{z} \mid \boldsymbol{\theta}) d\mathbf{z} \quad (3.1)$$

- In particular, one requires access to the *marginal* likelihood

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \int p(\mathbf{x}, \mathbf{z} \mid \boldsymbol{\theta}) d\mathbf{z} \quad (3.2)$$

# Expectation Maximization for Latent Variable Models

The **expectation maximization algorithm** estimates  $\theta$  by attempting to maximize the marginal likelihood.

The expectation maximization algorithm is

$$\theta^{(t+1)} = \operatorname{argmax}_{\theta} \mathbb{E}_{p(z \mid x, \theta^{(t)})} \left[ \ln p(x, z \mid \theta) \right] \quad (3.3)$$

# EM for Exponential Family Latent Variable Models

Let us assume that  $(\mathbf{x}, \mathbf{z}) = ((x_1, z_1), \dots, (x_n, z_n))$  are  $n$  independent samples from the same exponential family, where  $\mathbf{x}$  is observed data and  $\mathbf{z}$  is unobserved data. Moreover, let us assume that the complete data likelihood is in the exponential family

$$p(\mathbf{x}, \mathbf{z} \mid \theta) = \prod_{i=1}^n h(x_i, z_i) \exp \left\{ \eta(\theta)^T \sum_{i=1}^n t(x_i, z_i) - n a(\eta(\theta)) \right\} \quad (3.4)$$

Here we want to find  $\theta$  to optimize

$$f(\theta) = \mathbb{E}_{p(\mathbf{z} \mid \mathbf{x}, \theta^{(t)})} \left[ \ln p(\mathbf{x}, \mathbf{z} \mid \theta) \right]$$

Following the logic of the Exponential Family module, we determine that we should select  $\theta^{(t+1)}$  such that

$$\mu(\theta^{(t+1)}) = \frac{1}{n} \sum_{i=1}^n \mathbb{E}_{p(\mathbf{z} \mid \mathbf{x}, \theta^{(t)})} t(x_i, z_i)$$

where  $\mu := \mathbb{E}[t(x_1, z_1)]$  refers to the mean parametrization of the likelihood.

This is why an EM iteration is often described and implemented as iteratively computing **maximum likelihood with the expected sufficient statistics**.

# Convergence Properties

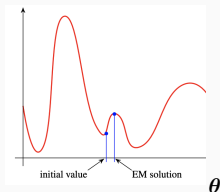
## Marginal likelihood

- It can be shown that the marginal likelihood  $p(x | \theta)$  always increases from each step to the next, unless  $\theta$  is already a stationary point.
- The theory guarantees only that the algorithm terminates at a stationary point. That point can be a saddle point rather than a maximum (very rare)

## The real problem: Local maxima

- EM is effectively a gradient method.
- The maxima it finds are the **local maxima of the log-likelihood**
- There are no guarantees on the global quality of the solution: The global maximum may differ arbitrarily from the one we find.

$\log p(x | \theta)$



Q: So what can we do about this?

## Comparing solutions

- If  $\theta$  and  $\theta'$  are two different EM solutions, we can always compute the log-likelihoods

$$\sum_i \log p(x_i | \theta) \quad \text{and} \quad \sum_i \log p(x_i | \theta')$$

- The solution with the higher likelihood is better.
- This is a very convenient feature of EM: Different solutions are comparable.

## Random restarts

In practice, the best way to use EM is often

- Restart EM repeatedly with randomly (or intelligently) chosen initial values.
- Compute the log-likelihoods of all solutions and compare them.
- Choose the solution achieving maximal log-likelihood

Q: What would be an intelligent way to initialize?

## **Some Cybersecurity Applications of EM**

---

## **Some Cybersecurity Applications of EM**

---

**Better malware ground truth**



# Introduction

- Commercial anti-virus software traditionally memorizes specific byte sequences (known as **signatures**) in the file contents of previously encountered malware.
- They could use these signatures to attempt to detect malware in the future.
- Malware authors can evade signature-based detection in many ways; for instance, by
  - tampering with existing malware signatures (sometimes flipping a single bit)
  - using obfuscation techniques (e.g. encryption or compression) to hide snippets of malicious code
  - writing metamorphic malware
- As a result, classical AV detections have **low false positive** rates *but also low true positive* rates.

Kantchelian et al. (2015) examine the problem of aggregating the results of multiple anti-virus (AV) vendors' detectors into a single authoritative ground-truth label for every binary

Kantchelian, A., Tschantz, M. C., Afroz, S., Miller, B., Shankar, V., Bachwani, R., ... & Tygar, J. D. (2015, October). Better malware ground truth: Techniques for weighting anti-virus vendor labels. In Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security (pp. 45-56).

# Model

## Random variables

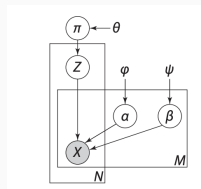
$X_{ij} \in \{0, 1\}$  AV label (i = instance, j = vendor)

$Z_i \in \{0, 1\}$  ground truth label

$\alpha, \beta$  vendor tp, fp rates

$\pi$  malware prevalence

In our terminology,  $\mathbf{Z}$  are latent variables and  $\alpha, \beta$  are parameters.



## Model

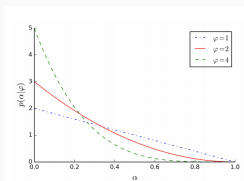
$$X_{ij} \mid Z_i, \alpha_j, \beta_j = \begin{cases} \alpha_j & \text{if } Z_i = 1 \text{ and } X_{ij} = 1 \text{ (TP)} \\ 1 - \alpha_j & \text{if } Z_i = 1 \text{ and } X_{ij} = 0 \text{ (FN)} \\ \beta_j & \text{if } Z_i = 0 \text{ and } X_{ij} = 1 \text{ (FP)} \\ 1 - \beta_j & \text{if } Z_i = 0 \text{ and } X_{ij} = 0 \text{ (TN)} \end{cases}$$

$$Z_i \mid \pi \sim \text{Bernouli}(\pi)$$

$$\pi \sim \text{Beta} \text{ (symmetric)}$$

$$\alpha \mid \psi \sim \text{Beta} \text{ (asymmetric)}$$

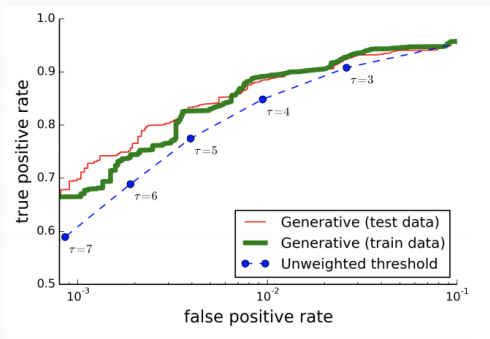
$$\beta \mid \phi \sim \text{Beta} \text{ (asymmetric)}$$



Asymmetric (right skewed)  
priors were used for  $\alpha, \beta$   
since both are expected to be  
low based on prior domain  
knowledge.

# Estimation and Results

- The model was fit using EM.
- The model far outperforms a common baseline in estimating ground truth.



- Moreover, it accomplished this *despite being fully unsupervised!* (No ground truth was available during training.)

# **Some Cybersecurity Applications of EM**

---

**Behavioral biometrics**

# Keystrokes biometrics

Gaussian mixture models (and other density models) have been applied to typing behavior to flag anomalous behavior.

