

# Statistical Machine Learning

**Pier Francesco Palamara**

Department of Statistics

University of Oxford

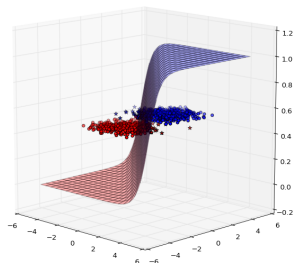
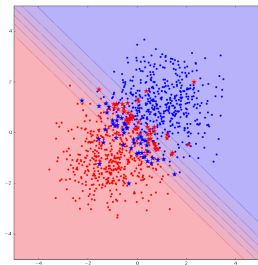
Slide credits and other course material can be found at:

[http://www.stats.ox.ac.uk/~palamara/SML19\\_BDI.html](http://www.stats.ox.ac.uk/~palamara/SML19_BDI.html)

# Logistic regression

# Review

- 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<sup>1</sup> is a popular way of doing this.



---

<sup>1</sup>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**:

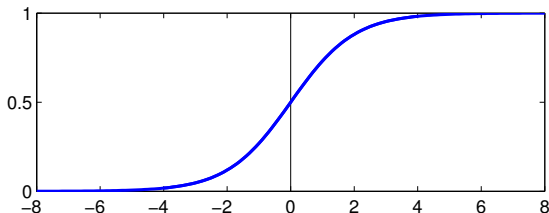
$$\log \frac{p(Y = +1|X = x; a, b)}{p(Y = -1|X = x; a, b)} = a + b^\top x.$$

- 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

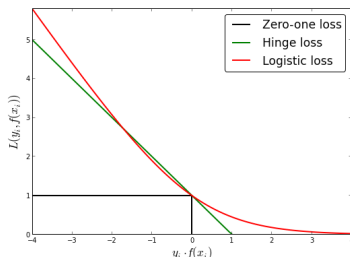
How to learn  $a$  and  $b$  given a training data set  $(x_i, y_i)_{i=1}^n$ ?

- 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**:

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



# 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:

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

- Differentiate wrt  $b$ :

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

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

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

## Logistic Function

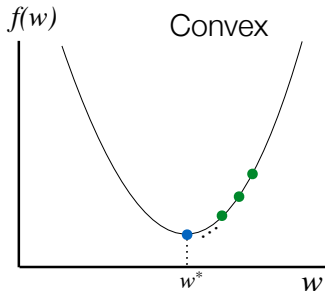
$$s(-z) = 1 - s(z)$$

$$\nabla_z s(z) = s(z)s(-z)$$

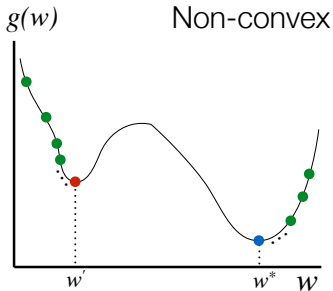
$$\nabla_z \log s(z) = s(-z)$$

$$\nabla_z^2 \log s(z) = -s(z)s(-z)$$

# 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**)

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

**How to determine convexity in the multivariate case?**

If the Hessian is positive semi-definite  $\mathbf{H} \succeq 0$ , then  $f$  is convex.

A matrix  $\mathbf{H}$  is positive semi-definite if and only if,  $\forall \mathbf{z}$ ,

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

# 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  $\epsilon_t \rightarrow 0$  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  $\mathbf{S}$  with  $\mu_i(1 - \mu_i)$  on its diagonal. Also define the vector  $\mathbf{c}$  where  $c_i = \mathbb{1}(y_i = +1)$ . Then

$$\begin{aligned}\nabla_b \hat{R}_{\log} &= \frac{1}{n} \sum_{i=1}^n -s(-y_i x_i^\top b) y_i x_i \\ &= \frac{1}{n} \sum_{i=1}^n x_i (\mu_i - c_i) \\ &= \mathbf{X}^\top (\boldsymbol{\mu} - \mathbf{c}) \\ \nabla_b^2 \hat{R}_{\log} &= \frac{1}{n} \sum_{i=1}^n s(y_i x_i^\top b) s(-y_i x_i^\top b) x_i x_i^\top \\ &= \mathbf{X}^\top \mathbf{S} \mathbf{X}\end{aligned}$$

# Iterative reweighted least squares (IRLS)

Let  $\mathbf{b}_t$  be the parameters after  $t$  “Newton steps”.

