TOPIC 4

Probabilistic modeling, inference and sampling

3 Probabilistic modeling: examples

Course: Math 535 (http://www.math.wisc.edu/~roch/mmids/) - Mathematical Methods in Data Science (MMiDS)
Author: Sebastien Roch (http://www.math.wisc.edu/~roch/), Department of Mathematics, University of Wisconsin-Madison
Updated: Dec 14, 2020

We give some examples of probabilistic models and associated inference tasks. In the process, we revisit previously encountered data science problems (classification, clustering) from a model-based point of view.

3.1 Example 1: Generalized linear models

Generalized linear models provide a vast generalization of linear regression using exponential families. Quoting from Wikipedia (https://en.wikipedia.org/wiki/Generalized_linear_model), the context in which they arise is the following:

> Ordinary linear regression predicts the expected value of a given unknown quantity (the response variable, a random variable) as a linear combination of a set of observed values (predictors). This implies that a constant change in a predictor leads to a constant change in the response variable (i.e. a linear-response model). This is appropriate when the response variable can vary, to a good approximation, indefinitely in either direction, or more generally for any quantity that only varies by a relatively small amount compared to the variation in the predictive variables, e.g. human heights. However, these assumptions are inappropriate for some types of response variables.

> For example, in cases where the response variable is expected to be always positive and varying over a wide range, constant input changes lead to geometrically (i.e. exponentially) varying, rather than constantly varying, output changes. [...] Similarly, a model that predicts a probability of making a yes/no choice (a Bernoulli variable) is even less suitable as a linear-response model, since probabilities are bounded on both ends (they must be between 0 and 1). [...] Generalized linear models cover all these situations by allowing for response variables that have arbitrary distributions (rather than simply normal distributions), and for an arbitrary function of the response variable (the link function) to vary linearly with the predicted values (rather than assuming that the response itself must vary linearly).

In its simplest form, a generalized linear model assumes that an outcome variable $y \in \mathbb{R}$ is generated from an exponential family p_θ, where θ is a linear combination of the predictor variables $\mathbf{x} \in \mathbb{R}^d$. That is, we assume that $\theta = \mathbf{w}^T\mathbf{x}$ for unknown $\mathbf{w} \in \mathbb{R}^d$ and the probability distribution of y is of the form

$$p_{\mathbf{w}^T\mathbf{x}}(y) = h(y)\exp\big((\mathbf{w}^T\mathbf{x})\phi(y) - A(\mathbf{w}^T\mathbf{x})\big)$$

for some sufficient statistic $\phi(y)$.

Given data points $(\mathbf{x}_i, y_i)_{i=1}^n$, the model is fitted using maximum likelihood as follows. Under independence of the samples, the likelihood of the data is $\prod_{i=1}^n p_{\mathbf{w}^T\mathbf{x}_i}(y_i)$, which we seek to maximize over $\mathbf{w}$. As before, we work with the minus log-likelihood

$$L_n(\mathbf{w}; \{(\mathbf{x}_i, y_i)_{i=1}^n\}) = -\sum_{i=1}^n \log p_{\mathbf{w}^T\mathbf{x}_i}(y_i).$$

The gradient with respect to $\mathbf{w}$ is given by

$$\begin{aligned}
\nabla_{\mathbf{w}} L_n(\mathbf{w}; \{(\mathbf{x}_i, y_i)_{i=1}^n\}) &= -\sum_{i=1}^n \nabla_{\mathbf{w}} \log\big[h(y_i)\exp\big(\mathbf{w}^T\mathbf{x}_i\phi(y_i) - A(\mathbf{w}^T\mathbf{x}_i)\big)\big] \\
&= -\sum_{i=1}^n \nabla_{\mathbf{w}} \left[\log h(y_i) + \mathbf{w}^T\mathbf{x}_i\phi(y_i) - A(\mathbf{w}^T\mathbf{x}_i)\right] \\
&= -\sum_{i=1}^n \left[\mathbf{x}_i\phi(y_i) - \nabla_{\mathbf{w}} A(\mathbf{w}^T\mathbf{x}_i)\right].
\end{aligned}$$

By the *Chain Rule* and our previous formulas,

$$\nabla_{\mathbf{w}} A(\mathbf{w}^T\mathbf{x}_i) = A'(\mathbf{w}^T\mathbf{x}_i)\,\mathbf{x}_i = \mu(\mathbf{w}; \mathbf{x}_i)\,\mathbf{x}_i$$

where $\mu(\mathbf{w}; \mathbf{x}_i) = \mathbb{E}[\phi(Y_i)]$ with $Y_i \sim p_{\mathbf{w}^T\mathbf{x}_i}$. That is,

$$\nabla_{\mathbf{w}} L_n(\mathbf{w}; \{(\mathbf{x}_i, y_i)_{i=1}^n\}) = -\sum_{i=1}^n \mathbf{x}_i(\phi(y_i) - \mu(\mathbf{w}; \mathbf{x}_i)).$$

The Hessian of $A(\mathbf{w}^T\mathbf{x}_i)$, again by the *Chain Rule* and our previous formulas, is

$$A''(\mathbf{w}^T\mathbf{x}_i)\,\mathbf{x}_i\mathbf{x}_i^T = \sigma^2(\mathbf{w};\mathbf{x}_i)\,\mathbf{x}_i\mathbf{x}_i^T$$

where $\sigma^2(\mathbf{w};\mathbf{x}_i) = \mathrm{K}_{\phi(Y_i),\phi(Y_i)} = \mathrm{Var}[\phi(Y_i)]$with $Y_i \sim p_{\mathbf{w}^T\mathbf{x}_i}$. So the Hessian of the minus log-likelihood is

$$\mathrm{H}_{L_n}(\mathbf{w}) = \sum_{i=1}^n \sigma^2(\mathbf{w};\mathbf{x}_i)\,\mathbf{x}_i\mathbf{x}_i^T$$

which is positive semidefinite.

Exercise: Show that $\mathrm{H}_{L_n}(\mathbf{w})$ is positive semidefinite. ⊲

As a result, the minus log-likelihood is convex and the maximum likelihood estimator $\hat{\mathbf{w}}_{\mathrm{MLE}}$ solves the equation

$$\sum_{i=1}^n \mathbf{x}_i\,\mu(\mathbf{w};\mathbf{x}_i) = \sum_{i=1}^n \mathbf{x}_i\,\phi(y_i).$$

We revisit linear and logistic regression next.

Example (Linear regression): Consider the case where p_θ is a univariate Gaussian with mean θ and fixed variance 1. That is,

$$\begin{aligned}
p_\theta(y) &= \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(y-\theta)^2}{2}\right)\\
&= \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}[y^2 - 2y\theta + \theta^2]\right)\\
&= \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)\exp\left(\theta y - \frac{\theta^2}{2}\right)\\
&= h(y)\exp(\theta\phi(y) - A(\theta)),
\end{aligned}$$

where $\phi(y) = y$ and $A(\theta) = \theta^2/2$. We now assume that $\theta = \mathbf{x}^T\mathbf{w}$ to obtain the corresponding generalized linear model.

