CS 229 Midterm Review

Course Staff Fall 2018

11/2/2018

Outline

Today:

- SVMs
- Kernels
- Tree Ensembles
- EM Algorithm / Mixture Models

[Focus on building intuition, less so on solving specific problems. Ask questions!]

SVMs

Optimal margin classifier

Two classes separable by linear decision boundary.

But first … what is a hyperplane?

- In d-dimensional space, a (d−1)-dimensional affine subspace
- Examples: line in 2D, plane in 3D

Hyperplane in d-dimensional space:

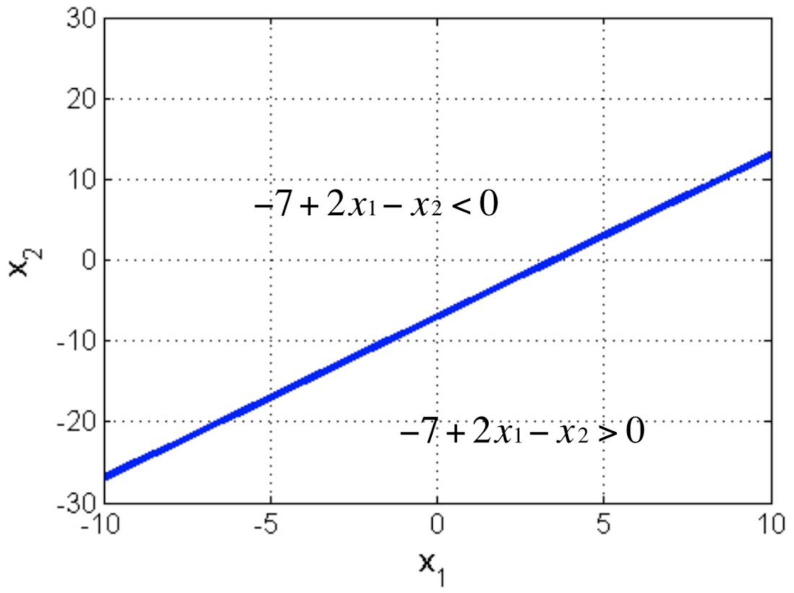
$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d = 0$$

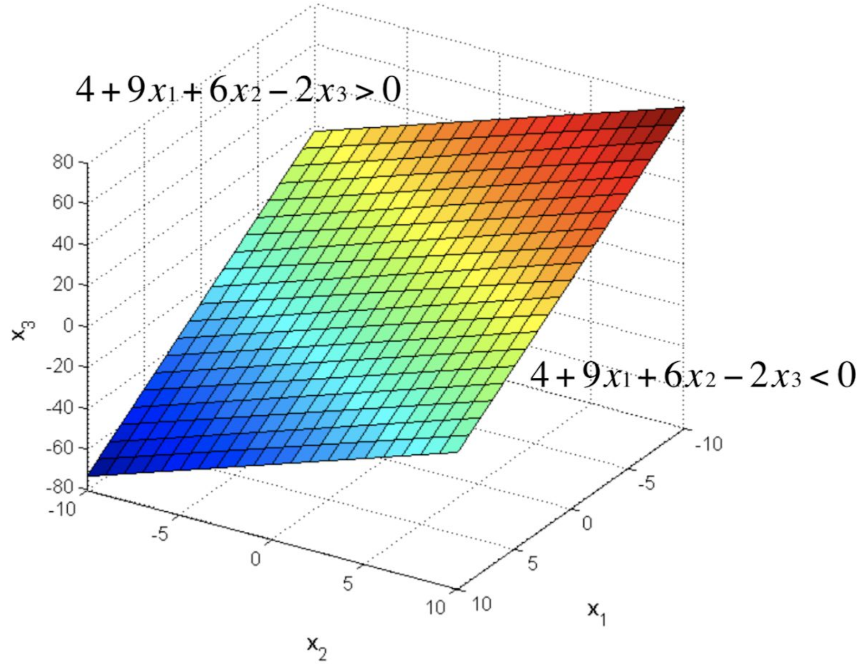
[Separates space into two half-spaces.]

Hyperplanes

$-7 + 2x_1 - x_2 = 0$

$4 + 9x_1 + 6x_2 - 2x_3 = 0$





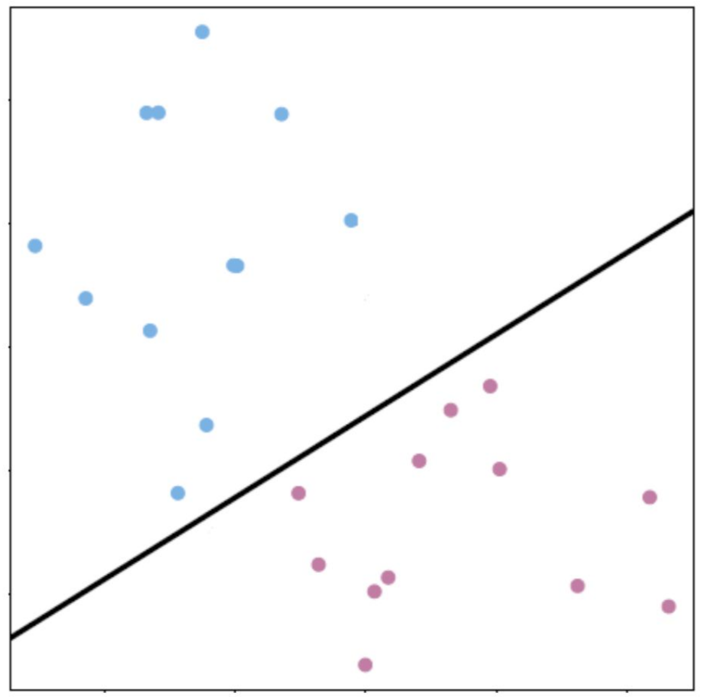
Optimal margin classifier

Idea:

Use a separating hyperplane for binary classification.

Key assumption:

Classes can be separated by a linear decision boundary.

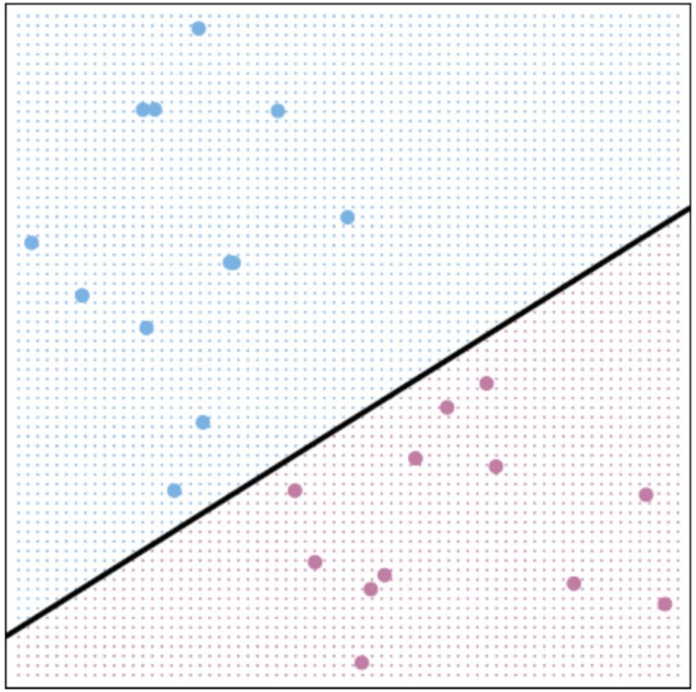


Optimal margin classifier

To classify new data points:

Assign class by location of new data point with respect to hyperplane:

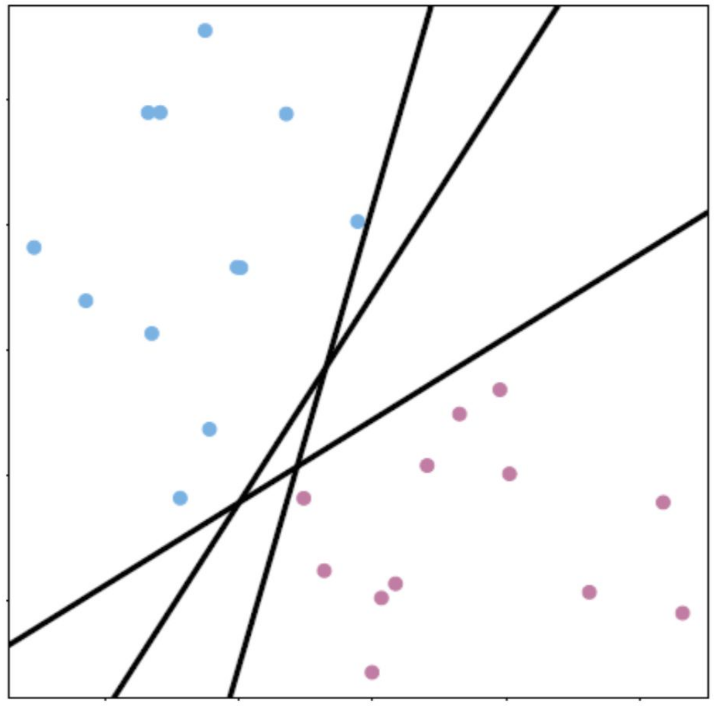
$$\hat{Y} = \text{sign}\left(\beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d\right)$$



Optimal margin classifier

Problem

Many possible separating hyperplanes!

