Lecture 22

From Cross-Validation to mixture models with MCMC

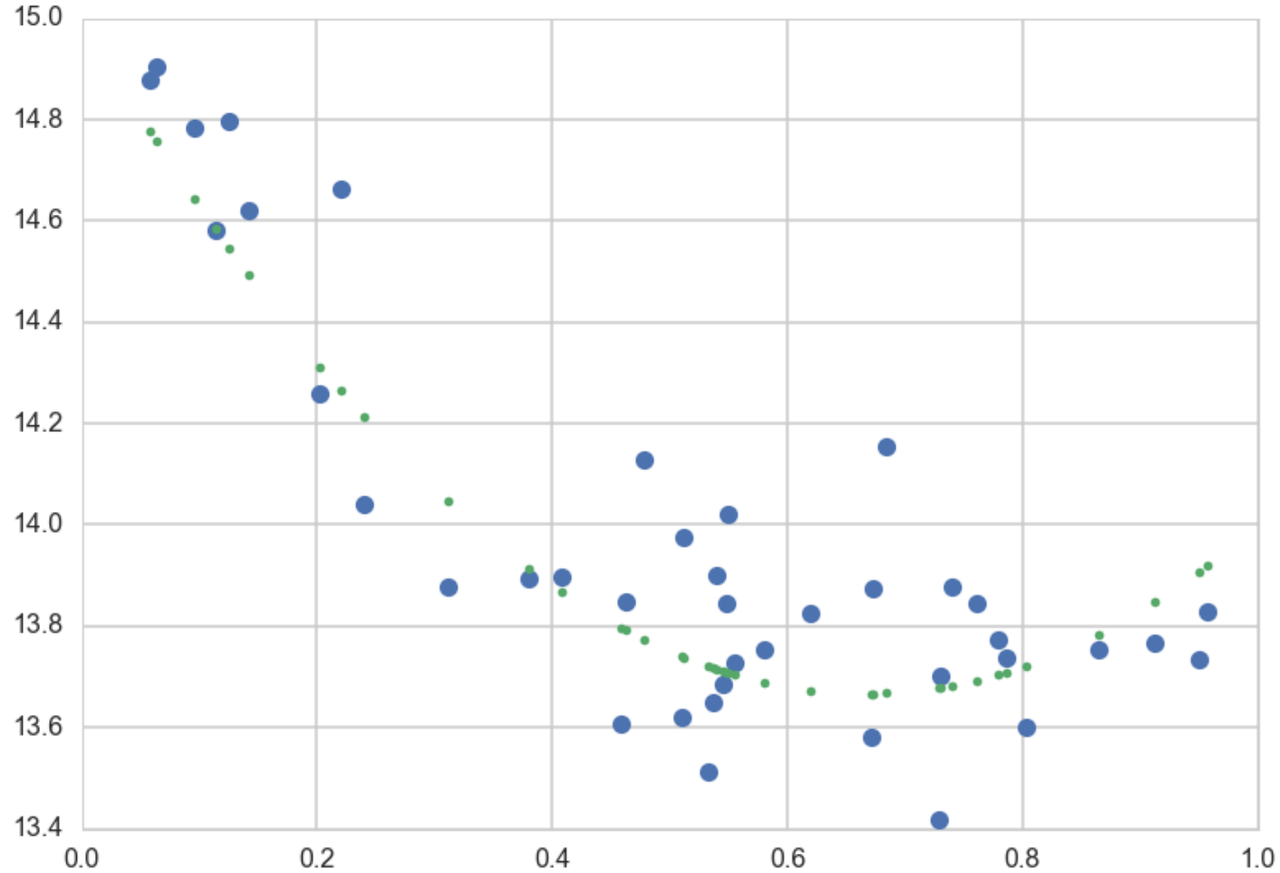
Previously

- Decision theory
- point estimates for decisions
- classification risk
- probabilistic estimates for estimation
- which model to use
- probabilistic estimates for model comparison

Today

- cross-validation and LOOCV
- latent variables
- mixture models
- supervised learning, MCMC
- unsupervised learning, MCMC
- semi-supervised learning, MCMC

Dataset



- 20 data points
- keep train and test, we'll see these later
- center x for speed of sampling
- fit multiple order polynomials
- The `logp` method of a node gives us the conditional (log) probability of the node given its parents.
- Conditional probability of the observed node, which is also the model node, ...

model.logp({paramdict})

gives us likelihood of data $p(y|etc)$

```
ridge=3
sigma=0.2
pf=PolynomialFeatures(msize, include_bias=False).fit_transform(xtrain.reshape(-1,1))
print(pf.shape)
with pm.Model() as m:
    alpha = pm.Normal('alpha', 0, 100)
    beta = pm.Normal('beta', mu=0, sd=ridge, shape=msize)
    mu = alpha + pm.math.dot(pf, beta)
    o = pm.Normal('o', mu, sigma, observed=ytrain)
    trace=pm.sample(5000, init='MAP')
```

```
meanpoint={'alpha': trace['alpha'].mean(), 'beta': trace['beta'].mean(0)}
{'alpha': 13.840500632702547, 'beta': array([-0.99418043,  2.31483997])}
m.logp(meanpoint), mtest.logp(meanpoint)
(array(-1.699085634529018), array(-5.251454815218393))
```

Deviance

$$D(q) = -2 \sum_i log(q_i),$$

then

$$D_{KL}(p, q) - D_{KL}(p, r) = \frac{2}{N}(D(q) - D(r))$$

More generally: $D(q) = -\frac{N}{2} E_p[log(q)]$

Bayesian deviance

$D(q) = -\frac{N}{2} E_p[log(pp(y))]$ posterior predictive for points y on the test set or future data

replace joint posterior predictive over new points y by product of marginals:

ELPD: $\sum_i E_p[log(pp(y_i))]$

Since we do not know the true distribution p,

replace elpd: $\sum_i E_p[log(pp(y_i))]$

by the computed "log pointwise predictive density" (lppd) **in-sample**

$$\sum_j log\left\langle p(y_j|\theta)\right\rangle = \sum_j log\left(\frac{1}{S}\sum_s p(y_j|\theta_s)\right)$$

WAIC

$$WAIC = lppd + 2p_W$$

where

$$p_W = 2\sum_i (log(E_{post}[p(y_i|\theta)] - E_{post}[log(p(y_i|\theta))])$$

Once again this can be estimated by

$$\sum_i Var_{post}[log(p(y_i|\theta))]$$

```
For Model with 1 slope, waic is WAIC_r(WAIC=18.086154924759676, WAIC_se=10.162063571524614, p_WAIC=4.6685686490640128)
For Model with 2 slope, waic is WAIC_r(WAIC=-7.9746057666052668, WAIC_se=4.8913389425902176, p_WAIC=2.8290346198804275)
For Model with 4 slope, waic is WAIC_r(WAIC=-5.1476015964310298, WAIC_se=4.5248859948089732, p_WAIC=5.0163702157862842)
For Model with 10 slope, waic is WAIC_r(WAIC=91.825697802482608, WAIC_se=34.336737894534828, p_WAIC=53.519631187786722)
For Model with 19 slope, waic is WAIC_r(WAIC=61.178349788662352, WAIC_se=25.002444963302214, p_WAIC=38.140875766495071)
```

it is tempting to use information criteria to compare models with different likelihood functions. Is a Gaussian or binomial better? Can't we just let WAIC sort it out?
Unfortunately, WAIC (or any other information criterion) cannot sort it out. The problem is that deviance is part normalizing constant. The constant affects the absolute magnitude of the deviance, but it doesn't affect fit to data.

— *McElreath*

cross-validation

- estimate the out-of-sample risk as an average, thus gaining robustness to odd validation sets
- providing some measure of uncertainty on the out-of-sample performance.
- less data to fit so biased models
- we are not talking here about cross-validation to do hyperparameter optimization

hyperparameter fitting

- part of the prior specification, uses entire data set
- or we can use empirical bayes, and use entire data set.
- faster than cross-val but prone to model mis-specification
- but EB is not a model selection procedure

LOOCV

- The idea here is that you fit a model on N-1 data points, and use the Nth point as a validation point. Clearly this can be done in N ways.
- the N-point and N-1 point posteriors are likely to be quite similar, and one can sample one from the other by using importance sampling.

$$E_f[h] = \frac{\sum_s w_s h_s}{\sum_s w_s} \text{ where } w_s = f_s/g_s.$$

Fit the full posterior once. Then we have

$$w_s = \frac{p(\theta_s|y_{-i})}{p(\theta_s|y)} \propto \frac{1}{p(y_i|\theta_s, y_{-i})}$$

- the importance sampling weights can be unstable out in the tails.
- importance weights have a long right tail, pymc (`pm.loo`) fits a generalized pareto to the tail (largest 20% importance ratios) for each held out data point i (a MLE fit). This smooths out any large variations.

$$elpd_{loo} = \sum_i log(p(y_i|y_{-i}))$$

$$= \sum_i log\left(\frac{\sum_s w_{is} p(y_i|\theta_s)}{\sum_s w_{is}}\right)$$

over the training sample.

```
For Model with 1 slope, loo is LOO_r(LOO=19.345559544708721, LOO_se=10.448387931742174, p_LOO=5.2982709590385335)
For Model with 2 slope, loo is LOO_r(LOO=-6.4907292356212007, LOO_se=5.6639101289730434, p_LOO=3.5709728853724583)
For Model with 4 slope, loo is LOO_r(LOO=-7.8411331189326319, LOO_se=4.9343760037143145, p_LOO=3.6696044545354827)
For Model with 10 slope, loo is LOO_r(LOO=-1.3783161338355767, LOO_se=6.6512996064759902, p_LOO=6.917624219627621)
For Model with 19 slope, loo is LOO_r(LOO=1.1788924202325761, LOO_se=7.3342281925207269, p_LOO=8.1411470822801828)
```

What should you use?

1. LOOCV and WAIC are fine. The former can be used for models not having the same likelihood, the latter can be used with models having the same likelihood.

2. WAIC is fast and computationally less intensive, so for same-likelihood models (especially nested models where you are really performing feature selection), it is the first line of attack

3. One does not always have to do model selection. Sometimes just do posterior predictive checks to see how the predictions are, and you might deem it fine.

4. For hierarchical models, WAIC is best for predictive performance within an existing cluster or group. Cross validation is best for new observations from new groups

Latent variables

- instead of bayesian vs frequentist, think hidden vs not hidden
- key concept: full data likelihood vs partial data likelihood
- probabilistic model is a *joint distribution* $p(\mathbf{x}, \mathbf{z})$
- observed variables $\mathbf{x}$ corresponding to data, and latent variables $\mathbf{z}$

From `edwardlib`: $p(\mathbf{x} \mid \mathbf{z})$

describes how any data $\mathbf{x}$ depend on the latent variables $\mathbf{z}$.

- **The likelihood posits a data generating process**, where the data $\mathbf{x}$ are assumed drawn from the likelihood conditioned on a particular hidden pattern described by $\mathbf{z}$.
- The *prior* $p(\mathbf{z})$ is a probability distribution that describes the latent variables present in the data. **The prior posits a generating process of the hidden structure**.

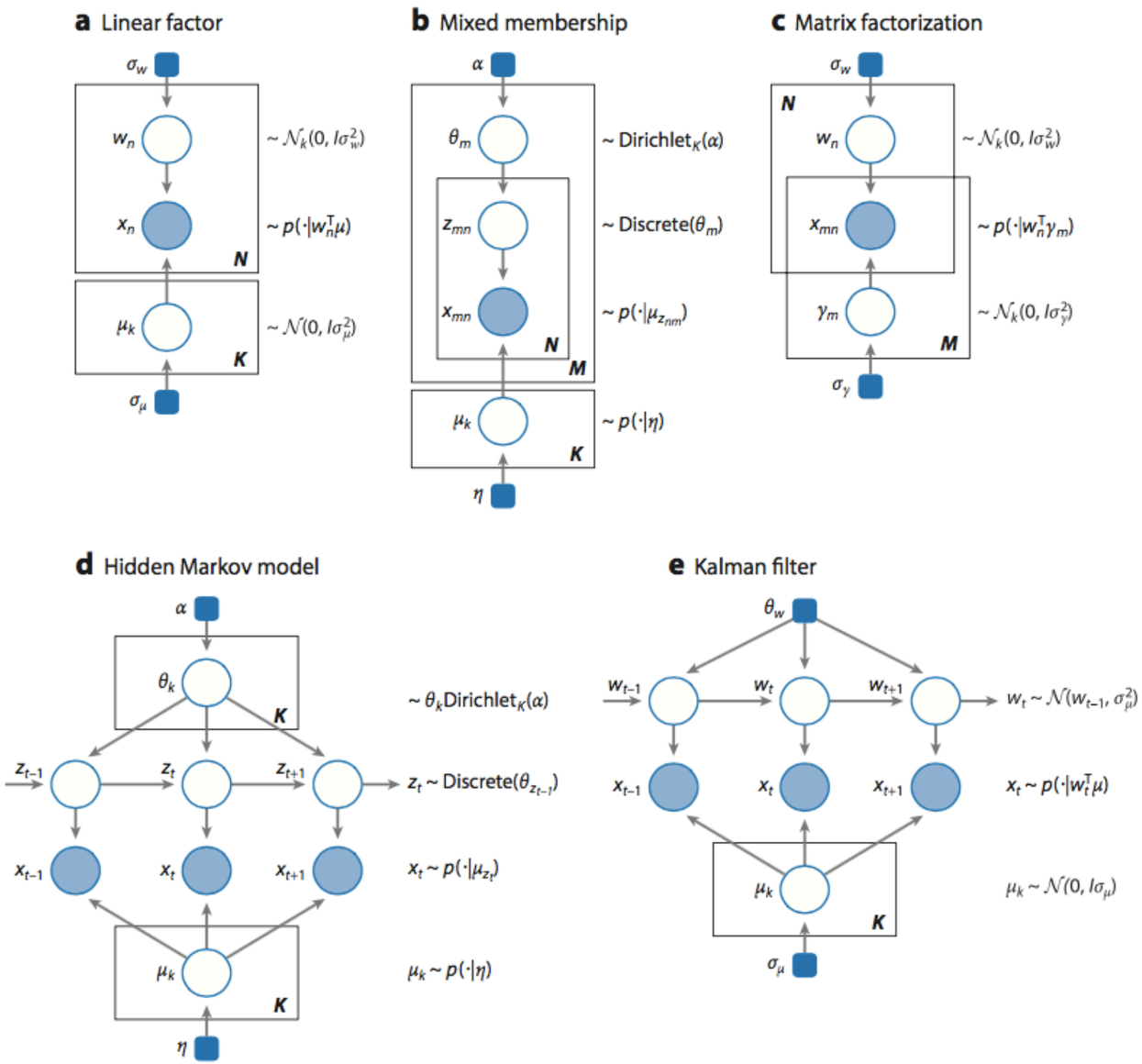
Any bayesian parameters can be considered as $\mathbf{z}$.

More generally posit hidden structure $\rightarrow$

e.g. **Ratings Latent Factor Model**:

$$Y_{um} = \mu + \theta_u[0] + \gamma_m[0] + \theta_u[1:]^\top \gamma_m[1:] + \epsilon_{um}$$

where $\epsilon_{um} \sim N(0, \sigma)$ and γ_m is an item-specific with first element item-specific bias and remaining latent factors for item m; θ_u is ditto for users; μ overall ratings mean and σ is residual variance of ratings.



Mixture Models

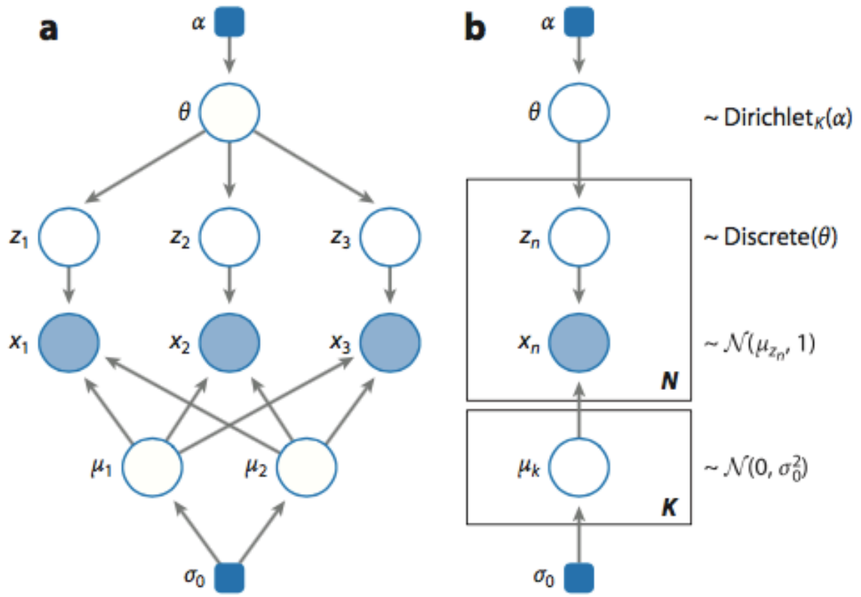


Figure 3

(*a*) A graphical model for a mixture of two Gaussians. There are three data points. The shaded nodes are observed variables, the unshaded nodes are hidden variables, and the blue square boxes are fixed hyperparameters (such as the Dirichlet parameters). (*b*) A graphical model for a mixture of K Gaussians with N data points.

A distribution $p(x|\{\theta_k\})$ is a mixture of K component distributions $p_1, p_2, \ldots p_K$ if:

$$p(x|\{\theta_k\}) = \sum_k \lambda_k p_k(x|\theta_k)$$

with the λ_k being mixing weights, $\lambda_k > 0$, $\sum_k \lambda_k = 1$.

Example: Zero Inflated Poisson

Generative Model (Sense 1): How to simulate from it?

$$Z \sim Categorical(\lambda_1, \lambda_2, \ldots, \lambda_K)$$

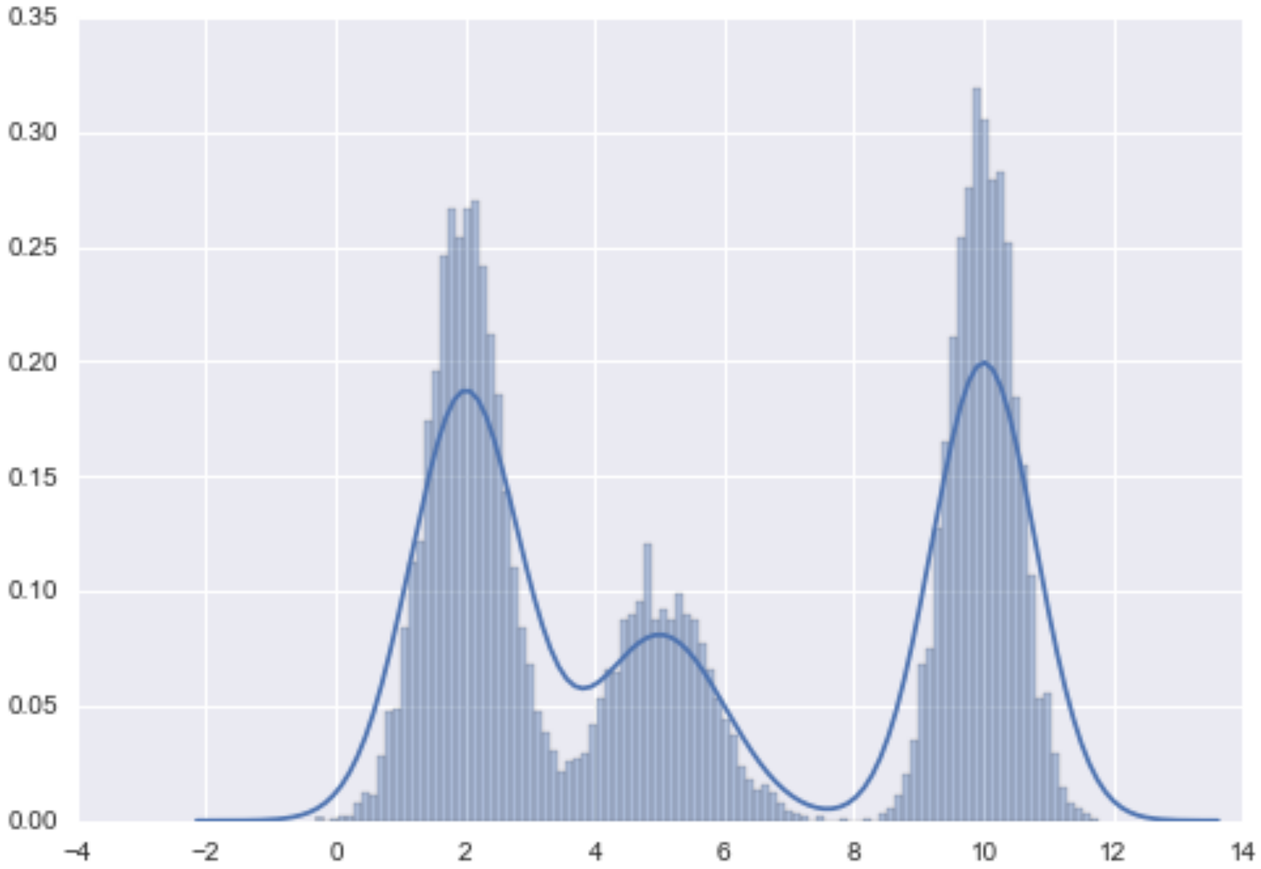
where Z says which component X is drawn from.

Thus λ_j is the probability that the hidden class variable $z = j$.

Then: $X \sim p_z(x|\theta_z)$ and general structure is:

$$p(x|\theta) = \sum_z p(x, z) = \sum_z p(z)p(x|z, \theta) \text{ where } \theta = \{\theta_k\}.$$

Gaussian Mixture Model



$$p(x|\{\theta_k\}) = \sum_k \lambda_k N(x|\mu_k, \Sigma_k)$$

Generative:

```
mu_true = np.array([2, 5, 10])
sigma_true = np.array([0.6, 0.8, 0.5])
lambda_true = np.array([.4, .2, .4])
n = 10000

# Simulate from each distribution according to mixing proportion psi
z = multinomial.rvs(1, lambda_true, size=n) #categorical
x=np.array([np.random.normal(mu_true[i.astype('bool')][0],\
    sigma_true[i.astype('bool')][0]) for i in z])
```

```
multinomial.rvs(1,[0.6,0.1, 0.3], size=10)
array([[1, 0, 0],[0, 0, 1],...[1, 0, 0],[1, 0, 0]])
```

Generative Sense 2 (for classifiers)

For a feature vector x, we use Bayes rule to express the posterior of the class-conditional as:

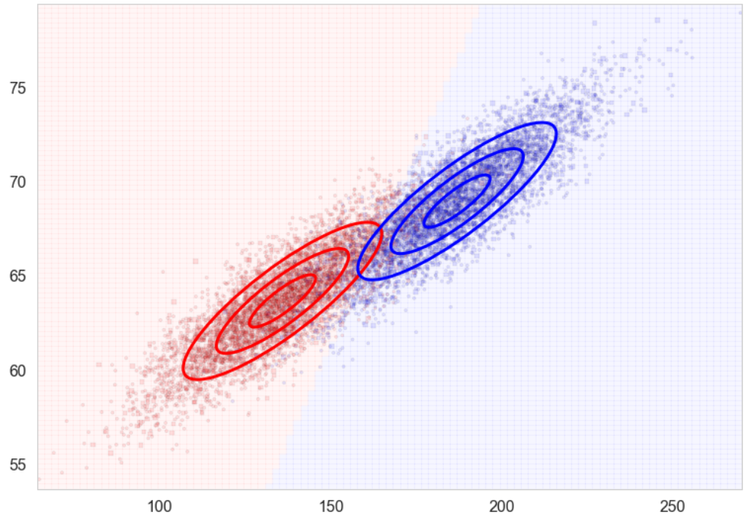
$$p(z = c|x, \theta) = \frac{p(z = c|\theta)p(x|z = c, \theta)}{\sum_{c'} p(z = c'|\theta)p(x|z = c', \theta)}$$

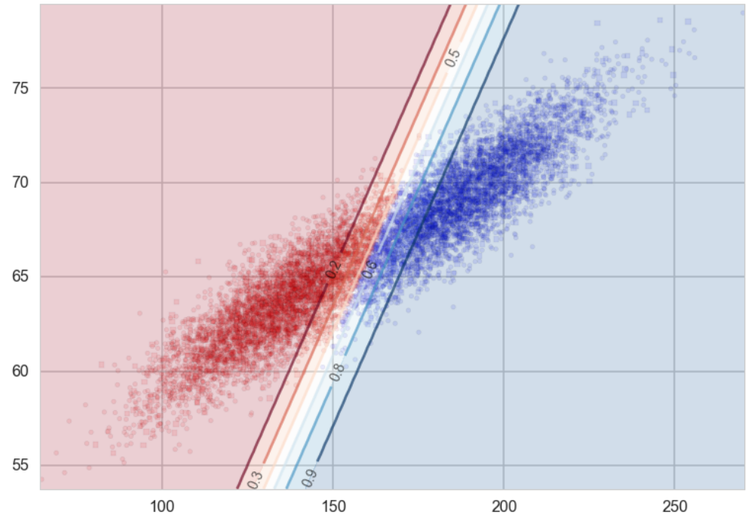
This is a **generative classifier**, since it specifies how to generate the data using the class-conditional density $p(x|z = c, \theta)$ and the class prior $p(z = c|\theta)$.

Discriminative classifier

Directly fit the class posterior, $p(z = c|x, \theta)$.

For example, a Gaussian Mixture model vs logistic regression.





Supervised vs Unsupervised Learning

In **Supervised Learning**, Latent Variables $\mathbf{z}$ are observed.

In other words, we can write the full-data likelihood $p(\mathbf{x}, \mathbf{z})$

In **Unsupervised Learning**, Latent Variables $\mathbf{z}$ are hidden.

We can only write the observed data likelihood:

$$p(\mathbf{x}) = \sum_z p(\mathbf{x}, \mathbf{z}) = \sum_z p(\mathbf{z}) p(\mathbf{x}|\mathbf{z})$$

GMM supervised formulation

$$Z \sim \text{Bernoulli}(\lambda)$$
$$X|z=0 \sim \mathcal{N}(\mu_0, \Sigma_0),\ X|z=1 \sim \mathcal{N}(\mu_1, \Sigma_1)$$

Full-data loglike: $l(x, z|\lambda, \mu_0, \mu_1, \Sigma) = -\sum_{i=1}^{m} \log((2\pi)^{n/2}|\Sigma|^{1/2})$

$$-\frac{1}{2}\sum_{i=1}^{m}\sum_{i=1}^{m}(x-\mu_{z_i})^T \Sigma^{-1}(x-\mu_{z_i}) + \sum_{i=1}^{m}[z_i \log\lambda + (1-z_i)\log(1-\lambda)]$$

Solution to MLE

$$\lambda = \frac{1}{m}\sum_{i=1}^{m} \delta_{z_i,1}$$

$$\mu_0 = \frac{\sum_{i=1}^{m} \delta_{z_i,0}\, x_i}{\sum_{i=1}^{m} \delta_{z_i,0}}$$

$$\mu_1 = \frac{\sum_{i=1}^{m} \delta_{z_i,1}\, x_i}{\sum_{i=1}^{m} \delta_{z_i,1}}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{z_i})(x_i - \mu_{z_i})^T$$

Classification

We can use the log likelihood at a given x as a classifier: assign class depending upon which probability $p(x_j|\lambda, z, \Sigma)$ is larger. (JUST x likelihood, as we want to compare probabilities at fixed zs).

$$log\, p(x_j|\lambda, z, \Sigma) = -\sum_{i=1}^{m} \log((2\pi)^{n/2}|\Sigma|^{1/2}) - \frac{1}{2}\sum_{i=1}^{m}(x-\mu_{z_i})^T\,\Sigma^{-1}(x-\mu_{z_i})$$

The first term of the likelihood does not matter since it is independent of z.

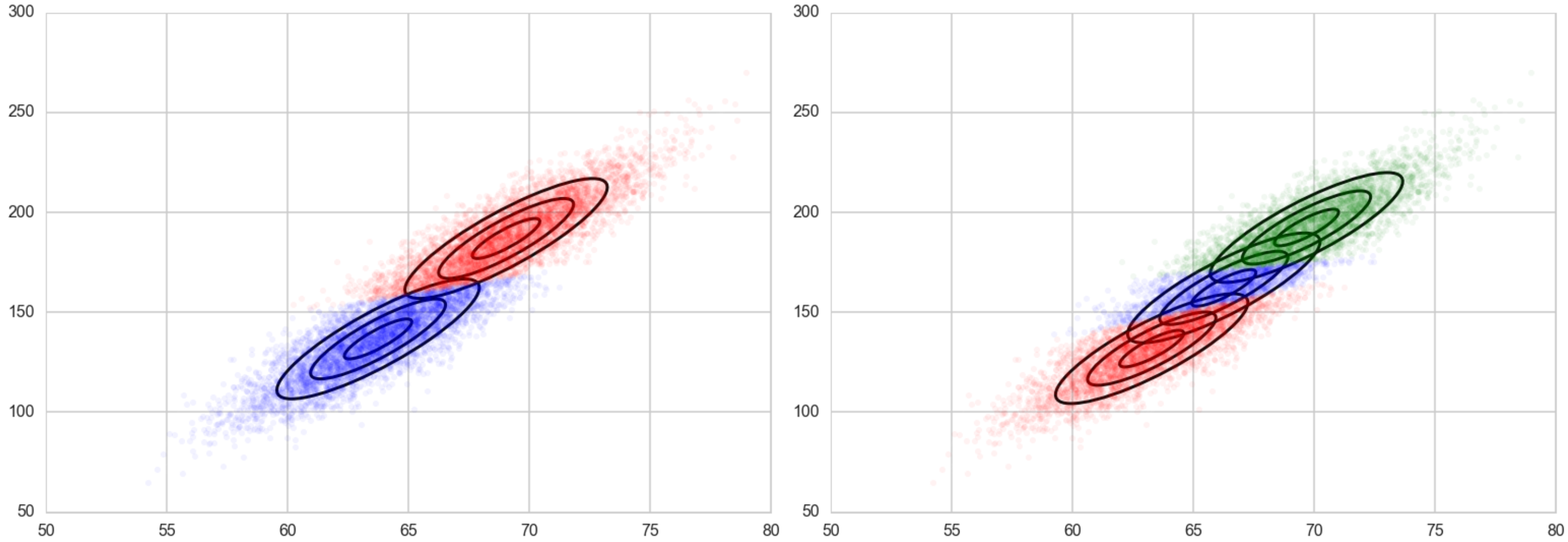
Unsupervised: Soft k-means

responsibility of cluster k for point i, and can be computed as before using Bayes rule as follows:

$$p(z_k = c|x_i, \theta) = \frac{p(z_k = c|\theta)p(x_i|z_k = c, \theta)}{\sum_{c'} p(z_k = c'|\theta)p(x_i|z_k = c', \theta)}$$

Here we never observe z_k for any samples, whereas before with the generative GDA classifier, we did observe z_k on the training set.

How many clusters? How many $\mathbf{z}$?



Concrete Formulation of unsupervised learning

Estimate Parameters by $\mathbf{x}$-MLE:

$$
\begin{aligned}
l(x|\lambda, \mu, \Sigma) &= \sum_{i=1}^{m} \log p(x_i|\lambda, \mu, \Sigma) \\
&= \sum_{i=1}^{m} \log \sum_{z} p(x_i|z_i, \mu, \Sigma)\, p(z_i|\lambda)
\end{aligned}
$$

Not Solvable analytically! EM and Variational. Or do MCMC.

Semi-supervised learning

We have some labels, but typically very few labels: not enough to form a good training set. Likelihood a combination.

$$l(\{x_i\}, \{x_j\}, \{z_i\}|\theta, \lambda) = \sum_i log\, p(x_i, z_i|\lambda, \theta) + \sum_j log\, p(x_j|\lambda, \theta)$$

$$= \sum_i log\, p(z_i|\lambda) p(x_i|z,\theta) + \sum_j log \sum_z p(z_j|\lambda) p(x_j|z_j, \theta)$$

Here i ranges over the data points where we have labels, and j over the data points where we dont.

Semi-supervised learning

Basic Idea: there is structure in $p(x)$ which might help us divine the conditionals, thus combine full-data and $\mathbf{x}$-likelihood.

Include x on the validation set in the likelihood, and x and z on the training set in the likelihood.

Has been very useful for Naive Bayes.