The gradient and Hessian at step  $t$  are given by:

$$\begin{aligned}\mathbf{g}_t &= \mathbf{X}^\top (\boldsymbol{\mu}_t - \mathbf{c}) = -\mathbf{X}^\top (\mathbf{c} - \boldsymbol{\mu}_t) \\ \mathbf{H}_t &= \mathbf{X}^\top \mathbf{S}_t \mathbf{X}\end{aligned}$$

The Newton Update Rule is:

$$\begin{aligned}\mathbf{b}_{t+1} &= \mathbf{b}_t - \mathbf{H}_t^{-1} \mathbf{g}_t \\ &= \mathbf{b}_t + (\mathbf{X}^\top \mathbf{S}_t \mathbf{X})^{-1} \mathbf{X}^\top (\mathbf{c} - \boldsymbol{\mu}_t) \\ &= (\mathbf{X}^\top \mathbf{S}_t \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{S}_t (\mathbf{X} \mathbf{b}_t + \mathbf{S}_t^{-1} (\mathbf{c} - \boldsymbol{\mu}_t)) \\ &= (\mathbf{X}^\top \mathbf{S}_t \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{S}_t \mathbf{z}_t\end{aligned}$$

Where  $\mathbf{z}_t = \mathbf{X} \mathbf{b}_t + \mathbf{S}_t^{-1} (\mathbf{c} - \boldsymbol{\mu}_t)$ . Then  $\mathbf{b}_{t+1}$  is a solution of the “weighted least squares” problem:

$$\text{minimise} \quad \sum_{i=1}^N S_{t,ii} (z_{t,i} - \mathbf{b}^\top \mathbf{x}_i)^2$$

# Linearly separable data

Assume that the data is linearly separable, i.e. there is a scalar  $\alpha$  and a vector  $\beta$  such that  $y_i(\alpha + \beta^\top x_i) > 0$ ,  $i = 1, \dots, n$ . Let  $c > 0$ . The empirical risk for  $a = c\alpha$ ,  $b = c\beta$  is

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

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

**Regularization** provides a solution to this problem.

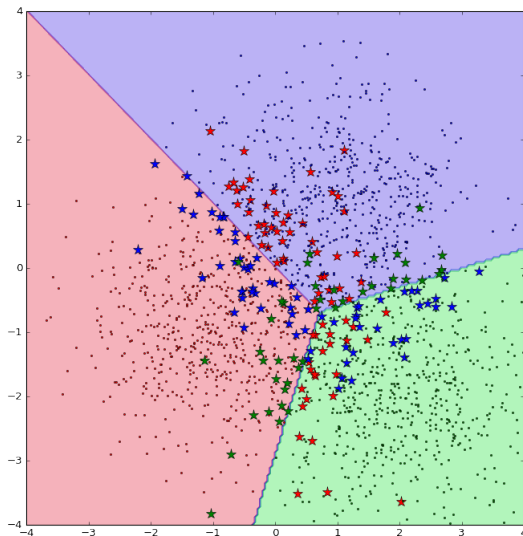
# Multi-class logistic regression

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  $k = 1, \dots, K$ , i.e.,

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

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



# Crab Dataset

```
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)
```

# Crab Dataset

```
## 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])

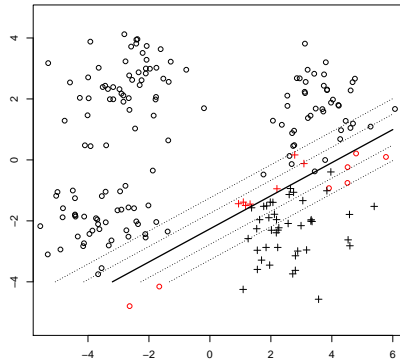
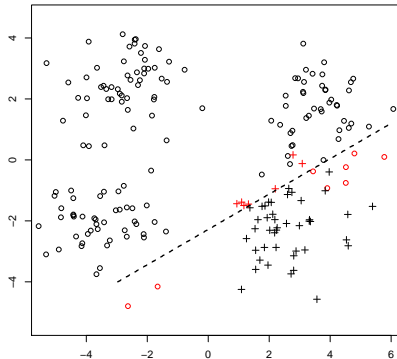
lda <- lda(x,y)
y.lda <- predict(lda,x)$class
eqsplot(x,pch=2*y+1,col=2-as.numeric(y==y.lda))
y.lda.grid <- predict(lda,gdf)$class
contour(gx1,gx2,matrix(y.lda.grid,gm,gn),
        levels=c(0.5), add=TRUE,d=FALSE,lty=2,lwd=2)
```