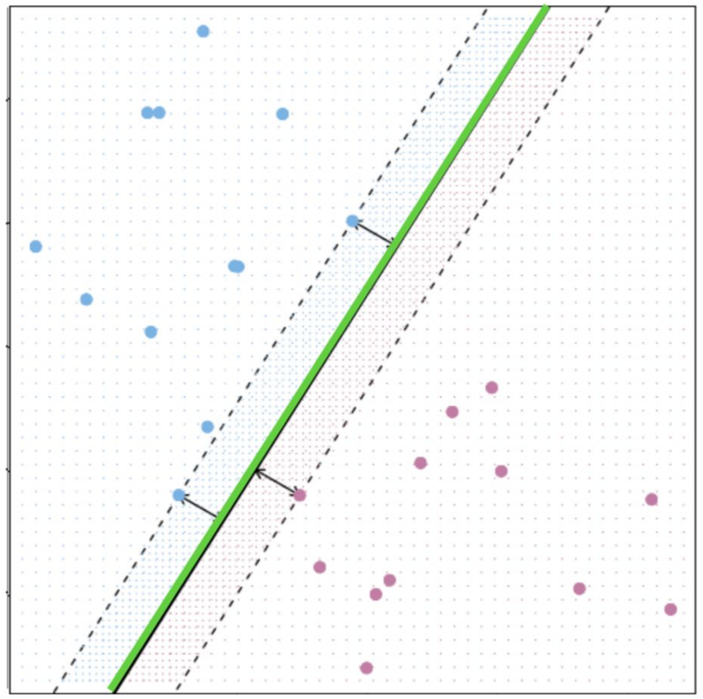


Optimal margin classifier

Which linear decision boundary?

Separating hyperplane "farthest" from the training data.

➔ "Optimal margin"



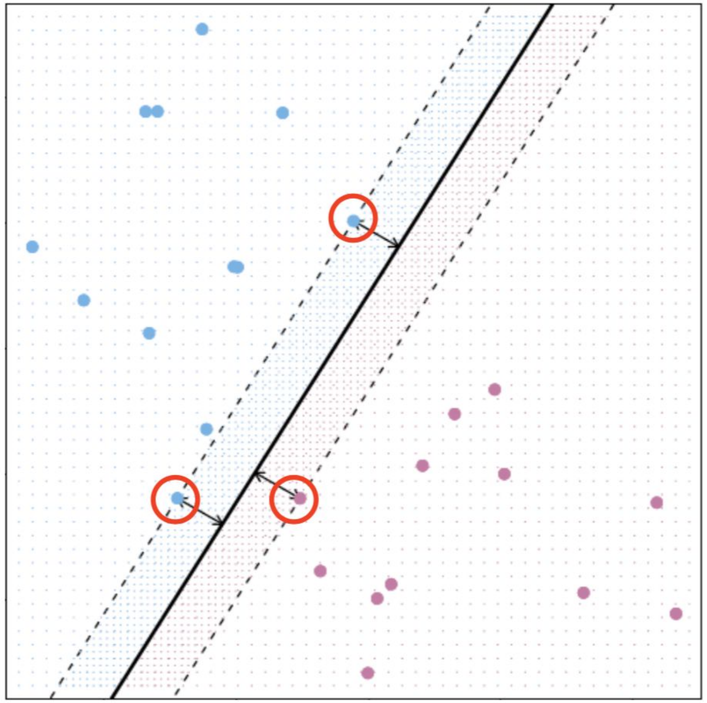
Optimal margin classifier

Which linear decision boundary?

Separating hyperplane "farthest" from the training data

Margin: smallest distance between any training observation and the hyperplane

Support vectors: the training observations equidistant from the hyperplane

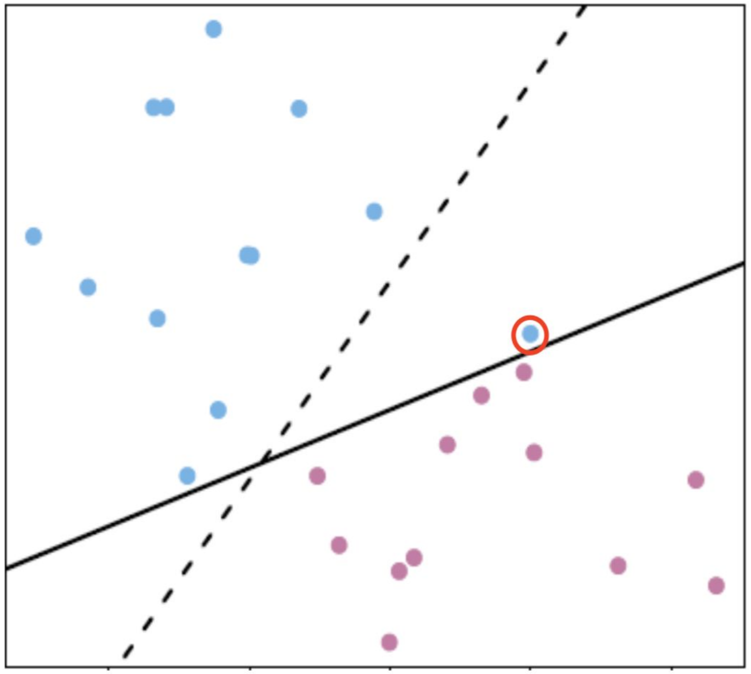


Regularization and the non-separable case

Disadvantage

Can be sensitive to individual observations.

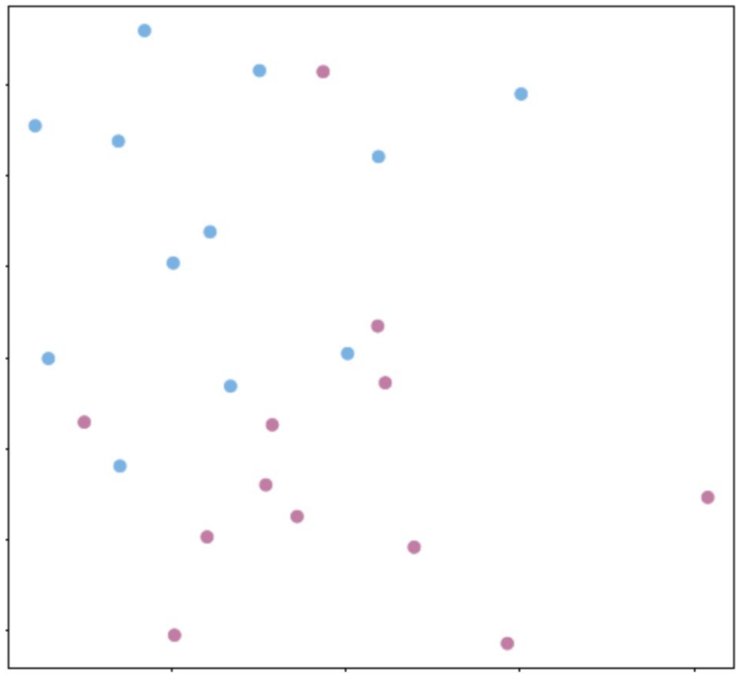
May overfit training data.



Regularization and the non-separable case

So far we've assumed that classes can be separated by a linear decision boundary.

What if there's no separating hyperplane?

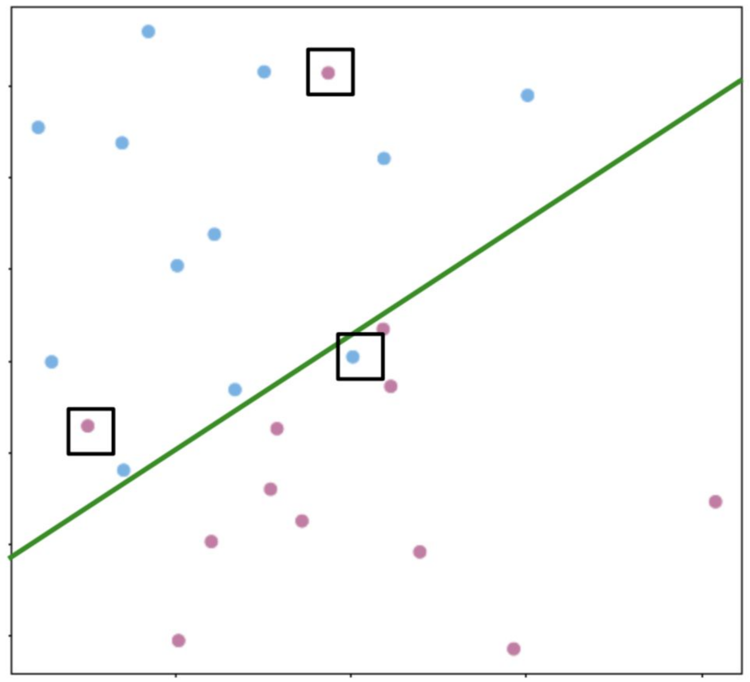


Regularization and the non-separable case

What if there's no separating hyperplane?

Support Vector Classifier:

Allows training samples on the "wrong side" of the margin or hyperplane.



