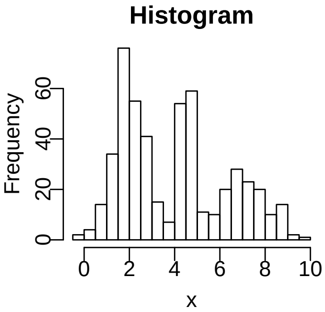
Section 5.
Mixture Methods

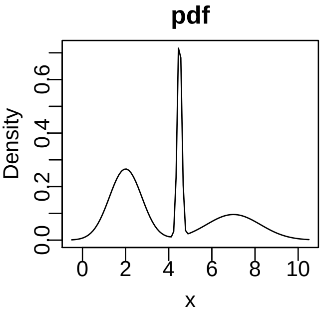
Vincent Dorie

New York University

Mixture Models

- Class of models with observations coming from than one distribution, membership unknown





Mixture Model Density

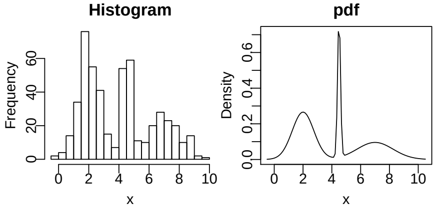
- Depends on a **latent** class membership

$$y_i \mid z_i \sim \begin{cases} p_1(y_i) & \text{if } z_i = 1 \\ p_2(y_i) & \text{if } z_i = 2 \\ \vdots & \\ p_K(y_i) & \text{if } z_i = K \end{cases},$$

$$z_i \sim \text{Categorical}(\lambda).$$

$\lambda \geq 0, \sum_{k=1}^{K} \lambda_k = 1$

Example Density



- $y_i \mid z_i \sim$ Normal(4.5, 0.1) if $z_i = 1$,
- $y_i \mid z_i \sim$ Normal(2, 2) if $z_i = 2$,
- $y_i \mid z_i \sim$ Normal(7, 1.25) if $z_i = 3$,
- $\lambda = (0.2, 0.5, 0.3)$

```
data {
  int<lower = 0> N;
  int<lower = 0> K;
  real y[N];
}
parameters {
  simplex[K] lambda;
  int z[N];

  real[K] mu;
  real<lower = 0> sigma[K];
}
model {
  z ~ categorical(lambda);
  for (n in 1:n)
    y[n] ~ normal(mu[z[n]], sigma[z[n]])
}
```

```
data {
  int<lower = 0> N;
  int<lower = 0> K;
  real y[N];
}
parameters {
  simplex[K] lambda;
  int z[N];

  real mu[K];
  real<lower = 0> sigma[K];
}
model {
  z ~ categorical(lambda);
  for (n in 1:n)
    y[n] ~ normal(mu[z[n]], sigma[z[n]])
}
integer parameters or transformed parameters are not allowed;
found declared type int, parameter name=z Problem with
declaration.
```

Integrate/Sum Out z

$$p(y_i, z_i) = \Big[\mathsf{I}[z_i = 1]p_1(y_i) + \mathsf{I}[z_i = 2]p_2(y_i) + \cdots + \mathsf{I}[z_i = K]p_K(y_i)\Big] \prod_{k=1}^{K} \lambda_k^{\mathsf{I}[z_i = k]}.$$

$$\sum_{z_i=1}^{K} p(y_i, z_i) = \lambda_1 p_1(y_i) + \lambda_2 p_2(y_i) + \cdots + \lambda_K p_K(y_i),$$

$$= p(y_i).$$

Multiply together to get likelihood:

$$p(y) = \prod_{i=1}^{N} \left[\lambda_1 p_1(y_i) + \lambda_2 p_2(y_i) + \cdots + \lambda_K p_K(y_i)\right].$$

`log` Problem

- Operate on log probability as multiplication leads to underflow
  - `small #` $\times$ `small #` $\times$ ...
  - $\log($`small #`$) + \log($`small #`$)+$ ...
- Log-likelihood is unwieldy:

$$\log p(y) = \sum_{i=1}^{N} \log\left[\lambda_1 p_1(y_i) + \lambda_2 p_2(y_i) + \cdots + \lambda_K p_K(y_i)\right].$$

log_sum_exp

- Stan efficiently computes from individual components

$$
\begin{aligned}
\text{log\_sum\_exp}(a, b) &= \log\left(e^a + e^b\right), \\
\text{log\_sum\_exp}(v) &= \log \sum_i e^{v_i}.
\end{aligned}
$$

```
data {
  int<lower = 0> N;
  int<lower = 0> K;
  real y[N];
}
parameters {
  simplex[K] lambda;
  real mu[K];
  real<lower = 0> sigma[K];
}
model {
  real probs[K];
  for (n in 1:N) {
    for (k in 1:K) {
      probs[k] = log(lambda[k]) +
                 normal_lpdf(y[n] | mu[k], sigma[k]);
    }
    target += log_sum_exp(probs);
  }
}
```

- For person i, $\mathrm{probs[k]} = \log p_k + \log p(y_i)$

Recovering Classes

```
generated quantities {
  matrix[N, K] classProbs;

  for (n in 1:N) {
    vector[K] indivProbs;
    for (k in 1:K) {
      indivProbs[k] = log(lambda[k]) +
        normal_lpdf(y[n] | mu[k], sigma[k]);
    }
    classProbs[n,] = softmax(indivProbs)';
  }
}
```