# Crab Dataset

```
## logistic regression
xdf <- data.frame(x)
logreg <- glm(y ~ LD1 + LD2, data=xdf, family=binomial)
y.lr <- predict(logreg,type="response")
eqsplot(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)

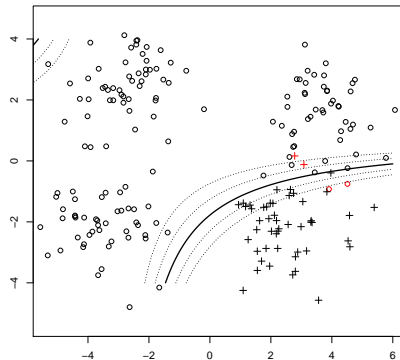
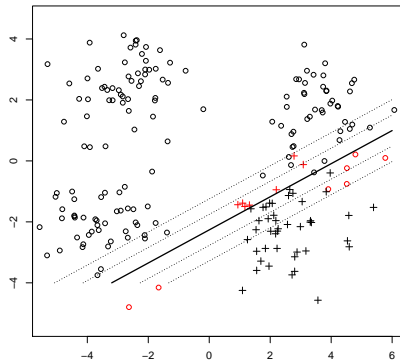
## logistic regression with quadratic interactions
logreg <- glm(y ~ (LD1 + LD2)^2, data=xdf, family=binomial)
y.lr <- predict(logreg,type="response")
eqsplot(x,pch=2*y+1,col=2-as.numeric(y==(y.lr>.5)))
y.lr.grid <- predict(logreg,newdata=gdf,type="response")
contour(gx1,gx2,matrix(y.lr.grid,gm,gn),
        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.

# Crab Dataset



Comparing logistic regression with and without quadratic interactions.

# Logistic regression Python demo

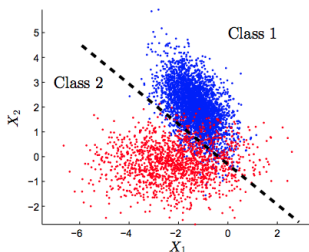
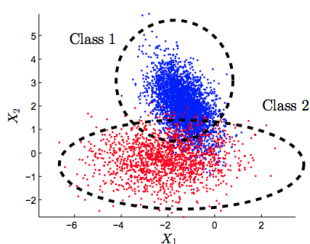
**Single-class:** <https://github.com/vkanade/mlmt2017/blob/master/lecture11/Logistic%20Regression.ipynb>

**Multi-class:** <https://github.com/vkanade/mlmt2017/blob/master/lecture11/Multiclass%20Logistic%20Regression.ipynb>

# Generative vs. Discriminative

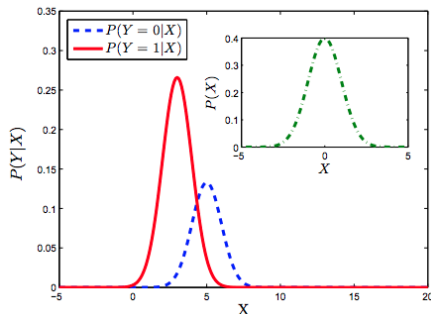
# Generative vs Discriminative Learning

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



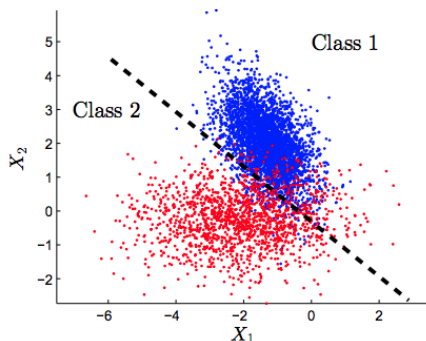
# Generative Learning

- **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.

$$\hat{\theta} = \operatorname{argmax}_{\theta} \sum_{i=1}^n \log p(x_i, y_i | \theta)$$

- 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.

$$\hat{\theta} = \operatorname{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^n L(y_i, f_{\theta}(x_i)) \quad \text{or} \quad \hat{\theta} = \operatorname{argmax}_{\theta} \sum_{i=1}^n \log p(y_i | x_i, \theta)$$

- 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

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

# Naïve Bayes Classifier (NBC)

- In order to fit a generative model, we'll express the joint distribution as

$$p(\mathbf{x}, y \mid \boldsymbol{\theta}, \boldsymbol{\pi}) = p(y \mid \boldsymbol{\pi}) \cdot p(\mathbf{x} \mid y, \boldsymbol{\theta})$$

- To model  $p(y \mid \boldsymbol{\pi})$ , we'll use parameters  $\pi_c$  such that  $\sum_c \pi_c = 1$

$$p(y = c \mid \boldsymbol{\pi}) = \pi_c$$

- For class-conditional densities, for class  $c = 1, \dots, C$ , we will have a model:

$$p(\mathbf{x} \mid y = c, \boldsymbol{\theta}_c)$$

- We assume that the features are conditionally independent given the class label

$$p(\mathbf{x} \mid y = c, \boldsymbol{\theta}_c) = \prod_{j=1}^D p(x_j \mid y = c, \theta_{jc})$$

- 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

$$p(\mathbf{x} \mid y = \text{trump}, \theta_{\text{trump}})$$

models the distribution over  $\mathbf{x}$  given  $y = \text{trump}$  and  $\theta_{\text{trump}}$

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

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

Estimate the parameters  $\pi_c, \theta_c$  for  $c = 1, \dots, C$

# 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  $\{1, \dots, K\}$
- 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  $a_c + b_c^\top x$  for class  $c$ . Verify this.)
- If we have  $C$  classes, overall we have only  $O(CD)$  parameters,  $\theta_{jc}$  for each  $j = 1, \dots, D$  and  $c = 1, \dots, C$
- 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!

# Maximum Likelihood for the NBC

- Let us suppose we have data  $\langle (\mathbf{x}_i, y_i) \rangle_{i=1}^N$  i.i.d. from some joint distribution  $p(\mathbf{x}, y)$
- The probability for a single datapoint is given by:

$$p(\mathbf{x}_i, y_i | \boldsymbol{\theta}, \boldsymbol{\pi}) = p(y_i | \boldsymbol{\pi}) \cdot p(\mathbf{x}_i | \boldsymbol{\theta}, y_i) = \prod_{c=1}^C \pi_c^{\mathbb{I}(y_i=c)} \cdot \prod_{c=1}^C \prod_{j=1}^D p(x_{ij} | \boldsymbol{\theta}_{jc})^{\mathbb{I}(y_i=c)}$$

- 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.:

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

- The log-likelihood is easily separated into sums involving different parameters!

# Maximum Likelihood for the NBC

- We have the log-likelihood for the NBC

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

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

$$\begin{aligned} &\text{maximize} && \sum_{c=1}^C N_c \log \pi_c \\ &\text{subject to :} && \sum_{c=1}^C \pi_c = 1 \end{aligned}$$

- This constrained optimization problem can be solved using Lagrange multipliers

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

# Maximum Likelihood for the NBC

We can write the Lagrangean form:

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

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

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

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

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

- We obtained the estimates,  $\pi_c = \frac{N_c}{N}$
- We can estimate  $\boldsymbol{\theta}_{jc}$  by taking a similar approach
- To estimate  $\boldsymbol{\theta}_{jc}$  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!

# NBC: Handling Missing Data

Let's recall our example about trying to predict voter preferences

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

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

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

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

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

Since  $x_1$  is missing, we can marginalize it out,

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

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

# NBC: Handling Missing Data

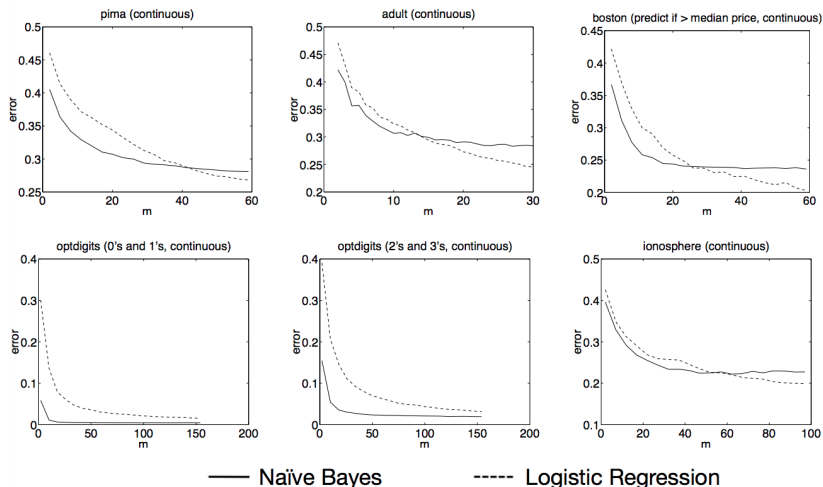
For Naïve Bayes Classifiers, training with missing entries is quite easy

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

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**.

$$\begin{aligned}\log p(y = c|\mathbf{x}) &= \log \left[ \frac{p(y = c)p(\mathbf{x}|y = c)}{\sum_{c'} p(y = c')p(\mathbf{x}|y = c')} \right] \\ &= b_c - \log \left[ \sum_{c'=1}^C e^{b_{c'}} \right]\end{aligned}$$

$$b_c \triangleq \log p(\mathbf{x}|y = c) + \log p(y = c)$$

- 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<sup>2</sup>. For example:

$$\log(e^{-120} + e^{-121}) = \log(e^{-120}(e^0 + e^{-1})) = \log(e^0 + e^{-1}) - 120.$$

- In general, having defined  $B = \max_c b_c$ :

$$\log \sum_c e^{b_c} = \log \left[ \left( \sum_c e^{b_c - B} \right) e^B \right] = \left[ \log \left( \sum_c e^{b_c - B} \right) \right] + B$$

<sup>2</sup>Also see Murphy 3.5.3.

# Naïve Bayes code example: Titanic data I

Predicting Titanic survival from passenger data using Naïve Bayes<sup>3</sup>:

```
#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
Titanic_dataset=Titanic_df[repeating_sequence,]
```

# Naïve Bayes code example: Titanic data II

```
#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)
```

---

<sup>3</sup>code from