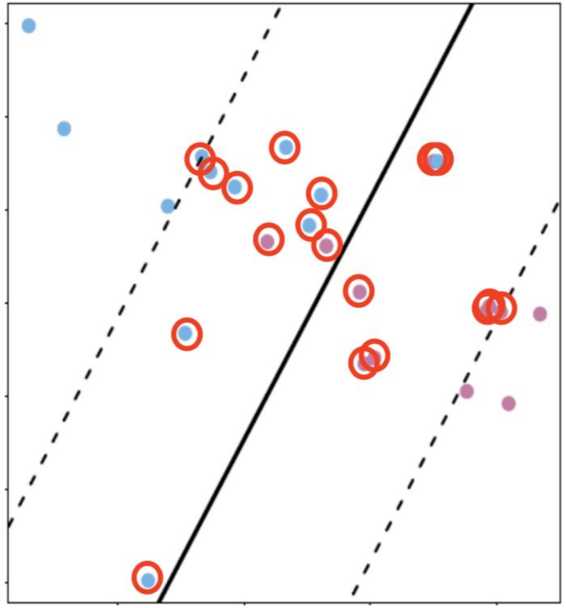
Regularization and the non-separable case

Penalty parameter C

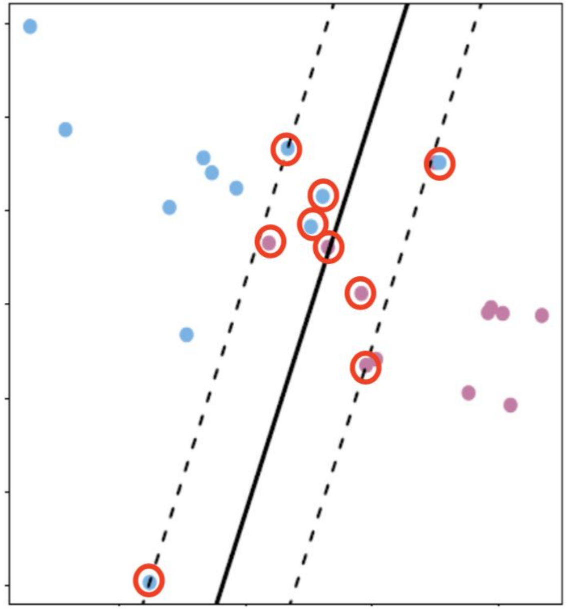
“Budget” for violations, allows at most C misclassifications on training set.

Support vectors

Observations on margin or violating margin.



Large budget
many Support Vectors



Small budget
fewer Support Vectors

Quizz

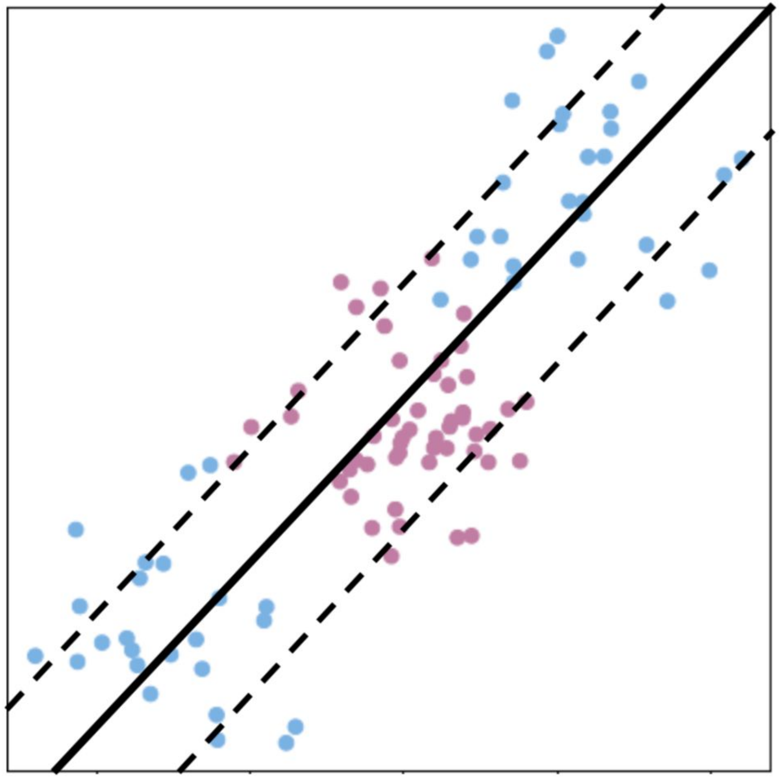
[4 points] Suppose we trained a linear SVM classifier to perform binary classification using the hinge loss $L(\theta^T x, y) = \max\{0, 1 - y\theta^T x\}$. For each of the following scenarios, does the optimal decision boundary necessarily remain the same? Explain your reasoning, perhaps by sketching a picture. Assume that after we perform the action described in each scenario we still have at least one training example in the positive class as well as in the negative class.

i. Remove all examples $(x^{(i)}, y^{(i)})$ with margin > 1.
ii. Remove all examples $(x^{(i)}, y^{(i)})$ with margin < 1.
iii. Add an ℓ_2-regularization term $\frac{\lambda}{2}\theta^T\theta = \frac{\lambda}{2}\|\theta\|_2^2$ to the training loss.
iv. Scale all $x^{(i)}$ by a constant factor α.

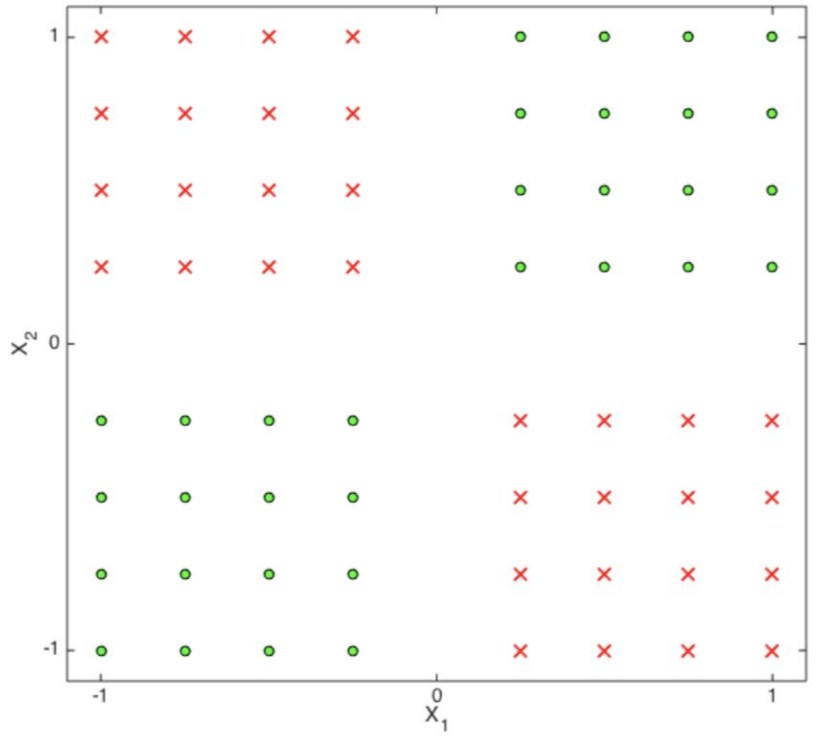
Non-linear decision boundary

Disadvantage

What if we have a non-linear decision boundary?



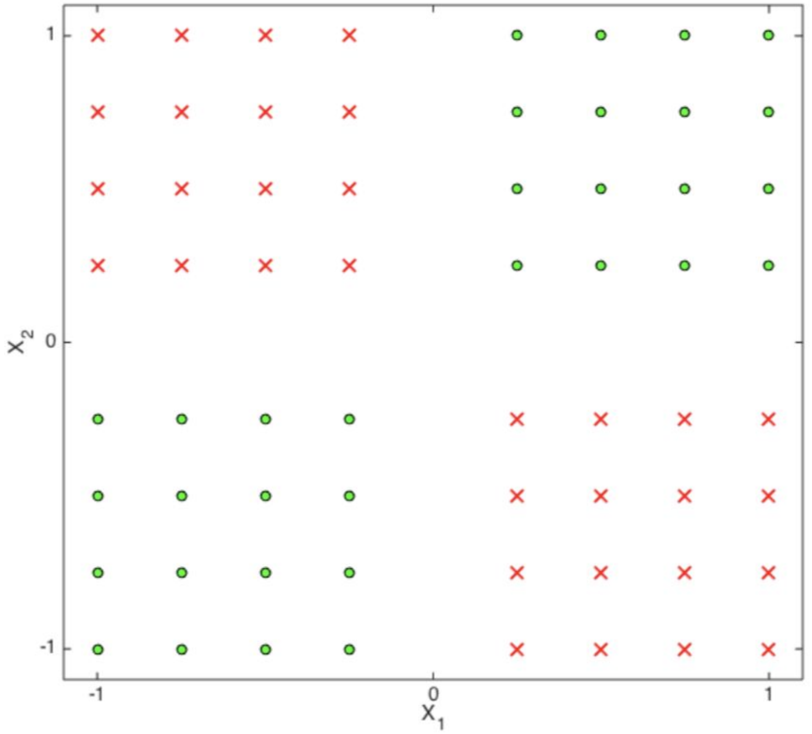
Expanding feature space



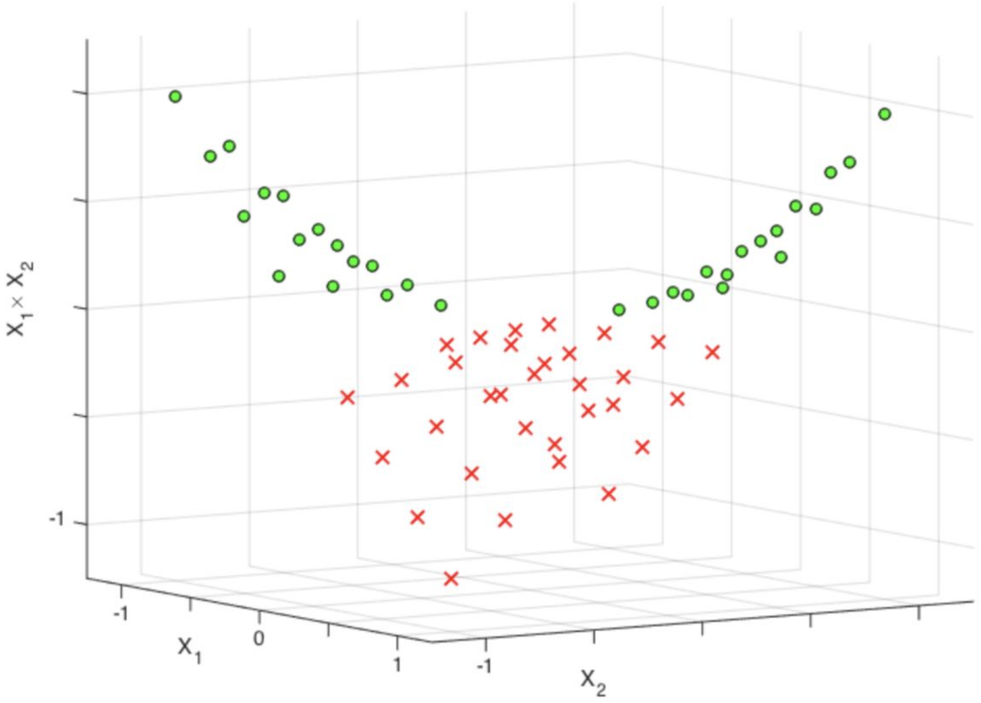
Some data sets are not linearly separable…