Given data points $(\mathbf{x}_i, y_i)_{i=1}^n$, recall that the maximum likelihood estimator $\hat{\mathbf{w}}_{\mathrm{MLE}}$ solves the equation

$$\sum_{i=1}^n \mathbf{x}_i\,\mu(\mathbf{w};\mathbf{x}_i) = \sum_{i=1}^n \mathbf{x}_i\,\phi(y_i)$$

where $\mu(\mathbf{w};\mathbf{x}_i) = \mathbb{E}[\phi(Y_i)]$ with $Y_i \sim p_{\mathbf{x}_i^T\mathbf{w}}$. Here $\mathbb{E}[\phi(Y_i)] = \mathbb{E}[Y_i] = \mathbf{x}_i^T\mathbf{w}$. So the equation reduces to

$$\sum_{i=1}^n \mathbf{x}_i\mathbf{x}_i^T\mathbf{w} = \sum_{i=1}^n \mathbf{x}_i\,y_i.$$

You may not recognize this equation, but we have encountered it before in a different form. Let A be the matrix with row i equal to $\mathbf{x}_i$ and let $\mathbf{y}$ be the vector with i-th entry equal to y_i. Then

$$\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T = A^T A \qquad \text{and} \qquad \sum_{i=1}^{n} \mathbf{x}_i y_i = A^T \mathbf{y}$$

as can be checked entry by entry for instance. Therefore, the equation above is equivalent to $A^T A \mathbf{w} = A^T \mathbf{y}$ - the normal equations of linear regression.

To make sense of this finding, we look back at the minus log-likelihood

$$\begin{aligned} L_n(\mathbf{w}; \{(\mathbf{x}_i, y_i)_{i=1}^n\}) &= -\sum_{i=1}^{n} \log p_{\mathbf{x}_i^T \mathbf{w}}(y_i) \\ &= -\sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(y_i - \mathbf{x}_i^T \mathbf{w})^2}{2} \right) \right) \\ &= -\log(\sqrt{2\pi}) + \frac{1}{2} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \mathbf{w})^2. \end{aligned}$$

Observe that mimizing this expression over $\mathbf{w}$ is equivalent to solving the least-squares problem as the first term does not depend on $\mathbf{w}$ and the factor of $1/2$ does not affect the optimum.

While we have rederived the least squares problem from a probabilistic model, it should not be noted that the Gaussian assumption is not in fact required for linear regression to be efffective. Rather, it gives a different perspective on the same problem. ◁

Exercise: Assume instead that, for each i, p_{θ_i} is a univariate Gaussian with mean $\theta_i = \mathbf{x}_i^T \mathbf{w}$ and known variance σ_i^2. Show that the maximum likelihood estimator of $\mathbf{w}$ solves the weighted least squares problem, as defined in a previous assignment. ◁

Example (Logistic regression): Consider the case where p_θ is a Bernoulli distribution. That is, for $y \in \{0, 1\}$,

$$p_\theta(y) = h(y) \exp(\theta\, \phi(y) - A(\theta)),$$

where $h(y) = 1$, $\phi(y) = y$ and $A(\theta) = \log(1 + e^\theta)$. We now assume that $\theta = \mathbf{x}^T \mathbf{w}$ to obtain the corresponding generalized linear model. Given data points $(\mathbf{x}_i, y_i)_{i=1}^n$, the maximum likelihood estimator $\hat{\mathbf{w}}_{\text{MLE}}$ solves the equation

$$\sum_{i=1}^n \mathbf{x}_i\, \mu(\mathbf{w}; \mathbf{x}_i) = \sum_{i=1}^n \mathbf{x}_i \phi(y_i)$$

where $\mu(\mathbf{w}; \mathbf{x}_i) = \mathbb{E}[\phi(Y_i)]$ with $Y_i \sim p_{\mathbf{x}_i^T \mathbf{w}}$. Here, by our formula for the gradient of A,

$$\mathbb{E}[\phi(Y_i)] = \mathbb{E}[Y_i] = A'(\mathbf{x}_i^T \mathbf{w}) = \frac{e^{\mathbf{x}_i^T \mathbf{w}}}{1 + e^{\mathbf{x}_i^T \mathbf{w}}} = \sigma(\mathbf{x}_i^T \mathbf{w}),$$

where σ is the sigmoid function. So the equation reduces to

$$\sum_{i=1}^n \mathbf{x}_i \sigma(\mathbf{x}_i^T \mathbf{w}) = \sum_{i=1}^n \mathbf{x}_i\, y_i.$$

The equation in this case cannot be solved explicitly. Instead we can use gradient descent, or a variant, to minimize the minus log-likelihood directly. The lattter is

$$\begin{aligned}
L_n(\mathbf{w}; \{(\mathbf{x}_i, y_i)_{i=1}^n\}) &= -\sum_{i=1}^n \log p_{\mathbf{x}_i^T \mathbf{w}}(y_i) \\
&= -\sum_{i=1}^n \log\Big(\exp((\mathbf{x}_i^T \mathbf{w}) y_i - \log(1 + e^{\mathbf{x}_i^T \mathbf{w}}))\Big) \\
&= -\sum_{i=1}^n \left[(\mathbf{x}_i^T \mathbf{w}) y_i - \log(1 + e^{\mathbf{x}_i^T \mathbf{w}})\right] \\
&= -\sum_{i=1}^n \left[y_i \log(e^{\mathbf{x}_i^T \mathbf{w}}) - (y_i + (1 - y_i)) \log(1 + e^{\mathbf{x}_i^T \mathbf{w}})\right] \\
&= -\sum_{i=1}^n \left[y_i \log(\sigma(\mathbf{x}_i^T \mathbf{w})) + (1 - y_i) \log(1 - \sigma(\mathbf{x}_i^T \mathbf{w}))\right].
\end{aligned}$$

Minimizing $L_n(\mathbf{w}; \{(\mathbf{x}_i, y_i)_{i=1}^n\})$ is equivalent to logistic regression.

To use gradient descent, we compute

$$\begin{aligned}
\nabla_{\mathbf{w}} L_n(\mathbf{w}; \{(\mathbf{x}_i, y_i)_{i=1}^n\}) &= -\sum_{i=1}^n \mathbf{x}_i (\phi(y_i) - \mu(\mathbf{w}; \mathbf{x}_i)) \\
&= -\sum_{i=1}^n \mathbf{x}_i (y_i - \sigma(\mathbf{x}_i^T \mathbf{w})).
\end{aligned}$$

This expression is indeed consistent with what we previously derived for logistic regression. ◁

In practice, generalized linear models of moderate dimension are often fitted using the second-order method Iterative Reweighted Least Squares (IRLS), which generalizes the special case for logistic regression we encountered in a previous assignment.

Exercise: The exponential family form of the Poisson distribution with mean λ has sufficient statistic $\phi(y) = y$ and natural parameter $\theta = \log \lambda$. In Poisson regression, we assume that $p_\theta(y)$ is Poisson with $\theta = \mathbf{x}^T \mathbf{w}$. Compute the gradient and Hessian of the minus log-likelihood in this case. ◁

3.2 Example 2: Naive Bayes

The model-based justification for logistic regression in the previous section used a so-called discriminative approach (https://en.wikipedia.org/wiki/Discriminative_model), where the conditional distribution of the target y given the features $\mathbf{x}$ is specified - but not the full distribution of the data $(\mathbf{x}, y)$. Here we give an example of the generative approach (https://en.wikipedia.org/wiki/Generative_model), which models the full distribution. For a discussion of the benefits and drawbacks of each approach, see for example here (https://en.wikipedia.org/wiki/Discriminative_model#Contrast_with_generative_model).

