

ADEC 7430: Big Data Econometrics

Classification Models

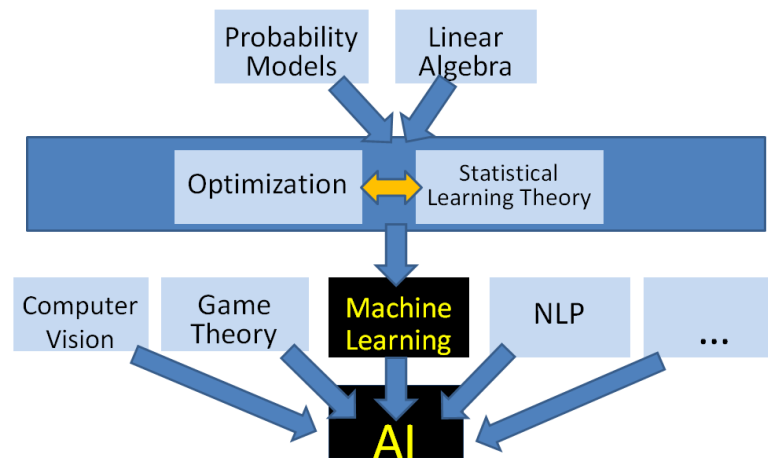
Dr. Nathan Bastian

Woods College of Advancing Studies

Boston College

Assignment

- **Reading:** Ch. 4; Handouts
- **Study:** Lecture Slides, Lecture Videos
- **Activity:** Quiz 5, R Lab 5, Discussion #5, Individual Project #2



References

- *An Introduction to Statistical Learning, with Applications in R* (2013), by G. James, D. Witten, T. Hastie, and R. Tibshirani.
- *The Elements of Statistical Learning* (2009), by T. Hastie, R. Tibshirani, and J. Friedman.
- *Machine Learning: A Probabilistic Perspective* (2012), by K. Murphy



Lesson Goals

- Express the basic setting of a classification problem.
- Explain the Bayes classification rule and the Naïve Bayes classifier
- Describe the statistical model of logistic regression.
- Demonstrate the binary logistic regression algorithm.
- Summarize the basic concepts of the multi-class logistic regression algorithm.
- Explain the statistical model used by Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA).
- Discuss how to estimate the Gaussian distributions within each class.
- Explain the LDA and QDA classification rule.
- Describe the similarities and differences between LDA and logistic regression.
- Review the K -nearest neighbor classification algorithm.
- Recall how the number of neighbors, K , adjusts the model complexity.
- Describe the basic concepts of one of the oldest machine learning methods, the perceptron learning algorithm, for the problem of separating hyperplanes.
- Explain artificial neural networks and the back-propagation algorithm for both regression and classification problems.
- Differentiate between all of the classification methods.

Classification

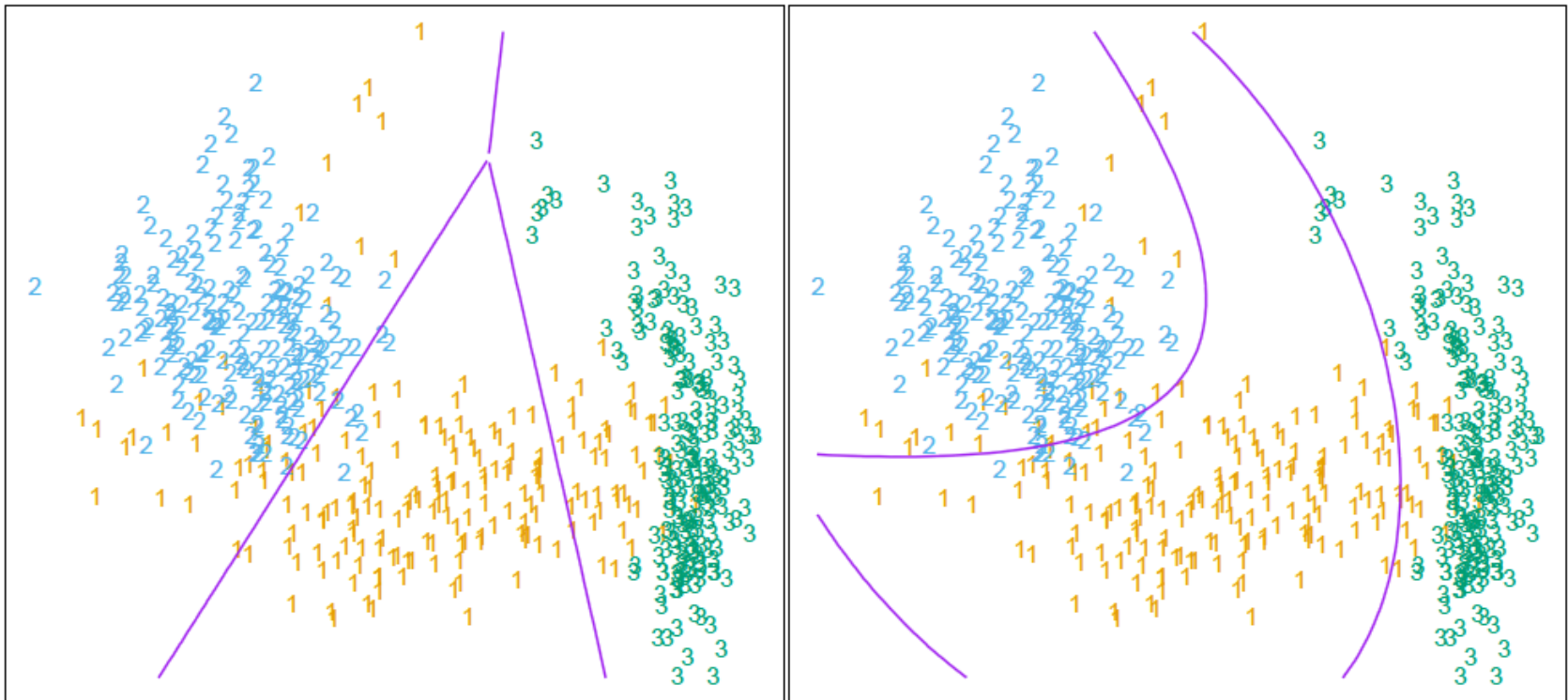
- Predicting a *qualitative* response for an observation can be referred to as *classifying* that observation, since it involves assigning the observation to a category, or class.
- Thus, *classification models* are supervised learning methods for which the true class labels for the data points are given in the training data.
- The methods used for classification often predict the probability of each of the categories of a qualitative variable as the basis for making the classification decision.

Classification Setting

- Training data: $\{(x_1, g_1), (x_2, g_2), \dots, (x_N, g_N)\}$
- The feature vector $X = (X_1, X_2, \dots, X_p)$, where each X_j is quantitative.
- The response variable G is categorical s.t. $G \in \mathcal{G} = \{1, 2, \dots, K\}$
- Form a predictor $G(x)$ to predict G based on X .
- Note that $G(x)$ divides the input space (feature vector space) into a collection of regions, each labeled by one class.

Classification Setting (cont.)

- For each plot, the feature vector space is divided into three pieces, each assigned with a particular class.



Classification Error Rate

- The *classification error rate* is the number of observations that are misclassified over the sample size:

$$\frac{1}{n} \sum_{i=1}^n I(\hat{Y}_i \neq Y_i)$$

where $I(\hat{Y}_i \neq Y_i) = 1$ if $\hat{Y}_i \neq Y_i$ and 0 otherwise.

- For binary classification let \hat{Y} be a 0-1 vector of the predicted class labels and Y be a 0-1 vector of the observed class labels.

Review of Probability Basics

- Prior, conditional and joint probability for random variables
 - Prior probability: $P(X)$
 - Conditional probability: $P(X_1 | X_2), P(X_2 | X_1)$
 - Joint probability: $\mathbf{X} = (X_1, X_2), P(\mathbf{X}) = P(X_1, X_2)$
 - Relationship: $P(X_1, X_2) = P(X_2 | X_1)P(X_1) = P(X_1 | X_2)P(X_2)$
 - Independence: $P(X_2 | X_1) = P(X_2), P(X_1 | X_2) = P(X_1), P(X_1, X_2) = P(X_1)P(X_2)$
- **Bayes' Rule**

$$P(C | \mathbf{X}) = \frac{P(\mathbf{X} | C)P(C)}{P(\mathbf{X})} \quad \text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$$

Bayes Classification Rule

- Suppose the marginal distribution of G is specified by the probability mass function $p_G(g)$, $g = 1, 2, \dots, K$.
- The conditional distribution of X given $G = g$ is $f_{X|G}(x | G = g)$.
- The training data (x_i, g_i) , $i = 1, 2, \dots, N$, are independent samples from the joint distribution of X and G :

$$f_{X,G}(x,g) = p_G(g) f_{X|G}(x | G = g)$$

Bayes Classification Rule (cont.)

- Assume the loss of predicting G as $G(X) = \hat{G}$ is $L(\hat{G}, G)$.

- Goal of classification: minimize the expected loss:

$$E_{X,G}L(G(X), G) = E_X (E_{G|X}L(G(X), G)).$$

- To minimize the left-hand side, it suffices to minimize $E_{X,G}L(G(X), G)$ for each X . Thus, the optimal predictor:

$$\hat{G}(x) = \operatorname{argmin}_g E_{G|X=x} L(g, G)$$

- The above decision rule is called the *Bayes classification rule*.

Bayes Classification Rule (cont.)

- For 0-1 loss, i.e.

$$L(g, g') = \begin{cases} 1 & g \neq g' \\ 0 & g = g' \end{cases}$$

$$E_{G|X=x} L(g, G) = 1 - Pr(G = g|X = x)$$

- The Bayes rule becomes the rule of maximum posterior probability:

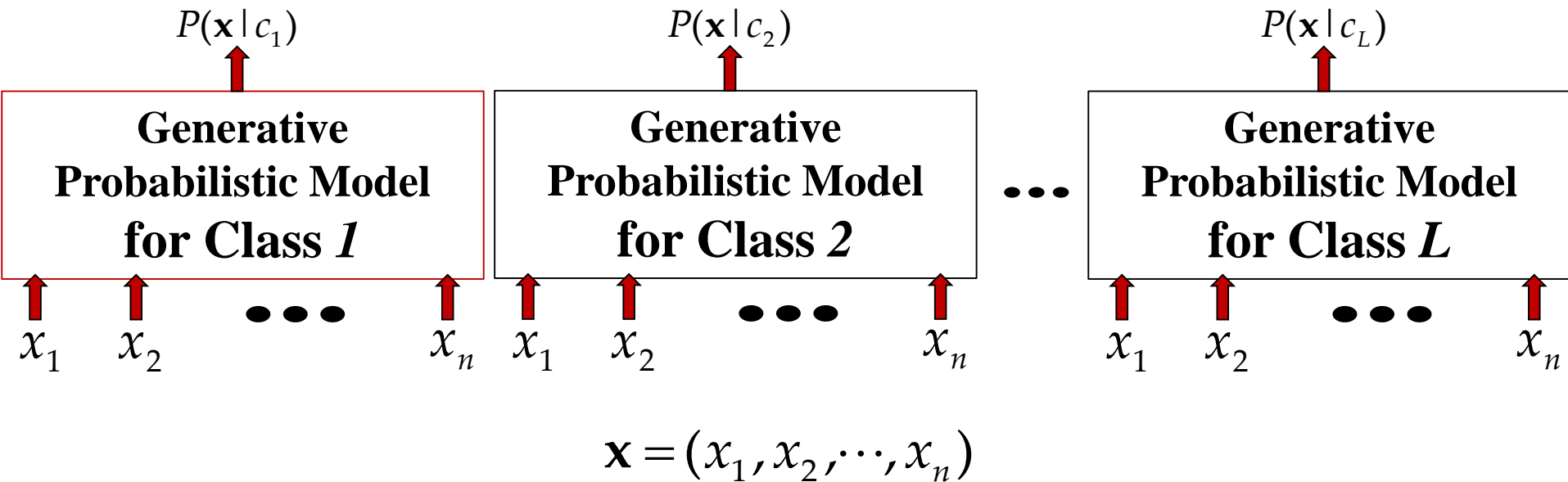
$$\begin{aligned} \hat{G}(x) &= \arg \min_g E_{G|X=x} L(g, G) \\ &= \arg \max_g Pr(G = g|X = x) \end{aligned}$$

- Many classification algorithms attempt to estimate $Pr(G = g | X = x)$, then apply the Bayes rule.

Probabilistic Classification

- Establishing a probabilistic model for classification
 - Generative model**

$$P(\mathbf{X}|C) \quad C = c_1, \dots, c_L, \mathbf{X} = (X_1, \dots, X_n)$$



Probabilistic Classification (cont.)

- MAP classification rule
 - **MAP: M**aximum **A P**osterior
 - Assign \mathbf{x} to c^* if

$$P(C = c^* | \mathbf{X} = \mathbf{x}) > P(C = c | \mathbf{X} = \mathbf{x}) \quad c \neq c^*, \quad c = c_1, \dots, c_L$$

- Generative classification with the MAP rule
 - Apply Bayes' rule to convert them into posterior probabilities

$$\begin{aligned} P(C = c_i | \mathbf{X} = \mathbf{x}) &= \frac{P(\mathbf{X} = \mathbf{x} | C = c_i)P(C = c_i)}{P(\mathbf{X} = \mathbf{x})} \\ &\propto P(\mathbf{X} = \mathbf{x} | C = c_i)P(C = c_i) \\ &\quad \text{for } i = 1, 2, \dots, L \end{aligned}$$

- Then apply the MAP rule

Naïve Bayes Classifier

- Bayes classification

$$P(C | \mathbf{X}) \propto P(\mathbf{X} | C)P(C) = P(X_1, \dots, X_n | C)P(C)$$

Difficulty: learning the joint probability $P(X_1, \dots, X_n | C)$

- Naïve Bayes classification

- Assume that **all input features are conditionally independent!**

$$\begin{aligned} P(X_1, X_2, \dots, X_n | C) &= P(X_1 | X_2, \dots, X_n, C)P(X_2, \dots, X_n | C) \\ &= P(X_1 | C)P(X_2, \dots, X_n | C) \\ &= P(X_1 | C)P(X_2 | C) \cdots P(X_n | C) \end{aligned}$$

- MAP classification rule: for $\mathbf{x} = (x_1, x_2, \dots, x_n)$

$$[P(x_1 | c^*) \cdots P(x_n | c^*)]P(c^*) > [P(x_1 | c) \cdots P(x_n | c)]P(c), \quad c \neq c^*, c = c_1, \dots, c_L$$

Naïve Bayes Classifier (cont.)

- Algorithm: Discrete-Valued Features
 - Learning Phase: Given a training set S of F features and L classes,
For each target value of c_i ($c_i = c_1, \dots, c_L$)
 $\hat{P}(C = c_i) \leftarrow$ estimate $P(C = c_i)$ with examples in S ;
For every feature value x_{jk} of each feature X_j ($j = 1, \dots, F; k = 1, \dots, N_j$)
 $\hat{P}(X_j = x_{jk} | C = c_i) \leftarrow$ estimate $P(X_j = x_{jk} | C = c_i)$ with examples in S ;

Output: $F * L$ conditional probabilistic (generative) models
 - Test Phase: Given an unknown instance $\mathbf{X}' = (a'_1, \dots, a'_n)$
“Look up tables” to assign the label c^* to \mathbf{X}' if
$$[\hat{P}(a'_1 | c^*) \cdots \hat{P}(a'_n | c^*)] \hat{P}(c^*) > [\hat{P}(a'_1 | c) \cdots \hat{P}(a'_n | c)] \hat{P}(c), \quad c \neq c^*, c = c_1, \dots, c_L$$

Example

- Example: Play Tennis

PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example (cont.)

- Learning Phase

Outlook	Play=Yes	Play=No
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Temperature	Play=Yes	Play=No
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Humidity	Play=Yes	Play=No
High	3/9	4/5
Normal	6/9	1/5

Wind	Play=Yes	Play=No
Strong	3/9	3/5
Weak	6/9	2/5

$$P(\text{Play=Yes}) = 9/14 \quad P(\text{Play=No}) = 5/14$$

Example (cont.)

- Test Phase

- Given a new instance, predict its label

$\mathbf{x}' = (\text{Outlook}=\text{Sunny}, \text{Temperature}=\text{Cool}, \text{Humidity}=\text{High}, \text{Wind}=\text{Strong})$

- Look up tables achieved in the learning phrase

$$P(\text{Outlook}=\text{Sunny} \mid \text{Play}=\text{Yes}) = 2/9$$

$$P(\text{Outlook}=\text{Sunny} \mid \text{Play}=\text{No}) = 3/5$$

$$P(\text{Temperature}=\text{Cool} \mid \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Temperature}=\text{Cool} \mid \text{Play}=\text{No}) = 1/5$$

$$P(\text{Humidity}=\text{High} \mid \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Humidity}=\text{High} \mid \text{Play}=\text{No}) = 4/5$$

$$P(\text{Wind}=\text{Strong} \mid \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Wind}=\text{Strong} \mid \text{Play}=\text{No}) = 3/5$$

$$P(\text{Play}=\text{Yes}) = 9/14$$

$$P(\text{Play}=\text{No}) = 5/14$$

- Decision making with the MAP rule

$$P(\text{Yes} \mid \mathbf{x}') \approx [P(\text{Sunny} \mid \text{Yes})P(\text{Cool} \mid \text{Yes})P(\text{High} \mid \text{Yes})P(\text{Strong} \mid \text{Yes})]P(\text{Play}=\text{Yes}) = 0.0053$$

$$P(\text{No} \mid \mathbf{x}') \approx [P(\text{Sunny} \mid \text{No})P(\text{Cool} \mid \text{No})P(\text{High} \mid \text{No})P(\text{Strong} \mid \text{No})]P(\text{Play}=\text{No}) = 0.0206$$

Given the fact $P(\text{Yes} \mid \mathbf{x}') < P(\text{No} \mid \mathbf{x}')$, we label \mathbf{x}' to be “No”.



Linear Methods for Classification

- *Decision boundaries* are linear.
- Two class problem:
 - The decision boundary between the two classes is a *hyperplane* in the feature vector space,
 - A hyperplane in the p dimensional input space is the set:

$$\{x : \alpha_o + \sum_{j=1}^p \alpha_j x_j = 0\}$$

Linear Methods for Classification (cont.)

- The two regions separated by a hyperplane:

$$\{x : \alpha_o + \sum_{j=1}^p \alpha_j x_j > 0\} \qquad \{x : \alpha_o + \sum_{j=1}^p \alpha_j x_j < 0\}$$

- For more than two classes, the decision boundary between any pair of classes k and l is a hyperplane.
- Example methods for deciding the hyperplane:
 - Logistic regression
 - Linear discriminant analysis

Can we use Linear Regression?

- Here, each of the response categories are coded via an indicator variable (giving us an *indicator response matrix*).
- If G has K classes, there will be K such indicators Y_k , $k = 1, \dots, K$.

g	y₁	y₂	y₃	y₄
3	0	0	1	0
1	1	0	0	0
2	0	1	0	0
4	0	0	0	1
1	1	0	0	0

Can we use Linear Regression?

- Fit a linear regression model for each Y_k , $k = 1, 2, \dots, K$, using X :

$$\hat{Y}_k = X(X^T X)^{-1} X^T Y_k$$

- Define $Y = (Y_1, Y_2, \dots, Y_K) \rightarrow \hat{Y} = X(X^T X)^{-1} X^T Y$

- Classification procedure:

- Define $\hat{B} = (X^T X)^{-1} X^T Y$
- Compute the fitted output $\hat{f}(x)^T = (1, x^T) \hat{B}$
- Identify the largest component and classify accordingly,

$$\hat{G}(x) = \operatorname{argmax}_{k \in \mathcal{G}} \hat{f}_k(x)$$

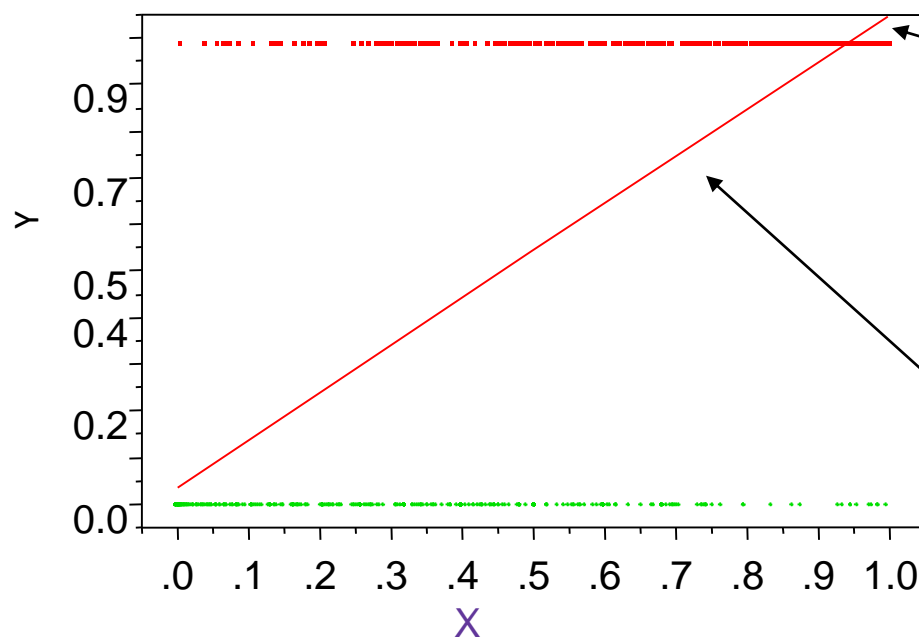
Can we use Linear Regression?

- One rather formal justification is to view the linear regression as an estimate of conditional expectation.
- However, how good an approximation to conditional expectation is the rather rigid linear regression model?
- The key issue is that the fitted values can be negative or greater than 1; this is a consequence of the rigid nature of linear regression, especially if we make predictions outside the domain of the training data.



Can we use Linear Regression?

- When Y only takes on values of 0 and 1, we can see why is OLS linear regression is often not appropriate.



How do we interpret values greater than 1?

How do we interpret values of Y between 0 and 1?

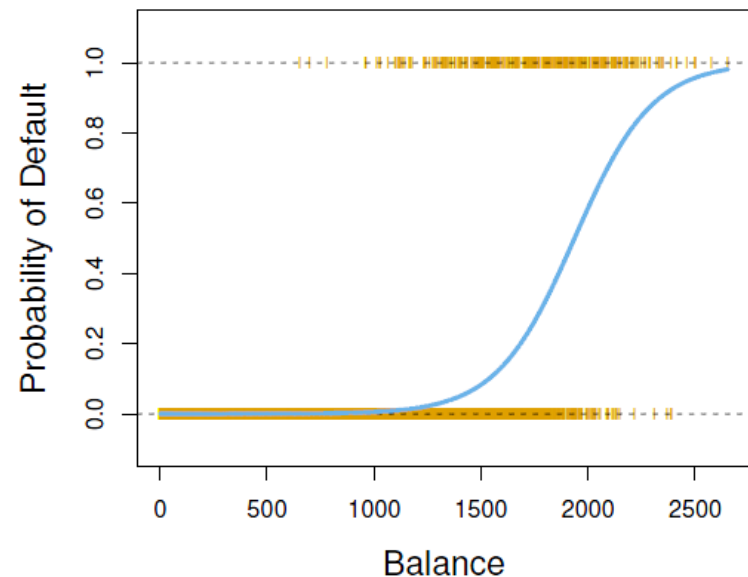
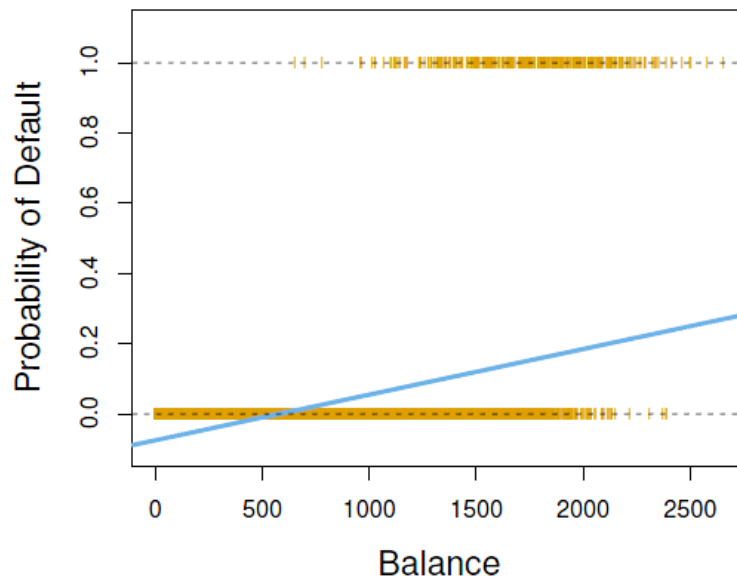
Linear vs. Logistic Regression

- Linear regression might produce probabilities less than zero or bigger than one because the regression line can take on any value between negative and positive infinity.
- Thus, the regression line almost always predicts the wrong value for Y in classification problems.
- *Logistic regression* is more appropriate, especially because we are interested in estimating the *probabilities* that X belong to each category, or class.



Linear vs. Logistic Regression (cont.)

- Below, the orange marks indicate the response Y , either 0 or 1. Linear regression does not estimate $\Pr(Y = 1 | X)$ well, so logistic regression seems well suited to the task.



Logistic Regression

- There are two big branches of methods for classification. One is called *generative* modeling (which we saw with Naïve Bayes classification), and the other is called *discriminative* modeling.
 - A **generative** model learns the joint probability distribution.
 - A **discriminative** model learns the conditional probability distribution.
- *Logistic regression* for classification is a discriminative modeling approach, where we estimate the *posterior probabilities* of classes given X directly without assuming the marginal distribution on X .
- As a result, this method preserves linear classification boundaries.

Logistic Regression (cont.)

- A review of Bayes rule shows us that when we use 0-1 loss, we pick the class k that has the maximum posterior probability:

$$\hat{G}(x) = \arg \max_k Pr(G = k|X = x)$$

- The decision boundary between classes k and l is determined by:

$$Pr(G = k|X = x) = Pr(G = l|X = x)$$

- That is, the x 's at which the two posterior probabilities of k and l are equal.

Logistic Regression (cont.)

- If we divide both sides by $Pr(G = l | X = x)$ and take the log of this ratio, the previous equation is equivalent to:

$$\log \frac{Pr(G = k | X = x)}{Pr(G = l | X = x)} = 0$$

- Since we want to enforce a linear classification boundary, we assume the function above is linear:

$$\log \frac{Pr(G = k | X = x)}{Pr(G = l | X = x)} = a_0^{(k,l)} + \sum_{j=1}^p a_j^{(k,l)} x_j$$

- This is the basic assumption of logistic regression.

Logistic Regression (cont.)

- We use the superscript (k, l) on the coefficients of the linear function because for every pair of k and l , the decision boundary would be different, determined by the different coefficients.
- For logistic regression, there are restrictive relations between $a^{(k,l)}$ for different pairs of (k, l) .
- We do not really need to specify this equation for every pair of k and l , but instead we only need to specify it for $K - 1$ such pairs.



Logistic Regression (cont.)

- If we take class K as the base class, the assumed equations are:

$$\begin{aligned}\log \frac{Pr(G=1|X=x)}{Pr(G=K|X=x)} &= \beta_{10} + \beta_1^T x \\ \log \frac{Pr(G=2|X=x)}{Pr(G=K|X=x)} &= \beta_{20} + \beta_2^T x \\ &\vdots \\ \log \frac{Pr(G=K-1|X=x)}{Pr(G=K|X=x)} &= \beta_{(K-1)0} + \beta_{K-1}^T x\end{aligned}$$

- This indicates that we do not have to specify the decision boundary for every pair of classes. We only need to specify the decision boundary between class j and the base class K .

Logistic Regression (cont.)

- Once we have specified the parameters for these $K - 1$ log ratios, then for any pair of classes (k, l) , we can derive the log ratios without introducing new parameters:

$$\log \frac{Pr(G = k|X = x)}{Pr(G = l|X = x)} = \beta_{k0} + \beta_{l0} + (\beta_k - \beta_l)^T x$$

- Number of parameters: $(K - 1)(p + 1)$
- We denote the entire parameter set by θ and arrange them as:

$$\theta = \{\beta_{10}, \beta_1^T, \beta_{20}, \beta_2^T, \dots, \beta_{(K-1)0}, \beta_{K-1}^T\}$$

Logistic Regression (cont.)

- The log ratios of posterior probabilities are called *log-odds* or *logit* transformations.
- Under the previously stated assumptions, the posterior probabilities are given by the following two equations:

$$Pr(G = k|X = x) = \frac{\exp(\beta_{k0} + \beta_k^T x)}{1 + \sum_{l=1}^{K-1} \exp(\beta_{l0} + \beta_l^T x)} \text{ for } k = 1, \dots, K - 1$$

$$Pr(G = K|X = x) = \frac{1}{1 + \sum_{l=1}^{K-1} \exp(\beta_{l0} + \beta_l^T x)}$$

Logistic Regression (cont.)

- For $\Pr(G = k \mid X = x)$ given previously:
 - These must sum to 1: $\sum_{k=1}^K \Pr(G = k \mid X = x) = 1$
- Similarities with linear regression on indicators:
 - Both attempt to estimate $\Pr(G = k \mid X = x)$, both have linear classification boundaries, and the posterior probabilities sum to 1 across classes
- Differences with linear regression on indicators:
 - For linear regression, approximate $\Pr(G = k \mid X = x)$ by a linear function of x ; it is not guaranteed to fall between 0 and 1.
 - For logistic regression, $\Pr(G = k \mid X = x)$ is a nonlinear (sigmoid) function of x ; it is guaranteed to range from 0 to 1.

Logistic Regression (cont.)

- How do we estimate the parameters and how do we fit a logistic regression model?
- What we want to do is find parameters that *maximize* the conditional *likelihood* of class labels G given X using the training data.
- We are not interested in the distribution of X , but our focus is on the conditional probabilities of the class labels given X .



Logistic Regression (cont.)

- Given point x_i , the posterior probability for the class to be k is:

$$p_k(x_i; \theta) = \Pr(G = k | X = x_i; \theta)$$

- Given the first input x_1 , the posterior probability of its class, denoted as g_1 , is computed by: $\Pr(G = g_1 | X = x_1)$
- Since samples in the training data are summed independent, the posterior probability for the N sample points each have class g_i , $i = 1, 2, \dots, N$, given their inputs x_1, x_2, \dots, x_N is:

$$\prod_{i=1}^N \Pr(G = g_i | X = x_i)$$

Logistic Regression (cont.)

- The joint conditional likelihood is the product of the conditional probabilities of the classes given every data point.
- Thus, the conditional *log-likelihood* of the class labels in the training data set becomes a summation:

$$\begin{aligned} l(\theta) &= \sum_{i=1}^N \log \Pr(G = g_i | X = x_i) \\ &= \sum_{i=1}^N \log p_{g_i}(x_i; \theta) \end{aligned}$$

Logistic Regression (cont.)

- We discuss in detail the two-class case (i.e. binary classification), since the algorithms simplify considerably.
- It is convenient to code the two-class g_i via a 0/1 response y_i , where $y_i = 1$ when $g_i = \text{class 1}$, and $y_i = 0$ when $g_i = \text{class 2}$.
- Let $p_1(x; \theta) = p(x; \theta)$ and let $p_2(x; \theta) = 1 - p_1(x; \theta) = 1 - p(x; \theta)$ because the posterior probabilities of the two classes must sum up to one.
- Since $K = 2$, there is only one decision boundary between the two classes.



Logistic Regression (cont.)

- Thus, the log-likelihood function can be written as:

$$\begin{aligned} l(\beta) &= \sum_{i=1}^N \log p_{g_i}(x_i; \beta) \\ &= \sum_{i=1}^N [y_i \log p(x_i; \beta) + (1 - y_i) \log (1 - p(x_i; \beta))] \\ &= \sum_{i=1}^N \left[y_i \beta^T x_i - \log (1 + e^{\beta^T x_i}) \right] \end{aligned}$$

- There are $p + 1$ parameters in $\beta = \{\beta_{10}, \beta_1\}$, and we assume that the vector of inputs x_i includes the constant term 1 to accommodate the intercept.

Logistic Regression (cont.)

- If we want to maximize the log-likelihood function, we set the first order partial derivatives of the function $l(\beta)$ with respect to β_{1j} to zero for all $j = 0, 1, \dots, p$:

$$\begin{aligned}\frac{\partial l(\beta)}{\beta_{1j}} &= \sum_{i=1}^N y_i x_{ij} - \sum_{i=1}^N \frac{x_{ij} e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \\ &= \sum_{i=1}^N y_i x_{ij} - \sum_{i=1}^N p(x; \beta) x_{ij} \\ &= \sum_{i=1}^N x_{ij} (y_i - p(x_i; \beta))\end{aligned}$$

- In matrix form:
$$\frac{\partial l(\beta)}{\beta_{1j}} = \sum_{i=1}^N x_i (y_i - p(x_i; \beta))$$

Logistic Regression (cont.)

- To solve the set of $p + 1$ nonlinear equations, we use the Newton-Raphson algorithm.
- This algorithm requires the second-order derivatives (i.e. Hessian matrix), given as follows:

$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} = - \sum_{i=1}^N x_i x_i^T p(x_i; \beta) (1 - p(x_i; \beta))$$

Logistic Regression (cont.)

- We now derive the Hessian matrix, where the element on the j th row and n th column is (counting from 0):

$$\begin{aligned} & \frac{\partial^2 l(\beta)}{\partial \beta_{1j} \partial \beta_{1n}} \\ &= - \sum_{i=1}^N \frac{(1 + e^{\beta^T x_i}) e^{\beta^T x_i} x_{ij} x_{in} - (e^{\beta^T x_i})^2 x_{ij} x_{in}}{(1 + e^{\beta^T x_i})^2} \\ &= - \sum_{i=1}^N x_{ij} x_{in} p(x_i; \beta) - x_{ij} x_{in} p(x_i; \beta)^2 \\ &= - \sum_{i=1}^N x_{ij} x_{in} p(x_i; \beta) (1 - p(x_i; \beta)) \end{aligned}$$

Logistic Regression (cont.)

- Starting with β^{old} , a single Newton-Raphson update is given by this matrix formula:

$$\beta^{new} = \beta^{old} - \left(\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} \right)^{-1} \frac{\partial l(\beta)}{\partial \beta}$$

where the derivatives are evaluated at β^{old} .

- If given an old set of parameters, we update the new set of parameters by taking β^{old} minus the inverse of the Hessian matrix times the first-order derivative vector.

Logistic Regression (cont.)

- This procedure can also be expressed compactly in matrix form.
- Let \mathbf{y} be the column vector of y_i , let \mathbf{X} be the $N \times (p + 1)$ input matrix, let \mathbf{p} be the N -vector of fitted probabilities, and let \mathbf{W} be an $N \times N$ diagonal matrix of weights.
- Then, the first order derivative vector is:
$$\frac{\partial l(\beta)}{\partial \beta} = \mathbf{X}^T (\mathbf{y} - \mathbf{p})$$
- The Hessian matrix is:
$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} = -\mathbf{X}^T \mathbf{W} \mathbf{X}$$

Logistic Regression (cont.)

- The Newton-Raphson step is:

$$\begin{aligned}\beta^{new} &= \beta^{old} + (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{y} - \mathbf{p}) \\ &= (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} (\mathbf{X} \beta^{old} + \mathbf{W}^{-1} (\mathbf{y} - \mathbf{p})) \\ &= (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{z}\end{aligned}$$

where $\mathbf{z} \triangleq \mathbf{X} \beta^{old} + \mathbf{W}^{-1} (\mathbf{y} - \mathbf{p})$

- If \mathbf{z} is viewed as a response and \mathbf{X} is the input matrix, then β^{new} is the solution to a weighted least squares problem:

$$\beta^{new} \leftarrow \arg \min_{\beta} (\mathbf{z} - \mathbf{X} \beta)^T \mathbf{W} (\mathbf{z} - \mathbf{X} \beta)$$

Logistic Regression (cont.)

- Recall that linear regression by OLS solves:

$$\min_{\beta} (\mathbf{z} - \mathbf{X}\beta)^T \mathbf{W} (\mathbf{z} - \mathbf{X}\beta)$$

- \mathbf{Z} is referred to as the *adjusted response*, and the algorithm is referred to as *iteratively reweighted least squares* (IRLS).
- The pseudo code of a relative computationally efficient algorithm for IRLS is provided on the next slide.

Logistic Regression (cont.)

1. $0 \rightarrow \beta$
2. Compute y by setting its elements to:

$$y_i = \begin{cases} 1 & \text{if } g_i = 1 \\ 0 & \text{if } g_i = 2 \end{cases}$$

3. Compute p by setting its elements to:

$$p(x_i; \beta) = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \quad i = 1, 2, \dots, N$$

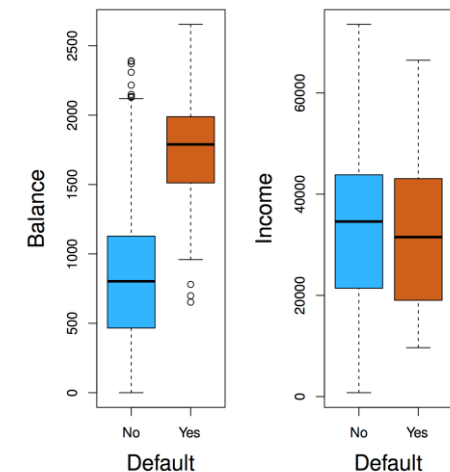
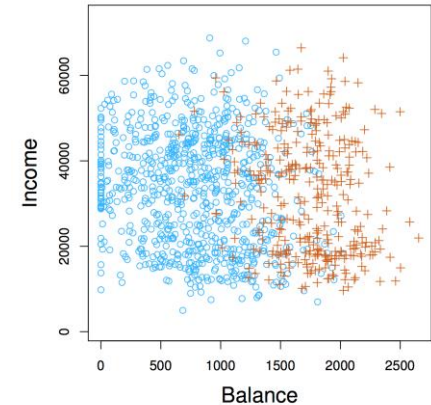
4. Compute the $N \times (p + 1)$ matrix $\tilde{\mathbf{X}}$ by multiplying the i th row of matrix \mathbf{X} by $p(x_i; \beta)(1 - p(x_i; \beta))$, $i = 1, 2, \dots, N$:

$$\mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \dots \\ x_N^T \end{pmatrix} \quad \tilde{\mathbf{X}} = \begin{pmatrix} p(x_1; \beta)(1 - p(x_1; \beta))x_1^T \\ p(x_2; \beta)(1 - p(x_2; \beta))x_2^T \\ \dots \\ p(x_N; \beta)(1 - p(x_N; \beta))x_N^T \end{pmatrix}$$

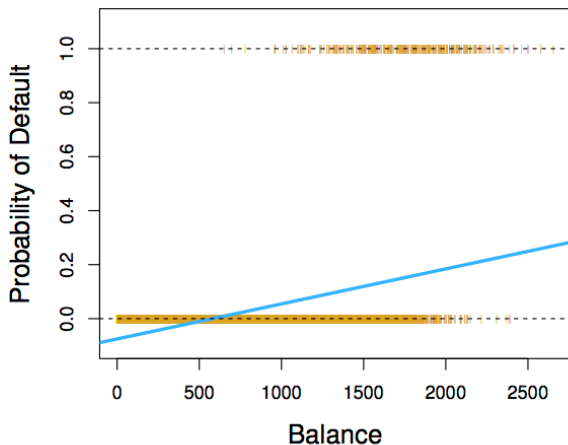
5. $\beta \leftarrow \beta + (\mathbf{X}^T \tilde{\mathbf{X}})^{-1} \mathbf{X}^T (\mathbf{y} - \mathbf{p})$.
6. If the stopping criteria is met, stop; otherwise go back to step 3.

Logistic Regression: Credit Card Default Example

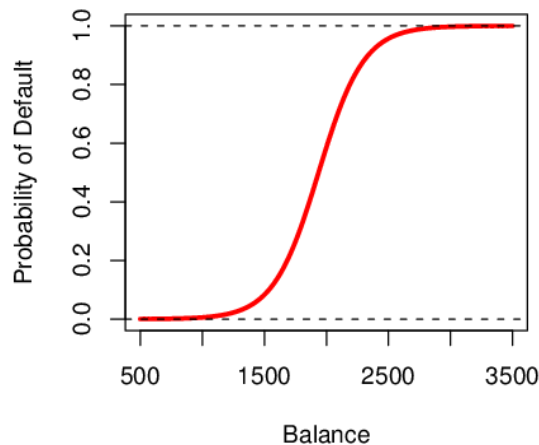
- We would like to be able to predict customers that are likely to default on their credit card.
- Possible X variables:
 - Annual Income
 - Monthly Credit Card Balance
- The Y variable (Default) is categorical: Yes or No



Logistic Regression: Credit Card Default Example (cont.)



- If we fit a linear regression model to the Default data, then:
 - For very low balances we predict a negative probability!
 - For high balances we predict a probability above 1!
- If we fit a logistic regression model, then the probability of default is between 0 and 1 for all balances.



$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} \quad \log \left(\frac{p(X)}{1 - p(X)} \right) = \beta_0 + \beta_1 X$$

Logistic Regression: Credit Card Default Example (cont.)

- Interpreting what β_1 means is not very easy with logistic regression, simply because we are predicting $\Pr(Y)$ and not Y .
- In a logistic regression model, increasing X by one unit changes the *log odds* by β_1 , or equivalently it multiplies the odds by e^{β_1} .
- If $\beta_1 = 0$, then there is no relationship between Y and X . If $\beta_1 > 0$, then when X gets larger so does the probability that $Y = 1$. If $\beta_1 < 0$, then when X gets larger the probability that $Y = 1$ gets smaller.
- How much increase or decrease in probability depends on the slope.



Logistic Regression: Credit Card Default Example (cont.)

- We still want to perform a hypothesis test to see whether we can be sure that θ_0 and θ_1 are significantly different from zero.
- We use a Z test and interpret the p-value as usual.
- In this example, the p-value for balance is very small and $\hat{\beta}_1$ is positive. This means that if the balance increases, then the probability of default will increase as well.

	Coefficient	Std. Error	Z-statistic	P-value
Intercept	-10.6513	0.3612	-29.5	< 0.0001
balance	0.0055	0.0002	24.9	< 0.0001

Logistic Regression: Credit Card Default Example (cont.)

- What is the estimated probability of default for someone with a balance of \$1000?

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 X}} = \frac{e^{-10.6513 + 0.0055 \times 1000}}{1 + e^{-10.6513 + 0.0055 \times 1000}} = 0.006$$

- What is the estimated probability of default for someone with a balance of \$2000?

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 X}} = \frac{e^{-10.6513 + 0.0055 \times 2000}}{1 + e^{-10.6513 + 0.0055 \times 2000}} = 0.586$$

Logistic Regression: Credit Card Default Example (cont.)

- We can also use student (0,1) as a predictor to estimate the probability that an individual defaults:

	Coefficient	Std. Error	Z-statistic	P-value
Intercept	-3.5041	0.0707	-49.55	< 0.0001
student [Yes]	0.4049	0.1150	3.52	0.0004

$$\widehat{\Pr}(\text{default}=\text{Yes}|\text{student}=\text{Yes}) = \frac{e^{-3.5041+0.4049 \times 1}}{1 + e^{-3.5041+0.4049 \times 1}} = 0.0431,$$

$$\widehat{\Pr}(\text{default}=\text{Yes}|\text{student}=\text{No}) = \frac{e^{-3.5041+0.4049 \times 0}}{1 + e^{-3.5041+0.4049 \times 0}} = 0.0292.$$

Multiple Logistic Regression

- Note that we use the maximum likelihood method (as previously discussed) to estimate the beta coefficients when considering the problem of predicting a binary response using multiple predictors.

$$\log \left(\frac{p(X)}{1 - p(X)} \right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}}$$

L_1 Regularized Logistic Regression

- For logistic regression, we would maximize a penalized version of the log-likelihood function:

$$\max_{\beta_0, \beta} \left\{ \sum_{i=1}^N \left[y_i(\beta_0 + \beta^T x_i) - \log(1 + e^{\beta_0 + \beta^T x_i}) \right] - \lambda \sum_{j=1}^p |\beta_j| \right\}$$

- As with the lasso method, we typically do not penalize the intercept term, and we standardize the predictors for the penalty to be meaningful.
- Because this optimization criterion is concave, a solution can be found using nonlinear programming methods or via quadratic approximations as used in the Newton-Raphson algorithm.

Multi-Class Logistic Regression

- We sometimes wish to classify a response variable that has more than two classes.
- The two-class logistic regression models have multiple class extensions.
- This method is often not used in practice because *discriminant analysis* is a more popular method for multi-class classification.

Multi-Class Logistic Regression (cont.)

- When the number of classes $K > 2$, the parameter β is a $(K - 1)(p + 1)$ vector:

- Also, let

$$\bar{\beta}_l = \begin{pmatrix} \beta_{10} \\ \beta_l \end{pmatrix}$$

$$\beta = \begin{pmatrix} \beta_{10} \\ \beta_1 \\ \beta_{20} \\ \beta_2 \\ \vdots \\ \beta_{(K-1)0} \\ \beta_{K-1} \end{pmatrix} = \begin{pmatrix} \beta_{10} \\ \beta_{11} \\ \vdots \\ \beta_{1p} \\ \beta_{20} \\ \vdots \\ \beta_{2p} \\ \vdots \\ \beta_{(K-1)0} \\ \vdots \\ \beta_{(K-1)p} \end{pmatrix}$$

Multi-Class Logistic Regression (cont.)

- The log-likelihood function becomes:

$$\begin{aligned} l(\beta) &= \sum_{i=1}^N \log p_{g_i}(x_i; \beta) \\ &= \sum_{i=1}^N \log \left(\frac{e^{\bar{\beta}_{g_i}^T x_i}}{1 + \sum_{l=1}^{K-1} e^{\bar{\beta}_{g_l}^T x_i}} \right) \\ &= \sum_{i=1}^N \left[\bar{\beta}_{g_i}^T x_i - \log \left(1 + \sum_{l=1}^{K-1} e^{\bar{\beta}_{g_l}^T x_i} \right) \right] \end{aligned}$$

Multi-Class Logistic Regression (cont.)

- The indicator function $I(.)$ equals 1 when the argument is *true* and 0 *otherwise*.
- Thus, the first order derivatives of the log-likelihood function are:

$$\begin{aligned}\frac{\partial l(\beta)}{\partial \beta_{kj}} &= \sum_{i=1}^N \left[I(g_i = k) x_{ij} - \frac{e^{\bar{\beta}_k^T x_i} x_{ij}}{1 + \sum_{l=1}^{K-1} e^{\bar{\beta}_l^T x_i}} \right] \\ &= \sum_{i=1}^N x_{ij} (I(g_i = k) - p_k(x_i; \beta))\end{aligned}$$

Multi-Class Logistic Regression (cont.)

- The second order derivatives of the log-likelihood function are:

$$\begin{aligned}\frac{\partial^2 l(\beta)}{\partial \beta_{kj} \partial \beta_{mn}} &= \sum_{i=1}^N x_{ij} \cdot \frac{1}{(1 + \sum_{l=1}^{K-1} e^{\tilde{\beta}_l^T x_i})^2} \cdot \left[-e^{\tilde{\beta}_k^T x_i} I(k=m) x_{in} (1 + \sum_{l=1}^{K-1} e^{\tilde{\beta}_l^T x_i}) + e^{\tilde{\beta}_k^T x_i} x_{in} \right] \\ &= \sum_{i=1}^N x_{ij} x_{in} (-p_k(x_i; \beta) I(k=m) + p_k(x_i; \beta) p_m(x_i; \beta)) \\ &= - \sum_{i=1}^N x_{ij} x_{in} p_k(x_i; \beta) [I(k=m) - p_m(x_i; \beta)]\end{aligned}$$

- We next express the solution in matrix form because it is much more compact. First, we introduce several notations.

Multi-Class Logistic Regression (cont.)

- \mathbf{y} is the concatenated indicator vector, a column vector of dimension $N(K - 1)$.

$$\mathbf{y} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_{K-1} \end{pmatrix} \quad \mathbf{y}_k = \begin{pmatrix} I(g_1 = k) \\ I(g_2 = k) \\ \vdots \\ I(g_N = k) \end{pmatrix} \quad 1 \leq k \leq K - 1$$

- \mathbf{p} is the concatenated vector of fitted probabilities, a column vector of dimension $N(K - 1)$.

$$\mathbf{p} = \begin{pmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_{K-1} \end{pmatrix} \quad \mathbf{p}_k = \begin{pmatrix} p_k(x_1; \beta) \\ p_k(x_2; \beta) \\ \vdots \\ p_k(x_N; \beta) \end{pmatrix} \quad 1 \leq k \leq K - 1$$

Multi-Class Logistic Regression (cont.)

- $\tilde{\mathbf{X}}$ is an $N(K - 1) \times (p + 1)(K - 1)$ matrix, where \mathbf{X} is the $N(p + 1)$ input matrix defined before:

$$\tilde{\mathbf{X}} = \begin{pmatrix} \mathbf{X} & 0 & \dots & 0 \\ 0 & \mathbf{X} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \mathbf{X} \end{pmatrix}$$

- Matrix \mathbf{W} is an $N(K - 1) \times N(K - 1)$ square matrix:

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_{11} & \mathbf{W}_{12} & \dots & \mathbf{W}_{1(K-1)} \\ \mathbf{W}_{21} & \mathbf{W}_{22} & \dots & \mathbf{W}_{2(K-1)} \\ \dots & \dots & \dots & \dots \\ \mathbf{W}_{(K-1),1} & \mathbf{W}_{(K-1),2} & \dots & \mathbf{W}_{(K-1),(K-1)} \end{pmatrix}$$

Multi-Class Logistic Regression (cont.)

- Each submatrix \mathbf{W}_{km} , $1 \leq k, m \leq K - 1$, is an $N \times N$ diagonal matrix.
- When $k = m$, the i th diagonal element in \mathbf{W}_{kk} is $p_k(x_i; \beta^{old})(1 - p_k(x_i; \beta^{old}))$.
- – When $k \neq m$, the i th diagonal element in \mathbf{W}_{km} is $-p_k(x_i; \beta^{old})(1 - p_m(x_i; \beta^{old}))$.

- As with binary classification:

$$\frac{\partial l(\beta)}{\partial \beta} = \tilde{\mathbf{X}}^T (\mathbf{y} - \mathbf{p})$$
$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} = \tilde{\mathbf{X}}^T \mathbf{W} \tilde{\mathbf{X}}$$

Multi-Class Logistic Regression (cont.)

- The formula for updating β^{new} in the binary classification case holds for multi-class, except that the definitions of the matrices are more complicated.
- The formula itself may look simple because the complexity is wrapped in the definitions of the matrices.

$$\beta^{new} = (\tilde{\mathbf{X}}^T \mathbf{W} \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{W} \mathbf{z}$$

where $\mathbf{z} \triangleq \tilde{\mathbf{X}} \beta^{old} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$ or simply:

$$\beta^{new} = \beta^{old} + (\tilde{\mathbf{X}}^T \mathbf{W} \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T (\mathbf{y} - \mathbf{p})$$

Discriminant Analysis

- Let the feature vector be X and the class labels be Y .
- The Bayes rule says that if you have the joint distribution of X and Y , and if X is given, under 0-1 loss, the optimal decision on Y is to choose a class with maximum posterior probability given X .
- *Discriminant analysis* belongs to the branch of classification methods called generative modeling, where we try to estimate the within class density of X given the class label.
- Combined with the prior probability (unconditioned probability) of classes, the posterior probability of Y can be obtained by the Bayes formula.

Discriminant Analysis (cont.)

- Assume the *prior* probability or the marginal probability mass function for class k is denoted as π_k , $\sum_{k=1}^K \pi_k = 1$
- π_k is the probability that a given observation is associated with the k th category of the response variable Y .
- Note that π_k is usually estimated simply by empirical frequencies of the training set:

$$\hat{\pi}_k = \frac{\text{\# of Samples in class } k}{\text{Total \# of samples}}$$

Discriminant Analysis (cont.)

- You have the training data set and you count what percentage of data come from a certain class.
- Then we need the class-conditional density of X .
- This is the *density function* of X conditioned on the class k , or class $G = k$ denoted by $f_k(x)$.
- Note that $f_k(x)$ is relatively large if there is a high probability that an observation in the k th class has $X \approx x$, and $f_k(x)$ is relatively small if it is very unlikely that an observation in the k th class has $X \approx x$.



Discriminant Analysis (cont.)

- According to the Bayes rule, we can now compute the *posterior probability* that an observation $X = x$ belongs to the k th class:

$$Pr(G = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{l=1}^K f_l(x)\pi_l}$$

- This is a *conditional* probability of class G (or denoted Y in the book) given X (the predictor value for that observation).

- By the Bayes rule for 0-1 loss:
- $$\begin{aligned}\hat{G}(x) &= \arg \max_k Pr(G = k|X = x) \\ &= \arg \max_k f_k(x)\pi_k\end{aligned}$$

Discriminant Analysis (cont.)

- Notice that the denominator is identical no matter what class k you are using. Thus, for maximization, it does not make a difference in the choice of k .
- The *maximum a posteriori* rule (i.e. Bayes rule for 0-1 loss) is essentially trying to maximize π_k times $f_k(x)$.
- Note that the Bayes classifier, which classifies an observation to the class for which the posterior probability is largest, has the lowest possible error rate out of all classifiers.



Class Density Estimation

- Depending on which algorithms you use, you end up with different ways of *density estimation* within every class.
- In both **Linear Discriminant Analysis** (LDA) and **Quadratic Discriminant Analysis** (QDA), we assume that every density within each class is a Gaussian distribution.
- In LDA we assume those Gaussian distributions for different classes share the same covariance structure. In QDA, however, we do not have such a constraint.



Class Density Estimation (cont.)

- You can also use **general nonparametric density estimates**, for instance kernel estimates and histograms.
- **Naive Bayes**: assume each of the class densities are products of marginal densities, that is, all the variables are independent.
- In the well-known Naive Bayes algorithm, you can separately estimate the density (or probability mass function for discrete values of X) for every dimension and then multiply them to get the joint density (or joint probability mass function).

Linear Discriminant Analysis

- LDA undertakes the same task as logistic regression in that it classifies data based on categorical variables.
- When the classes are well separated, the parameter estimates for the logistic regression model are surprisingly unstable, where LDA does not suffer from this problem.
- Under LDA, we assume that the density for X given every class k is following a Gaussian distribution.

Linear Discriminant Analysis (cont.)

- The following is the density formula for a multivariate Gaussian distribution:

$$f_k(x) = \frac{1}{(2\pi)^{p/2} |\Sigma_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1} (x-\mu_k)}$$

where p is the dimension and Σ_k is the covariance matrix.

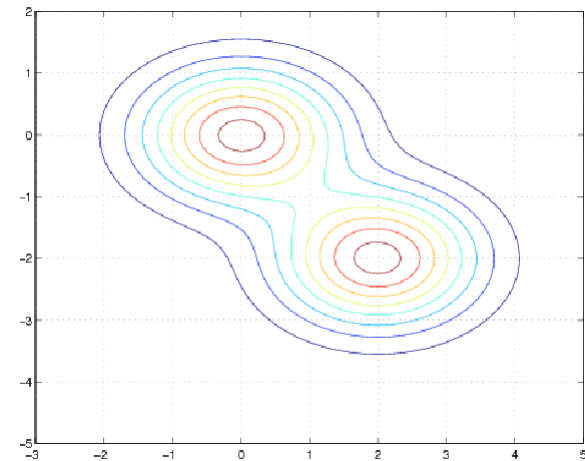
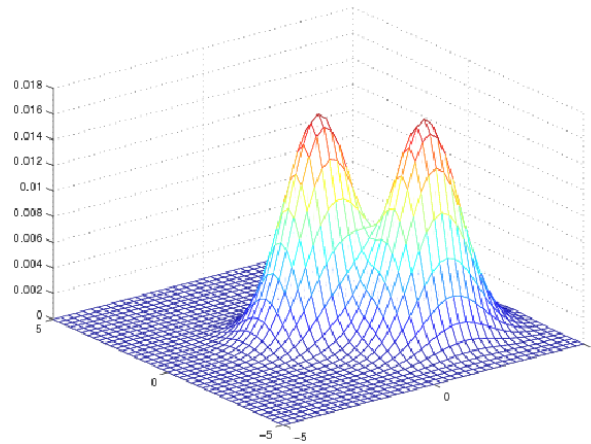
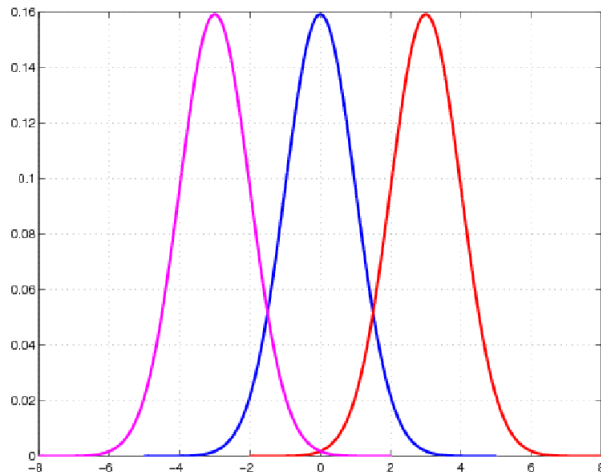
- This involves the square root of the determinant of this matrix. In this case, we are doing matrix multiplication.
- The vector x and the mean vector μ_k are both column vectors.

Linear Discriminant Analysis (cont.)

- For LDA, the covariance matrix $\Sigma_k = \Sigma \forall k$
- In LDA, you simply assume for different k that the covariance matrix is identical.
- By making this assumption, the classifier becomes linear. The only difference from QDA is that we do not assume that the covariance matrix is identical for different classes.
- For QDA, the decision boundary is determined by a quadratic function.

Linear Discriminant Analysis (cont.)

- Since the covariance matrix determines the shape of the Gaussian density, in LDA, the Gaussian densities for different classes have the same shape, but are shifted versions of each other (different mean vectors).



Linear Discriminant Analysis (cont.)

- For the moment, we will assume that we already have the covariance matrix for every class.
- In terms of optimal classification, Bayes rule says that we should pick a class that has the maximum posterior probability given the feature vector X .
- If we are using the generative modeling approach, this is equivalent to maximizing the product of the prior and the within class density.

Linear Discriminant Analysis (cont.)

- Since the log function is an increasing function, the maximization is equivalent because whatever gives you the maximum should also give you a maximum under a log function.
- Thus, we plug in the density of the Gaussian distribution assuming common covariance and then multiplying the prior probabilities:

$$\begin{aligned}\hat{G}(x) &= \arg \max_k Pr(G = k | X = x) \\ &= \arg \max_k f_k(x) \pi_k \\ &= \arg \max_k \log(f_k(x) \pi_k) \\ &= \arg \max_k \left[-\log((2\pi)^{p/2} |\Sigma|^{1/2}) - \frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k) + \log(\pi_k) \right] \\ &= \arg \max_k \left[-\frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k) + \log(\pi_k) \right]\end{aligned}$$

Linear Discriminant Analysis (cont.)

- Note that: $-\frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k - \frac{1}{2} x^T \Sigma^{-1} x$
- After simplification, we obtain the Bayes classifier:

$$\hat{G}(x) = \arg \max_k \left[x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log(\pi_k) \right]$$

- Given any x , you simply plug into this formula and see which k maximizes this.
- Usually the number of classes is pretty small, and very often only two classes, so an exhaustive search over the classes is effective.

Linear Discriminant Analysis (cont.)

- LDA provides a linear boundary because the quadratic term is dropped.

- Thus, we define the linear discriminant function as:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log(\pi_k)$$

$$\hat{G}(x) = \arg \max_k \delta_k(x)$$

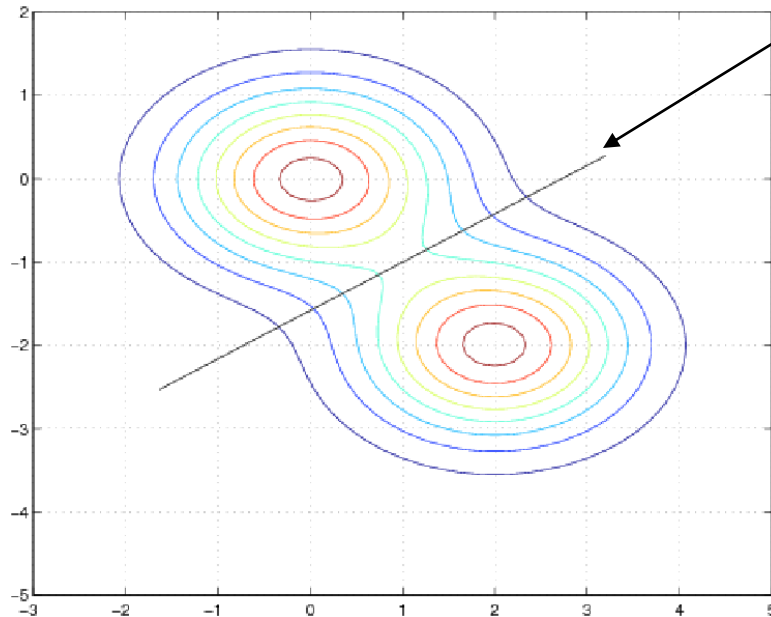
- The decision boundary between class k and l is $\{x : \delta_k(x) = \delta_l(x)\}$
or equivalently $\log \frac{\pi_k}{\pi_l} - \frac{1}{2} (\mu_k + \mu_l)^T \Sigma^{-1} (\mu_k - \mu_l) + x^T \Sigma^{-1} (\mu_k - \mu_l) = 0$

Linear Discriminant Analysis (cont.)

- In binary classification, for instance if we let $(k = 1, l = 2)$, then we would define constant a_0 , where π_1 and π_2 are prior probabilities for the two classes and μ_1 and μ_2 are mean vectors.
- Define: $a_0 = \log \frac{\pi_1}{\pi_2} - \frac{1}{2} (\mu_1 + \mu_2)^T \Sigma^{-1} (\mu_1 - \mu_2)$
- Define: $(a_1, a_2, \dots, a_p)^T = \Sigma^{-1} (\mu_1 - \mu_2)$
- Classify to class 1 if $a_0 + \sum_{j=1}^p a_j x_j > 0$ or to class 2 otherwise.

Linear Discriminant Analysis (cont.)

- For example:



Decision Boundary

$$*\pi_1 = \pi_2 = 0.5$$

$$*\mu_1 = (0, 0)^T, \mu_2 = (2, -2)^T$$

$$*\Sigma = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 0.5625 \end{pmatrix}$$

$$*\text{Decision boundary: } 5.56 - 2.00x_1 + 3.56x_2 = 0.0$$

- We have two classes and we know the within class density. The marginal density is simply the weighted sum of the within class densities, where the weights are the prior probabilities.

Linear Discriminant Analysis (cont.)

- Because we have equal weights and because the covariance matrix two classes are identical, we get these symmetric lines in the contour plot.
- The black diagonal line is the decision boundary for the two classes.
- If you are given an x , if it is above the line we would be classifying this x into the first-class. If it is below the line, it would be classified into the second class.

Linear Discriminant Analysis (cont.)

- In this example, we assumed that we have the prior probabilities for the classes and we also had the within class densities given to us.
- In practice, however, we do not have this. All we have is a set of training data.
- Thus, how do we find the π_k 's and the $f_k(x)$?
- We need to estimate the Gaussian distribution!!

Linear Discriminant Analysis (cont.)

- Here is the formula for estimating the π_k 's and the parameters in the Gaussian distributions.

- The formula below is actually the maximum likelihood estimator:

$$\hat{\pi}_k = N_k / N$$

where N_k is the number of class- k samples and N is the total number of points in the training data.

- To get the prior probabilities for class k , we simply count the frequency of data points in class k .

Linear Discriminant Analysis (cont.)

- The mean vector for every class is also simple. We take all of the data points in a given class and compute the sample mean:

$$\hat{\mu}_k = \sum_{g_i=k} x^{(i)} / N_k$$

- The covariance matrix formula looks slightly complicated. The reason is because we have to get a common covariance matrix for all of the classes.
- First, we divide the data points in two given classes according to the given labels.

Linear Discriminant Analysis (cont.)

- If we were looking at class k , for every point we subtract the corresponding mean which we computed earlier. Then multiply its transpose.
- Remember x is a column vector, therefore if we have a column vector multiplied by a row vector, we get a square matrix, which is what we need.

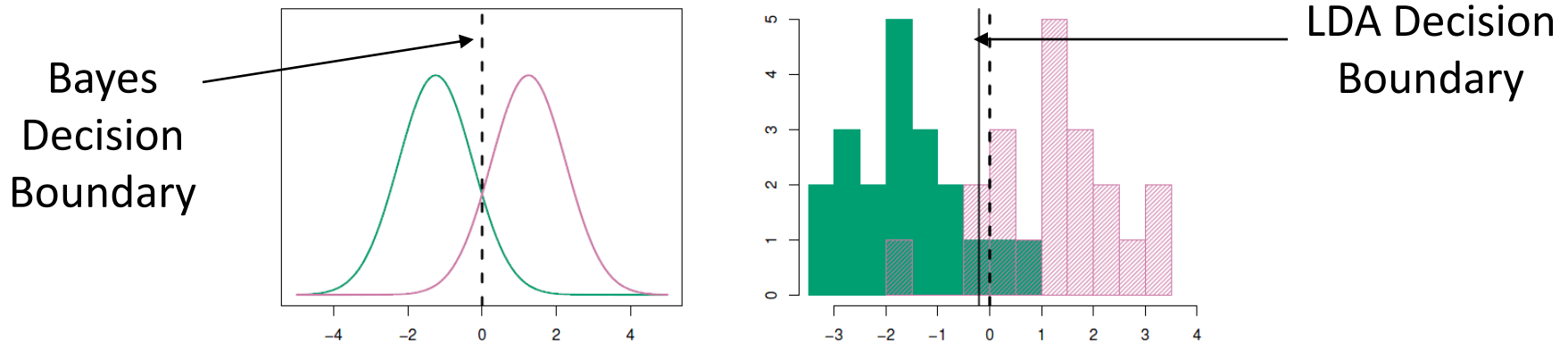
$$\hat{\Sigma} = \sum_{k=1}^K \sum_{g_i=k} \left(x^{(i)} - \hat{\mu}_k \right) \left(x^{(i)} - \hat{\mu}_k \right)^T / (N - K)$$

Linear Discriminant Analysis (cont.)

- First, we do the summation within every class k , then we have the sum over all of the classes. Next, we normalize by the scalar quantity, $N - K$.
- When we fit a MLE it should be divided by N , but if it is divided by $N - K$, we get an *unbiased estimator*.
- Remember, K is the number of classes. So, when N is large, the difference between N and $N - K$ is pretty small.
- Note that $x^{(i)}$ denotes the i th sample vector.

LDA: Example

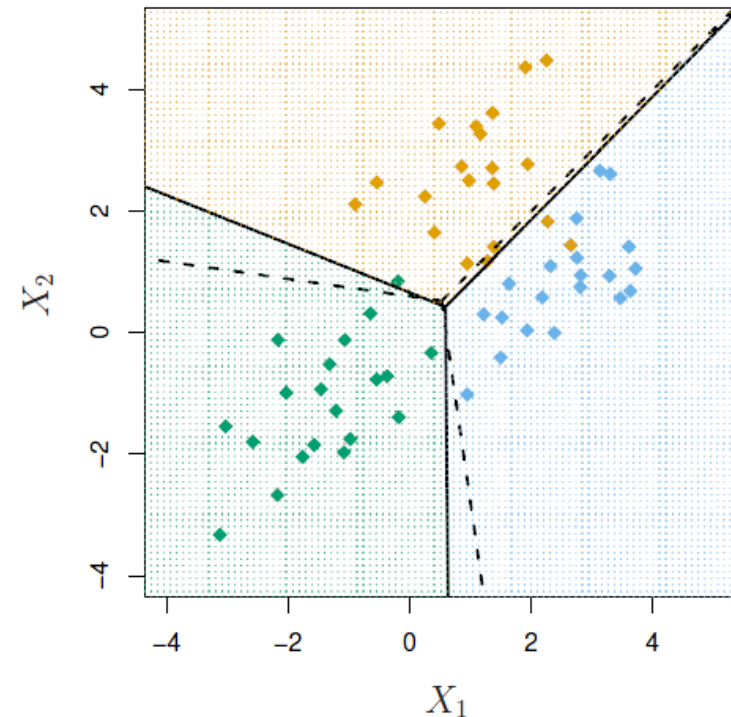
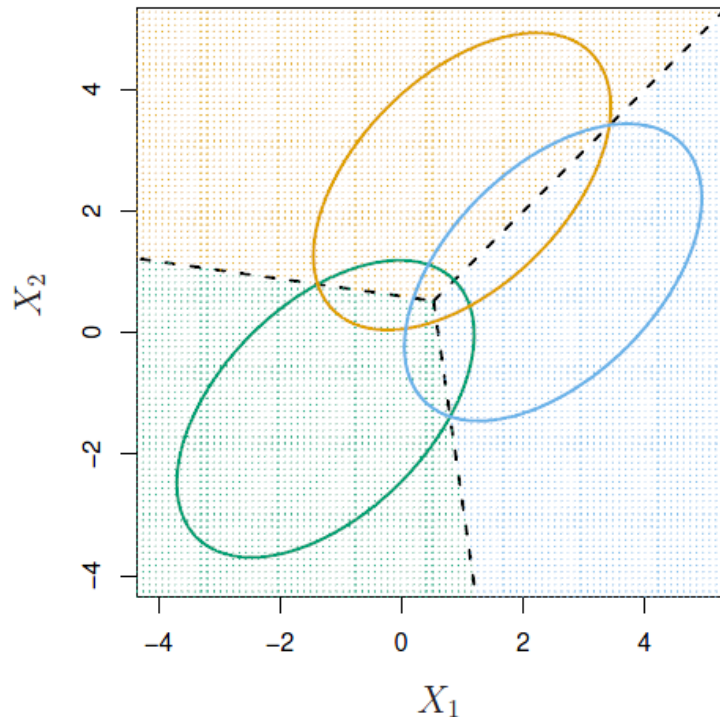
- Suppose we have only one predictor ($p = 1$), and two normal density functions $f_1(x)$ and $f_2(x)$, represent two distinct classes.
- The two density functions overlap, so there is some uncertainty about the class to which an observation with an unknown class belongs.



Example with $\mu_1 = -1.5$, $\mu_2 = 1.5$, $\pi_1 = \pi_2 = 0.5$, and $\sigma^2 = 1$.

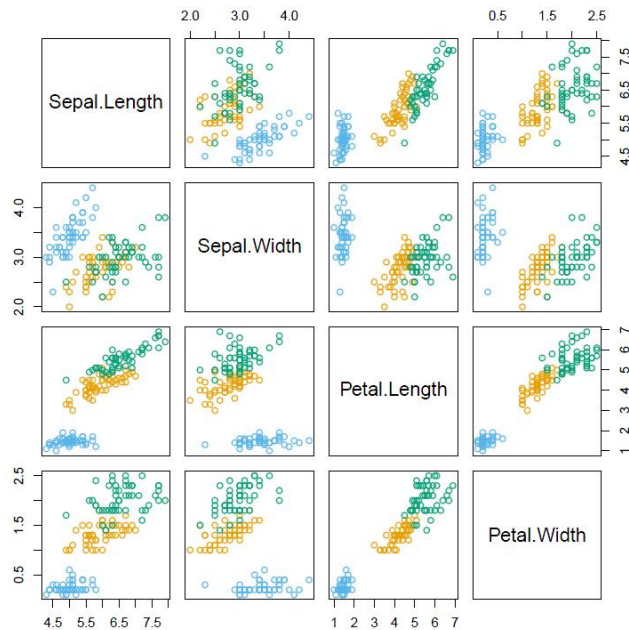
LDA: Example (cont.)

- If X is multidimensional where $p = 2$ and $K = 3$ classes:

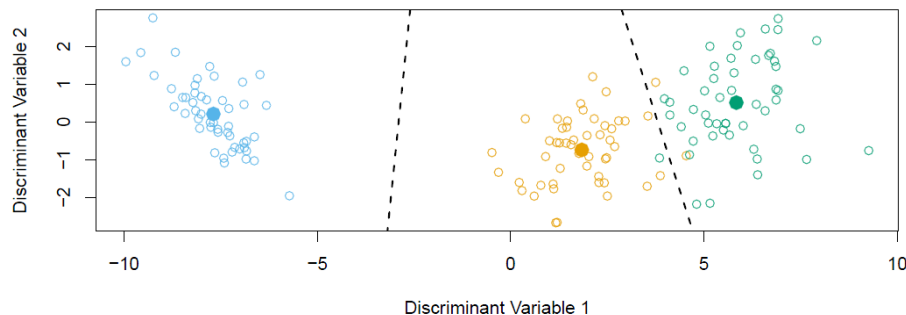


Here $\pi_1 = \pi_2 = \pi_3 = 1/3$.

LDA: Example (cont.)



- Using Fisher's Iris data set, we have:
 - 4 variables (sepal length, sepal width, petal length, petal width)
 - 3 species (setosa, versicolor, virginica)
 - 50 samples per class
- LDA correctly classifies all but 3 out of the 150 training samples.



Performance of a Classifier

- LDA is trying to approximate the Bayes classifier, which has the lowest total error rate out of all classifiers (if the Gaussian model is correct).
- In other words, the Bayes classifier will yield the smallest possible total number of misclassified observations, irrespective of which class the errors come from.

- Example of *confusion matrix*:

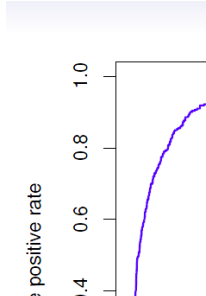
		<i>True Default Status</i>		
		No	Yes	Total
<i>Predicted Default Status</i>	No	9644	252	9896
	Yes	23	81	104
Total		9667	333	10000

Performance of a Classifier (cont.)

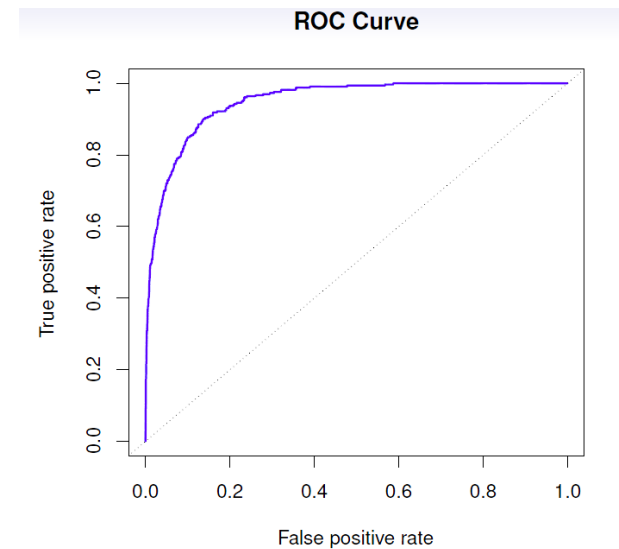
- Note that the results in the confusion matrix depend upon the establish *threshold* value, which is used to perform the classification based on the posterior probability.
- The *receiver operating characteristics* (ROC) curve is a popular graphic for simultaneously displaying the two types of errors for all possible thresholds.
- The overall performance of a classifier, summarized over all possible thresholds, is given by the *area under* the ROC curve.
 - The larger the area under the curve, the better the classifier!

Performance of a Classifier (cont.)

	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	
Test outcome	Test outcome positive	True positive	False positive (Type I error)	Positive predictive value (PPV, Precision) = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Test outcome positive}}$
	Test outcome negative	False negative (Type II error)	True negative	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$
	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	True positive rate (TPR, Sensitivity, Recall) = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR, Fall-out) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$	False negative rate (FNR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR, Specificity, SPC) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$		
	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR-}}$	http://en.wikipedia.org/wiki/Sensitivity_and_specificity			



A Receiver Operating Characteristic (ROC) curve is shown on the right side of the image. The y-axis is labeled 'positive rate' and ranges from 0.4 to 1.0. The x-axis represents the negative rate (1 - specificity). The curve is a blue line that starts at the bottom-left and curves towards the top-right, indicating a model's performance. The area under the curve is shaded in light purple.



Quadratic Discriminant Analysis

- QDA is not much different from LDA except that you assume that the covariance matrix can be different for each class.
- Thus, we will estimate the covariance matrix Σ_k separately for each class k , $k = 1, 2, \dots, K$.
- We get the following quadratic discriminant function:

$$\delta_k(x) = -\frac{1}{2} \log|\Sigma_k| - \frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) + \log \pi_k$$

Quadratic Discriminant Analysis (cont.)

- This quadratic discriminant function is very much like the linear discriminant function except that because Σ_k is not identical, you cannot throw away the quadratic terms.
- This discriminant function is a quadratic function and will contain second order terms.
- We get the following classification rule:

$$\hat{G}(x) = \arg \max_k \delta_k(x)$$

Quadratic Discriminant Analysis (cont.)

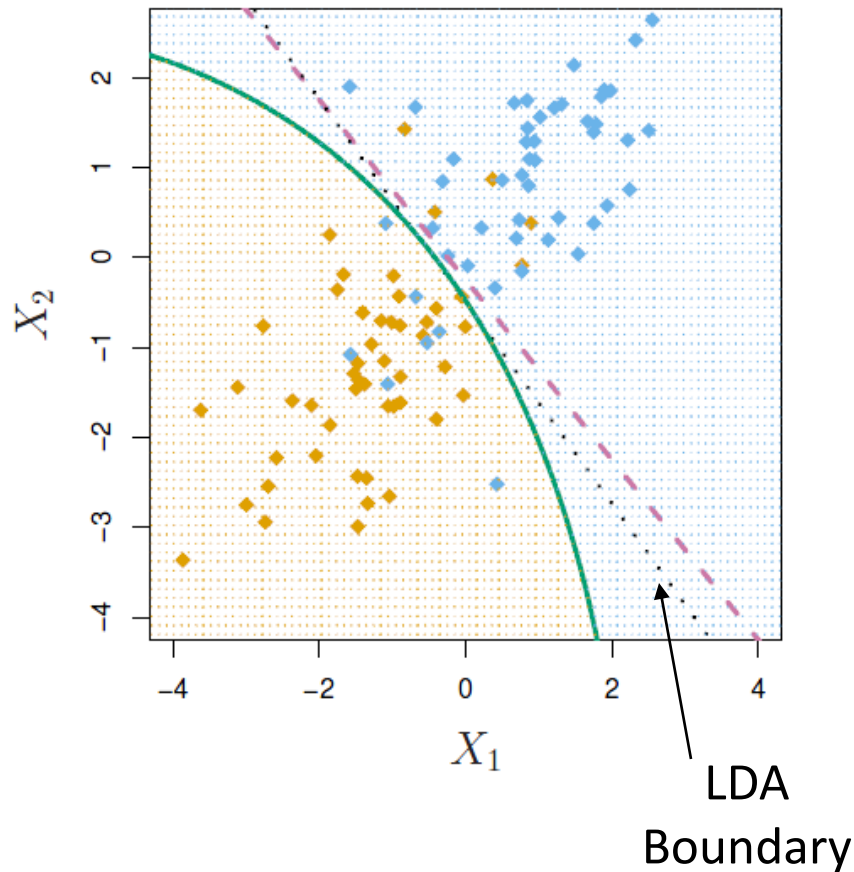
- For the classification rule, we find the class k which maximizes the quadratic discriminant function. The decision boundaries are quadratic equations in x .
- QDA, because it allows for more flexibility for the covariance matrix, tends to fit the data better than LDA, but then it has more parameters to estimate.
- The number of parameters increases significantly with QDA because of the separate covariance matrix for every class. If we have many classes and not so many sample points, this can be a problem.

Quadratic Discriminant Analysis (cont.)

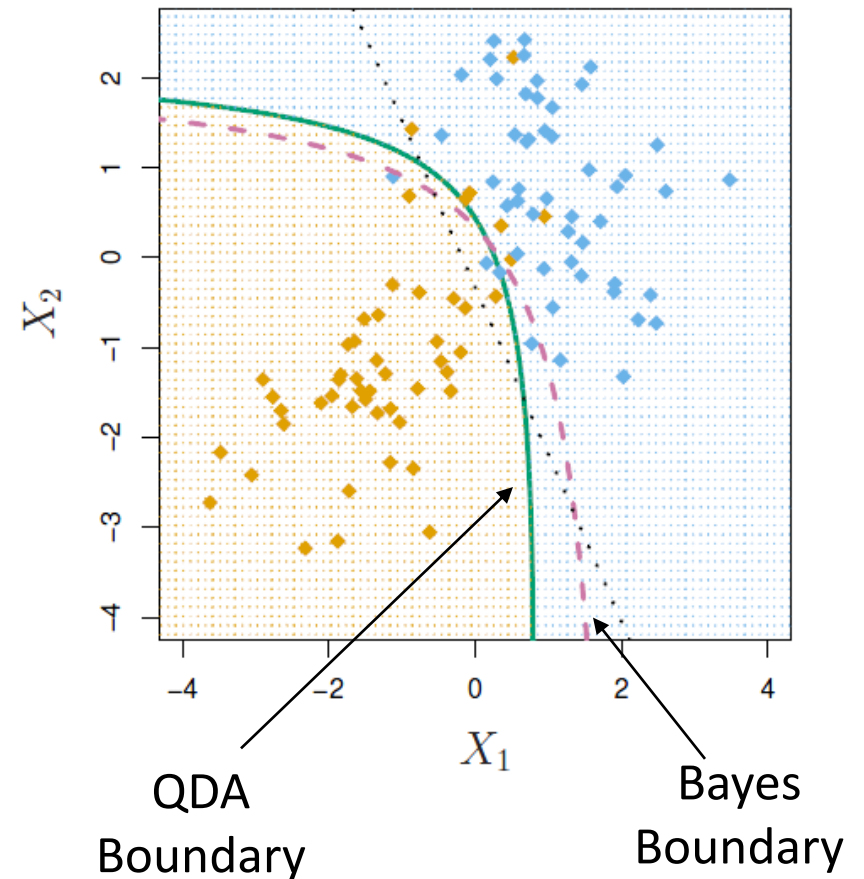
- As a result, there are trade-offs between fitting the training data well and having a simple model to work with.
- A simple model sometimes fits the data just as well as a complicated model.
- Even if the simple model does not fit the training data as well as a complex model, it still might be better on the test data because it is more robust.

LDA versus QDA

LDA is better!



QDA is better!



Review: Naive Bayes

- Assumes features are independent in each class.
- This is useful when p is large, and so multivariate methods like QDA and even LDA break down!
- Gaussian Naive Bayes assumes that each Σ_k is diagonal:

$$\delta_k(x) \propto \log \left[\pi_k \prod_{j=1}^p f_{kj}(x_j) \right] = -\frac{1}{2} \sum_{j=1}^p \frac{(x_j - \mu_{kj})^2}{\sigma_{kj}^2} + \log \pi_k$$

- We can use for *mixed* feature vectors (qualitative and quantitative).

LDA and Logistic Regression

- Under the model of LDA, we can compute the log-odds:

$$\begin{aligned} & \log \frac{Pr(G = k | X = x)}{Pr(G = K | X = x)} \\ &= \log \frac{\pi_k}{\pi_K} - \frac{1}{2} (\mu_k + \mu_K)^T \Sigma^{-1} (\mu_k - \mu_K) \\ &= a_{k0} + a_k^T x \end{aligned}$$

- The model of LDA satisfies the assumption of the linear logistic model.

LDA and Logistic Regression (cont.)

- The difference between linear logistic regression and LDA is that the linear logistic model only specifies the conditional distribution $\Pr(G = k \mid X = x)$.
- No assumption is made about $\Pr(X)$, while the LDA model specifies the joint distribution of X and G .
- $\Pr(X)$ is a mixture of Gaussians (where ϕ is the Gaussian density function):

$$\Pr(X) = \sum_{k=1}^K \pi_k \phi(X; \mu_k, \Sigma)$$

LDA and Logistic Regression (cont.)

- Moreover, linear logistic regression is solved by maximizing the conditional likelihood of G given X : $\Pr(G = k \mid X = x)$, while LDA maximizes the joint likelihood of G and X : $\Pr(X = x, G = k)$.
- If the additional assumption made by LDA is appropriate, LDA tends to estimate the parameters more efficiently by using more information about the data.
- Another advantage of LDA is that samples without class labels can be used under the model of LDA. On the other hand, LDA is not robust to gross outliers.
- Because logistic regression relies on fewer assumptions, it seems to be more robust to non-Gaussian type of data.



Review: K -Nearest Neighbors

- These classifiers are *memory-based* and require no model to be fit.
 - Training data: $(g_i, x_i), i = 1, 2, \dots, N$
1. Define distance on input x (e.g. Euclidian distance)
 2. Classify new instance by looking at the label of the single closest sample in the training set:

$$\hat{G}(x^*) = \operatorname{argmin}_i d(x_i, x^*)$$

Review: K -Nearest Neighbors (cont.)

- By looking at only the closest sample, overfitting the data can be a huge problem.
- To prevent overfitting, we can smooth the decision boundary by K nearest neighbors instead of 1.
- Find the K training samples x_r , $r = 1, \dots, K$ closest in distance to x^* , and then classify using majority vote among the k neighbors.
- The amount of computation can be intense when the training data is large since the distance between a new data point and every training point has to be computed and sorted.

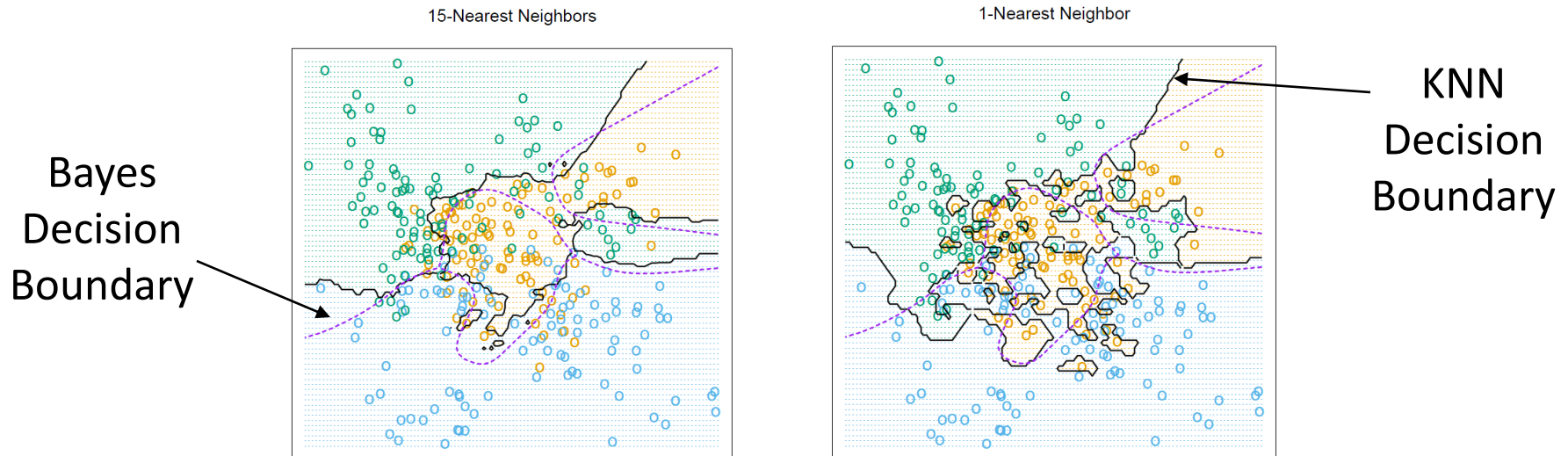


Review: K -Nearest Neighbors (cont.)

- Feature *standardization* is often performed in pre-processing.
- Because standardization affects the distance, if one wants the features to play a similar role in determining the distance, standardization is recommended.
- However, whether to apply normalization is rather subjective.
- One has to decide on an individual basis for the problem in consideration.

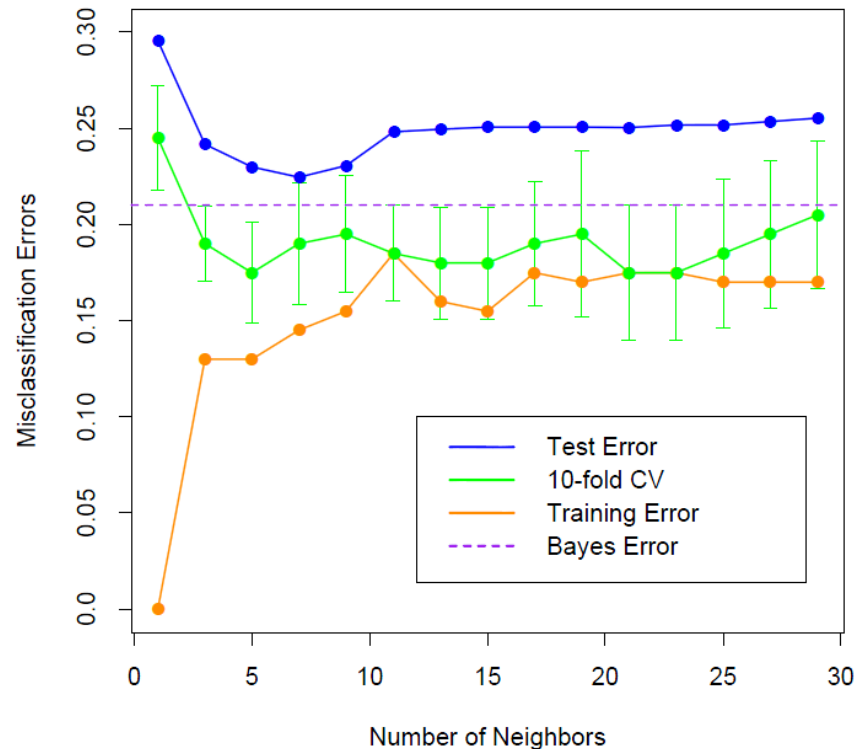
Review: K -Nearest Neighbors (cont.)

- The only parameter that can adjust the complexity of KNN is the number of neighbors k .
- The larger k is, the smoother the classification boundary. Or we can think of the complexity of KNN as lower when k increases.



Review: K -Nearest Neighbors (cont.)

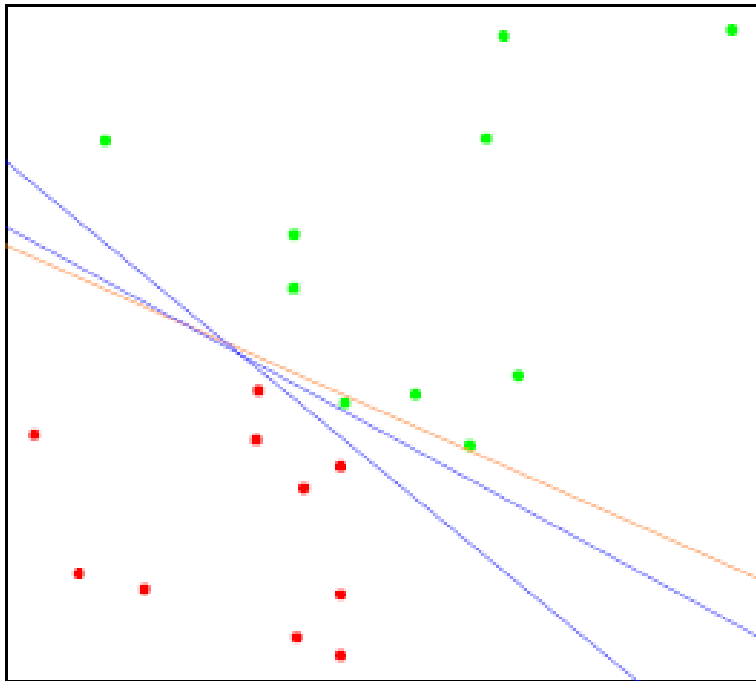
- For another simulated data set, there are two classes. The error rates based on the training data, test data, and 10-fold cross validation are plotted against K , the number of neighbors.
- We can see that the training error rate tends to grow when k grows, which is not the case for the error rate based on a separate test data set or cross-validation.



Separating Hyperplanes

- We seek to construct linear decision boundaries that explicitly try to separate the data into different classes as well as possible.
- We have seen that LDA and logistic regression both estimate linear decision boundaries in similar but slightly different ways.
- Even when the training data can be perfectly separated by hyperplanes, LDA or other linear methods developed under a statistical framework may not achieve perfect separation.
- Instead of using a probabilistic argument to estimate parameters, here we consider a geometric argument.

Separating Hyperplanes (cont.)



- The blue lines are two of the infinitely many possible *separating hyperplanes*.
- The orange line is the OLS solution, obtained by regressing the -1/1 response Y on X :

$$\{x : \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0\}$$

- The OLS does not do a perfect job separating the points.

Separating Hyperplanes (cont.)

- *Perceptrons*: classifiers that compute a linear combination of the input features and return the sign (-1, +1).
- Perceptrons set the foundations for neural network models, and they provide the basis for support vector classifiers.
- Before we continue, let's first review some vector algebra.
- A hyperplane or *affine set* L is defined by the equation:

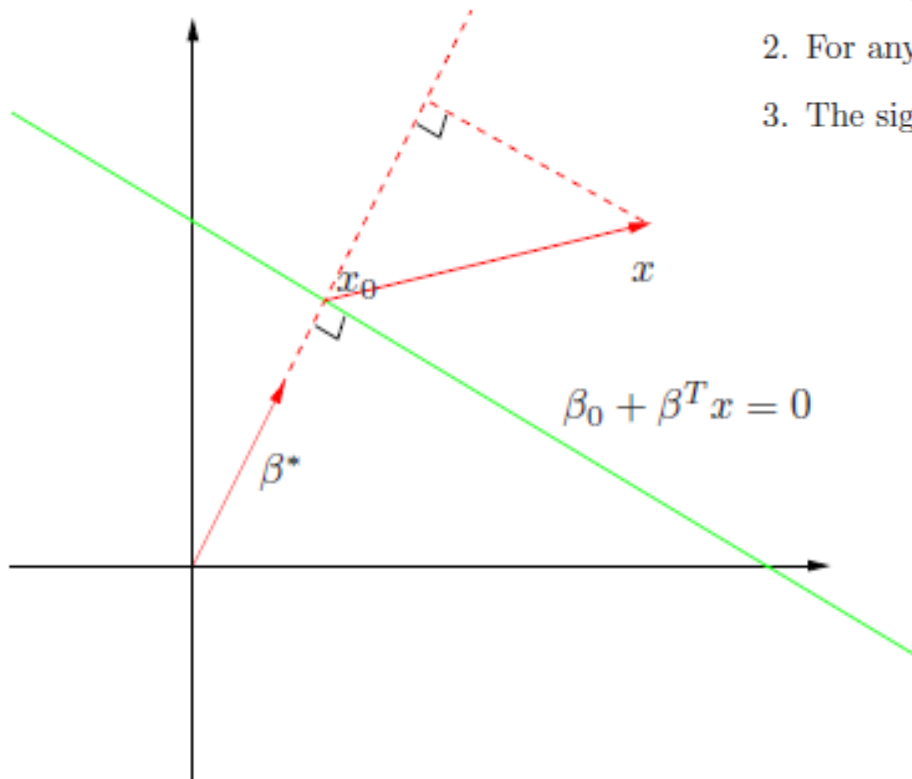
$$L = \{x : f(x) = \beta_0 + \beta^T x = 0\}$$

Separating Hyperplanes (cont.)

Some properties:

1. For any two points x_1 and x_2 lying in L , $\beta^T(x_1 - x_2) = 0$, and hence $\beta^* = \beta / \|\beta\|$ is the vector normal to the surface of L .
2. For any point x_0 in L , $\beta^T x_0 = -\beta_0$.
3. The signed distance of any point x to L is given by

$$\begin{aligned} \beta^{*T}(x - x_0) &= \frac{1}{\|\beta\|}(\beta^T x + \beta_0) \\ &= \frac{1}{\|f'(x)\|} f(x). \end{aligned} \quad (4.40)$$



Hence $f(x)$ is proportional to the *signed distance* from x to the hyperplane defined by $f(x) = 0$.

Perceptron Learning Algorithm

- One of the older approaches to this problem of finding a separating hyperplane in the machine learning literature is called the *perceptron learning algorithm*, developed by Frank Rosenblatt in 1956.
- **Goal:** The algorithm tries to find a separating hyperplane by minimizing the distance of misclassified points to the decision boundary.
- The algorithm starts with an initial guess as to the separating plane's parameters and then updates that guess when it makes mistakes.

Perceptron Learning Algorithm (cont.)

- Code the two classes by $y_i = +1, -1$.
- If a response $y_i = +1$ is misclassified, then $\beta^T x_i + \beta_0 < 0$.
- If a response $y_i = -1$ is misclassified, then $\beta^T x_i + \beta_0 > 0$.
- Since the signed distance from x to the decision boundary is $\frac{\beta^T x_i + \beta_0}{\|\beta\|}$, then the distance from a misclassified x_i to the decision boundary is
$$\frac{-y_i(\beta^T x_i + \beta_0)}{\|\beta\|}$$

Perceptron Learning Algorithm (cont.)

- Denote the set of misclassified points by M .
- The goal is to minimize: $D(\beta, \beta_0) = -\sum_{i \in M} y_i(\beta^T x_i + \beta_0)$
- The quantity is non-negative and proportional to the distance of the misclassified points to the decision boundary.
- To minimize $D(\beta, \beta_0)$, compute the gradient (assuming M is fixed):

$$\begin{aligned}\frac{\partial D(\beta, \beta_0)}{\partial \beta} &= -\sum_{i \in M} y_i x_i, \\ \frac{\partial D(\beta, \beta_0)}{\partial \beta_0} &= -\sum_{i \in M} y_i.\end{aligned}$$

Perceptron Learning Algorithm (cont.)

- The algorithm uses *stochastic gradient descent* to minimize the piecewise linear criterion.
- This means that rather than computing the sum of the gradient contributions of each observation followed by a step in the negative gradient direction, a step is taken after each observation is visited.
- Hence, the misclassified observations are visited in some sequence, and the parameters β are updated via:

$$\begin{pmatrix} \beta \\ \beta_0 \end{pmatrix} \leftarrow \begin{pmatrix} \beta \\ \beta_0 \end{pmatrix} + \rho \begin{pmatrix} y_i x_i \\ y_i \end{pmatrix}$$

Perceptron Learning Algorithm (cont.)

- Note that ρ is the learning rate, which in this case can be taken to be 1 without loss in generality.
- If the classes are linearly separable, the algorithm converges to a separating hyperplane in a finite number of steps.
- There are a number of problems with this algorithm:
 1. When the data are separable, there are many solutions, and which one is found depends on the starting values.
 2. The number of steps can be very large. The smaller the gap, the longer the time to find it.
 3. When the data are not separable, the algorithm will not converge, and cycles develop. The cycles can be long and therefore hard to detect.

Perceptron Learning Algorithm (cont.)

Pseudo Code

- Input: select random sample from the linearly separable training data set $x_i \in \mathbb{R}^D$ and $y_i \in \{-1, +1\} \forall i = 1, \dots, N$;
 - Initialize the vector of parameters β_0 ;
 - $k \leftarrow 0$;
 - **repeat**
 - $k \leftarrow k + 1$
 - $i \leftarrow k \bmod N$
 - **if** $\hat{y}_i \neq y_i$ **then**
 - $\beta_{k+1} = \beta_k + y_i x_i$
 - **else**
 - no-op
 - **until** *converged*;
- Note that if $\hat{y}_i y_i = -1$ (i.e. < 0), then we misclassified!
 - At each step, we update the weight vector by adding on the gradient (assuming the learning rate is one).

Perceptron Learning Algorithm – R Code

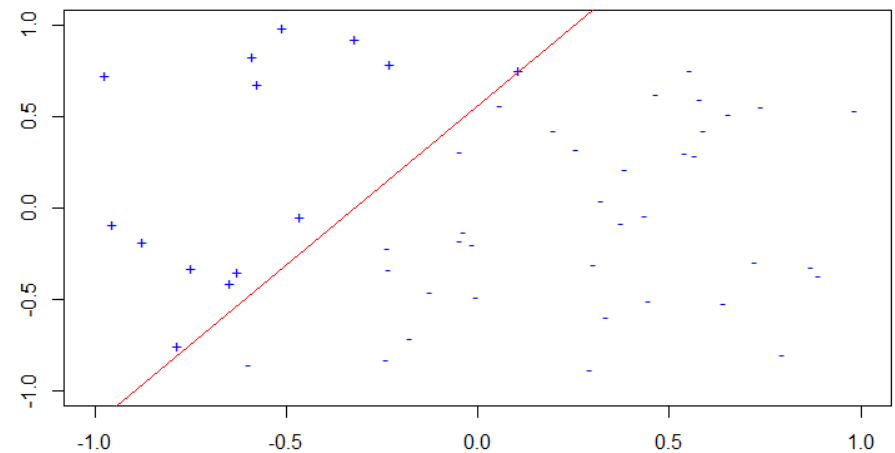
```
# Perceptron Learning Algorithm
# Input:
# x: linearly separable training data set
# y: label coding (assumed -1 and +1)
# beta0: initial beta parameters, can be single number or vector of size ncol(x)
# tol: tolerance for convergence check
# minepochs: minimum number of times to cycle all training points
# maxepochs: maximum number of times to cycle all training points
# verbose: print messages as iterations go
#
# Output: vector of estimated beta parameters

perceptron <- function(x, y, beta0 = 1, tol = 1e-8, minepochs = 2, maxepochs = 100, verbose = TRUE)
{
  N <- nrow(x)
  p <- ncol(x)
  if (length(y) != N) {
    stop("The number of rows in x is not the length of y")
  }
  if (length(beta0) == 1) {
    beta0 <- rep(beta0, p)
  }
  if (length(beta0) != p) {
    stop("The dimension of beta0 is not the number of columns in x")
  }
  # Make it stochastic!
  o <- sample(N)
  x <- x[o,]
  y <- y[o]

  itlim <- maxepochs * N
  itmin <- minepochs * N
  eps <- .Machine$double.xmax
  beta <- beta0
  k <- 0
  while ((eps > tol && k < itlim) || (k < itmin)) {
    beta0 <- beta
    i <- (k %% N) + 1
    # Check if this point is misclassified
    d <- y[i] * crossprod(beta, x[i,])
    if (d < 0) {
      # If it is misclassified then update beta
      beta <- beta0 + y[i] * x[i,]
    }
    # Update epsilon
    eps <- sqrt(sum((beta0 - beta)^2))/sqrt(sum(beta0^2))
    k <- k + 1
    if (verbose) {
      cat(sprintf("It: %d i:%d d:%.4f eps: %.4f\n", k, i, d, eps))
    }
  }
  if (k == itlim) {
    warning("Failed to converge")
  }
  return(beta)
}
```

- In the plot below, there are 50 randomly generated points, which are linearly separated using the perceptron() function:

Separating Hyperplanes - Perceptron Learning Algorithm

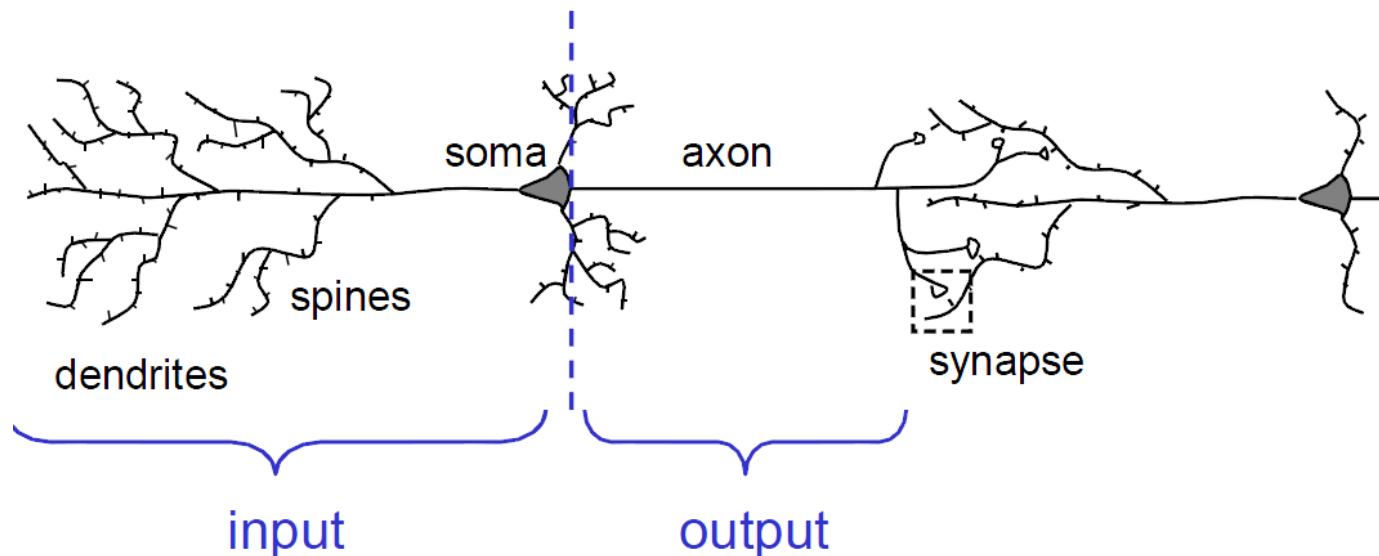


Artificial Neural Networks

- The central idea of *artificial neural networks* (ANN) is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features.
- This machine learning method originated as an algorithm trying to mimic neurons and the brain.
- The brain has extraordinary computational power to represent and interpret complex environments.
- ANN attempts to capture this mode of computation.

Artificial Neural Networks (cont.)

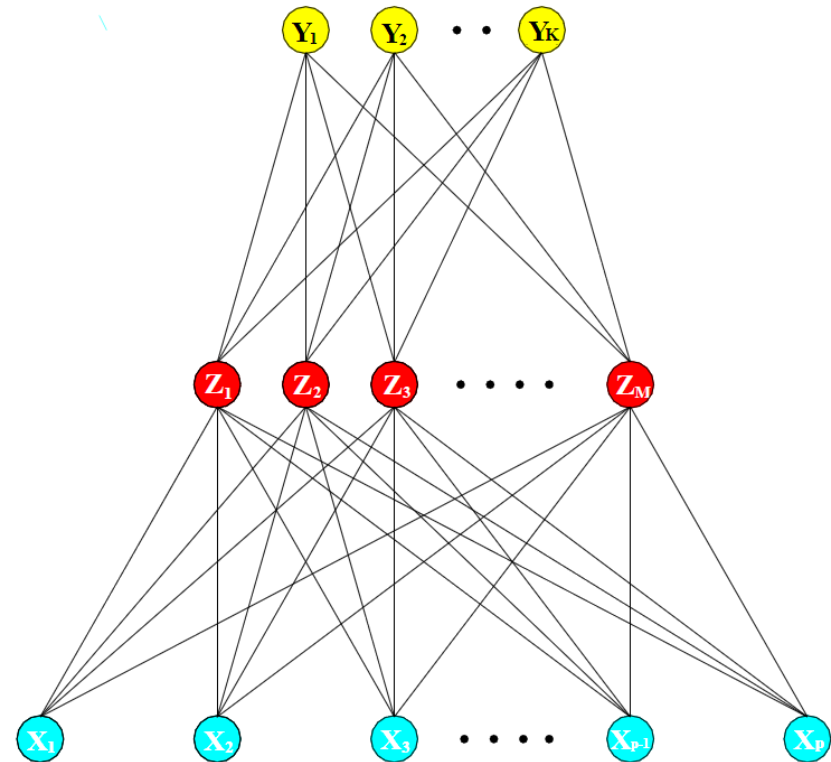
- Specifically, ANN is a formalism for representing functions, inspired from biological systems and composed of parallel computing units, each computing a simple function.



- Parts of the neuron: dendrites, soma (body), axon, synapses

Artificial Neural Networks (cont.)

- We will describe the most widely used ANN, the single layer perceptron (single hidden layer back-propagation network).
- A neural network is a two-stage regression or classification model, typically represented by a network diagram.



Artificial Neural Networks (cont.)

- For regression, typically $K = 1$ and there is only one output unit Y_1 at the top of the network diagram.
- These networks, however, can handle multiple quantitative responses in a seamless fashion.
- For K -class classification, there are K units at the top of the network diagram, with the k th unit modeling the probability of class k .
- There are K target measurements Y_k , $k = 1, \dots, K$, each being coded as a 0 – 1 variable for the k th class.

Artificial Neural Networks (cont.)

- Derived features Z_m are created from linear combinations of the inputs, and then the target Y_k is modeled as a function of the linear combinations of the Z_m :

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, \dots, M,$$

$$T_k = \beta_{0k} + \beta_k^T Z, \quad k = 1, \dots, K,$$

$$f_k(X) = g_k(T), \quad k = 1, \dots, K,$$

where $Z = (Z_1, Z_2, \dots, Z_M)$ and $T = (T_1, T_2, \dots, T_K)$

- The *activation function* $\sigma(v)$ is typically the sigmoid (logistic) function: $\frac{1}{1+e^{-v}}$

Artificial Neural Networks (cont.)

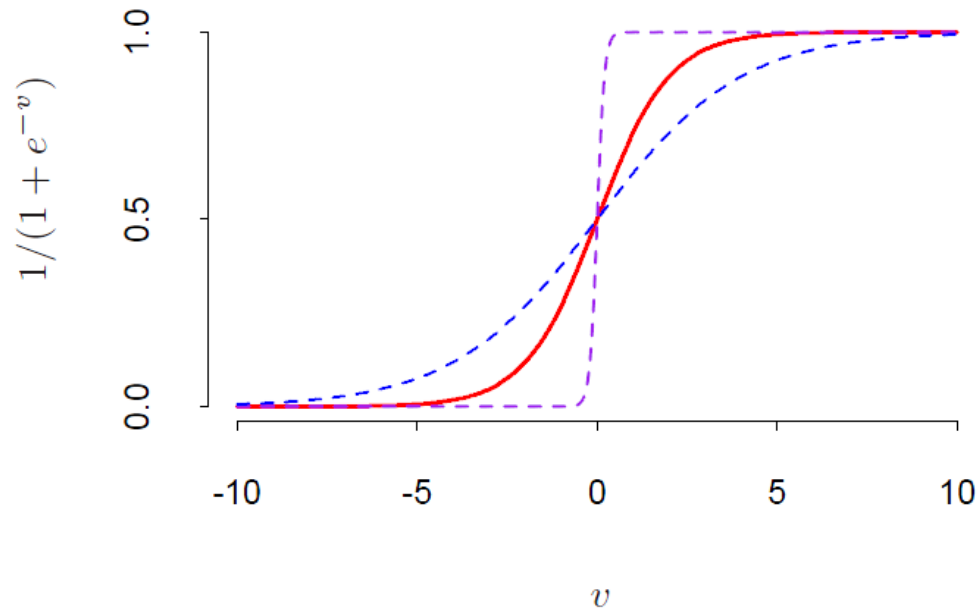


FIGURE 11.3. Plot of the sigmoid function $\sigma(v) = 1/(1 + \exp(-v))$ (red curve), commonly used in the hidden layer of a neural network. Included are $\sigma(sv)$ for $s = \frac{1}{2}$ (blue curve) and $s = 10$ (purple curve). The scale parameter s controls the activation rate, and we can see that large s amounts to a hard activation at $v = 0$. Note that $\sigma(s(v - v_0))$ shifts the activation threshold from 0 to v_0 .

Artificial Neural Networks (cont.)

- Neural network diagrams are sometimes drawn with an additional *bias* unit feeding into every unit in the hidden and output layers.
- Thinking of the constant “1” as an additional input feature, this bias unit captures the intercepts α_{0m} and β_{0k} in the model.
- The output function $g_k(T)$ allows a final transformation of the vector of outputs T .
- For regression, we typically choose the identity function $g_k(T) = T_k$

Artificial Neural Networks (cont.)

- Early work in K -class classification also used the identity function, but this was later abandoned in favor of the *softmax* function:

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^K e^{T_\ell}}$$

- Note that this is exactly the transformation used in the multilogit model, which produces positive estimates that sum to one.
- The units in the middle of the network, computing the derived features Z_m , are called *hidden units* because these values are not directly observed.

Artificial Neural Networks (cont.)

- We can think of the Z_m as a basis expansion of the original inputs X ; the neural network is then a standard linear model, or linear multilogit model, using these transformations as inputs.
- However, the parameters of the basis functions are learned from the data, which is an important enhancement over the basis expansion techniques.
- Notice that if σ is the identity function, then the entire model collapses to a linear model in the inputs.

Artificial Neural Networks (cont.)

- Thus, a neural network can be thought of as a nonlinear generalization of the linear model, both for regression and classification.
- By introducing the nonlinear transformation σ , it greatly enlarges the class of linear models.
- In general, there can be more than one hidden layer.
- The neural network model has unknown parameters, often called *weights*.

Artificial Neural Networks (cont.)

- We seek values for the weights that make the model fit the training data well.
- We denote the complete set of weights by ϑ , which consists of:

$$\begin{aligned} \{\alpha_{0m}, \alpha_m; m = 1, 2, \dots, M\} & \quad M(p + 1) \text{ weights,} \\ \{\beta_{0k}, \beta_k; k = 1, 2, \dots, K\} & \quad K(M + 1) \text{ weights.} \end{aligned}$$

- For regression, we use RSS as our measure of fit (error function):

$$R(\theta) = \sum_{k=1}^K \sum_{i=1}^N (y_{ik} - f_k(x_i))^2.$$

Artificial Neural Networks (cont.)

- For classification, we use squared error or cross-entropy (deviance):

$$R(\theta) = - \sum_{i=1}^N \sum_{k=1}^K y_{ik} \log f_k(x_i),$$

and the corresponding classifier $G(x) = \operatorname{argmax}_k f_k(x)$.

- With the softmax activation function and the cross-entropy error function, the neural network model is exactly a linear logistic regression model in the hidden units, and all the parameters are estimated by maximum likelihood.

Artificial Neural Networks (cont.)

- Typically, we do not want the global minimizer $R(\vartheta)$, as this is likely to be an overfit solution.
- Instead, some regularization is needed; this is achieved directly through a penalty term, or indirectly by early stopping.
- The generic approach to minimizing $R(\vartheta)$ is by gradient descent, called *back-propagation* in this setting.
- The gradient can be easily derived using the chain rule for differentiation.

Artificial Neural Networks (cont.)

- This can be computed by a forward and backward sweep over the network, keeping track only of quantities local to each unit.
- Here is back-propagation in detail for squared error loss.
- Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ and $z_i = (z_{1i}, z_{2i}, \dots, z_{Mi})$. Then:

$$\begin{aligned} R(\theta) &\equiv \sum_{i=1}^N R_i \\ &= \sum_{i=1}^N \sum_{k=1}^K (y_{ik} - f_k(x_i))^2, \end{aligned}$$

Artificial Neural Networks (cont.)

- With the following derivatives:

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)z_{mi},$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^K 2(y_{ik} - f_k(x_i))g'_k(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{i\ell}.$$

- Given these derivatives, a gradient descent update at the $(r + 1)$ st iteration has the form (where γ_r is the *learning rate*):

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^N \frac{\partial R_i}{\partial \beta_{km}^{(r)}},$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^N \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}},$$

Artificial Neural Networks (cont.)

- We can simplify the derivatives to:

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi},$$
$$\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}.$$

- The quantities δ_{ki} and s_{mi} are “errors” from the current model at the output and hidden layer units, respectively. Note that the output layer errors $\delta_{ki} = (\hat{f}_k(x_i) - f_k(x_i))$
- From their definitions, these errors satisfy the *back-propagation equations*:

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^K \beta_{km} \delta_{ki},$$

Artificial Neural Networks (cont.)

- Using this, the gradient descent updates can be implemented with a two-pass algorithm.
- In the *forward pass*, the current weights are fixed and the predicted values $\hat{f}_k(x_i)$ are computed.
- In the *backward pass*, the errors δ_{ki} are computed, and then back-propagated via the back-propagation equations to give the errors s_{mi}
- Both sets of errors are then used to compute the gradients for the updates.

Artificial Neural Networks (cont.)

- This two-pass procedure is what is known as back-propagation.
- It has also been called the *delta rule*.
- The computational components for cross-entropy have the same form as those for the sum of squares function.
- The advantage of back-propagation are its simple, local nature. In the back-propagation algorithm, each hidden unit passes and receives information only to and from units that share a connection.



Artificial Neural Networks (cont.)

- Usually starting values for weights are chosen to be random values near zero.
- Hence, the model starts out nearly linear and becomes nonlinear as the weights increase.
- Use of exact zero weights leads to zero derivatives and perfect symmetry, and the algorithm never moves.
- Starting instead with large weights often leads to poor solutions.

Artificial Neural Networks (cont.)

- Often neural networks have too many weights and will overfit the data at the global minimum of R .
- An explicit method for regularization is *weight decay*, which is analogous to ridge regression used for linear models.
- We add a penalty to the error function $R(\theta) + \lambda J(\theta)$, where:

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{m\ell} \alpha_{m\ell}^2$$

and λ is a tuning parameter.

Artificial Neural Networks (cont.)

- Larger values of λ will tend to shrink the weights toward zero; typically cross-validation is used to estimate λ .
- Other forms for the penalty (such as *weight elimination*) are:

$$J(\theta) = \sum_{km} \frac{\beta_{km}^2}{1 + \beta_{km}^2} + \sum_{m\ell} \frac{\alpha_{m\ell}^2}{1 + \alpha_{m\ell}^2},$$

- Also, since the scaling of the inputs determines the effective scaling of the weights in the bottom layer, it can have a large effect on the quality of the final solution; thus, it is best to standardize all inputs.

Artificial Neural Networks (cont.)

- Generally, it is better to have too many hidden units than too few.
- With too few hidden units, the model might not have enough flexibility to capture the nonlinearities in the data.
- With too many hidden units, the extra weights can be shrunk toward zero if appropriate regularization is used.
- Typically the number of hidden units is somewhere in the range of 5 to 100, with the number increasing with the number of inputs and number of training cases.

Artificial Neural Networks (cont.)

- The error function is nonconvex, possessing many local minima, so one must try a number of random starting configurations and then choose the solution giving the lowest error.
- Probably a better approach is to use the average predictions over the collection of networks as the final prediction.
- Another approach is via *bagging*, which averages the predictions of networks training from randomly perturbed versions of the training data.

ANN – Example 1 R Code

- In this first example, we use the R library “neuralnet” to train and build a neural network that is able to take a number and calculate the square root.
- The ANN will take a single input (the number you want square rooting) and produce a single output (the square root of the input).
- The ANN will contain 10 hidden neurons to be trained within each layer.
- The ANN does a reasonable job at finding the square root!

ANN – Example 1 R Code (cont.)

```
## Example 1: Create a ANN to perform square rooting

library(neuralnet) # For Neural Network

# Generate 50 random numbers uniformly distributed between 0 and 100
# and store them as a dataframe
traininginput <- as.data.frame(runif(50, min = 0, max = 100))
trainingoutput <- sqrt(traininginput)

# Column bind the data into one variable
trainingdata <- cbind(traininginput, trainingoutput)
colnames(trainingdata) <- c("Input", "Output")

# Train the neural network
# Going to have 10 hidden layers
# Threshold is a numeric value specifying the threshold for the partial
# derivatives of the error function as stopping criteria.
net.sqrt <- neuralnet(Output ~ Input, trainingdata, hidden = 10, threshold = 0.01)
print(net.sqrt)

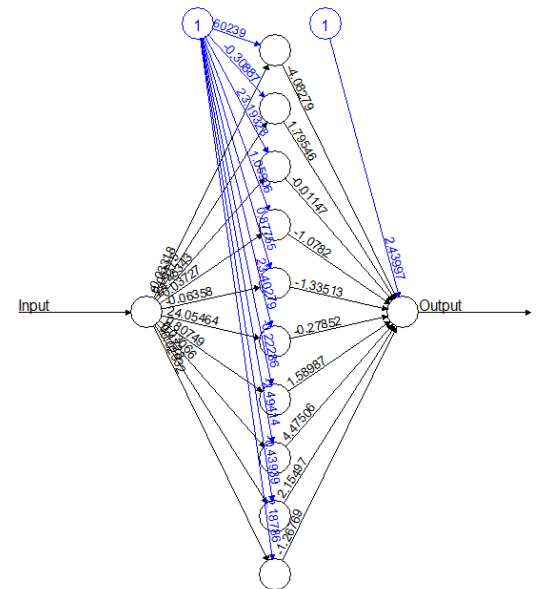
# Plot the neural network
plot(net.sqrt)

# Test the neural network on some training data
testdata <- as.data.frame((1:10)^2) # generate some squared numbers
net.results <- compute(net.sqrt, testdata) # run them through the neural network

# Let's see what properties net.sqrt has
ls(net.results)

# Let's see the results
print(net.results$net.result)

# Let's display a better version of the results
cleanoutput <- cbind(testdata, sqrt(testdata), as.data.frame(net.results$net.result))
colnames(cleanoutput) <- c("Input", "Expected output", "Neural Net Output")
print(cleanoutput)
```



	Input	Expected output	Neural Net output
1	1	1	0.9715460215
2	4	2	2.0247643739
3	9	3	2.9885151138
4	16	4	4.0083153896
5	25	5	5.0058957028
6	36	6	5.9979280544
7	49	7	6.9966862119
8	64	8	8.0000630348
9	81	9	9.0070629151
10	100	10	9.9741177808

ANN – Example 2 R Code

- In this second example, we use the R library “nnet” to train and build a neural network that is able to predict median value of owner-occupied homes.
- This example of regression with neural networks is compared with multiple linear regression.
- The data set is housing data for 506 census tracts of Boston from the 1970 census.
- We see that the ANN outperforms linear regression in terms of lower MSE.

ANN – Example 2 R Code (cont.)

```
# Example 2: Create a ANN for prediction

# We give a brief example of regression with neural networks and comparison with
# multivariate linear regression. The data set is housing data for 506 census tracts of
# Boston from the 1970 census. The goal is to predict median value of owner-occupied homes.

# Load the data and inspect the range (which is 1 - 50)
library(mlbench)
data(BostonHousing)
summary(BostonHousing$medv)

# Build the multiple linear regression model
lm.fit <- lm(medv ~ ., data = BostonHousing)
lm.predict <- predict(lm.fit)

# Calculate the MSE and plot
mean((lm.predict - BostonHousing$medv)^2) # MSE = 21.89483
par(mfrow = c(2,1))
plot(BostonHousing$medv, lm.predict, main = "Linear Regression Predictions vs Actual (MSE = 21.9)",
     xlab = "Actual", ylab = "Predictions", pch = 19, col = "brown")

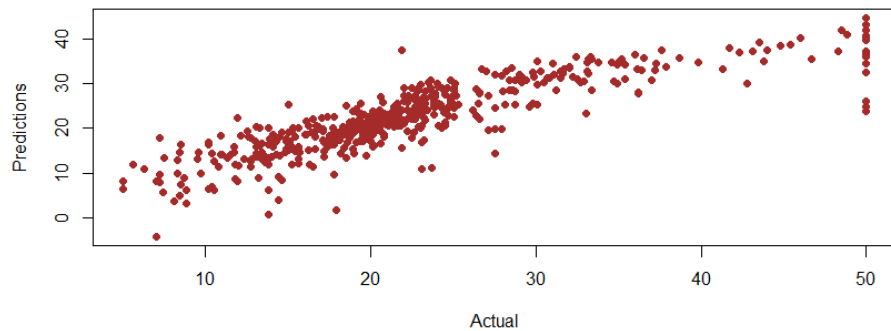
# Build the feed-forward ANN (w/ one hidden layer)
library(nnet) # For Neural Network
nnet.fit <- nnet(medv/50 ~ ., data = BostonHousing, size = 2) # scale inputs: divide by 50 to get 0-1 range
nnet.predict <- predict(nnet.fit)*50 # multiply 50 to restore original scale

# Calculate the MSE and plot
mean((nnet.predict - BostonHousing$medv)^2) # MSE = 16.56974
plot(BostonHousing$medv, nnet.predict, main = "Artificial Neural Network Predictions vs Actual (MSE = 16.6)",
     xlab = "Actual", ylab = "Predictions", pch = 19, col = "blue")

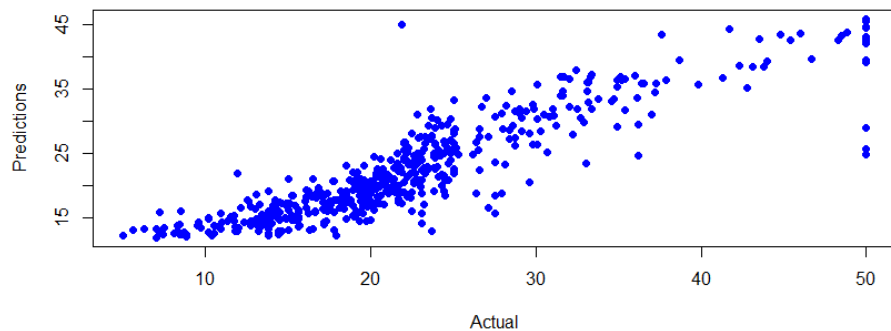
# Next, we use the function train() from the package caret to optimize the ANN
# hyperparameters decay and size, Also, caret performs resampling to give a better
# estimate of the error. We scale linear regression by the same value, so the error
# statistics are directly comparable.
```

ANN – Example 2 R Code (cont.)

Linear Regression Predictions vs Actual (MSE = 21.9)



Artificial Neural Network Predictions vs Actual (MSE = 16.6)



- Next, we use the function `train()` from the R library “caret” to optimize the ANN hyperparameters decay and size.
- This library also performs resampling to give a better estimate of the error.
- Note that we scale linear regression by the same value, so the error statistics are directly comparable.

ANN – Example 2 R Code (cont.)

```
# Next, we use the function train() from the package caret to optimize the ANN
# hyperparameters decay and size. Also, caret performs resampling to give a better
# estimate of the error. We scale linear regression by the same value, so the error
# statistics are directly comparable.

library(mlbench)
data(BostonHousing)
library(caret)

# Optimize the ANN hyperparameters and print the results
mygrid <- expand.grid(.decay = c(0.5, 0.1), .size = c(4, 5, 6))
nnet.fit2 <- train(medv/50 ~ ., data = BostonHousing, method = "nnet", maxit = 1000,
  tuneGrid = mygrid, trace = FALSE)
print(nnet.fit2)

# Scale the linear regression and print the results
lm.fit2 <- train(medv/50 ~ ., data = BostonHousing, method = "lm")
print(lm.fit2)
```

- Tuned ANN RMSE = 0.0822
- Linear Regression RMSE = 0.1003

```
Neural Network

506 samples
13 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 506, 506, 506, 506, 506, 506, ...
Resampling results across tuning parameters:
```

decay	size	RMSE	Rsquared
0.1	4	0.08351023015	0.7899223528
0.1	5	0.08306895714	0.7908757420
0.1	6	0.08220087875	0.7955866187
0.5	4	0.08960617815	0.7648468036
0.5	5	0.08919464524	0.7655984833
0.5	6	0.08857583326	0.7693882032

```
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were size = 6 and decay = 0.1.
```

```
Linear Regression

506 samples
13 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 506, 506, 506, 506, 506, 506, ...
Resampling results:
```

RMSE	Rsquared
0.1002723129	0.7099376352

```
Tuning parameter 'intercept' was held constant at a value of TRUE
```

ANN – Example 3 R Code

- In this third example, we use the R library “RSNNS” to train and build a neural network that is able to predict the identification of Iris plant Species.
- The ANN performs the classification on the basis of plant attribute measurements: Sepal Length, Sepal Width, Petal Length, Petal Width.
- This example of classification with neural networks uses the standard back-propagation algorithm.

ANN – Example 3 R Code (cont.)

```
# Example 3: Create a ANN for classification

# Predict the identification of Iris plant Species on the basis of plant
# attribute measurements: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width

library(RSNNS)

# Load and store the 'iris' data
data(iris)

# Generate a sample from the 'iris' data set
irisSample <- iris[sample(1:nrow(iris), length(1:nrow(iris))), 1:ncol(iris)]
irisValues <- irisSample[, 1:4]
head(irisValues)
irisTargets <- irisSample[, 5]
head(irisTargets)

# Generate a binary matrix from an integer-valued input vector representing class labels
irisDecTargets <- decodeClassLabels(irisTargets)
head(irisDecTargets)

# Split the data into the training and testing set, and then normalize
irisSample <- splitForTrainingAndTest(irisValues, irisDecTargets, ratio = 0.15)
irisSample <- normTrainingAndTestSet(irisSample)

# Train the Neural Network (Multi-Layer Perceptron)
nn3 <- mlp(irisSample$inputsTrain, irisSample$targetsTrain, size = 2, learnFuncParams = 0.1, maxit = 100,
  inputsTest = irisSample$inputsTest, targetsTest = irisSample$targetsTest)
print(nn3)

# Predict using the testing data
testPred6 <- predict(nn3, irisSample$inputsTest)

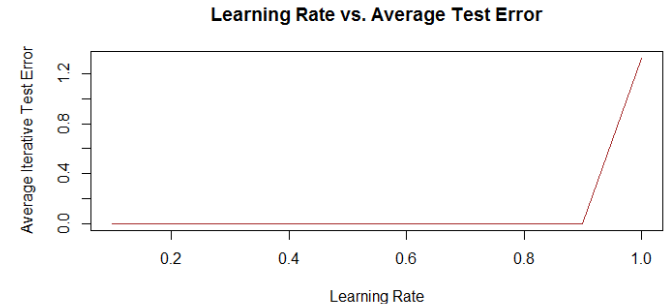
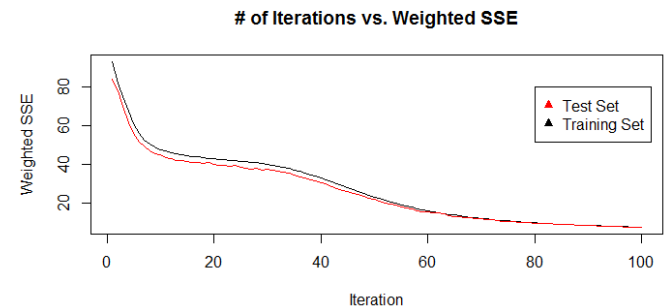
# Calculate the Confusion Matrices for the Training and Testing Sets
confusionMatrix(irisSample$targetsTrain, fitted.values(nn3))
confusionMatrix(irisSample$targetsTest, testPred6)

# Calculate the weights of the Newly Trained Network
weightMatrix(nn3)

# Plot the Iterative Error of both training (black) and test (red) error
# This shows hows the Number of Iterations Affects the weighted SSE
plotIterativeError(nn3, main = "# of Iterations vs. Weighted SSE")
legend(80, 80, legend = c("Test Set", "Training Set"), col = c("red", "black"), pch = 17)

# See how changing the Learning Rate Affects the Average Test Error
err <- vector(mode = "numeric", length = 10)
learnRate = seq(0.1, 1, length.out = 10)
for (i in 10){
  fit <- mlp(irisSample$inputsTrain, irisSample$targetsTrain, size = 2, learnFuncParams = learnRate[i], maxit = 50,
    inputsTest = irisSample$inputsTest, targetsTest = irisSample$targetsTest)
  err[i] <- mean(fit$iterativeTestError)
}

# Plot the Effect of Learning Rate vs. Average Iterative Test Error
plot(learnRate, err, xlab = "Learning Rate", ylab = "Average Iterative Test Error",
  main = "Learning Rate vs. Average Test Error", type = "l", col = "brown")
```



We can see the iterative error of both training and test sets, and we see how changing the learning rate affects the average test error!

Comparison of Classification Methods

- KNN is completely non-parametric, as there are no assumptions made about the shape of the decision boundary.
- We can expect KNN to dominate both LDA and logistic regression when the decision boundary is highly non-linear.
- On the other hand, KNN does not tell us which predictors are important; there is no table of coefficients.
- QDA serves as a compromise between the non-parametric KNN method and the linear LDA and logistic regression approaches.

Comparison of Classification Methods (cont.)

- Logistic regression is very popular for classification, especially when $K = 2$.
- LDA is useful when n is small, the classes are well separated (and Gaussian assumptions are reasonable), and when $K > 2$.
- Naive Bayes is useful when p is very large.
- Finally, ANNs is useful for modeling complex, non-linear underlying relationships. The key for strong model performance is hyper-parameter tuning.

Summary

- Basic setting of a classification problem.
- Bayes classification rule and Naïve Bayes classifier
- Statistical model of logistic regression.
- Binary logistic regression algorithm.
- Basic concepts of the multi-class logistic regression algorithm.
- Statistical model used by LDA and QDA.
- Estimate the Gaussian distributions within each class.
- LDA and QDA classification rule.
- Similarities and differences between LDA and logistic regression.
- Review of the K -nearest neighbor classification algorithm.
- How the number of neighbors, K , adjusts the model complexity.
- Perceptron learning algorithm for separating hyperplanes.
- Artificial neural networks and the back-propagation algorithm for regression and classification problems.
- Differences between all of the classification methods.