But they *become* linearly separable when transformed into a *higher* dimensional space

Expanding feature space



Variables: X1, X2



Variables: X1, X2, X1X2

Non-linear decision boundary

Suppose our original data has *d* features:

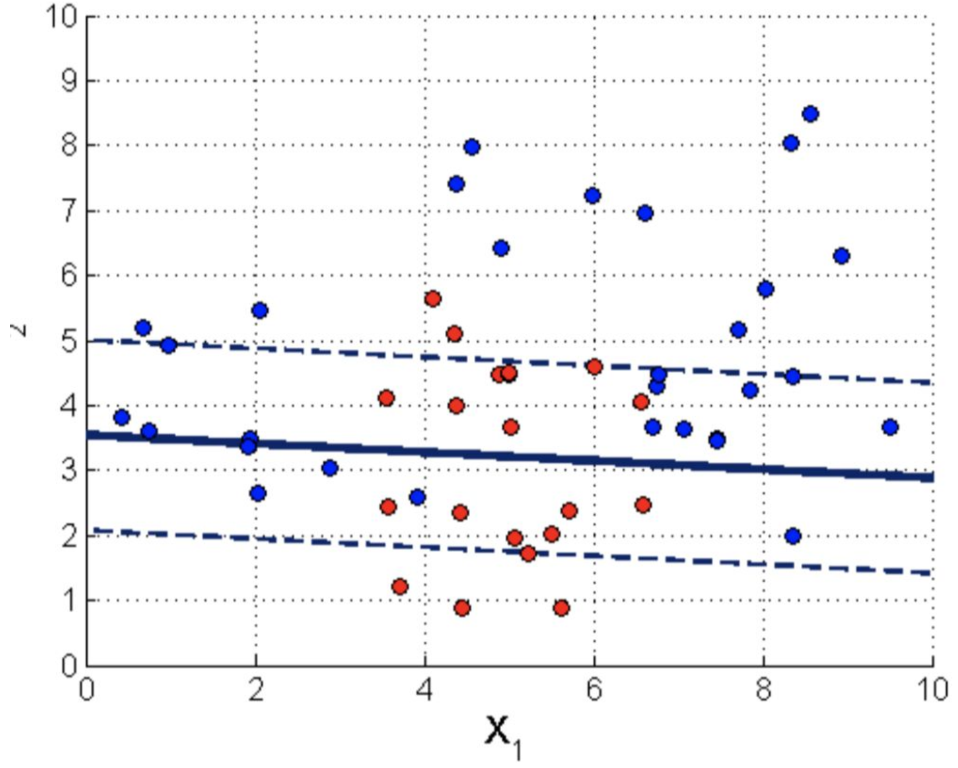
$$X = [X_1, X_2, \cdots, X_d]$$

Expand feature space to include quadratic terms:

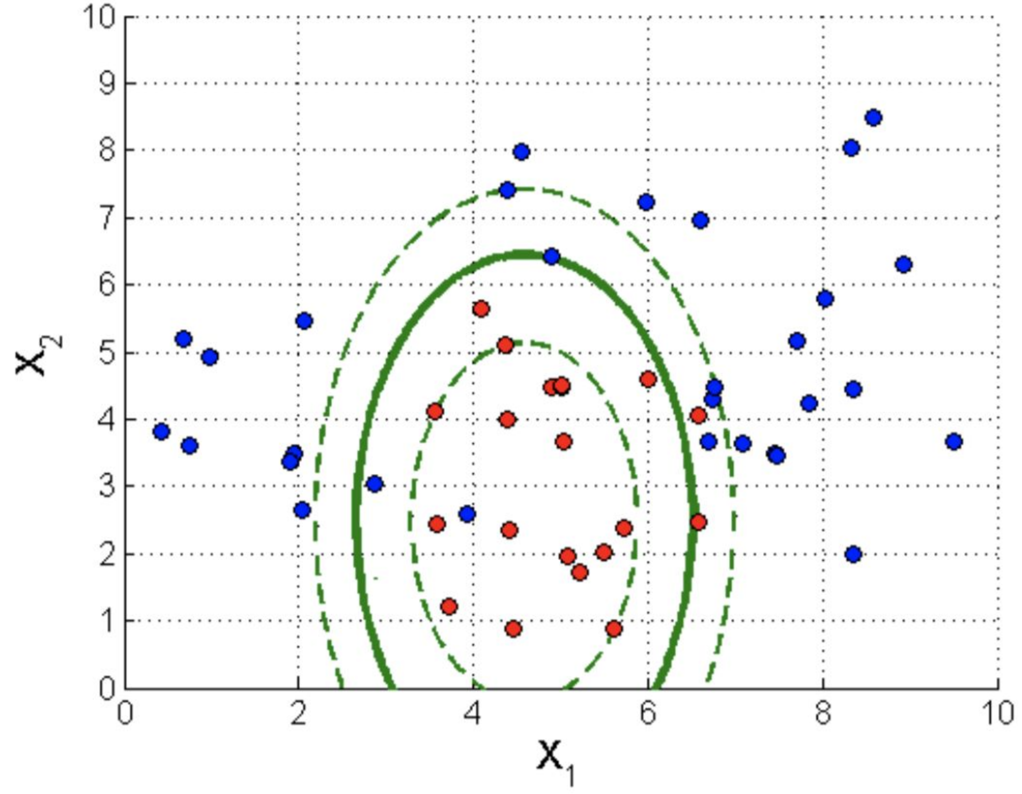
$$\tilde{X} = [\underbrace{X_1}_{\tilde{X}_1}, \underbrace{(X_1)^2}_{\tilde{X}_2}, \underbrace{X_2}_{\tilde{X}_3}, \underbrace{(X_2)^2}_{\tilde{X}_4}, \cdots, \underbrace{X_d}_{\tilde{X}_{2d-1}}, \underbrace{(X_d)^2}_{\tilde{X}_{2d}}]$$

Decision boundary will be non-linear in original feature space (ellipse), but linear in the expanded feature space.

Non-linear decision boundary



Non-linear decision boundary



Non-linear decision boundary

Large number of features becomes computationally challenging.

We need an efficient way to work with large number of features.

Kernels

Kernels

Kernel: Generalization of inner product.

$$\phi : \mathbb{R}^n \longrightarrow \mathbb{R}^d$$

$$K(x, x') = \langle \phi(x), \phi(x') \rangle$$

Kernels (implicitly) map data into higher-dimensional space

Why use kernels instead of explicitly constructing larger feature space?

- Computational advantage when $n << d$ [see the following slide]

Kernels

Consider two equivalent ways to represent a linear classifier.

$$\hat{y}(x) = w^\top \phi(x) = \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} K(x^{(i)}, x)$$

Left: $\{w\}, w \in \mathbb{R}^d$

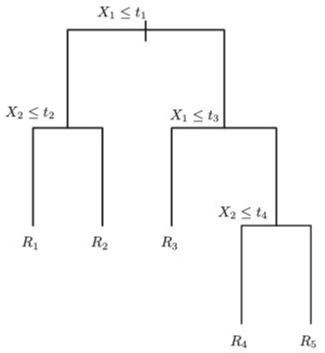
Right: $\{(x^{(i)}, y^{(i)}, \alpha^{(i)})\}_{i=1}^{m}, x^{(i)} \in \mathbb{R}^n, y^{(i)} \in \mathbb{R}, \alpha^{(i)} \in \mathbb{R}$

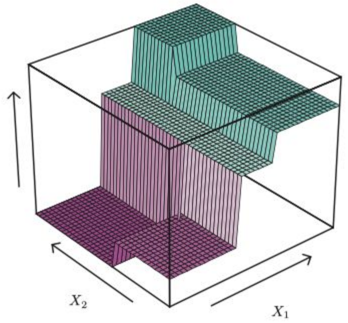
If $d \gg nm$, much more space efficient to use kernelized representation.

Tree Ensembles

Tree Ensembles

Decision Tree: recursively partition space to make predictions.