The Naive Bayes model is a simple discrete model for supervised learning. It is useful for document classification for instance, and we will use that terminology here to be concrete. We assume that a document has a single topic C from a list $\mathcal{C} = \{1, \dots, K\}$ with probability distribution $\pi_k = \mathbb{P}[C = k]$. There is a vocabulary of size M and we record the presence or absence of a word m in the document with a Bernoulli variable X_m, where $p_{km} = \mathbb{P}[X_m = 1 | C = k]$. We denote by $\mathbf{X} = (X_1, \dots, X_M)$ the corresponding vector.

The conditional independence assumption comes next: we assume that, given a topic C, the word occurrences are independent. That is,

$$\begin{aligned}\mathbb{P}[\mathbf{X} = \mathbf{x} | C = k] &= \prod_{m=1}^{M} \mathbb{P}[X_m = x_m | C = k] \\ &= \prod_{m=1}^{M} p_{km}^{x_m} (1 - p_{km})^{1 - x_m}.\end{aligned}$$

Finally, the joint distribution is

$$\begin{aligned}\mathbb{P}[C = k, \mathbf{X} = \mathbf{x}] &= \mathbb{P}[\mathbf{X} = \mathbf{x} | C = k] \, \mathbb{P}[C = k] \\ &= \pi_k \prod_{m=1}^{M} p_{km}^{x_m} (1 - p_{km})^{1 - x_m}.\end{aligned}$$

Model fitting: Before using the model for prediction, one must first fit the model from training data $\{\mathbf{x}_i, c_i\}_{i=1}^n$. In this case, it means estimating the unknown parameters $\boldsymbol{\pi}$ and $\{\mathbf{p}_k\}_{k=1}^K$, where $\mathbf{p}_k = (p_{k1}, \ldots, p_{kM})$. For each k, m let

$$N_{km} = \sum_{i=1}^{n} \mathbf{1}_{\{c_i=k\}} x_{i,m}, \qquad N_k = \sum_{i=1}^{n} \mathbf{1}_{\{c_i=k\}}.$$

We use maximum likelihood estimation which, recall, entails finding the parameters that maximize the probability of observing the data

$$\mathcal{L}(\boldsymbol{\pi}, \{\mathbf{p}_k\}; \{\mathbf{x}_i, c_i\}) = \prod_{i=1}^{n} \pi_{c_i} \prod_{m=1}^{M} p_{c_i,m}^{x_{i,m}} (1 - p_{c_i,m})^{1-x_{i,m}}.$$

Here, as usual, we assume that the samples are independent and identically distributed. We take a logarithm to turn the products into sums and consider the negative log-likelihood

$$\begin{aligned} &L_n(\boldsymbol{\pi}, \{\mathbf{p}_k\}; \{\mathbf{x}_i, c_i\}) \\ &\quad = -\sum_{i=1}^{n} \log \pi_{c_i} - \sum_{i=1}^{n} \sum_{m=1}^{M} [x_{i,m} \log p_{c_i,m} + (1 - x_{i,m}) \log(1 - p_{c_i,m})] \\ &\quad = -\sum_{k=1}^{K} N_k \log \pi_k - \sum_{k=1}^{K} \sum_{m=1}^{M} [N_{km} \log p_{km} + (N_k - N_{km}) \log(1 - p_{km})]. \end{aligned}$$

The negative log-likelihood can be broken up naturally into several terms that depend on different sets of parameters -- and therefore can be optimized separately. First, there is a term that depends only on the π_k's

$$J_0(\boldsymbol{\pi}; \{\mathbf{x}_i, c_i\}) = -\sum_{k=1}^{K} N_k \log \pi_k.$$

The rest of the sum can be further split into KM terms, each depending only on p_{km} for a fixed k and m

$$J_{km}(p_{km}; \{\mathbf{x}_i, c_i\}) = -N_{km} \log p_{km} - (N_k - N_{km}) \log(1 - p_{km}).$$

So

$$L_n(\boldsymbol{\pi}, \{\mathbf{p}_k\}; \{\mathbf{x}_i, c_i\}) = J_0(\boldsymbol{\pi}; \{\mathbf{x}_i, c_i\}) + \sum_{k=1}^{K} \sum_{m=1}^{M} J_{km}(p_{km}; \{\mathbf{x}_i, c_i\}).$$

We minimize these terms separately. We assume that $N_k > 0$ for all k.

We use a special case of maximum likelihood estimation, which we previously worked out in an example, where we consider the space of all probability distributions over a finite set. The maximum likelihood estimator in that case is given by the empirical frequencies. Notice that minimizing $I_0(\boldsymbol{\pi}; \{\mathbf{x}_i, c_i\})$ is precisely of this form: we observe N_k samples from class k and we seek the maximum likelihood estimator of, π_k, the probability of observing k. Hence the solution is simply

$$\hat{\pi}_k = \frac{N_k}{N},$$

for all k. Similarly, for each k, m, I_{km} is of that form as well. Here the states correspond to word m being present or absent in a document of class k, and we observe N_{km} documents of type k where m is present. So the solution is

$$\hat{p}_{km} = \frac{N_{km}}{N_k}$$

for all k, m.

Prediction: To predict the class of a new document, it is natural to maximize over k the probability that $\{C = k\}$ given $\{\mathbf{X} = \mathbf{x}\}$. By Bayes' rule,

$$\begin{aligned}\mathbb{P}[C = k|\mathbf{X} = \mathbf{x}] &= \frac{\mathbb{P}[C = k, \mathbf{X} = \mathbf{x}]}{\mathbb{P}[\mathbf{X} = \mathbf{x}]} \\ &= \frac{\pi_k \prod_{m=1}^M p_{km}^{x_m}(1 - p_{km})^{1-x_m}}{\sum_{k'=1}^K \pi_{k'} \prod_{m=1}^M p_{k'm}^{x_m}(1 - p_{k'm})^{1-x_m}}.\end{aligned}$$

As the denominator does not in fact depend on k, maximizing $\mathbb{P}[C = c_k|\mathbf{X} = \mathbf{x}]$ boils down to maximizing the numerator $\pi_k \prod_{m=1}^M p_{km}^{x_m}(1 - p_{km})^{1-x_m}$, which is straighforward to compute. Since the parameters are unknown, we use $\hat{\pi}_k$ and $\hat{p}_{km}$ in place of π_k and p_{km}. As we did previously, we often take a negative logarithm, which has some numerical advantages, and we refer to it as the score

$$-\log\left(\pi_k \prod_{m=1}^M p_{km}^{x_m}(1 - p_{km})^{1-x_m}\right) = -\log \pi_k - \sum_{m=1}^M [x_m \log p_{km} + (1 - x_m) \log(1 - p_{km})].$$

While maximum likehood estimation has desirable theoretical properties (https://en.wikipedia.org/wiki/Maximum_likelihood_estimation#Properties), it does suffer from overfitting (https://towardsdatascience.com/parameter-inference-maximum-aposteriori-estimate-49f3cd98267a). If for instance a particular word does not occur in any training document, then the probability of observing a new document containing that word is 0 for any class and the maximization problem above is not well-defined.

