Machine Learning

Xuzhe Xia

January 12, 2023

Contents

Chapter 1

Linear/Nonlinear Regression

1.0 Notation

- X is an $n \times d$ matrix, where n the number of sample, and d is the number of feature.
- x_i is a $d \times 1$ vector, represent the i-th sample.
- $\hat{y}$ is the prediction, which is an $n \times 1$ vector.
- y is the true value.
- $r_i = \hat{y}_i - y_i$ is the residual.

1.1 One feature without y-intercept (d=1)

The model:

$$\hat{y}_i = wx_i \tag{1.1}$$

Linear Least Squares Objective Function:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(wx_i - y_i)^2 \tag{1.2}$$

> **Remark.** $cf(w) + d$ has the same set of minimizer as (1.2), where c, d are constant.

Least Squares Solution Derivation:

$$f(w) = \frac{w^2}{2}\sum_{i=1}^{n}x_i^2 - w\sum_{i=1}^{n}x_iy_i + \frac{1}{2}\sum_{i=1}^{n}y_i^2 \tag{1.3}$$

$$\left(\text{Let } a = \sum_{i=1}^{n}x_i^2; \quad b = \sum_{i=1}^{n}x_iy_i; \quad c = \sum_{i=1}^{n}y_i^2\right) \tag{1.4}$$

$$f(w) = \frac{w^2}{2}a - wb + c \tag{1.5}$$

Take derivative:

$$f'(w) = wa - b \tag{1.6}$$

Set $f'(w) = 0$:

Formula 1 (Least Squares Solution - One feature).

$$w = \frac{b}{a} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2} \tag{1.7}$$

1.2 Least square in d-dimension

Modify equation (1.1) and (1.2) to d-dimension:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + \cdots + w_d x_{id} = \sum_{j=1}^{d} w_j x_{ij} = w^T x_i \tag{1.8}$$

$$f(w) = f(w_1, w_2, \cdots, w_d) = \frac{1}{2} \sum_{i=1}^{n} (w^T x_i - y_i)^2 = \frac{1}{2} \|Xw - y\|^2 \tag{1.9}$$

Least Squares Solution Derivation:

$$f(w) = \frac{1}{2} w^T X^T X w - w^T X^T y + \frac{1}{2} y^T y \tag{1.10}$$

$$\left(\text{Let } A = X^T X; \qquad b = X^T y; \qquad c = \frac{1}{2} y^T y \right) \tag{1.11}$$

$$f(w) = \frac{1}{2} w^T A w - w^T b + c \tag{1.12}$$

Take gradient:

$$\nabla f(w) = Aw - b \tag{1.13}$$

Set $\nabla f(w) = 0$:

Formula 2 (Normal equation).

$$X^T X w = X^T y \tag{1.14}$$

Linear solve to get w.

1.3 y-intercept and Nonlinear regression with one feature

Idea: Form a new matrix Z by adding column to X (Polynomial Transformations).

The model:

$$\hat{y_i} = w_0 + w_1 x_i + w_2 x_i^2 + \cdots + w_p x_i^p \tag{1.15}$$

Let

$$Z = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^p \\ 1 & x_2 & x_2^2 & \cdots & x_2^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^p \end{bmatrix} \text{ and } v = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} \tag{1.16}$$

Linear Least Squares Objective Function:

$$f(w) = \frac{1}{2} \|Zv - y\|^2 \tag{1.17}$$

Apply equation (1.14) to find solution.

> **Remark.** Polynomials are not the only possible transformation, we can also use Exponential, logarithms, trigonometric functions, and so on.

1.4 Computation time

Normal equations find w with $\nabla f(w) = 0$ in $O(nd^2 + d^3)$ time.
This leads us to iterative method: Gradient Descent.

Chapter 2

Convex Functions

Some useful tricks for showing a function is convex:

- 1-variable, twice-differentiable function is convex iff $f''(w) \geq 0$ for all w.
- A convex function multiplied by non-negative constant is convex.
- Norms and squared norms are convex.
- The sum of convex functions is a convex function.
- The max of convex functions is a convex function.
- Composition of a convex function and a linear function is convex.

> **Remark.** Not true that composition of convex with convex is convex:

Chapter 3

Gradient Descent

Goal: find zero of $\nabla f(w) = X^T X w - X^T y$.

3.1 Gradient Descent

Algorithm 1 (Gradient Descent).

