Logistic regression
In LDA and QDA, we estimate $p(x|y)$, but for classification we are mainly interested in $p(y|x)$.

Why not estimate that directly? Logistic regression\(^1\) is a popular way of doing this.

\(^1\)Despite the name “regression”, we are using it for classification!
Linearity of log-odds and logistic function

- $a + b^\top x$ models the **log-odds ratio**:

$$\log \frac{p(Y = +1|X = x; a, b)}{p(Y = -1|X = x; a, b)} = a + b^\top x.$$ 

- Solve explicitly for conditional class probabilities (using $p(Y = +1|X = x; a, b) + p(Y = -1|X = x; a, b) = 1$):

$$p(Y = +1|X = x; a, b) = \frac{1}{1 + \exp(-(a + b^\top x))} =: s(a + b^\top x)$$

$$p(Y = -1|X = x; a, b) = \frac{1}{1 + \exp((a + b^\top x))} = s(-a - b^\top x)$$

where $s(z) = 1/(1 + \exp(-z))$ is the **logistic function**.
Fitting the parameters of the hyperplane

How to learn $a$ and $b$ given a training data set $(x_i, y_i)_{i=1}^{n}$?

- Consider maximizing the **conditional log likelihood**:

$$
\ell(a, b) = \sum_{i=1}^{n} \log p(y_i | x_i) = \sum_{i=1}^{n} \log s(y_i (a + b^\top x_i)).
$$

- Equivalent to minimizing the empirical risk associated with the **log loss**:

$$
\hat{R}_{\text{log}}(f_a, b) = \frac{1}{n} \sum_{i=1}^{n} - \log s(y_i (a + b^\top x_i)) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i (a + b^\top x_i))).
$$
Logistic Regression

- Log-loss is differentiable, but it is not possible to find optimal $a, b$ analytically.
- For simplicity, absorb $a$ as an entry in $b$ by appending ’1’ into $x$ vector, as we did before.
- Objective function:

$$\hat{R}_{\text{log}} = \frac{1}{n} \sum_{i=1}^{n} -\log s(y_i x_i^\top b)$$

- Differentiate wrt $b$:

$$\nabla_b \hat{R}_{\text{log}} = \frac{1}{n} \sum_{i=1}^{n} -s(-y_i x_i^\top b) y_i x_i$$

$$\nabla^2_b \hat{R}_{\text{log}} = \frac{1}{n} \sum_{i=1}^{n} s(y_i x_i^\top b)s(-y_i x_i^\top b)x_i x_i^\top \succeq 0.$$  

- We cannot set $\nabla_b \hat{R}_{\text{log}} = 0$ and solve: no closed form solution. We’ll use numerical methods.

Logistic Function

$$s(-z) = 1 - s(z)$$
$$\nabla_z s(z) = s(z)s(-z)$$
$$\nabla_z \log s(z) = s(-z)$$
$$\nabla^2_z \log s(z) = -s(z)s(-z)$$
Where Will We Converge?

Any local minimum is a global minimum

Multiple local minima may exist

Least Squares, Ridge Regression and Logistic Regression are all convex!
Convexity

How to determine convexity? $f(x)$ is convex if

$$f''(x) \geq 0$$

Examples:

$$f(x) = x^2, f''(x) = 2 > 0$$

How to determine convexity in this case?

Matrix of second-order derivatives (Hessian)

$$H = \begin{pmatrix}
\frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_D} \\
\frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_D} \\
\cdots & \cdots & \cdots & \cdots \\
\frac{\partial^2 f(x)}{\partial x_1 \partial x_D} & \frac{\partial^2 f(x)}{\partial x_2 \partial x_D} & \cdots & \frac{\partial^2 f(x)}{\partial x_D^2}
\end{pmatrix}$$

How to determine convexity in the multivariate case?