One approach to deal with this is Laplace smoothing (https://en.wikipedia.org/wiki/Additive_smoothing)

$$\bar{\pi}_k = \frac{N_k + \alpha}{N + K\alpha}, \quad \bar{p}_{km} = \frac{N_{km} + \beta}{N_k + 2\beta}$$

where $\alpha, \beta > 0$, which can be justified using a Bayesian perspective.

NUMERICAL CORNER We implement the Naive Bayes model. We use Laplace smoothing (https://en.wikipedia.org/wiki/Additive_smoothing) to avoid overfitting issues.

We use a simple example from [Towards Data Science (https://towardsdatascience.com/all-about-naive-bayes-8e13cef044cf)](https://towardsdatascience.com/all-about-naive-bayes-8e13cef044cf):

> **Example:** let's say we have data on 1000 pieces of fruit. The fruit being a Banana, Orange or some other fruit and imagine we know 3 features of each fruit, whether it's long or not, sweet or not and yellow or not, as displayed in the table below.

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	**500**	**650**	**800**	**1000**

> [...] Which should provide enough evidence to predict the class of another fruit as it's introduced.

We encode the data into a table, where the rows are the classes and the columns are the features. The entries are the corresponding N_{km}'s. In addition we provide the vector N_k, which is the last column above, and the value N, which is the sum of the entries of N_k.

In [1]:
```
# Julia version: 1.5.1
ENV["JULIA_CUDA_SILENT"] = true # silences warning about GPUs

using Images, ImageMagick
using Flux, Flux.Data.MNIST
```

In [2]:
```
N_km = [400. 350. 450.; 0. 150. 300.; 100. 150. 50.]
N_k = [500., 300., 200.]
N = 1000;
```

In [3]:
```
function mmids_nb_fit_table(N_km, N_k, N; alpha=1., beta=1.)

    K, M = size(N_km)

    # MLE for pi_k's
    pi_k = (N_k.+alpha)/(N.+K*alpha)

    # MLE for p_km's
    p_km = (N_km.+beta)./(N_k.+2*beta)

    return pi_k, p_km

end
```

Out[3]:
```
mmids_nb_fit_table (generic function with 1 method)
```

We run it on our simple dataset.

```
In [4]: pi_k, p_km = mmids_nb_fit_table(N_km, N_k, N);
```

```
In [5]: pi_k
```

```
Out[5]: 3-element Array{Float64,1}:
         0.4995014955134596
         0.3000997008973081
         0.20039880358923232
```

```
In [6]: p_km
```

```
Out[6]: 3×3 Array{Float64,2}:
         0.798805    0.699203  0.898406
         0.00331126  0.5       0.996689
         0.5         0.747525  0.252475
```

Continuing on with our previous example:

> So let's say we're given the features of a piece of fruit and we need to predict the class. If we're told that the additional fruit is Long, Sweet and Yellow, we can classify it using the [prediction] formula and subbing in the values for each outcome, whether it's a Banana, an Orange or Other Fruit. The one with the highest probability (score) being the winner.

The next function computes the negative logarithm of $\pi_k \prod_{m=1}^{M} p_{km}^{x_m}(1-p_{km})^{1-x_m}$, that is, the score of k, and outputs a k achieving the minimum score.

In [7]:
```
function mmids_nb_predict(pi_k, p_km, x, label_set)

    K = length(pi_k)

    # Computing the score for each k
    score_k = zeros(Float64, K)
    for k=1:K

        score_k[k] += - log(pi_k[k])
        score_k[k] += - sum(x .* log.(p_km[k,:]) .+ (1 .- x).*log.(1 .-
p_km[k,:]))

    end

    # Computing the minimum
    (minscr, argmin) = findmin(score_k, dims=1)

    return label_set[argmin]

end
```

Out[7]:
```
mmids_nb_predict (generic function with 1 method)
```

We run it on our dataset with the additional fruit from the quote above.

In [8]:
```
label_set = ["Banana" "Orange" "Other"]
x = [1., 1., 1.];
```

In [9]:
```
mmids_nb_predict(pi_k, p_km, x, label_set)
```

Out[9]:
```
1-element Array{String,1}:
 "Banana"
```

3.3 Example 3: Mixtures of multivariate Bernoullis

We consider now a mixture version of the previous example. Let again $\mathcal{C} = \{1, \ldots, K\}$ be a collection of classes. Let C be a random variable taking values in $\mathcal{C}$ and, for $m = 1, \ldots, M$, let X_i take values in $\{0, 1\}$. Define $\pi_k = \mathbb{P}[C = c_k]$ and $p_{km} = \mathbb{P}[X_m = 1 | C = c_k]$ for $m = 1, \ldots, M$. We denote by $\mathbf{X} = (X_1, \ldots, X_M)$ the corresponding vector of X_i's and assume that the entries are conditionally independent given C.

However, we assume this time that C itself is *not observed*. So the resulting joint distribution is the mixture

$$\begin{aligned}\mathbb{P}[\mathbf{X} = \mathbf{x}] &= \sum_{k=1}^{K} \mathbb{P}[C = k, \mathbf{X} = \mathbf{x}] \\ &= \sum_{k=1}^{K} \mathbb{P}[\mathbf{X} = \mathbf{x} | C = k] \, \mathbb{P}[C = k] \\ &= \sum_{k=1}^{K} \pi_k \prod_{m=1}^{M} p_{km}^{x_m} (1 - p_{km})^{1 - x_m}.\end{aligned}$$

This type of model is useful in particular for clustering tasks, where the c_k's can be thought of as different clusters. Similarly to what we did in the previous section, our goal is to infer the parameters from samples and then predict the class of an old or new sample given its features. The main - substantial - difference is that the true labels of the samples are not observed. As we will see, that complicates the task considerably.

3.3.1 Model fitting

We first fit the model from training data $\{\mathbf{x}_i\}_{i=1}^n$. Recall that the corresponding class labels c_i's are not observed. In this type of model, they are referred to as hidden or latent variables and we will come back to their inference below.

We would like to use maximum likelihood estimation, that is, maximize the probability of observing the data

$$\mathcal{L}(\boldsymbol{\pi}, \{\mathbf{p}_k\}; \{\mathbf{x}_i\}) = \prod_{i=1}^{n} \left(\sum_{k=1}^{K} \pi_k \prod_{m=1}^{M} p_{k,m}^{x_{i,m}} (1 - p_{k,m})^{1 - x_{i,m}} \right).$$

As usual, we assume that the samples are independent and identically distributed. Consider the negative log-likelihood

$$L_n(\boldsymbol{\pi}, \{\mathbf{p}_k\}; \{\mathbf{x}_i\}) = -\sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \prod_{m=1}^{M} p_{k,m}^{x_{i,m}} (1 - p_{k,m})^{1 - x_{i,m}} \right).$$

Already, we see that things are potentially more difficult than they were in the supervised (or fully observed) case. The negative log-likelihood does not decompose into a sum of terms depending on different sets of parameters.