Prediction: simply predict the average of labels in leaf.

$$\hat{y} = \frac{1}{|R_m|} \sum_{i \in R_m} y^{(i)}$$

Tree Ensembles

Recursive partitioning: split on thresholds of features.

$$S_p(j,t) = (\{x : x_j < t\}, \{x : x_j \geq t\})$$

How to choose? Take average loss in children produced by split.

$$j, t = \arg\min_{j,t}(L(R_1) + L(R_2))$$

Classification: cross-entropy loss $$L_{\text{cross-entropy}} = -\sum_{x \in \mathcal{X}} \hat{p}_x \log \hat{p}_x$$

Regression: mean squared-error loss $$L_{\text{MSE}} = \frac{1}{|R_m|} \sum_{i \in R_m} (y^{(i)} - \bar{y}_m)^2$$

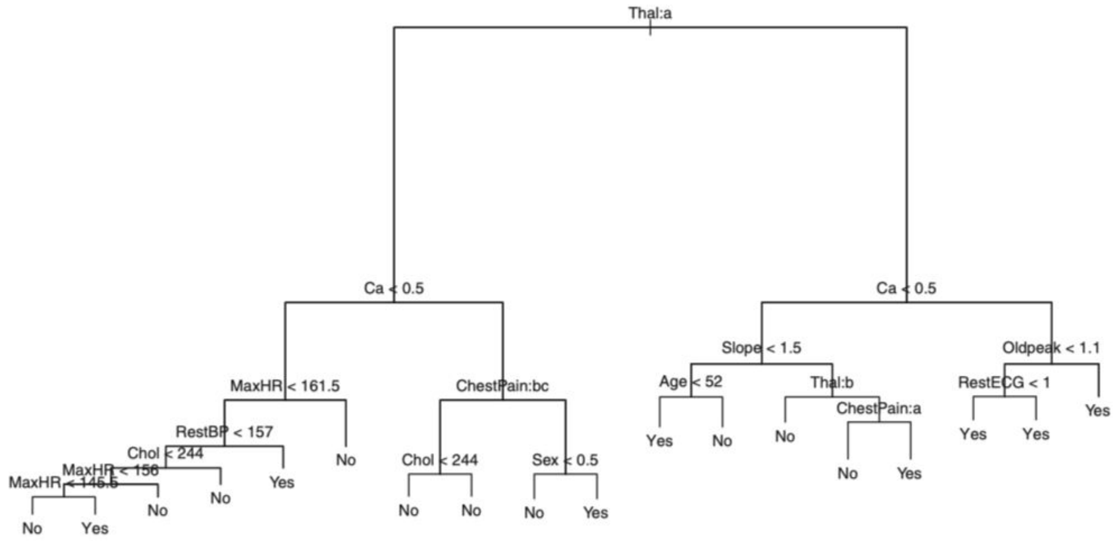
Tree Ensembles

Decision tree tuning:

1. Minimum leaf size
2. Maximum depth
3. Maximum number of nodes
4. Minimum decrease in loss
5. Pruning with validation set

Advantage: easy to interpret!

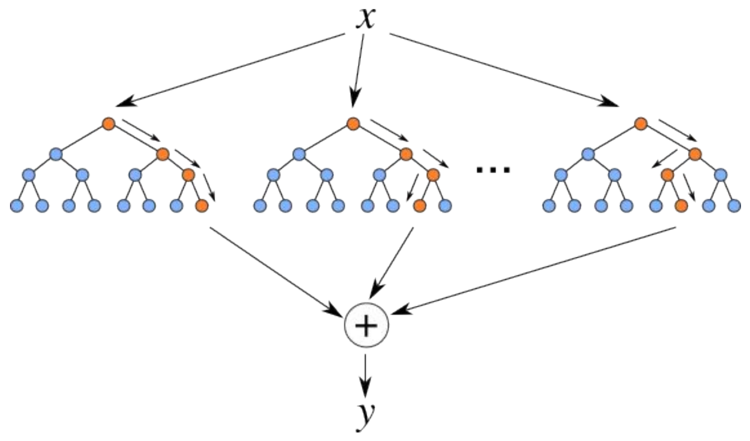
Disadvantage: easy to overfit.



Tree Ensembles

Random forests: Take the average prediction of many decision trees,

1. Each constructed on a *bagged* dataset of the original.
2. Only use a subset (typically √p) features per split.



Each decision tree has high bias, but averaging them together yields low variance.

Bagging: resample of the same size as the original dataset, with replacement.

EM Algorithm / Mixtures

Mixture Models

Gaussian Mixture Model

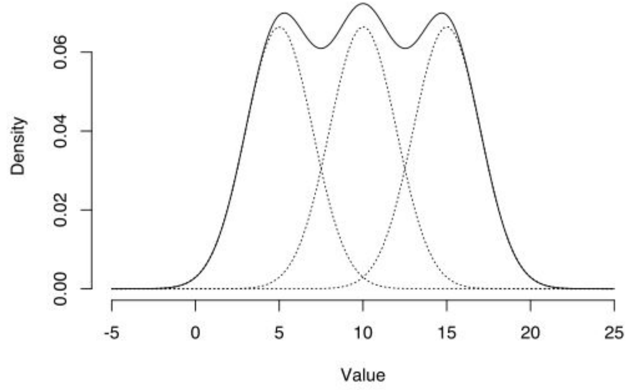
We have n data points, which we suppose come from k Gaussians.

$$p(x) = \prod_{i=1}^{n} \sum_{z=1}^{k} p(z^{(i)} = z) p(x^{(i)} | z^{(i)} = z)$$

Here, $\{z^{(i)}\}_{i=1}^{n} \in \{1, \ldots, k\}^n$

We hypothesize

$$x^{(i)} | z^{(i)} = z \sim \mathcal{N}(\mu_z, \Sigma_z)$$



Mixture Models

GMMs can be extremely effective at modeling distributions!

[Richardson and Weiss 2018: On GANs and GMMs]

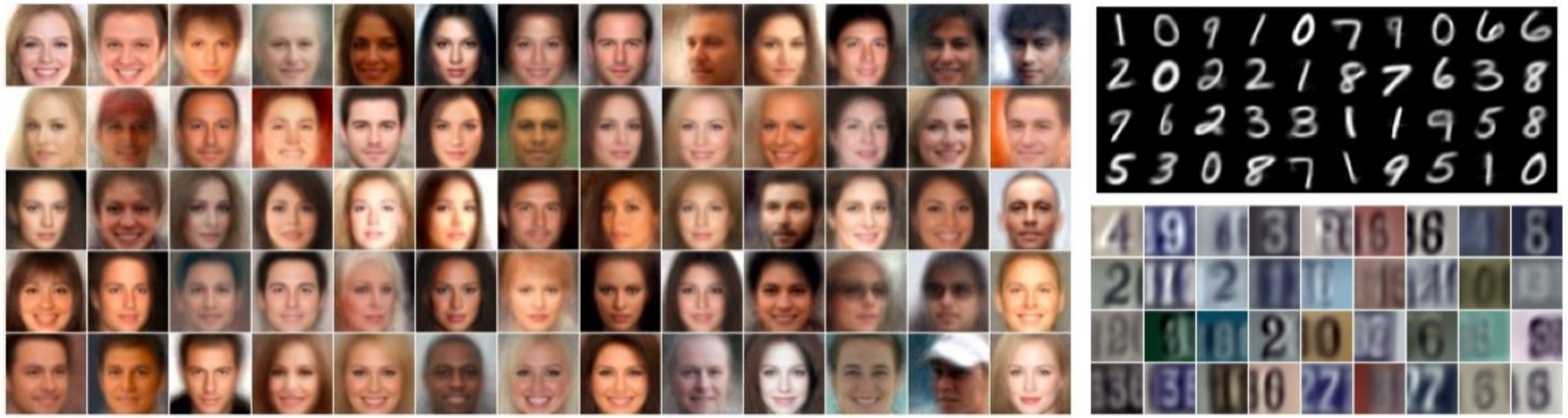


Figure 3: Random samples generated by our MFA model trained on CelebA, MNIST and SVHN

EM Algorithm

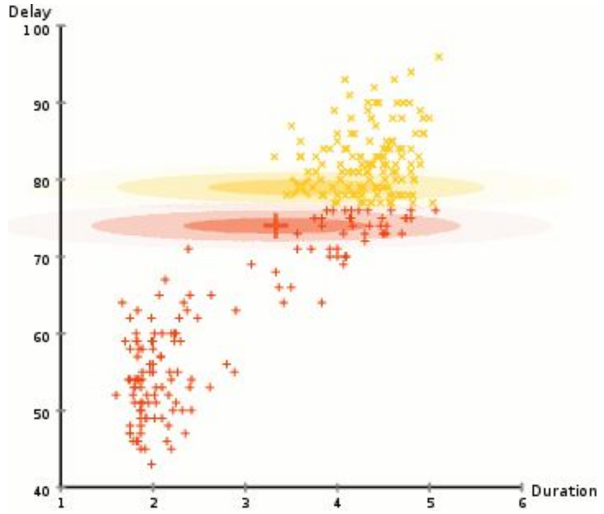
Problem: how do we estimate parameters when there are latent variables?