If the Hessian is positive semi-definite $H \succeq 0$, then $f$ is convex. A matrix $H$ is positive semi-definite if and only if, $\forall z$,

$$z^T H z = \sum_{j,k} H_{j,k} z_j z_k \geq 0$$
Logistic Regression

- Hessian is positive-definite: objective function is **convex** and there is a **single unique global minimum**.
- Many different algorithms can find optimal $b$, e.g.:
  - Gradient descent:
    \[
    b^\text{new} = b + \epsilon \frac{1}{n} \sum_{i=1}^{n} s(-y_i x_i^\top b) y_i x_i
    \]
  - Stochastic gradient descent:
    \[
    b^\text{new} = b + \epsilon_t \frac{1}{|I(t)|} \sum_{i \in I(t)} s(-y_i x_i^\top b) y_i x_i
    \]
    where $I(t)$ is a subset of the data at iteration $t$, and $\epsilon_t \to 0$ slowly ($\sum_t \epsilon_t = \infty, \sum_t \epsilon_t^2 < \infty$).
  - Conjugate gradient, LBFGS and other methods from numerical analysis.
  - Newton-Raphson:
    \[
    b^\text{new} = b - (\nabla_b^2 \hat{R}_{\log})^{-1} \nabla_b \hat{R}_{\log}
    \]
    This is also called **iterative reweighted least squares**.
Iterative reweighted least squares (IRLS)

We can write gradient and Hessian in a more compact form. Define 
\[ \mu_i = s(x_i^\top b), \] and the diagonal matrix \( S \) with \( \mu_i(1 - \mu_i) \) on its diagonal. Also define the vector \( c \) where \( c_i = 1(y_i = +1) \). Then

\[
\nabla_b \widehat{R}_{\log} = \frac{1}{n} \sum_{i=1}^{n} -s(-y_i x_i^\top b)y_i x_i
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} x_i (\mu_i - c_i)
\]

\[
= X^\top (\mu - c)
\]

\[
\nabla_b^2 \widehat{R}_{\log} = \frac{1}{n} \sum_{i=1}^{n} s(y_i x_i^\top b)s(-y_i x_i^\top b)x_i x_i^\top
\]

\[
= X^\top S X
\]
Iterative reweighted least squares (IRLS)

Let $b_t$ be the parameters after $t$ “Newton steps”. The gradient and Hessian at step $t$ are given by:

$$g_t = X^T(\mu_t - c) = -X^T(c - \mu_t)$$
$$H_t = X^T S_t X$$

The Newton Update Rule is:

$$b_{t+1} = b_t - H_t^{-1} g_t$$
$$= b_t + (X^T S_t X)^{-1} X^T (c - \mu_t)$$
$$= (X^T S_t X)^{-1} X^T S_t (Xb_t + S_t^{-1}(c - \mu_t))$$
$$= (X^T S_t X)^{-1} X^T S_t z_t$$

Where $z_t = Xb_t + S_t^{-1}(c - \mu_t)$. Then $b_{t+1}$ is a solution of the “weighted least squares” problem:

$$\text{minimise} \quad \sum_{i=1}^{N} S_{t,ii}(z_{t,i} - b^T x_i)^2$$
Assume that the data is linearly separable, i.e. there is a scalar $\alpha$ and a vector $\beta$ such that $y_i(\alpha + \beta^\top x_i) > 0$, $i = 1, \ldots, n$. Let $c > 0$. The empirical risk for $a = c\alpha$, $b = c\beta$ is

$$\widehat{R}_{\text{log}}(f_{a,b}) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-cy_i(\alpha + \beta^\top x_i)))$$

which can be made arbitrarily close to zero as $c \to \infty$, i.e. soft classification rule becomes $\pm \infty$ (overconfidence) $\to$ overfitting.

**Regularization** provides a solution to this problem.
The **multi-class/multinomial** logistic regression uses the **softmax** function to model the conditional class probabilities $p(Y = k | X = x; \theta)$, for $K$ classes $k = 1, \ldots, K$, i.e.,

$$p(Y = k | X = x; \theta) = \frac{\exp \left( w_k^\top x + b_k \right)}{\sum_{\ell=1}^{K} \exp \left( w_\ell^\top x + b_\ell \right)}.$$ 