At this point, one could fall back on the field of optimization and use a gradient-based method to minimize the negative log-likelihood. Indeed that is an option, although note that one must be careful to account for the constrained nature of the problem (here the parameters sum to one). There is a vast array of constrained optimization techniques suited for this task.

Instead a more popular approach in this context, the EM algorithm, is based on the general principle of majorization-minimization, which we have encountered implicitly in the k-means algorithm and the convergence proof of gradient descent in the smooth case. We detail this important principle in the next subsection before returning to model fitting in mixtures.

3.3.2 Majorization-minimization

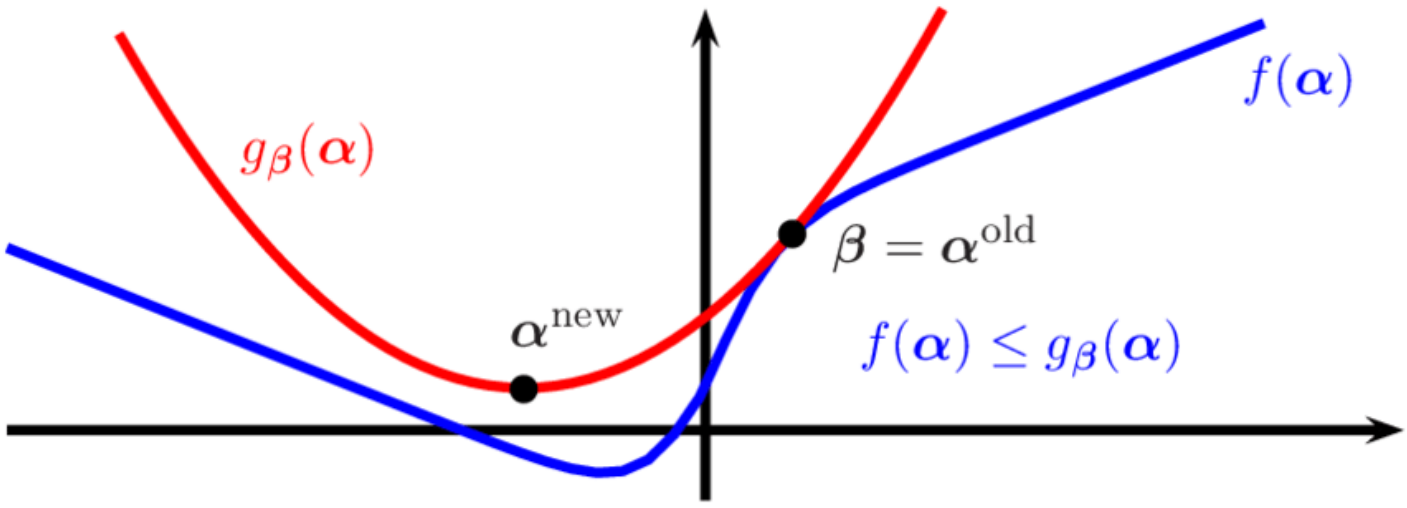
Here is a deceptively simple, yet powerful observation. Suppose we want to minimize a function $f : \mathbb{R}^d \to \mathbb{R}$. Finding a local minimum of f may not be easy. But imagine that for each $\mathbf{x} \in \mathbb{R}^d$ we have a surrogate function $U_{\mathbf{x}} : \mathbb{R}^d \to \mathbb{R}$ that (1) dominates f in the following sense

$$U_{\mathbf{x}}(\mathbf{z}) \geq f(\mathbf{z}), \quad \forall \mathbf{z} \in \mathbb{R}^d$$

and (2) equals f at $\mathbf{x}$

$$U_{\mathbf{x}}(\mathbf{x}) = f(\mathbf{x}).$$

We say that $U_{\mathbf{x}}$ majorizes f at $\mathbf{x}$. Then $U_{\mathbf{x}}$ can be used to make progress towards minimizing f, that is, find a point $\mathbf{x}'$ such that $f(\mathbf{x}') \leq f(\mathbf{x})$. If $U_{\mathbf{x}}$ is easier to minimize than f itself, say because an explicit minimum can be computed, then this observation proved in the next lemma naturally leads to an iterative algorithm.



(Source (https://www.researchgate.net/figure/Illustration-of-the-basic-majorization-minimization-principle-The-objective-function-f_fig3_268227141))

Lemma (Majorization-Minimization): Let $f : \mathbb{R}^d \to \mathbb{R}$ and suppose $U_{\mathbf{x}}$ majorizes f at $\mathbf{x}$. Let $\mathbf{x}'$ be a global minimum of $U_{\mathbf{x}}$. Then

$$f(\mathbf{x}') \leq f(\mathbf{x}).$$

Proof: Indeed

$$f(\mathbf{x}') \leq U_{\mathbf{x}}(\mathbf{x}') \leq U_{\mathbf{x}}(\mathbf{x}) = f(\mathbf{x}),$$

where the first inequality follows from the domination property of $U_{\mathbf{x}}$, the second inequality follows from the fact that $\mathbf{x}'$ is a global minimum of $U_{\mathbf{x}}$ and the equality follows from the fact that $U_{\mathbf{x}}$ equals f at $\mathbf{x}$. □

We have already encountered this idea.

Example (Minimizing a smooth function): Let $f : \mathbb{R}^d \to \mathbb{R}$ be L-smooth. By the *Quadratic Bound for Smooth Functions*, for all $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$ it holds that

$$f(\mathbf{z}) \leq U_{\mathbf{x}}(\mathbf{z}) := f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{z} - \mathbf{x}) + \frac{L}{2}\|\mathbf{z} - \mathbf{x}\|^2.$$

By showing that $U_{\mathbf{x}}$ is minimized at $\mathbf{z} = \mathbf{x} - (1/L)\nabla f(\mathbf{x})$, we previously obtained the descent guarantee

$$f(\mathbf{x} - (1/L)\nabla f(\mathbf{x})) \leq f(\mathbf{x}) - \frac{1}{2L}\|\nabla f(\mathbf{x})\|^2$$

for gradient descent, which played a central role in the analysis of its convergence. ◁

Example (k-means): Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be n vectors in $\mathbb{R}^d$. One way to formulate the k-means clustering problem is as the minimization of

$$f(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K) = \sum_{i=1}^{n} \min_{j \in [K]} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2$$

over the centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, where recall that $[K] = \{1, \ldots, K\}$. For fixed $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ and $\mathbf{m} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K)$, define

$$c_{\mathbf{m}}(i) \in \arg\min \left\{ \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \ : \ j \in [K] \right\}, \quad i = 1, \ldots, n$$

and

$$U_{\mathbf{m}}(\lambda_1, \ldots, \lambda_K) = \sum_{i=1}^{n} \|\mathbf{x}_i - \lambda_{c_{\mathbf{m}}(i)}\|^2$$

for $\lambda_1, \ldots, \lambda_K \in \mathbb{R}^d$. That is, we fix the optimal cluster assignments under $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ and then vary the centers.