Idea: maximize marginal likelihood.

$$p(x;\theta) = \sum_z p(x, z; \theta)$$

But this is difficult to compute!

$$z \in \{1,2\}^n \implies \text{sum over } 2^n \text{ possibilities}$$



Example: Mixture of Gaussians. x : data points, z : clusters, $\theta : \{\mu_i, \Sigma_i\}_{i=1}^k$

EM Algorithm

Algorithm

1. Begin with an initial guess for $\theta^{(0)}$
2. Alternate between:
   a. [E-step] Hallucinate missing values by computing, for all possible values z,

   $$p(z|x;\theta^{(t)})$$

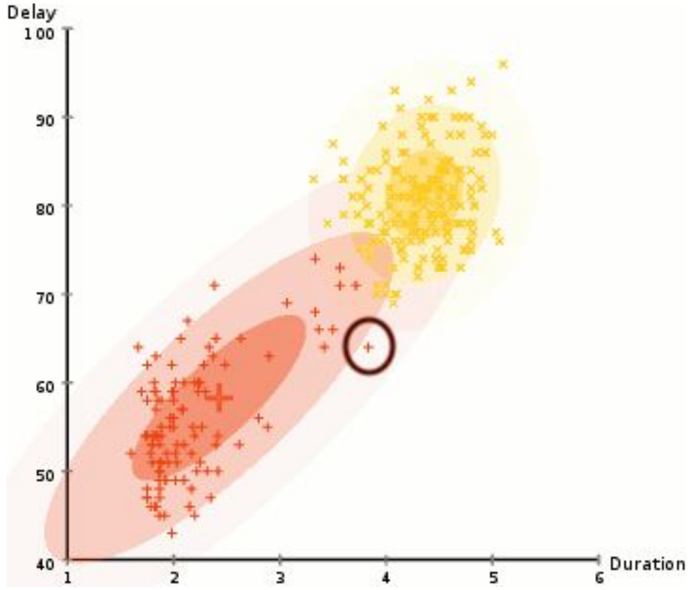
   b. [M-step] Use hallucinated dataset to maximize lower bound on log-likelihood

   $$\begin{aligned}\theta^{(t+1)} &= \arg\max_{\theta} \sum_{z} p(z|x;\theta^{(t)}) \log p(x,z;\theta) \\ &= \arg\max_{\theta} \mathbb{E}_{z\sim p(z|x;\theta^{(t)})}\left[\log p(x,z;\theta)\right]\end{aligned}$$

EM Algorithm

[E-step] Hallucinate missing values by computing, for all possible values z,

$$p(z|x;\theta^{(t)})$$



In the GMM example: for this data point, compute:

$$p(z^{(i)} = 1|x^{(i)}; \mu_1, \mu_2, \Sigma_1, \Sigma_2)$$
$$p(z^{(i)} = 2|x^{(i)}; \mu_1, \mu_2, \Sigma_1, \Sigma_2)$$

Now repeat for *all* data points.

Creates “augmented” dataset, weighted by probabilities.

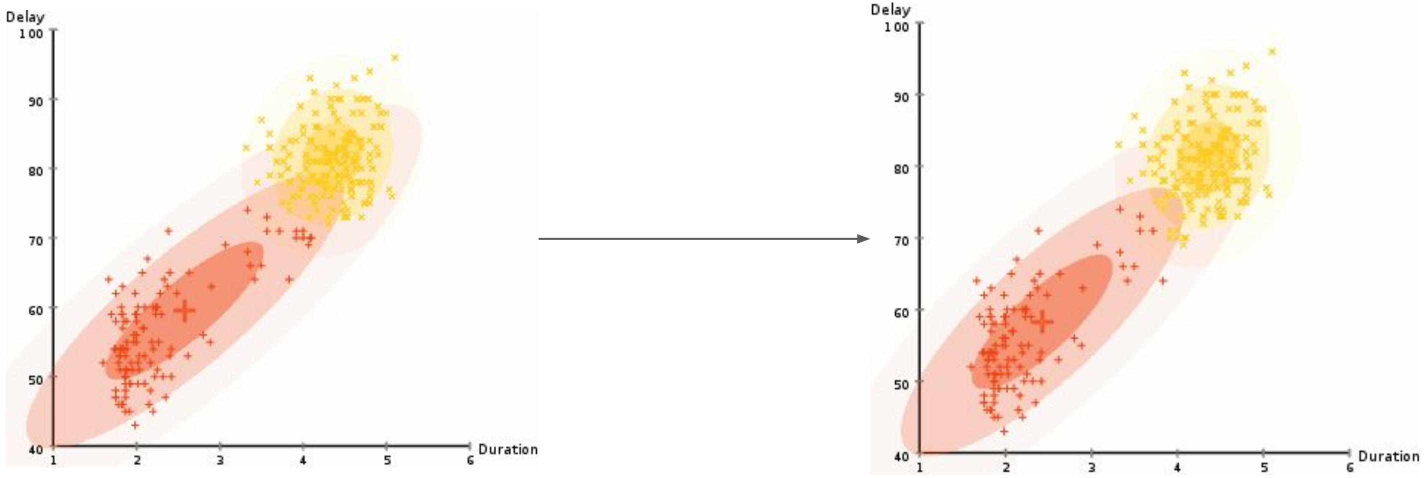
$$\{x^{(i)}, z^{(i)} = 1\}_{i=1}^{n} \quad \{x^{(i)}, z^{(i)} = 2\}_{i=1}^{n}$$

EM Algorithm

[M-step] Use hallucinated dataset to maximize lower bound on log-likelihood

$$\theta^{(t+1)} = \arg\max_{\theta} \sum_{z} p(z|x;\theta^{(t)}) \log p(x,z;\theta) = \arg\max_{\theta} \mathbb{E}_{z\sim p(z|x;\theta^{(t)})}\left[\log p(x,z;\theta)\right]$$

We simply now fit $\mu_1, \mu_2, \Sigma_1, \Sigma_2$ using the augmented dataset with weights.

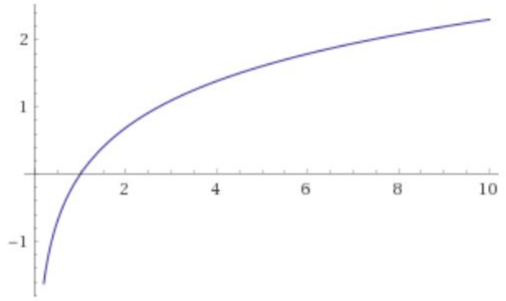


EM Algorithm

Theory

It turns out that we're maximizing a lower bound on the true log-likelihood.

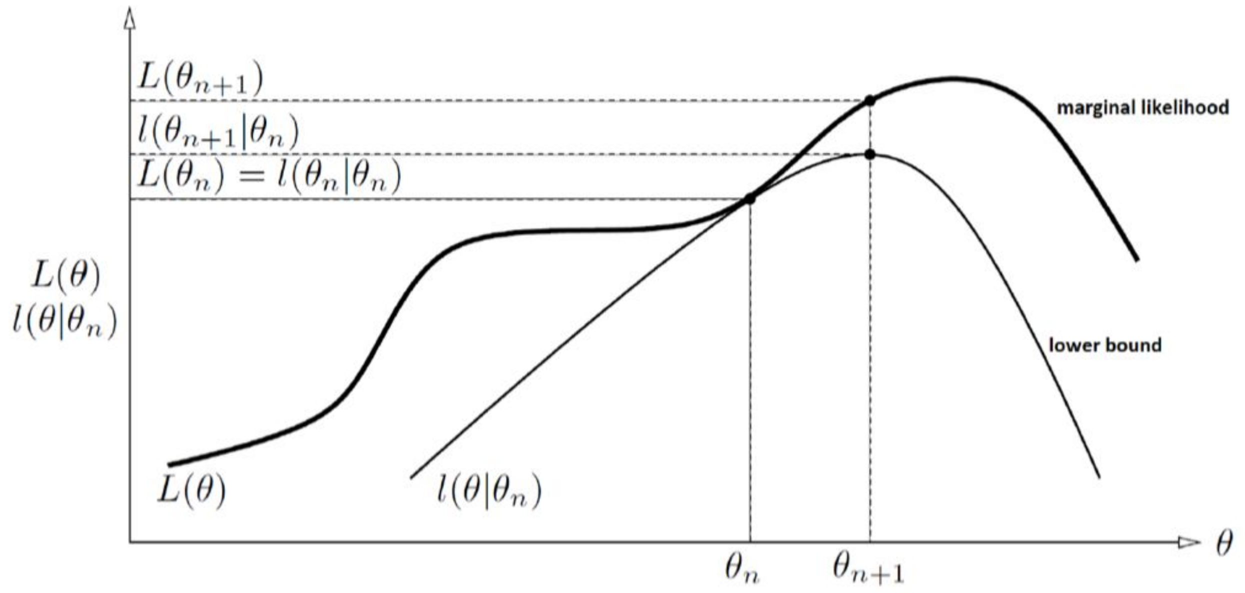
$$\begin{aligned}
\log p(x;\theta) &= \log \sum_z p(x,z;\theta) \\
&= \log \sum_z p(z|x;\theta^{(t)}) \frac{p(x,z;\theta)}{p(z|x;\theta^{(t)})} \\
&= \log \mathbb{E}_{z\sim p(z|x;\theta^{(t)})} \left[\frac{p(x,z;\theta)}{p(z|x;\theta^{(t)})} \right] \\
&\geq \mathbb{E}_{z\sim p(z|x;\theta^{(t)})} \left[\log \frac{p(x,z;\theta)}{p(z|x;\theta^{(t)})} \right] \\
&= \mathbb{E}_{z\sim p(z|x;\theta^{(t)})} [\log p(x,z;\theta)] + \text{constant}
\end{aligned}$$



Here we use Jensen's inequality.

EM Algorithm

Intuition



(Figure adapted from tutorial by Sean Borman)

EM Algorithm

Sanity Check

EM is guaranteed to converge to a *local optimum*.

Why?

Runtime per iteration?

EM Algorithm

Sanity Check

EM is guaranteed to converge to a *local optimum*.

Why?

Our estimated $\log p(x;\theta^{(t)})$ only ever increases.

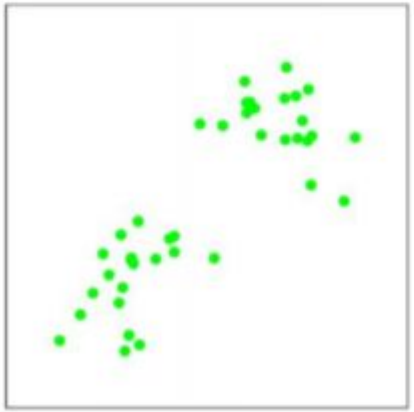
Runtime per iteration?

In general, need to hallucinate one new data point per possible value of z.