$w \leftarrow$ Random guess
while $\|\nabla f(w)\| > \varepsilon$ **do**
    $w \leftarrow w - \alpha \nabla f(w)$
return w

Remark. Time complexity: $O(ndt)$.

Compare to least square, gradient descent can be faster when d is very large.

If step size α is too large gradient descent may not converge.
If step size α is too small gradient descent may be too slow.

Chapter 4

Robust Regression

Motivation. Least squares are very sensitive to outliers.

4.1 Smooth Approximations to the L1-Norm

Goal: Making our model less sensitive to the outlier.

Problem with using L1-norm: non-differentiable. To solve the problem, we use:

Formula 3 (Huber loss).

$$f(w) = \sum_{i=1}^{n} h(w^T x_i - y_i) \tag{4.1}$$

$$\left(\text{Let } w^T x_i - y_i = r_i\right) \tag{4.2}$$

$$h(r_i) = \begin{cases} \frac{1}{2} r_i^2 & \text{for } |r_i| \leq \varepsilon \\ \varepsilon(|r_i| - \frac{1}{2}\varepsilon) & \text{otherwise} \end{cases} \tag{4.3}$$

4.2 "Brittle" Regression

Goal: Making our model **more** sensitive to the outlier.

Solution: Use infinity-norm:

$$f(w) = \|Xw - y\|_\infty \tag{4.4}$$

Problem with using infinity-norm: also non-differentiable. To solve the problem, we use log-sum-exp function defined as follow:

$$\max_i \{z_i\} \approx \ln(\sum_i \exp(z_i)) \tag{4.5}$$

Observe. The largest element in $\{z_i\}$ is magnified exponentially, thus summing does not affect too much on the value of $\exp(z_i)$, and taking ln we could approximate $\max_i\{z_i\}$.

Chapter 4

Feature Selection

Motivation. Find the features (columns) of X that are important for predicting y.

4.1 Basic approaches

4.1.1 "Association" Approach

Idea. For each feature j, compute correlation between feature values x^j and y.

Problem. The problem of this approach is that it ignores **variable interactions**:

- Includes irrelevant variables:
- Excludes relevant variables:

4.1.2 "Regression Weight" Approach

Idea.

- Fit regression weights w based on all features (maybe with least squares).
- Take all features j where weight $|w_j|$ is greater than a threshold.

Problem. This approach has major problems with collinearity:

- If the variable A always equals the variable B, it could say that A is relevant but B is not (when in fact B is relevent).
- If you have two copies of an irrelevant feature, it could take both irrelevant copies.

4.2 Search and Score Methods

Idea.

1. Define score function $f(S)$ that measures quality of a set of features S.
2. Now search for the variables S with the best score.

Problem. The large number of sets of variables:

- If we have d variables, there are 2^d sets of variables.
- Optimization bias is high: we're optimizing over $2d$ models.

4.2.1 Validation error as score

We use validation error as the score to determine which subset of the features to use. So the method becomes "Find the set of features that gives the lowest validation error."

Why not using training error? Because training error goes down as you add features, so will select all features.

4.2.2 "Number of Features" Penalties

$$\text{score}(S) = \frac{1}{2}\sum_{i=1}^{n}(w_s^T x_{iS} - y_i)^2 + \lambda \text{size}(S)$$

Note.

- We're using x_{iS} as the features S of example x_i
- We minimize squared error plus a penalty on number of features.

4.2.3 L0-Norm

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \lambda \|w\|_0$$

4.2.4 Forward Selection

It is a greedy search procedure to search for the best S:

1. Compute score if we use no features.
2. Try adding each feature from $\{x_1, \cdots, x_d\}$and computing score for each.
3. Add the feature x_i with the best score.
4. Try $\{x_i, x_1\}, \cdots, \{x_i, x_d\}$ and computing the score of each variable with x_i.
5. Add the feature x_j with the best score combined with x_i.
6. keep going...

Remark.

- The runtime of forward selection: we fit $O(d^2)$ models out of 2^d possible models with different features.
- Not guaranteed to find the best set, but fitting fewer models reduces many problems.

Chapter 4

Regularization

Motivation. Complex models tend to overfit, but usually the mapping from x_i to y_i is complex. So we need regularization when we need a complex model.

4.1 L2-Regularization

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^t x_i - y_i)^2 + \frac{\lambda}{2}\sum_{j=1}^{d} w_j^2 = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2$$

Intuition.