We claim that $U_{\mathbf{m}}$ is majorizing f at $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$. Indeed

$$f(\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K) = \sum_{i=1}^n \min_{j \in [K]} \|\mathbf{x}_i - \boldsymbol{\lambda}_j\|^2 \leq \sum_{i=1}^n \|\mathbf{x}_i - \boldsymbol{\lambda}_{c_{\mathbf{m}}(i)}\|^2 = U_{\mathbf{m}}(\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K)$$

and

$$f(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K) = \sum_{i=1}^n \min_{j \in [K]} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 = \sum_{i=1}^n \|\mathbf{x}_i - \boldsymbol{\mu}_{c_{\mathbf{m}}(i)}\|^2 = U_{\mathbf{m}}(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K).$$

Moreover $U_{\mathbf{m}}(\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K)$ is easy to minimize. We showed previously that the optimal representatives are

$$\boldsymbol{\mu}'_j = \frac{1}{|C_j|} \sum_{i \in C_j} \mathbf{x}_i$$

where $C_j = \{i : c_{\mathbf{m}}(i) = j\}$.

The *Majorization-Minimization Lemma* implies that

$$f(\boldsymbol{\mu}'_1, \ldots, \boldsymbol{\mu}'_K) \leq f(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K).$$

This argument is equivalent to our previous analysis of the k-means algorithm. ◁

3.3.3 EM algorithm

The Expectation-Maximization (EM) algorithm (https://en.wikipedia.org/wiki/Expectation–maximization_algorithm) is an instantiation of the majorization-minimization principle that applies widely to parameter estimation of mixtures. Here we focus on the mixture of multivariate Bernoullis.

Here recall that the objective to be minimized is

$$L_n(\boldsymbol{\pi}, \{\mathbf{p}_k\}; \{\mathbf{x}_i\}) = -\sum_{i=1}^n \log\left(\sum_{k=1}^K \pi_k \prod_{m=1}^M p_{k,m}^{x_{i,m}} (1 - p_{k,m})^{1 - x_{i,m}}\right).$$

To simplify the notation and highlight the general idea, we let $\boldsymbol{\theta} = (\boldsymbol{\pi}, \{\mathbf{p}_k\})$, denote by Θ the set of allowed values for $\boldsymbol{\theta}$, and use $\mathbb{P}_{\boldsymbol{\theta}}$ to indicate that probabilities are computed under the parameters $\boldsymbol{\theta}$. We also return to the description of the model in terms of the unobserved latent variables $\{C_i\}$. That is, we write the negative log-likelihood as

$$\begin{aligned} L_n(\boldsymbol{\theta}) &= -\sum_{i=1}^n \log\left(\sum_{k=1}^K \mathbb{P}_{\boldsymbol{\theta}}[\mathbf{X}_i = \mathbf{x}_i | C_i = k] \, \mathbb{P}_{\boldsymbol{\theta}}[C_i = k]\right) \\ &= -\sum_{i=1}^n \log\left(\sum_{k=1}^K \mathbb{P}_{\boldsymbol{\theta}}[\mathbf{X}_i = \mathbf{x}_i, C_i = k]\right). \end{aligned}$$

To derive a majorizing function, we use the convexity of the negative logarithm. Indeed

$$\frac{\partial}{\partial z}[-\log z] = -\frac{1}{z} \quad \text{and} \quad \frac{\partial^2}{\partial^2 z}[-\log z] = \frac{1}{z^2} > 0, \quad \forall z > 0.$$

The first step of the construction is not obvious - it just works. For each $i = 1, \ldots, n$, we let $r_{k,i}^{\theta}$, $k = 1, \ldots, K$, be a stricly positive probability distribution on $[K]$. In other words, it defines a convex combination for every i. Then we use convexity to obtain the upper bound

$$\begin{aligned} L_n(\tilde{\boldsymbol{\theta}}) &= -\sum_{i=1}^{n} \log\left(\sum_{k=1}^{K} r_{k,i}^{\theta} \frac{\mathbb{P}_{\tilde{\boldsymbol{\theta}}}[\mathbf{X}_i = \mathbf{x}_i, C_i = k]}{r_{k,i}^{\theta}} \right) \\ &\leq -\sum_{i=1}^{n} \sum_{k=1}^{K} r_{k,i}^{\theta} \log\left(\frac{\mathbb{P}_{\tilde{\boldsymbol{\theta}}}[\mathbf{X}_i = \mathbf{x}_i, C_i = k]}{r_{k,i}^{\theta}} \right), \end{aligned}$$

which holds for any $\tilde{\boldsymbol{\theta}} = (\tilde{\boldsymbol{\pi}}, \{\tilde{\mathbf{p}}_k\}) \in \Theta$. We choose $r_{k,i}^{\theta} = \mathbb{P}_{\boldsymbol{\theta}}[C_i = k | \mathbf{X}_i = \mathbf{x}_i]$ (which for the time being we assume is strictly positive), denote the right-hand side of the inequality by $U_n(\tilde{\boldsymbol{\theta}}|\boldsymbol{\theta})$ (as a function of $\tilde{\boldsymbol{\theta}}$) and make two observations.

(1) *Dominating property*: For any $\tilde{\boldsymbol{\theta}} \in \Theta$, the inequality above implies immediately that $L_n(\tilde{\boldsymbol{\theta}}) \leq U_n(\tilde{\boldsymbol{\theta}}|\boldsymbol{\theta})$.

(2) *Equality at* $\boldsymbol{\theta}$: At $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}$,

$$\begin{aligned} U_n(\boldsymbol{\theta}|\boldsymbol{\theta}) &= -\sum_{i=1}^{n} \sum_{k=1}^{K} r_{k,i}^{\theta} \log\left(\frac{\mathbb{P}_{\boldsymbol{\theta}}[\mathbf{X}_i = \mathbf{x}_i, C_i = k]}{r_{k,i}^{\theta}} \right) \\ &= -\sum_{i=1}^{n} \sum_{k=1}^{K} r_{k,i}^{\theta} \log\left(\frac{\mathbb{P}_{\boldsymbol{\theta}}[C_i = k | \mathbf{X}_i = \mathbf{x}_i] \mathbb{P}_{\boldsymbol{\theta}}[\mathbf{X}_i = \mathbf{x}_i]}{r_{k,i}^{\theta}} \right) \\ &= -\sum_{i=1}^{n} \sum_{k=1}^{K} r_{k,i}^{\theta} \log \mathbb{P}_{\boldsymbol{\theta}}[\mathbf{X}_i = \mathbf{x}_i] \\ &= -\sum_{i=1}^{n} \log \mathbb{P}_{\boldsymbol{\theta}}[\mathbf{X}_i = \mathbf{x}_i] = L_n(\boldsymbol{\theta}). \end{aligned}$$

The two properties above show that $U_n(\tilde{\boldsymbol{\theta}}|\boldsymbol{\theta})$, as a function of $\tilde{\boldsymbol{\theta}}$, majorizes L_n at $\boldsymbol{\theta}$.