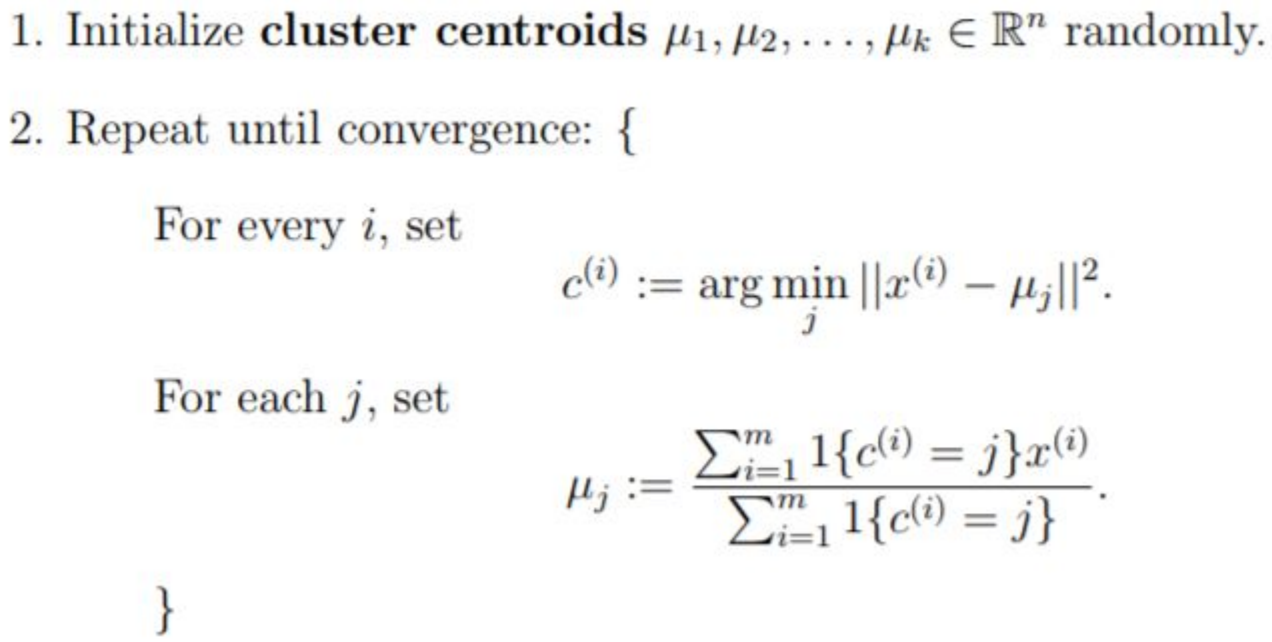
For GMMS, need to hallucinate kn data points thanks to independence.

K Means and More GMMs

K Means - Motivation

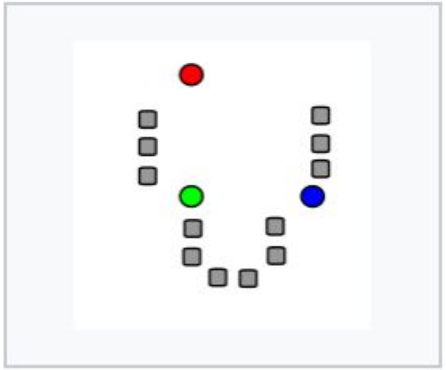


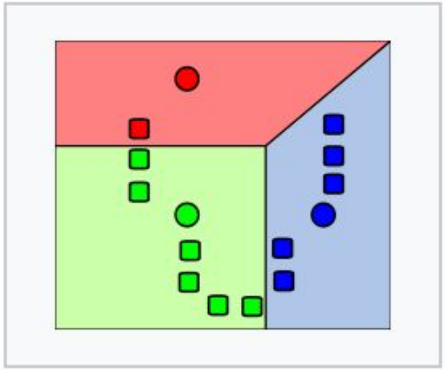
K Means - Algorithm(*)

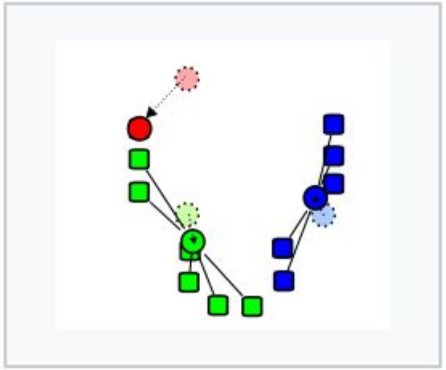


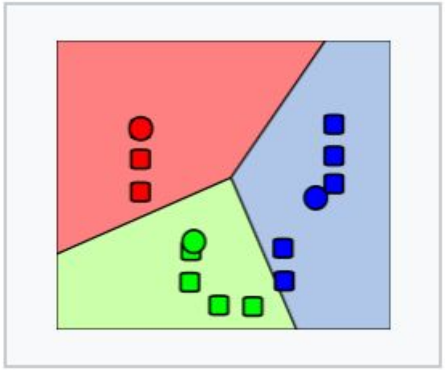
```
1. Initialize **cluster centroids** $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence: {
       For every $i$, set
           $c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$
       For each $j$, set
           $\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}.$
   }
```

K Means

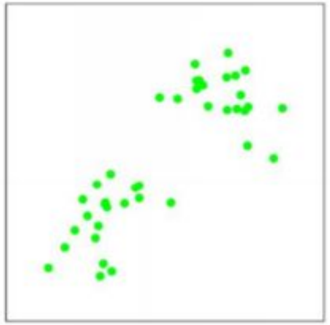

1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

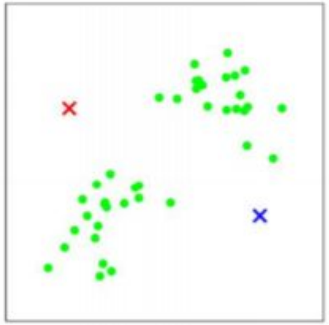

2. *k* clusters are created by associating every observation with the nearest mean.

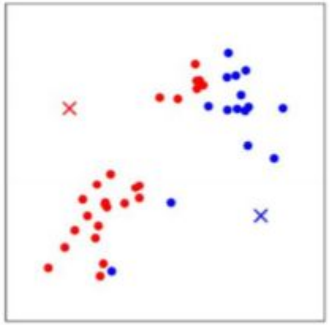

3. The centroid of each of the *k* clusters becomes the new mean.

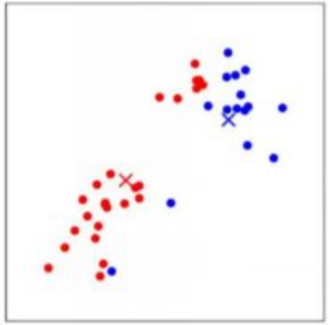

4. Steps 2 and 3 are repeated until convergence has been reached.

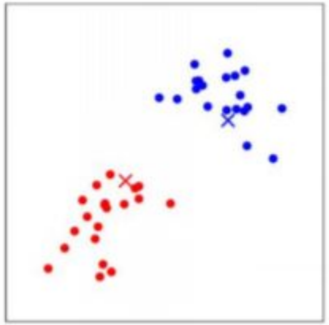
K Means

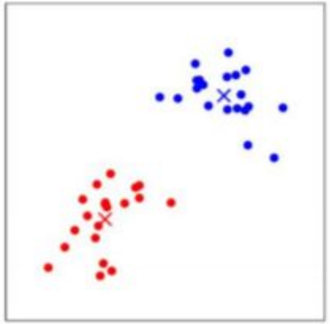

(a)


(b)

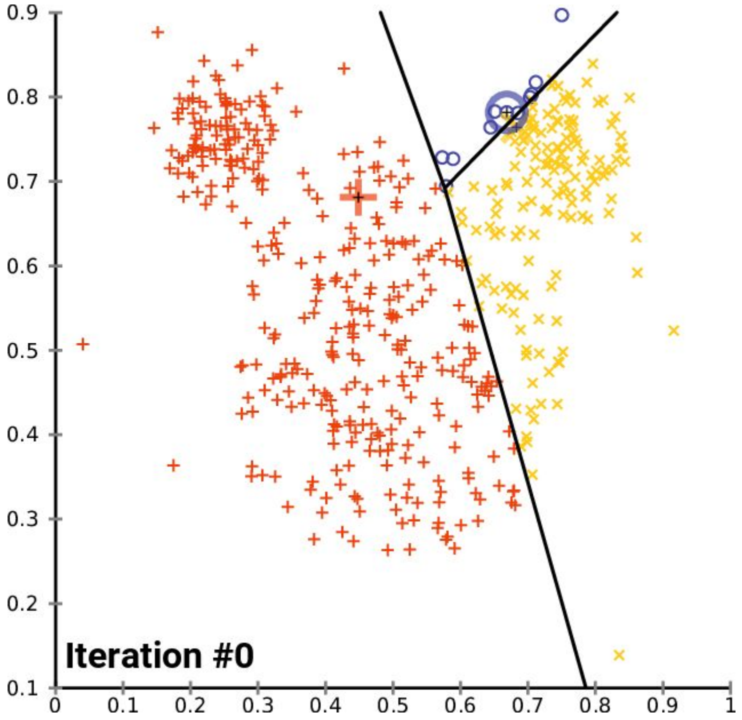

(c)


(d)

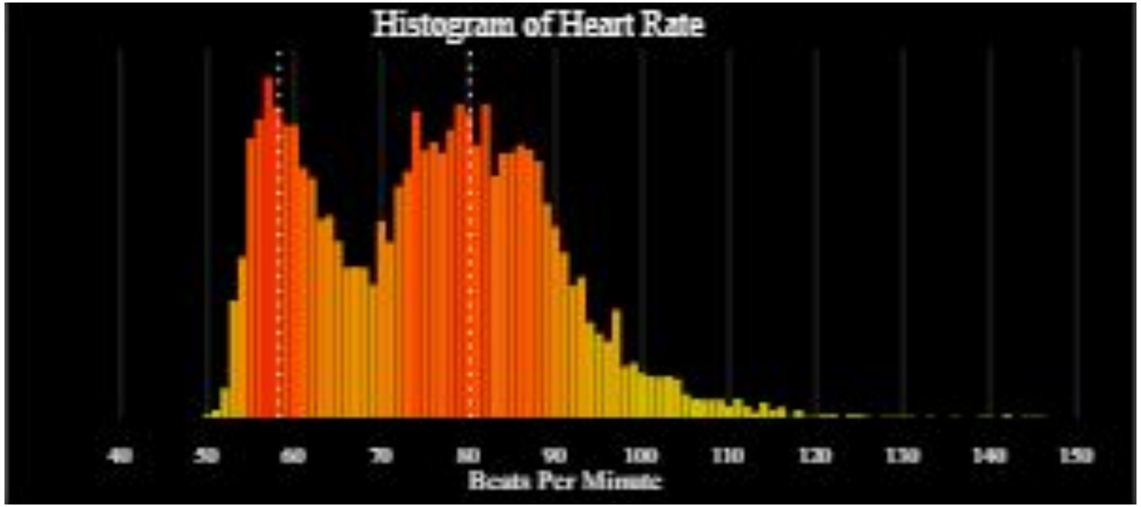

(e)


(f)

K Means



Real-World Example: Mixture of Gaussians



Note: data is unlabeled

How do we fit a GMM - the EM

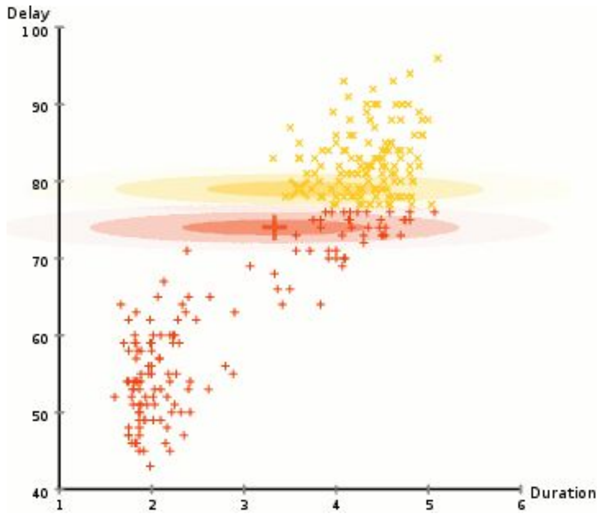
(E-step) For each i, j, set

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

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

(M-step) Update the parameters:

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



Observations

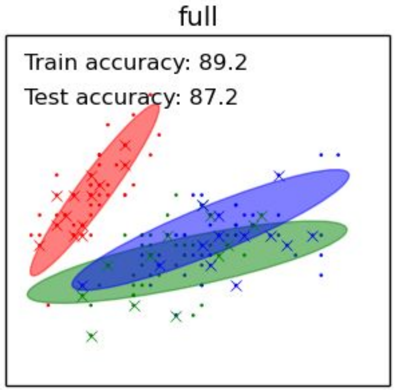
Only the distribution of X matters. You can change things like ordering of the components without affecting the distribution and hence not affecting the algorithm.

Mixing two distributions from a parametric family might give us a third distribution from the same family. A mixture of 2 Bernoullis is another Bernoulli .

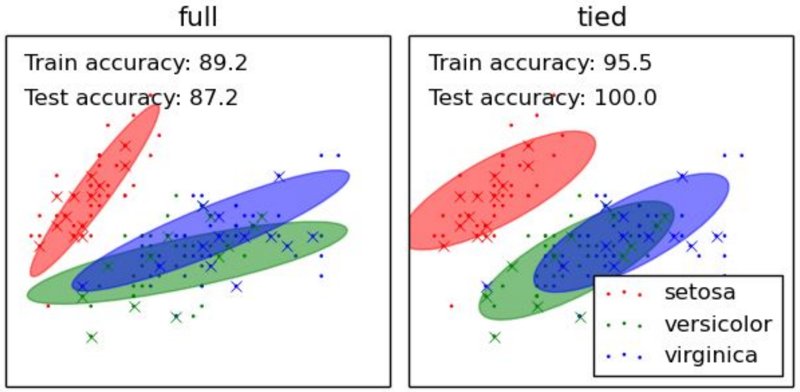
Probabilistic clustering - Putting similar data points together into “clusters”, where clusters are represented by the component distributions.

Sample Question

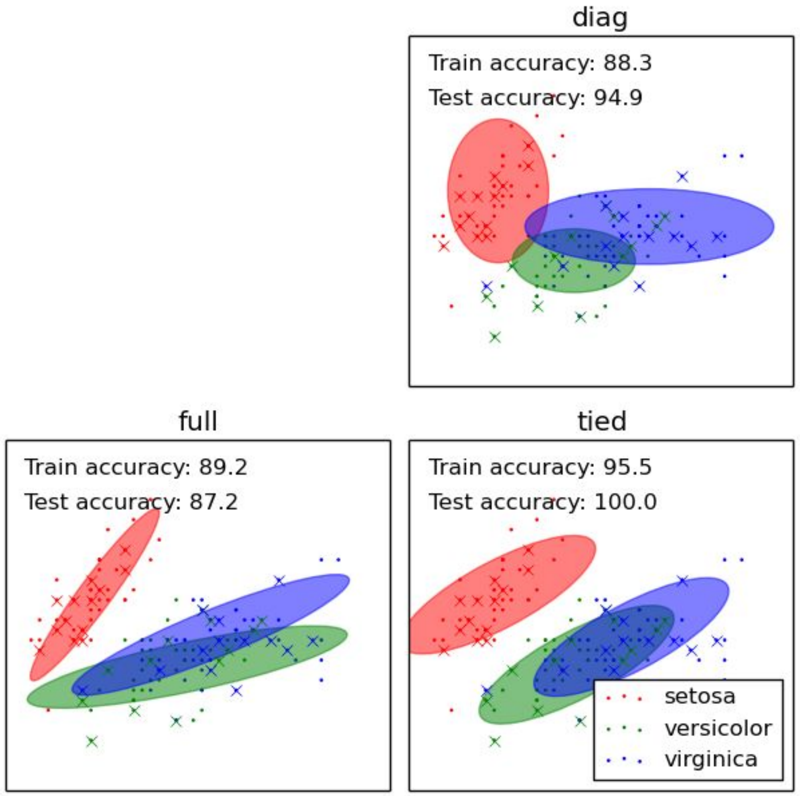
How do constraints on the covariance matrix change the gaussian that is being fit?



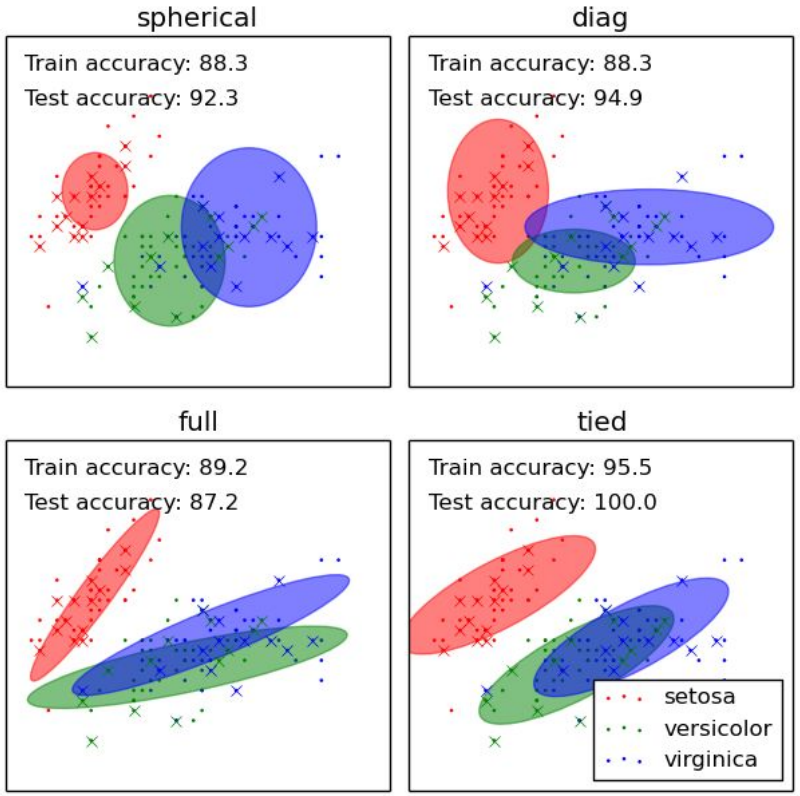
Tied


full
Train accuracy: 89.2
Test accuracy: 87.2
tied
Train accuracy: 95.5
Test accuracy: 100.0
setosa
versicolor
virginica

Diag


diag
Train accuracy: 88.3
Test accuracy: 94.9
full
Train accuracy: 89.2
Test accuracy: 87.2
tied
Train accuracy: 95.5
Test accuracy: 100.0
setosa
versicolor
virginica

Spherical (K-Means!)


spherical
Train accuracy: 88.3
Test accuracy: 92.3
diag
Train accuracy: 88.3
Test accuracy: 94.9
full
Train accuracy: 89.2
Test accuracy: 87.2
tied
Train accuracy: 95.5
Test accuracy: 100.0
setosa
versicolor
virginica