- Large slopes w_j tend to lead to overfitting.
- Regularization parameter $\lambda > 0$ controls "strength" of regularization, in another word, large λ puts large penalty on slopes.

4.1.1 Fundamental trade-off

- Regularization increases training error.
- Regularization decreases approximation error.

4.1.2 Solving L2-Regularized Least Squares Problem

$$\nabla f(w) = X^T X w - X^T y + \lambda w \tag{4.1}$$

$$w = (X^T X + \lambda I)^{-1} X^T y \tag{4.2}$$

> Remark. $X^T X + \lambda I$ is always invertible.

Gradient Descent for L2-Regularized Least Squares:

$$w^{t+1} = w^t - \alpha^t \left[X^T(Xw^t - y) + \lambda w^t\right] = w^t - \alpha^t \left[\nabla f(w^t)\right] \tag{4.3}$$

Remark. Number of iterations decrease as increases

4.2 L1-Regularization ("LASSO" regularization)

$$f(w) = \frac{1}{2} \|Xw - y\|^2 + \lambda \|w\|_1 \tag{4.4}$$

4.3 L2-Regularization vs L1-Regularization

L2-Regularization:	L1-Regularization:
Insensitive to changes in data.	Insensitive to changes in data.
Decreased variance: Lower test error.	– Decreased variance: Lower test error.
Closed-form solution.	Requires iterative solver.
Solution is unique.	Solution is not unique.
All w_j tend to be non-zero.	Many w_j tend to be zero.
Can learn with linear number of irrelevant features.	Can learn with exponential number of irrelevant feature

Chapter 5

Standardizing features

When the unit of features does not matter: Decision trees, Naive Bayes, and Least squares.
When the unit of features does matter: k-nearest neighbor, regularized least squares.

5.1 Standardize continuous features

Idea. For each feature:

- Compute mean and standard deviation:

$$\mu_j = \frac{1}{n}\sum_{i+1}^{n} x_{ij} \qquad \sigma_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij}-\mu_j)^2} \tag{5.1}$$

- Subtract mean and divide by standard deviation ("z-score"). (replace x_{ij} with $\frac{x_{ij}-\mu_j}{\sigma_j}$)

5.2 Standardize test data

Use mean and standard deviation of training data to standardize test data but do **not** use mean and standard deviation of test data.

5.3 Standardize the targets y_i

Replace y_i with $\frac{y_i-\mu_y}{\sigma_y}$

Chapter 4

Latent-factor Model

PCA is a latent-factor model, it learns the basis from the data. K-means is another example of (awful) latent-factor model.

Notation.

- Z is an $n \times k$ matrix.
- W is a $k \times d$ matrix.

4.1 PCA Computation

4.1.1 Alternating Minimization

4.1.2 SGD

4.2 Recommender Systems

Chapter 4

Neural Network

4.1 Neural Network with on hidden layer

As a function, single layer Neural Network can be expressed as follows:

$$\hat{y}_i = v^T h(Wx_i)$$

where it consists of three parts:

1. Linear transformation $z_i = Wx_i$ (like doing PCA).
   - W is a $k \times d$ parameter matrix.
   - x_i is the input vector.
2. Non-linear transformation h is often sigmoid applied element-wise.
   - (without a non-linear transformation $\hat{y}_i = v^T(Wx_i)$ is simply a linear model.)
   - (Using sigmoid allows us to differentiate.)
3. Second linear transformation $v^T h(z_i)$ gives final value.
   - v is a $k \times 1$ vector doing linear combination.

> **Remark** (Time Complexity). The cost of computing $\hat{y}_i$ is $O(kd)$

Intuition. I understand this as follows: given input, we use W to obtain the hidden features like in PCA, then applying h (sigmoid) to choose either we have this hidden feature or not. (Think of the case that we are doing number recognition, the hidden features are "parts" of digits. (WANT a figure here.))

4.1.1 Adding Bias Variables

Recall fitting linear models with a bias variable (so $\hat{y}_i \neq 0$ when $x_i = 0$):

$$\hat{y}_i = \sum_{j=1}^{d} w_j x_{ij} + \beta$$

In neural networks we often include biases on each z_{ic}:

$$\hat{y}_i = \sum_{c=1}^{k} v_c h(w_c^T x_i + \beta_c)$$

(We could implement this by adding a column of ones to X)
* We often also want a bias on the output:

$$\hat{y}_i = \sum_{c=1}^{k} v_c h(w_c^T x_i + \beta_c) + \beta$$

(For sigmoid h, we could implement by fixing one row of W to be 0, since $h(0)$ is constant which provides us a row of ones.)