Lemma (EM Guarantee): Let $\boldsymbol{\theta}^*$ be a global minimizer of $U_n(\tilde{\boldsymbol{\theta}}|\boldsymbol{\theta})$ as a function of $\tilde{\boldsymbol{\theta}}$, provided it exists. Then

$$L_n(\boldsymbol{\theta}^*) \leq L_n(\boldsymbol{\theta}).$$

Proof: The result follows directly from the *Majorization-Minimization Lemma*. □

What have we gained from this? As we mentioned before, using the *Majorization-Minimization Lemma* makes sense if Q_n is easier to minimize than L_n itself. Let us see why that is the case here.

The function Q_n naturally decomposes into two terms

$$\begin{aligned} Q_n(\tilde{\theta}|\theta) &= -\sum_{i=1}^{n}\sum_{k=1}^{K} r_{k,i}^{\theta} \log\left(\frac{\mathbb{P}_{\tilde{\theta}}[\mathbf{X}_i = \mathbf{x}_i, C_i = k]}{r_{k,i}^{\theta}}\right) \\ &= -\sum_{i=1}^{n}\sum_{k=1}^{K} r_{k,i}^{\theta} \log \mathbb{P}_{\tilde{\theta}}[\mathbf{X}_i = \mathbf{x}_i, C_i = k] + \sum_{i=1}^{n}\sum_{k=1}^{K} r_{k,i}^{\theta} \log r_{k,i}^{\theta}. \end{aligned}$$

Because $r_{k,i}^{\theta}$ depends on θ *but not* $\tilde{\theta}$, the second term is irrelevant to the opimization with respect to $\tilde{\theta}$.

If θ is our current estimate of the parameters, then the quantity $r_{k,i}^{\theta} = \mathbb{P}_{\theta}[C_i = k|\mathbf{X}_i = \mathbf{x}_i]$ is our estimate of the probability that the sample $\mathbf{x}_i$ comes from cluster k. So the first term above

$$Q_n(\tilde{\theta}|\theta) = -\sum_{i=1}^{n}\sum_{k=1}^{K} r_{k,i}^{\theta} \log \mathbb{P}_{\tilde{\theta}}[\mathbf{X}_i = \mathbf{x}_i, C_i = k]$$

is the expected negative log-likelihood of the fully observed data under our estimate of the cluster assignments. By fully observed, we mean that it includes the cluster variables C_i. Of course, in reality, those variables are not observed, but we have estimated their probability distribution given the observed data $\{\mathbf{x}_i\}$, and we are taking an expectation with respect to that distribution.

The key observation is that minimizing $Q_n(\tilde{\theta}|\theta)$ over $\tilde{\theta}$ is a variant of fitting a Naive Bayes model. And there is a straightforward formula for that. In essence that is because, when the cluster assignments are observed, the negative log-likehood decomposes: it naturally breaks up into terms that depend on separate sets of parameters, each of which can be optimized with a closed-form expression. The same happens with Q_n.

E Step: Specifically,

$$\begin{aligned} Q_n(\tilde{\theta}|\theta) &= -\sum_{i=1}^{n}\sum_{k=1}^{K} r_{k,i}^{\theta} \log \mathbb{P}_{\tilde{\theta}}[\mathbf{X}_i = \mathbf{x}_i, C_i = k] \\ &= -\sum_{i=1}^{n}\sum_{k=1}^{K} r_{k,i}^{\theta} \log\left(\prod_{i=1}^{n} \tilde{\pi}_k \prod_{m=1}^{M} \tilde{p}_{k,m}^{x_{i,m}}(1-\tilde{p}_{k,m})^{1-x_{i,m}}\right) \\ &= -\sum_{k=1}^{K} \eta_k^{\theta} \log \tilde{\pi}_k - \sum_{k=1}^{K}\sum_{m=1}^{M} [\eta_{k,m}^{\theta} \log \tilde{p}_{k,m} + (\eta_k^{\theta} - \eta_{k,m}^{\theta}) \log(1-\tilde{p}_{k,m})], \end{aligned}$$

where we defined, for $k = 1, \ldots, K$,

$$\eta_{k,m}^{\theta} = \sum_{i=1}^{n} x_{i,m} r_{k,i}^{\theta} \quad \text{and} \quad \eta_k^{\theta} = \sum_{i=1}^{n} r_{k,i}^{\theta}.$$

We have previously computed $r^{\theta}_{k,i} = \mathbb{P}_{\theta}[C_i = k | \mathbf{X}_i = \mathbf{x}_i]$ for prediction under the Naive Bayes model. We showed that

$$r^{\theta}_{k,i} = \frac{\pi_k \prod_{m=1}^{M} p_{k,m}^{x_{i,m}} (1 - p_{k,m})^{1 - x_{i,m}}}{\sum_{k'=1}^{K} \pi_{k'} \prod_{m=1}^{M} p_{k',m}^{x_{i,m}} (1 - p_{k',m})^{1 - x_{i,m}}},$$

which in this context is referred to as the responsibility that cluster k takes for data point i.

M Step: Adapting our previous calculutions for fitting a Naive Bayes model, we get that $Q_n(\tilde{\theta} | \theta)$ is minimized at

$$\pi^*_k = \frac{\eta^{\theta}_k}{n} \quad \text{and} \quad p^*_{k,m} = \frac{\eta^{\theta}_{k,m}}{\eta^{\theta}_k} \quad \forall k \in [K], m \in [M].$$

We used the fact that

$$\begin{aligned}
\sum_{k=1}^{K} \eta^{\theta}_k &= \sum_{k=1}^{K} \sum_{i=1}^{n} r^{\theta}_{k,i} \\
&= \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{P}_{\theta}[C_i = k | \mathbf{X}_i = \mathbf{x}_i] \\
&= \sum_{i=1}^{n} 1 \\
&= n,
\end{aligned}$$

since the conditional probability $\mathbb{P}_{\theta}[C_i = k | \mathbf{X}_i = \mathbf{x}_i]$ adds up to one when summed over k.

To summarize, the EM algorithm works as follows in this case. Assume we have data points $\{\mathbf{x}_i\}_{i=1}^{n}$, that we have fixed K and that we have some initial parameter estimate $\boldsymbol{\theta}^0 = (\boldsymbol{\pi}^0, \{\mathbf{p}^0_k\}) \in \Theta$ with strictly positive π^0_k's and $p^0_{k,m}$'s. For $t = 0, 1, \ldots, T-1$ we compute for all $i \in [n]$, $k \in [K]$, and $m \in [M]$

$$r^t_{k,i} = \frac{\pi^t_k \prod_{m=1}^{M} (p^t_{k,m})^{x_{i,m}} (1 - p^t_{k,m})^{1 - x_{i,m}}}{\sum_{k'=1}^{K} \pi^t_{k'} \prod_{m=1}^{M} (p^t_{k',m})^{x_{i,m}} (1 - p^t_{k',m})^{1 - x_{i,m}}}, \qquad \text{(E Step)}$$

$$\eta^t_{k,m} = \sum_{i=1}^{n} x_{i,m} r^t_{k,i} \quad \text{and} \quad \eta^t_k = \sum_{i=1}^{n} r^t_{k,i},$$

and

$$\pi^{t+1}_k = \frac{\eta^t_k}{n} \quad \text{and} \quad p^{t+1}_{k,m} = \frac{\eta^t_{k,m}}{\eta^t_k}. \qquad \text{(M Step)}$$

Provided $\sum_{i=1}^{n} x_{i,m} > 0$ for all m, the $\eta^t_{k,m}$'s and η^t_k's remain positive for all t and the algorithm is well-defined. The *EM Guarantee* stipulates that the negative log-likelihood cannot deteriorate, although note that it does not guarantee convergence to a global minimum.