Parameters are $\theta = (b, W)$ where $W = (w_{kj})$ is a $K \times p$ matrix of weights and $b \in \mathbb{R}^K$ is a vector of bias terms.
Multi-class logistic regression
library(MASS)
## load crabs data
data(crabs)
ct <- as.numeric(crabs[,1])-1+2*(as.numeric(crabs[,2])-1)
## project into first two LD
cb.lda <- lda(log(crabs[,4:8]),ct)
cb.ldp <- predict(cb.lda)
x <- cb.ldp$x[,1:2]
y <- as.numeric(ct==0)
eqscplot(x,pch=2*y+1,col=y+1)
# visualize decision boundary

gx1 <- seq(-6,6,.02)
gx2 <- seq(-4,4,.02)
gx <- as.matrix(expand.grid(gx1,gx2))
gm <- length(gx1)
gn <- length(gx2)
gdf <- data.frame(LD1=gx[,1],LD2=gx[,2])

lda <- lda(x,y)
y.lda <- predict(lda,x)$class
eqscplot(x,pch=2*y+1,col=2-as.numeric(y==y.lda))
y.lda.grid <- predict(lda,gdf)$class
contour(gx1,gx2,matrix(y.lda.grid,gm,gn),
        levels=c(0.5), add=TRUE,d=FALSE,lty=2,lwd=2)
Crab Dataset

```r
## logistic regression
xdf <- data.frame(x)
logreg <- glm(y ~ LD1 + LD2, data=xdf, family=binomial)
y.lr <- predict(logreg,type="response")
eqscplot(x,pch=2*y+1,col=2-as.numeric(y==(y.lr>.5)))
y.lr.grid <- predict(logreg,newdata=gdf,type="response")
contour(gx1,gx2,matrix(y.lr.grid,gm,gn),
    levels=c(.1,.25,.75,.9), add=TRUE,d=FALSE,lty=3,lwd=1)
contour(gx1,gx2,matrix(y.lr.grid,gm,gn),
    levels=c(.5), add=TRUE,d=FALSE,lty=1,lwd=2)

## logistic regression with quadratic interactions
logreg <- glm(y ~ (LD1 + LD2)^2, data=xdf, family=binomial)
y.lr <- predict(logreg,type="response")
eqscplot(x,pch=2*y+1,col=2-as.numeric(y==(y.lr>.5)))
y.lr.grid <- predict(logreg,newdata=gdf,type="response")
contour(gx1,gx2,matrix(y.lr.grid,gm,gn),
    levels=c(.1,.25,.75,.9), add=TRUE,d=FALSE,lty=3,lwd=1)
contour(gx1,gx2,matrix(y.lr.grid,gm,gn),
    levels=c(.5), add=TRUE,d=FALSE,lty=1,lwd=2)
```
Crab Dataset: Blue Female vs. rest

Comparing LDA and logistic regression.
Comparing logistic regression with and without quadratic interactions.
Logistic regression Python demo


Generative vs. Discriminative
Generative vs Discriminative Learning

- Machine learning: learn a (random) function that maps a variable $X$ (feature) to a variable $Y$ (class) using a (labeled) dataset $\mathcal{D} = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$.
  - Generative Approach: learn $P(Y, X) = P(Y|X)P(X)$.
  - Discriminative Approach: learn $P(Y|X)$.
Generative Learning

- **Generative Approach**: Finds a probabilistic model (a joint distribution $P(Y, X)$) that explicitly models the distribution of both the features and the corresponding labels (classes).

- Example techniques: LDA, QDA, Naive Bayes (coming soon), Hidden Markov Models, etc.
Discriminative Learning

- **Discriminative Approach**: Finds a good fit for $P(Y|X)$ without explicitly modeling the generative process.
- Example techniques: linear regression, logistic regression, K-nearest neighbors (coming soon), SVMs, perceptrons, etc.
- Example problem: 2 classes, separate the classes.
Generative vs Discriminative Learning

- **Generative Approach**: Finds parameters that explain all data.
  \[ \hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log p(x_i, y_i | \theta) \]
  - Makes use of all the data.
  - Flexible framework, can incorporate many tasks (e.g. classification, regression, semi-supervised learning, survival analysis, generating new data samples similar to the existing dataset, etc).
  - Stronger modeling assumptions, which may not be realistic (Gaussianity, independence of features).

- **Discriminative Approach**: Finds parameters that help to predict only relevant data.
  \[ \hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_\theta(x_i)) \quad \text{or} \quad \hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log p(y_i | x_i, \theta) \]
  - Weaker modeling assumptions (thus often fewer violated assumptions and better calibration of probabilities).
  - Learns to perform better on the given tasks.
  - Less immune to overfitting.
  - Easier to work with preprocessed data \( \phi(x) \).
Naïve Bayes
Naïve Bayes: overview

- Naïve Bayes: another **plug-in** classifier with a simple **generative** model - it assumes all measured variables/features are independent given the label.
- Easy to mix and match different types of features, handle missing data.
- Often used with categorical data, e.g. text document classification.
  - A basic standard model for text classification consists of considering a pre-specified dictionary of $p$ words and summarizing each document $i$ by a binary vector $x_i$ (“bag-of-words”):

$$x_i^{(j)} = \begin{cases} 
1 & \text{if word } j \text{ is present in document} \\
0 & \text{otherwise.}
\end{cases}$$

where the presence of the word $j$ is the $j$-th feature/dimension.
## Toy Example

Predict voter preference in US elections

<table>
<thead>
<tr>
<th>Voted in 2012?</th>
<th>Annual Income</th>
<th>State</th>
<th>Candidate Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>50K</td>
<td>OK</td>
<td>Clinton</td>
</tr>
<tr>
<td>N</td>
<td>173K</td>
<td>CA</td>
<td>Clinton</td>
</tr>
<tr>
<td>Y</td>
<td>80K</td>
<td>NJ</td>
<td>Trump</td>
</tr>
<tr>
<td>Y</td>
<td>150K</td>
<td>WA</td>
<td>Clinton</td>
</tr>
<tr>
<td>N</td>
<td>25K</td>
<td>WV</td>
<td>Johnson</td>
</tr>
<tr>
<td>Y</td>
<td>85K</td>
<td>IL</td>
<td>Clinton</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1050K</td>
<td>NY</td>
<td>Trump</td>
</tr>
<tr>
<td>N</td>
<td>35K</td>
<td>CA</td>
<td>Trump</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td><strong>100K</strong></td>
<td><strong>NY</strong></td>
<td>?</td>
</tr>
</tbody>
</table>
Naïve Bayes Classiﬁer (NBC)

- In order to ﬁt a generative model, we’ll express the joint distribution as
  \[ p(x, y \mid \theta, \pi) = p(y \mid \pi) \cdot p(x \mid y, \theta) \]

- To model \( p(y \mid \pi) \), we’ll use parameters \( \pi_c \) such that \( \sum_c \pi_c = 1 \)
  \[ p(y = c \mid \pi) = \pi_c \]

- For class-conditional densities, for class \( c = 1, \ldots, C \), we will have a model:
  \[ p(x \mid y = c, \theta_c) \]

- We assume that the features are conditionally independent given the class label
  \[ p(x \mid y = c, \theta_c) = \prod_{j=1}^{D} p(x_j \mid y = c, \theta_{jc}) \]

- Clearly, the independence assumption is “naïve” and never satisfied. But model ﬁtting becomes very very easy.

- Although the generative model is clearly inadequate, it actually works quite well. Goal is predicting class, not modelling the data!
Naïve Bayes Classifier (NBC)

In our example,

\[ p(y = \text{clinton} \mid \pi) = \pi_{\text{clinton}} \]
\[ p(y = \text{trump} \mid \pi) = \pi_{\text{trump}} \]
\[ p(y = \text{johnson} \mid \pi) = \pi_{\text{johnson}} \]

Given that a voter supports Trump

\[ p(x \mid y = \text{trump}, \theta_{\text{trump}}) \]
models the distribution over \( x \) given \( y = \text{trump} \) and \( \theta_{\text{trump}} \)

Similarly, we have \( p(x \mid y = \text{clinton}, \theta_{\text{clinton}}) \) and \( p(x \mid y = \text{johnson}, \theta_{\text{johnson}}) \)

We need to pick “model” for \( p(x \mid y = c, \theta_c) \)

Estimate the parameters \( \pi_c, \theta_c \) for \( c = 1, \ldots, C \)
Naïve Bayes Classifier (NBC)

Real-Valued Features

- $x_j$ is real-valued annual income
- Example: Use a Gaussian model, so $\theta_{jc} = (\mu_{jc}, \sigma_{jc}^2)$
- Can use other distributions, age is probably not Gaussian!

Categorical Features

- $x_j$ is categorical with values in $\{1, \ldots, K\}$
- Use the **multinoulli** distribution, i.e. $x_j = i$ with probability $\mu_{jc,i}$

\[
\sum_{i=1}^{K} \mu_{jc,i} = 1
\]

- The special case when $x_j \in \{0, 1\}$, use a single parameter $\theta_{jc} \in [0, 1]$
Naïve Bayes Classifier (NBC)

- Assume that all the features are binary, i.e. every $x_j \in \{0, 1\}$
- (In this case, the log-discriminant function of each class assumes the form $a_c + b_c^T x$ for class $c$. Verify this.)
- If we have $C$ classes, overall we have only $O(CD)$ parameters, $\theta_{jc}$ for each $j = 1, \ldots, D$ and $c = 1, \ldots, C$

Without the conditional independence assumption

- We have to assign a probability for each of the $2^D$ combination
- Thus, we have $O(C \cdot 2^D)$ parameters!
- The ‘naïve’ assumption breaks the curse of dimensionality and avoids overfitting!
Let us suppose we have data \( \langle (x_i, y_i) \rangle_{i=1}^{N} \) i.i.d. from some joint distribution \( p(x, y) \).

The probability for a single datapoint is given by:

\[
p(x_i, y_i | \theta, \pi) = p(y_i | \pi) \cdot p(x_i | \theta, y_i) = \prod_{c=1}^{C} \pi_{c}^{(y_i = c)} \cdot \prod_{c=1}^{C} \prod_{j=1}^{D} p(x_{ij} | \theta_{jc})^{(y_i = c)}
\]

Let \( N_c \) be the number of datapoints with \( y_i = c \), so that \( \sum_{c=1}^{C} N_c = N \).

We write the log-likelihood of the data, assuming points are i.i.d.:

\[
\log p(D | \theta, \pi) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{c=1}^{C} \sum_{j=1}^{D} \sum_{i:y_i = c} \log p(x_{ij} | \theta_{jc})
\]

The log-likelihood is easily separated into sums involving different parameters!
Maximum Likelihood for the NBC

- We have the log-likelihood for the NBC

\[
\log p(\mathcal{D} \mid \theta, \pi) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{c=1}^{C} \sum_{j=1}^{D} \sum_{i : y_i = c} \log p(x_{ij} \mid \theta_{jc})
\]

- We can use maximum likelihood to estimate the parameters (we have done this before). For instance, let’s estimate \( \pi \). We have the following optimization problem:

\[
\text{maximize} \quad \sum_{c=1}^{C} N_c \log \pi_c \\
\text{subject to} : \quad \sum_{c=1}^{C} \pi_c = 1
\]

- This constrained optimization problem can be solved using Lagrange multipliers

\[
\Lambda(\pi, \lambda) = \sum_{c=1}^{C} N_c \log \pi_c + \lambda \left( \sum_{c=1}^{C} \pi_c - 1 \right)
\]
Maximum Likelihood for the NBC

We can write the Lagrangean form:

$$\Lambda(\pi, \lambda) = \sum_{c=1}^{C} N_c \log \pi_c + \lambda \left( \sum_{c=1}^{C} \pi_c - 1 \right)$$

We can write the partial derivatives and set them to 0:

$$\frac{\partial \Lambda(\pi, \lambda)}{\partial \pi_c} = \frac{N_c}{\pi_c} + \lambda = 0; \quad \frac{\partial \Lambda(\pi, \lambda)}{\partial \lambda} = \sum_{c=1}^{C} \pi_c - 1 = 0$$

The solution is obtained by setting

$$\frac{N_c}{\pi_c} + \lambda = 0 \quad \rightarrow \quad \pi_c = -\frac{N_c}{\lambda}$$

As well as using the second condition,

$$\sum_{c=1}^{C} \pi_c - 1 = \left( \sum_{c=1}^{C} - \frac{N_c}{\lambda} \right) - 1 = 0 \quad \rightarrow \quad \lambda = -\sum_{c=1}^{C} N_c = -N$$

Thus, we get the estimates,

$$\pi_c = \frac{N_c}{N}$$
Maximum Likelihood for the NBC

- We have the log-likelihood for the NBC

\[
\log p(D \mid \theta, \pi) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{c=1}^{C} \sum_{j=1}^{D} \sum_{i:y_i=c} \log p(x_{ij} \mid \theta_{jc})
\]

- We obtained the estimates, \( \pi_c = \frac{N_c}{N} \)
- We can estimate \( \theta_{jc} \) by taking a similar approach
- To estimate \( \theta_{jc} \) we only need to use the \( j^{th} \) feature of examples with \( y_i = c \)
- Estimates depend on the model, e.g. Gaussian, Bernoulli, Multinoulli, etc.
- Fitting NBC is very very fast!
Let's recall our example about trying to predict voter preferences

<table>
<thead>
<tr>
<th>Voted in 2012?</th>
<th>Annual Income</th>
<th>State</th>
<th>Candidate Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>50K</td>
<td>OK</td>
<td>Clinton</td>
</tr>
<tr>
<td>N</td>
<td>173K</td>
<td>CA</td>
<td>Clinton</td>
</tr>
<tr>
<td>Y</td>
<td>80K</td>
<td>NJ</td>
<td>Trump</td>
</tr>
<tr>
<td>Y</td>
<td>150K</td>
<td>WA</td>
<td>Clinton</td>
</tr>
<tr>
<td>N</td>
<td>25K</td>
<td>WV</td>
<td>Johnson</td>
</tr>
<tr>
<td>Y</td>
<td>85K</td>
<td>IL</td>
<td>Clinton</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1050K</td>
<td>NY</td>
<td>Trump</td>
</tr>
<tr>
<td>N</td>
<td>35K</td>
<td>CA</td>
<td>Trump</td>
</tr>
<tr>
<td>?</td>
<td>100K</td>
<td>NY</td>
<td>?</td>
</tr>
</tbody>
</table>

Suppose a voter does not reveal whether or not they voted in 2012

For now, let’s assume we had no missing entries during training
NBC: Handling Missing Data

The prediction rule in a generative model is

\[
p(y = c \mid \mathbf{x}_{\text{new}}, \theta) = \frac{p(y = c \mid \theta) \cdot p(\mathbf{x}_{\text{new}} \mid y = c, \theta)}{\sum_{c' = 1}^{C} p(y = c' \mid \theta) p(\mathbf{x}_{\text{new}} \mid y = c', \theta)}
\]

Let us suppose our datapoint is \( \mathbf{x}_{\text{new}} = (? , x_2, \ldots, x_D) \), e.g. \((?, 100K, \text{NY})\)

\[
p(y = c \mid \mathbf{x}_{\text{new}}, \theta) = \frac{\pi_c \cdot \prod_{j=1}^{D} p(x_j \mid y = c, \theta_{cj})}{\sum_{c' = 1}^{C} p(y = c' \mid \theta) \prod_{j=1}^{D} p(x_j \mid y = c', \theta_{jc})}
\]

Since \( x_1 \) is missing, we can marginalize it out,

\[
p(y = c \mid \mathbf{x}_{\text{new}}, \theta) = \frac{\pi_c \cdot \prod_{j=2}^{D} p(x_j \mid y = c, \theta_{cj})}{\sum_{c' = 1}^{C} p(y = c' \mid \theta) \prod_{j=2}^{D} p(x_j \mid y = c', \theta_{jc})}
\]

This can be done for other generative models, but marginalization requires summation/integration
Naïve Bayes Model

NBC: Handling Missing Data

For Naïve Bayes Classifiers, training with missing entries is quite easy

<table>
<thead>
<tr>
<th>Voted in 2012?</th>
<th>Annual Income</th>
<th>State</th>
<th>Candidate Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>50K</td>
<td>OK</td>
<td>Clinton</td>
</tr>
<tr>
<td>N</td>
<td>173K</td>
<td>CA</td>
<td>Clinton</td>
</tr>
<tr>
<td>?</td>
<td>80K</td>
<td>NJ</td>
<td>Trump</td>
</tr>
<tr>
<td>Y</td>
<td>150K</td>
<td>WA</td>
<td>Clinton</td>
</tr>
<tr>
<td>N</td>
<td>25K</td>
<td>WV</td>
<td>Johnson</td>
</tr>
<tr>
<td>Y</td>
<td>85K</td>
<td>?</td>
<td>Clinton</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1050K</td>
<td>NY</td>
<td>Trump</td>
</tr>
<tr>
<td>N</td>
<td>35K</td>
<td>CA</td>
<td>Trump</td>
</tr>
<tr>
<td>?</td>
<td>100K</td>
<td>NY</td>
<td>?</td>
</tr>
</tbody>
</table>

Let’s say for Clinton voters, 103 had voted in 2012, 54 had not, and 25, didn’t answer

You can simply set $\theta = \frac{103}{157}$ as the probability that a voter had voted in 2012, conditioned on being a Clinton supporter
Naïve Bayes vs Logistic regression

"On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes" by A. Ng and M. Jordan, NIPS 2001. $m$ represents training dataset size.
Naïve Bayes vs Logistic regression

- For infinite data
  - If generative model is correct (independence assumption holds)
    \[ Error_{LR,\infty} \sim Error_{NB,\infty} \]
  - If generative model is inaccurate (independence assumption does not hold)
    \[ Error_{LR,\infty} < Error_{NB,\infty} \]

- For finite data (e.g. \( n \) points, \( d \) features), NB will require less training to converge to its (possibly asymptotically higher) error
  \[
  Error_{LR,n} \leq Error_{LR,\infty} + O\left(\sqrt{\frac{d}{n}}\right) \\
  Error_{NB,n} \leq Error_{NB,\infty} + O\left(\sqrt{\frac{\log d}{n}}\right)
  \]
Preventing numerical underflow (not examinable)

- Generative classifiers often require multiplying a large number of small quantities, leading to **numerical underflow**.
  \[
  \log p(y = c | \mathbf{x}) = \log \left( \frac{p(y = c) p(\mathbf{x} | y = c)}{\sum_{c'} p(y = c') p(\mathbf{x} | y = c')} \right) 
  \]
  \[
  = b_c - \log \left( \sum_{c'} e^{b_{c'}} \right) 
  \]
  \[
  b_c \triangleq \log p(\mathbf{x} | y = c) + \log p(y = c) 
  \]
- The terms \( e^{b_{c'}} \) are extremely small (e.g. in Naive Bayes), but we cannot sum in the log domain to evaluate \( \log \sum_{c'} e^{b_{c'}} \).
- Idea: factor out the largest term\(^2\). For example:
  \[
  \log(e^{-120} + e^{-121}) = \log(e^{-120}(e^0 + e^{-1})) = \log(e^0 + e^{-1}) - 120. 
  \]
- In general, having defined \( B = \max_c b_c \):
  \[
  \log \sum_{c} e^{b_{c}} = \log \left( \left( \sum_{c} e^{b_{c}-B} \right) e^{B} \right) = \left[ \log \left( \sum_{c} e^{b_{c}-B} \right) \right] + B 
  \]

\(^2\)Also see Murphy 3.5.3.
Naïve Bayes code example: Titanic data I

Predicting Titanic survival from passenger data using Naïve Bayes:

```r
#Install the package
install.packages("e1071")
#Loading the library
library(e1071)
?naiveBayes #The documentation also uses Titanic data
#Next load the Titanic dataset
data("Titanic")
#Save into a data frame and view it
Titanic_df=as.data.frame(Titanic)
#Creating data from table
#This will repeat each combination equal to the frequency
repeating_sequence=rep.int(seq_len(nrow(Titanic_df)),
    Titanic_df$Freq)

#Create the dataset by row repetition created
Naïve Bayes code example: Titanic data II

Titanic_dataset = Titanic_df[repeating_sequence,]
# We no longer need the frequency, drop the feature
Titanic_dataset$Freq = NULL

# Fitting the Naive Bayes model
Naive_Bayes_Model = naiveBayes(Survived ~ ., data=Titanic_dataset)
# What does the model say? Print the model summary
Naive_Bayes_Model

# Prediction on the dataset
NB_Predictions = predict(Naive_Bayes_Model, Titanic_dataset)
# Confusion matrix to check accuracy
table(NB_Predictions, Titanic_dataset$Survived)

3code from
https://r-posts.com/understanding-naive-bayes-classifier-using-r/