4.1.2 Regression and Binary Classification

We have the single layer Neural Network function, now we can either form a Regression problem or binary Classification problem:

Example. For Regression problem, our prediction (ignore biases) is:

$$\hat{y}_i = v^T h(W x_i)$$

And we might train to minimize the squared residual:

$$f(W, v) = \frac{1}{2} \sum_{i=1}^{n} (\tilde{y_i} - y_i)^2 = \frac{1}{2} \sum_{i=1}^{n} (v^T h(W x_i) - y_i)^2$$

Example. For Binary Classification, our prediction (ignore biases) is:

$$\text{sign}(v^T h(W x_i)) \quad \text{or} \quad \mathbb{P}(y_i | W, v, x_i) = \frac{1}{1 + \exp(-y_i v^T h(W x_i))}$$

And we might train to minimize the logistic loss:

$$f(W, v) = \sum_{i=1}^{n} (\log(1 + \exp(-y_i \tilde{y_i})))$$

4.1.3 Neural Network for multi-class Classification

Hidden layer is connected to multiple output units:

$$\hat{y}_1 = v_1^T h(Wx)$$
$$\hat{y}_2 = v_2^T h(Wx)$$
$$\hat{y}_3 = v_3^T h(Wx)$$

So, now we have a matrix of parameters:

$$V = \begin{bmatrix} \text{---} & v_1^T & \text{---} \\ & \vdots & \\ \text{---} & v_k^T & \text{---} \end{bmatrix}$$

which is a $k' \times k$ matrix, where k' is the number of classes.
We can also convert it into probabilities for each class using **softmax** of the $\hat{y}_c$ values.

$$\mathbb{P}(y_i = c | x_i, W, V) = \frac{\exp(\hat{y}_c)}{\sum_{c'=1}^{k'} \exp(\hat{y}_{c'})}$$

4.1.4 Training Neural Network

Recall that for binary Classification, the NLL under the sigmoid likelihood is:

$$f(W, v) = \sum_{i=1}^{n} (\log(1 + \exp(-y_i v^T h(W x_i))))$$

> **Remark.**
>
> - With W fixed this is convex, but with both W and v as variables it is non-convex.
> - And finding the global optimum is NP-hard in general.

Therefore, we nearly always train with variations on SGD.

$$W^{k+1} = W^k - \alpha^k \nabla_W f_{i_k}(W^k, v^k)$$
$$v^{k+1} = v^k - \alpha^k \nabla_v f_{i_k}(W^k, v^k)$$

where i_k is a training example chosen uniformly at random.

4.1.5 "skip" connections

Consider a neural network with one hidden layer and connections from input to output layer:

$$\hat{y}_i = w^T x_i + v^T h(W x_i)$$

4.2 Deep Neural Network

Deep learning models have more than one hidden layer.
As a function, Deep Neural Network can be expressed as follows:

$$\hat{y}_i = v^T (\bigcirc_{l=1}^{m} h(W^{(l)} x_i))$$

where $W^{(l)}$ are the parameter matrix in each layer, and can be different sizes.

Remark (Time Complexity). The cost of prediction, which is called "forward propagation":

- Cost of the matrix multiplies: $O(k^1 d + k^2 k^1 + k^3 k^2 + k^4 k^3)$.
- Cost of the non-linear transforms is $O(k^1 + k^2 + k^3 + k^4)$, so does not change cost.

Intuition. The intuition behind using deep neural network using digit example:

1. "Hierarchies of Parts" intuition: Each "neuron" might recognize a "part" of a digit, the "deeper" neurons might recognize combinations of parts.

4.2.1 Rectified Linear Units

Vanishing Gradients: Here is the problem, when we are using the sigmoid function as our activation function, the gradient is nearly zero away from the origin. The problem gets worse when you take the sigmoid of a sigmoid, so in deep networks, many gradients can be nearly zero everywhere, and numerically they will be set to 0, so SGD does not move.
Solution 1: Modern networks often replace sigmoid with perceptron loss (ReLU):

$$h(z_{ic}) = \max\{0, z_{ic}\}$$

Solution 2: Skip connections can also reduce vanishing gradient problem: Makes "shortcuts" between layers (so fewer transformations).

4.2.2 ResNet

An example of Skip Connections Deep Learning is Residual networks.
An ResNet "block" is defined as follows:

$$a^{l+2} = h(a^l + W^{l+1} h(W^l a^l))$$

4.2.3 Learning in Deep Neural Networks

The usual training procedure is stochastic gradient descent (SGD), but Deep networks are highly non-convex and notoriously difficult to tune. However, SGD will usually find the local minimum due to the over-parametrization (since deep neural Network has lots of parameters, so it is easy to over-parametrize). The problem need to be considered is that it is hard to get SG to close to a local minimum in reasonable time.

4.3 Automatic Differentiation (AD)

In the deep Neural Network, the objective function is large, so we want a way to avoid calculate gradient.

4.3.1 Automatic Differentiation – Single Input+Output

Example.

1. Our function written as a set of compositions:

$$f_5(f_4(f_3(f_2(f_1(x)))))$$

2. The derivative written using the chain rule::

$$f'(x) = f'_5(f_4(f_3(f_2(f_1(x))))) * f'_4(f_3(f_2(f_1(x)))) * f'_3(f_2(f_1(x))) * f'_2(f_1(x))f'_1(x)$$

Notice that this leads to repeated calculations. For example, we use f1(x) four different times. So We can use dynamic programming to avoid redundant calculations:

1. First, the "forward pass" will compute and store the expressions.

$$\alpha_1 = f_1(x), \alpha_2 = f_2(\alpha_1), \alpha_3 = f_3(\alpha_2), \alpha_4 = f_4(\alpha_3), \alpha_5 = f_5(\alpha_4) = f(x)$$

2. Next, the "backward pass" uses stored α_k values and f'_i functions.

$$\beta_5 = 1 * f'_5(\alpha_4), \beta_4 = \beta_5 * f'_4(\alpha_3), \beta_3 = \beta_4 * f'_3(\alpha_2), \beta_2 = \beta_3 * f'_2(\alpha_1), \beta_1 = \beta_2 * f'_1(x) = f'(x)$$

This is a generic method to make code computing $f'(x)$ for same cost as $f(x)$.

Chapter 4

Naive Bayes

Naive Bayes is a type of generative model. That is, when trying to predict the probability of an example x belonging to class t, the model first finds the conditional probability $p(x \mid t)$. The model then make the inference on $p(t \mid x)$ using Bayes rule as follows:

$$\begin{aligned} p(t \mid x) &= \frac{p(x \mid t)p(t)}{p(x)} && \text{(by Bayes rule)} \\ &= \frac{p(x \mid t)p(t)}{\sum_c p(x \mid c)p(c)} && \text{(by marginalizing conditional probability)} \end{aligned}$$

If the example x is a list of m features, Naive Bayes makes the simplifying assumption that $p(x \mid t) = \prod_i^m p(x_i \mid t)$. That is to say, the features x_i's are conditionally independent given the class t.

In a typical scenario, we observe a dataset of n examples $x^{(1)}, \ldots, x^{(n)}$, and each example $x^{(i)}$ has m binary features $x_1^{(i)}, \ldots, x_m^{(i)}$. Each example $x^{(i)}$ belongs to one of K classes $c_1, \ldots, c_K$, thus we denote the class label of example $x^{(i)}$ as a one-hot vector $c^{(i)}$ where $x^{(i)}$ belongs to class j is and only if $c_j^{(i)} = 1$.

We specify parameters θ_{ij} as the probability that feature x_i is true given that example belongs to class c_j, i.e., $\theta_{ij} = p(x_i = 1 \mid c_j)$. Specify π_j as the probability that an example belongs to class c_j, i.e., $\pi_j = p(c_j)$. We can estimate the θ's and π's by finding the maximum likelihood estimators (MLEs) for them that maximize the log-likelihood of observing the dataset. The log-likelihood can be decomposed as follows.

$$
\begin{aligned}
l(\theta, \pi) &= \log \prod_i^n p(c^{(i)} \mid x^{(i)}) \\
&= \sum_i^n \log p(c^{(i)} \mid x^{(i)}) \\
&\propto \sum_i^n \Big(\log p(x^{(i)} \mid c^{(i)}) + \log p(c^{(i)}) \Big) && \text{(by Bayes rule)} \\
&= \sum_i^n \log p(x^{(i)} \mid c^{(i)}) + \sum_i^n \log p(c^{(i)}) \\
&= \sum_i^n \log \prod_j^m p(x_j^{(i)} \mid c^{(i)}) + \sum_i^n \log p(c^{(i)}) && \text{(by Naive Bayes assumption)} \\
&= \sum_i^n \sum_j^m \log p(x_j^{(i)} \mid c^{(i)}) + \sum_i^n \log p(c^{(i)})
\end{aligned}
$$

The first term is the log-likelihood for the features, the second term is the log-likelihood for the labels. We will maximize $l(\theta, \pi)$ by maximizing each term separately.

First, expanding the terms, we have

$$
\begin{aligned}
p(c^{(i)}) &= \prod_k^K \pi_k^{c_k^{(i)}} \\
p(x_j^{(i)} \mid c^{(i)}) &= \prod_k^K \left(\theta_{jk}^{x_j^{(i)}} (1 - \theta_{jk})^{1 - x_j^{(i)}} \right)^{c_k^{(i)}}
\end{aligned}
$$

Therefore, we can write the log-likelihood for the labels as

$$
\begin{aligned}
\sum_i^n \log p(c^{(i)}) &= \sum_i^n \log \prod_k^K \pi_k^{c_k^{(i)}} \\
&= \sum_i^n \sum_k^K c_k^{(i)} \log \pi_k
\end{aligned}
$$

To find the MLE of π_j, $1 \le j \le K$, we first find the MLE of π_j for $1 \le j \le K-1$. In the end, we will show the MLE of every π_j is $\frac{s_j}{n}$, where $s_j = \sum_{i=1}^n c_j^{(i)}$.

We can rewrite the above log-likelihood expression and differentiate it with respect to π_j and set it to 0. We have

$$
\begin{aligned}
\frac{\partial}{\partial \pi_j} \sum_i^n \sum_k^K c_k^{(i)} \log \pi_k &= \frac{\partial}{\partial \pi_j} \sum_i^n \left(\left(\sum_k^{K-1} c_k^{(i)} \log \pi_k \right) + c_K^{(i)} \log \left(1 - \sum_{k'}^{K-1} \pi_{k'} \right) \right) \\
&= \sum_i^n \left(\frac{c_j^{(i)}}{\pi_j} - \frac{c_K^{(i)}}{1 - \sum_{k'=0}^{K-1} \pi_{k'}} \right) \\
&= \frac{s_j}{\pi_j} - \frac{s_K}{\pi_K} = 0 \Longrightarrow \frac{s_j}{\pi_j} = \frac{s_K}{\pi_K} \Longrightarrow \frac{\pi_j}{\pi_K} = \frac{s_j}{s_K} && (s_k = \textstyle\sum_{i=1}^n c_k^{(i)}) \ (*)
\end{aligned}
$$

Since $\sum_k^K \pi_k = 1$ and $\sum_k^K s_k = n$, by (*) we have

$$\sum_k^K \frac{\pi_k}{\pi_K} = \frac{1}{\pi_K} = \sum_k^K \frac{s_k}{s_K} = \frac{n}{s_K} \Longrightarrow \frac{1}{\pi_K} = \frac{n}{s_K} \Longrightarrow \pi_K = \frac{s_K}{n}$$

Then, by (*), for each $1 \leq j \leq K-1$ we have

$$\frac{\pi_j}{\pi_K} = \frac{\pi_j n}{s_K} = \frac{s_j}{s_K} \Longrightarrow \pi_j = \frac{s_j}{n}$$

Therefore, for each $1 \leq j \leq K$ the MLE for $\pi_j = \frac{s_j}{n}$.

It can be similarly shown that the MLE of θ_{jk} is $\frac{\sum_{i=1}^n c_k^{(i)} x_j^{(i)}}{\sum_{i=1}^n c_k^{(i)}}$. It will be left as an exercise for the reader to show this.

Having found the MLEs of the θ's and π's, we can use them for inference given a new x^* and classify which class it belongs to (using Bayes rule). That is, we can find the probability that x^* belongs to class c_j, i.e., $p(t = c_j|x^*)$ directly by using the formula

$$p(t = c_j \mid x^*) = \frac{p(x^* \mid t = c_j)p(t = c_j)}{\sum_c p(x^* \mid c)p(c)} \propto p(x^* \mid t = c_j)p(t = c_j) = \prod_i^m \left(\theta_{ij}^{x_i^*}(1 - \theta_{ij}^{1-x_i^*})\right) \cdot \pi_j$$

Since in doing category classification we only care about the relative magnitudes of the probability that x^* belongs to a certain class, we can ignore the denominator in the equation above in doing classification for simplicity.