NUMERICAL CORNER We implement the EM algorithm for mixtures of multivariate Bernoullis. For this purpose, we adapt our previous Naive Bayes routines. We also allow for the possibility of using Laplace smoothing.

In [10]:
```
function responsibility(pi_k, p_km, x)

    K = length(pi_k)

    # Computing the score for each k
    score_k = zeros(Float64, K)
    for k=1:K

        score_k[k] += - log(pi_k[k])
        score_k[k] += - sum(x .* log.(p_km[k,:]) .+ (1 .- x).*log.(1 .-
p_km[k,:]))

    end

    # Computing responsibilities for each k
    r_k = exp.(-score_k)./(sum(exp.(-score_k)))

    return r_k

end
```

Out[10]:
```
responsibility (generic function with 1 method)
```

In [11]:
```
function update_parameters(eta_km, eta_k, eta, alpha, beta)

    K = length(eta_k)

    # MLE for pi_k's
    pi_k = (eta_k.+alpha)/(eta.+K*alpha)

    # MLE for p_km's
    p_km = (eta_km.+beta)./(eta_k.+2*beta)

    return pi_k, p_km

end
```

Out[11]:
```
update_parameters (generic function with 1 method)
```

We implement the E and M Step next.

In [12]:
```
function mmids_em_bern(X, K, pi_0, p_0; maxiters = 10, alpha=0., beta=0.
)

    (n,M) = size(X)
    pi_k = pi_0
    p_km = p_0

    for _ = 1:maxiters

        # E Step
        r_ki = zeros(K,n)
        for i=1:n

            r_ki[:,i] = responsibility(pi_k, p_km, X[i,:])

        end

        # M Step
        eta_km = zeros(K,M)
        eta_k = sum(r_ki, dims=2)
        eta = sum(eta_k)
        for k=1:K
           for m=1:M
                eta_km[k,m] = sum(X[:,m] .* r_ki[k,:])
            end
        end
        pi_k, p_km = update_parameters(eta_km, eta_k, eta, alpha, beta)

    end

    return pi_k, p_km

end
```

Out[12]:
```
mmids_em_bern (generic function with 1 method)
```

We test the algorithm on a very simple dataset.

In [13]:
```
X = [1. 1. 1.; 1. 1. 1.; 1. 1. 1.; 1. 0. 1.; 0. 1. 1.; 0. 0. 0.; 0. 0.
0.; 0. 0. 1.]
(n,M) = size(X)
K = 2
```

Out[13]:
```
2
```

In [14]:
```
pi_0 = ones(K)./K
p_0 = rand(K,M);
```

In [15]:
```
pi_k, p_km = mmids_em_bern(X, K, pi_0, p_0);
```

In [16]:
```
pi_k
```

Out[16]:
```
2×1 Array{Float64,2}:
 0.3358119615500351
 0.6641880384499649
```

In [17]:
```
p_km
```

Out[17]:
```
2×3 Array{Float64,2}:
 6.28624e-5  4.37026e-5  0.255536
 0.752767    0.752777    1.0
```

We compute the probability that the vector $(0, 0, 1)$ is in each cluster.

In [18]:
```
x_test = [0., 0., 1.]
responsibility(pi_k, p_km, x_test)
```

Out[18]:
```
2-element Array{Float64,1}:
 0.6788244085256729
 0.32117559147432717
```

To give a more involved example, we return to the MNIST dataset. There are two common ways to write a 2. Let's see if a mixture of multivariate Bernoullis can find them. We load the dataset and extract the images labelled 2.

In [19]:
```
imgs = MNIST.images()
labels = MNIST.labels()
length(labels)
```

Out[19]:
```
60000
```

In [20]:
```
i2 = findall(l -> l==2, labels)
imgs2 = imgs[i2]
labels2 = labels[i2];
```

Next, we transform the images into vectors and convert into black and white by rounding.

In [21]:
```
X = reduce(hcat, [reshape(round.(Float32.(imgs2[i])),:) for i = 1:length
(imgs2)])';
```

We can convert back as follows.

In [22]:
```
Gray.(reshape(X[1,:],(28,28)))
```

Out[22]:

In this example, the probabilities involved are very small and the responsibilities are close to 0 or 1. We use a variant, called hard EM, which replaces responsibilities with the one-hot encoding of the largest responsibility.

In [23]:
```
function hard_responsibility(pi_k, p_km, x)

    K = length(pi_k)

    # Computing the score for each k
    score_k = zeros(Float64, K)
    for k=1:K

        score_k[k] += - log(pi_k[k])
        score_k[k] += - sum(x .* log.(p_km[k,:]) .+ (1 .- x).*log.(1 .-
p_km[k,:]))

    end

    # Computing responsibilities for each k
    (minscr, argmin) = findmin(score_k, dims=1)
    r_k = zeros(K)
    r_k[argmin[1]] = 1

    return r_k

end
```

Out[23]:
```
hard_responsibility (generic function with 1 method)
```

In [24]:
```
function mmids_hard_em_bern(X, K, pi_0, p_0; maxiters = 10, alpha=0., be
ta=0.)

    (n,M) = size(X)
    pi_k = pi_0
    p_km = p_0

    for _ = 1:maxiters

        # E Step
        r_ki = zeros(K,n)
        for i=1:n

            r_ki[:,i] = hard_responsibility(pi_k, p_km, X[i,:])

        end

        # M Step
        eta_km = zeros(K,M)
        eta_k = sum(r_ki, dims=2)
        eta = sum(eta_k)
        for k=1:K
           for m=1:M
                eta_km[k,m] = sum(X[:,m] .* r_ki[k,:])
            end
        end
        pi_k, p_km = update_parameters(eta_km, eta_k, eta, alpha, beta)

    end

    return pi_k, p_km

end
```

Out[24]:
```
mmids_hard_em_bern (generic function with 1 method)
```

We run the algorithm with 2 clusters. You may have to run it a few times to get a meaningful clustering.

In [25]:
```
K = 2
(n,M) = size(X)
pi_0 = ones(K)./K
p_0 = rand(K,M);
```

In [26]:
```
pi_k, p_km = mmids_hard_em_bern(X, K, pi_0, p_0, maxiters=10, alpha=1.,
beta=1.);
```

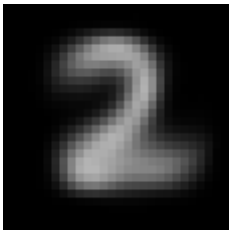
In [27]:
```
pi_k
```

Out[27]:
```
2×1 Array{Float64,2}:
 0.5714765100671141
 0.4285234899328859
```

We vizualize the two uncovered clusters by rendering their means as an image.

In [28]:
```
Gray.(reshape(p_km[1,:],(28,28)))
```

Out[28]:



In [29]:
```
Gray.(reshape(p_km[2,:],(28,28)))
```

Out[29]: