CAP 5625: Computational Foundations for Artificial Intelligence

Logistic regression

Relaxing the Gaussian assumption

The first classifier we covered specifically-tailored for performing classification was discriminant analysis.

Linear discriminant analysis (LDA) attempts to find hyperplanes forming boundaries of classes, and these hyperplanes are based on estimated linear discriminant functions for each class k

$$\hat{\delta}_k(x) = x^T \hat{\boldsymbol{\Sigma}}^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^T \hat{\boldsymbol{\Sigma}}^{-1} \hat{\mu}_k + \log \hat{\pi}_k$$

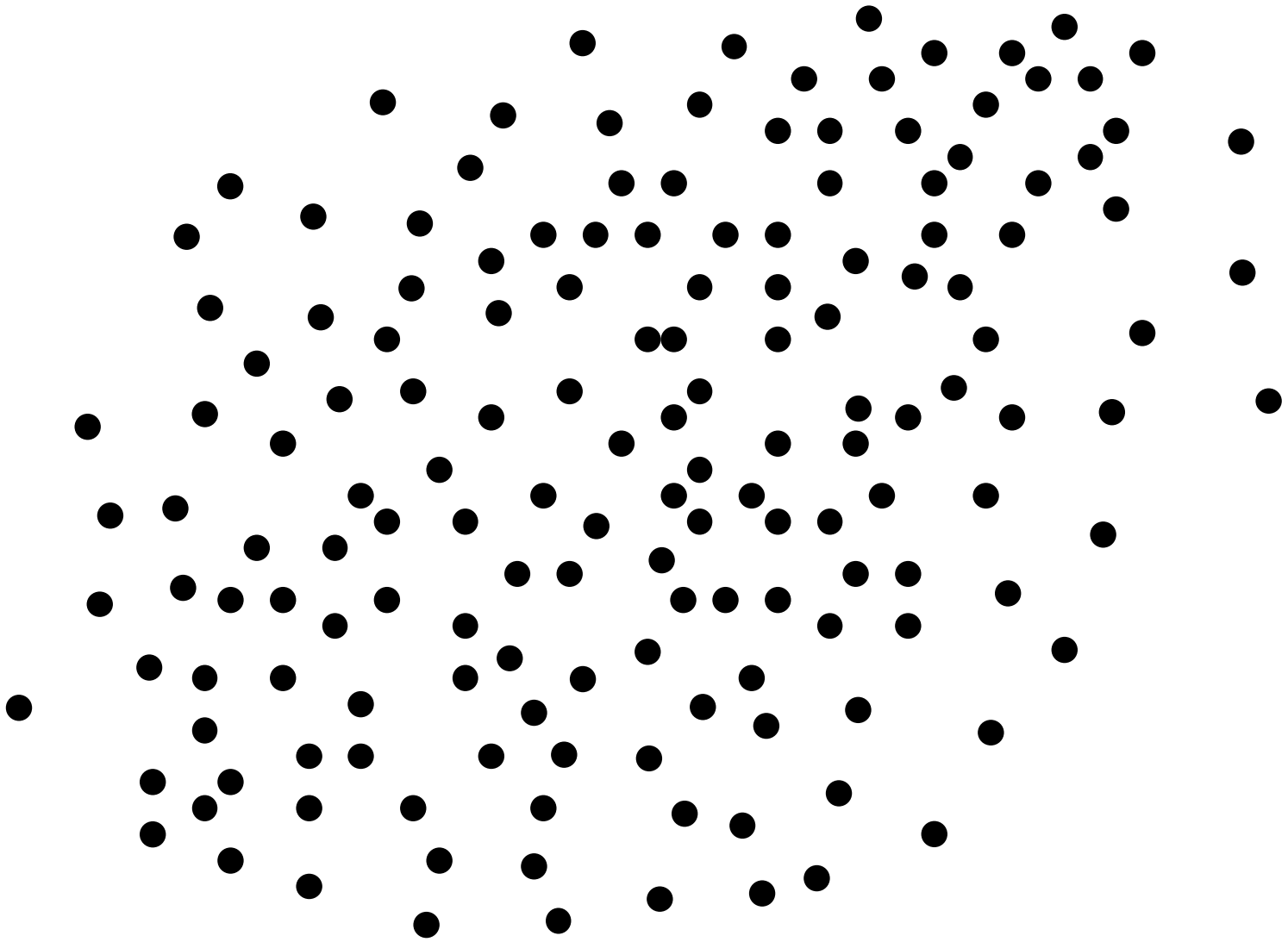
with the hyperplane separating classes k and ℓ

$$\hat{\delta}_k(x) - \hat{\delta}_\ell(x) = 0$$

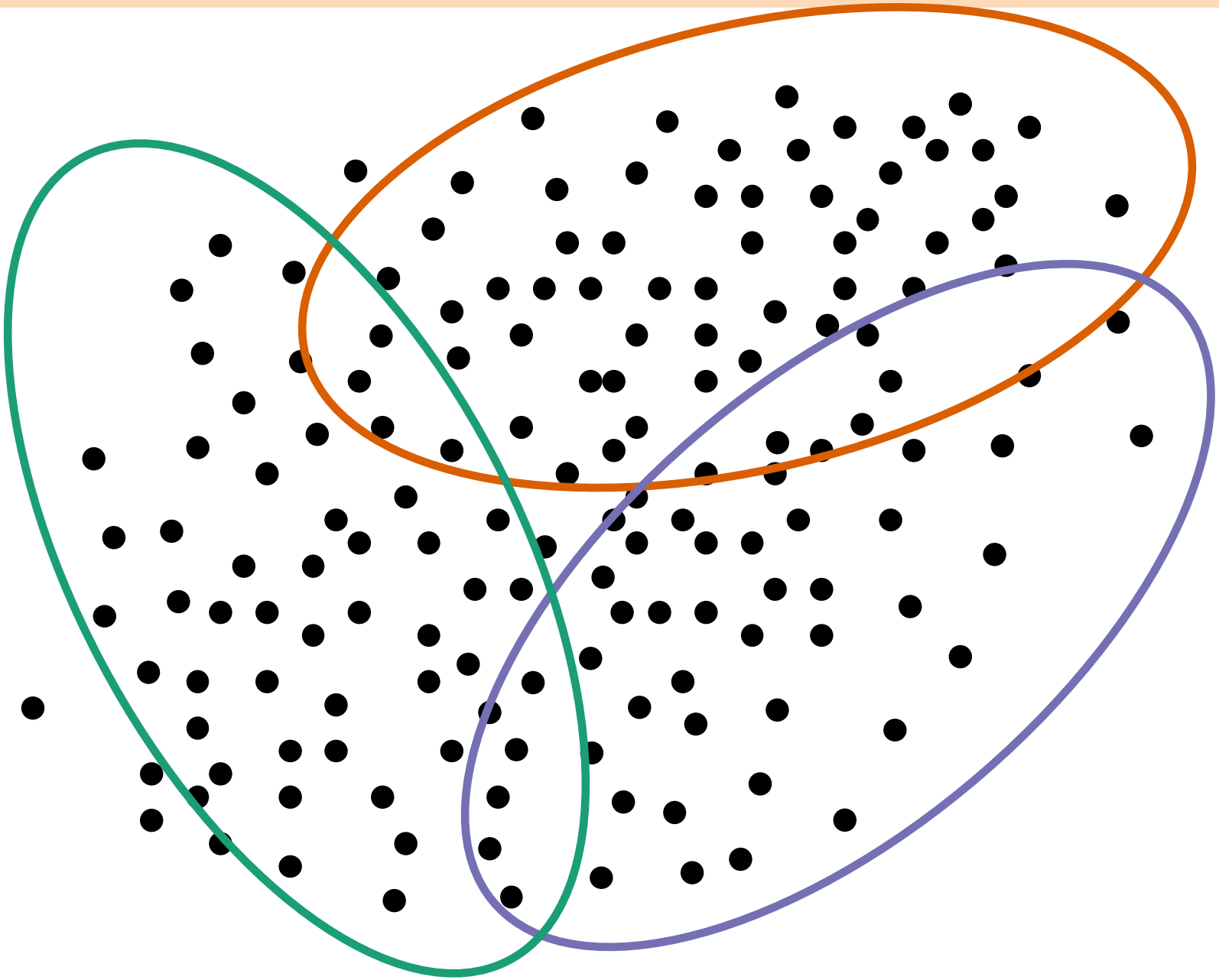
and the class label prediction for observation X

$$\hat{Y}(X) = \underset{k \in \{1,2,\ldots,K\}}{\operatorname{argmax}} \hat{\delta}_k(X)$$

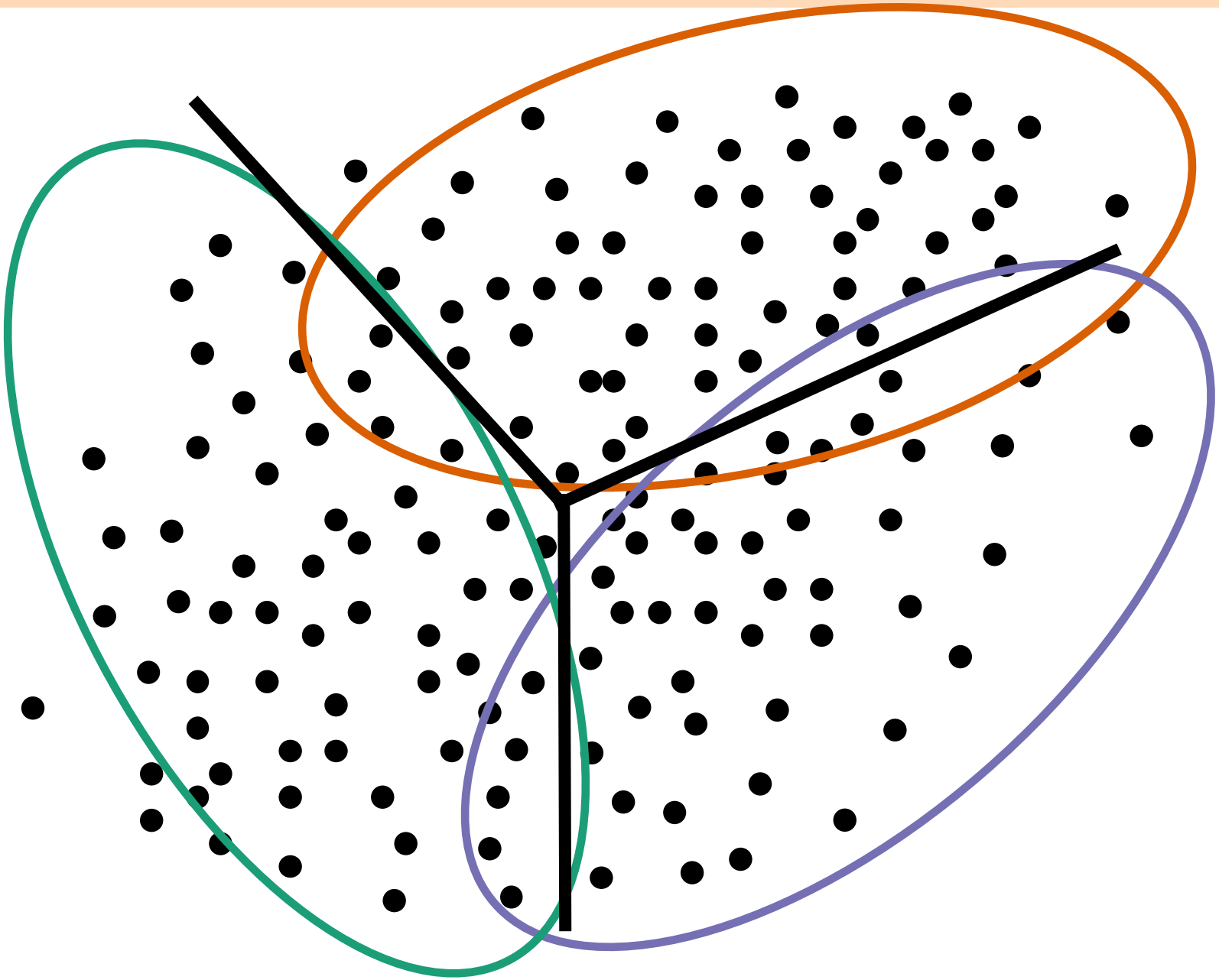
Discriminant analysis assumes normality



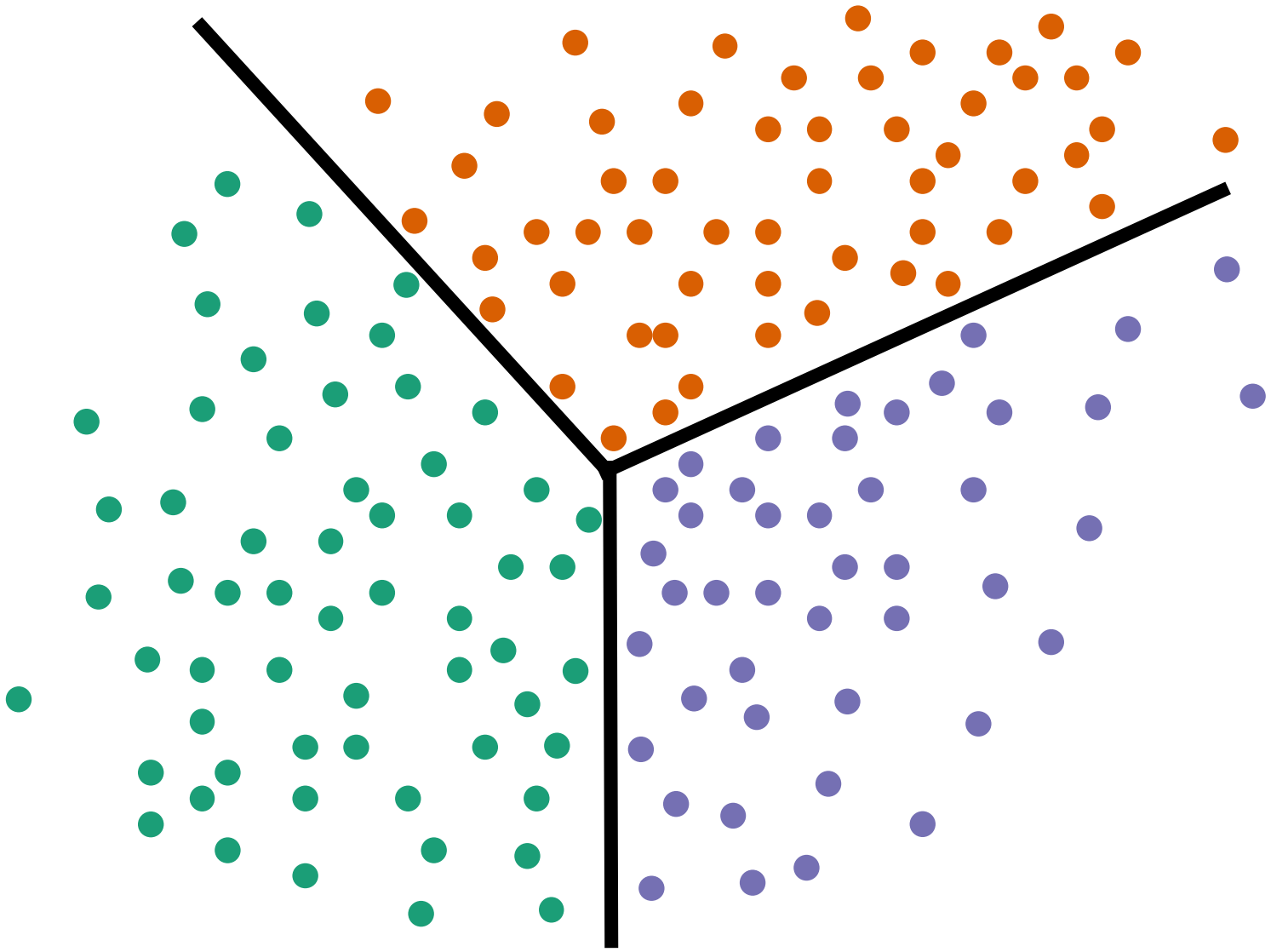
Discriminant analysis assumes normality



Discriminant analysis assumes normality



Discriminant analysis assumes normality



Relaxing the Gaussian assumption

Discriminant analysis is based on the assumption that observations are Gaussian-distributed.

That is, these hyperplanes (decision boundaries) are based on a specific generative model.

Such an assumption makes discriminant analysis incompatible with qualitative features, and may hamper classification ability if normality is violated.

Instead, it may be useful to learn the hyperplane directly from the data, rather than from an assumed generative model.

Consider a decision function based on sign with $K = 2$

Consider a situation with $K = 2$ classes.

The decision boundary is a hyperplane, which has the form

$$[1 \quad X^T]\beta = \beta_0 + \sum_{j=1}^{p} X_j \beta_j = 0$$

with

$$[1 \quad X^T]\beta > 0$$

belonging to one class and with

$$[1 \quad X^T]\beta < 0$$

belonging to the other class.

Consider a decision function based on sign with $K = 2$

Here, we only care whether $[1 \quad X^T]\beta$ is positive or negative (*i.e.*, what side of the hyperplane an observation X is on).

Let us assume that $X^T := [1 \quad X^T]$ such that the input data for an observation has been augmented to have first element equal to 1.

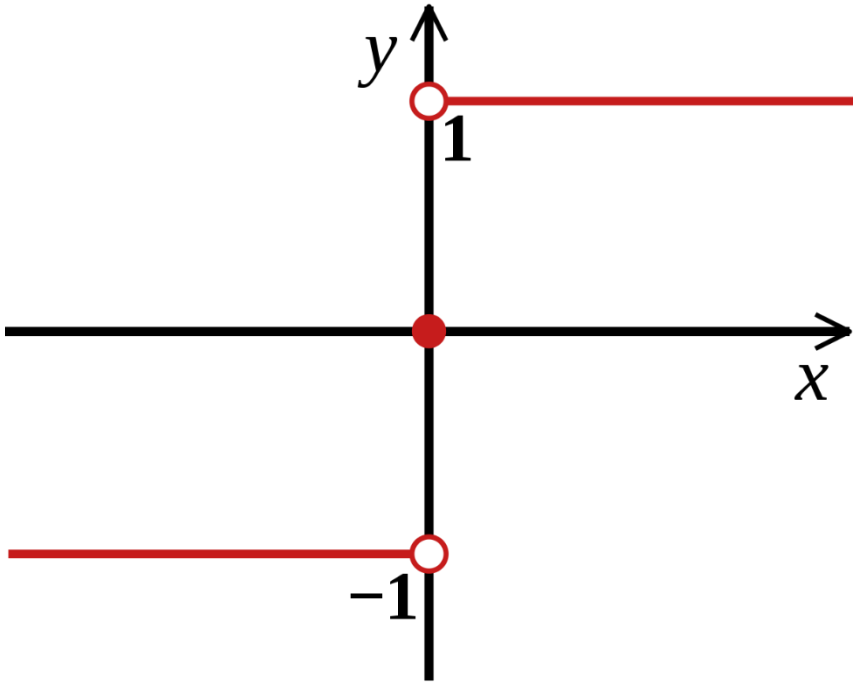
We can therefore model this hyperplane using the decision function

$$h(X;\beta) = \text{sign}(X^T\beta) = \begin{cases} -1 & \text{if } X^T\beta < 0 \\ 0 & \text{if } X^T\beta = 0 \\ 1 & \text{if } X^T\beta > 0 \end{cases}$$

and where the response $y \in \{-1,1\}$ for training observations takes values of -1 or 1.

Issue with modeling the hyperplane with sign function

The decision function $h(X;\beta) = \text{sign}(X^T\beta)$ is not differentiable



To model the hyperplane, it may be better to employ a differentiable function, as we can use optimization algorithms such as gradient descent to learn the model parameters $\beta \in \mathbb{R}^{p+1}$.

https://en.wikipedia.org/wiki/Sign_function.

What might this differentiable function look like?

As with our treatment of discriminant analysis, we will start by considering the general case of K classes.

Recall that in linear discriminant analysis, we were modeling the log odds between classes k and ℓ as

$$\log \frac{P(Y = k|X = x)}{P(Y = \ell|X = x)} = \delta_k(x) - \delta_\ell(x) = x^T\beta$$

for some coefficient vector $\beta \in \mathbb{R}^{p+1}$.

Let the denominator be the last class K, which we will use as a reference such that

$$\log \frac{P(Y = k|X = x)}{P(Y = K|X = x)} = \delta_k(x) - \delta_K(x) = x^T\beta_k$$

where $\beta_k^T = [\beta_{k0} \quad \beta_{k1} \quad \cdots \quad \beta_{kp}]$ is the coefficient vector for class k.

What might this differentiable function look like?

We are modeling the log odds between class $k \in \{1,2,\dots,K\}$ and a reference class K

$$\log \frac{P(Y=k|X=x)}{P(Y=K|X=x)} = x^T \beta_k$$

where $\beta_k^T = [\beta_{k0} \quad \beta_{k1} \quad \cdots \quad \beta_{kp}]$ is the coefficient vector for class k, except we are not assuming normality for the components of the conditional probability $P(Y=k|X=x)$.

Denote $p_k(x;\theta) = P(Y=k|X=x)$, where $\theta = \{\beta_1, \beta_2, \dots, \beta_K\}$ ($\beta_k \in \mathbb{R}^{p+1}$ for $k = 1,2,\dots,K$) is the set of model parameter vectors for classes 1 through K.

We wish to find the form of $p_k(x;\theta)$.

What might this differentiable function look like?

Exponentiating both sides, we obtain

$$\frac{p_k(x;\theta)}{p_K(x;\theta)} = \exp(x^T\beta_k)$$

and thus

$$p_k(x;\theta) = p_K(x;\theta)\exp(x^T\beta_k)$$

The sum of all the conditional probabilities must equal 1, and so

$$1 = \sum_{\ell=1}^{K} p_\ell(x;\theta) = \sum_{\ell=1}^{K} p_K(x;\theta)\exp(x^T\beta_\ell)$$

$$= p_K(x;\theta)\sum_{\ell=1}^{K}\exp(x^T\beta_\ell)$$

What might this differentiable function look like?

Solving for $p_K(x;\theta)$ gives

$$p_K(x;\theta) = \frac{1}{\sum_{\ell=1}^{K} \exp(x^T \beta_\ell)}$$

and therefore

$$p_k(x;\theta) = \frac{\exp(x^T \beta_k)}{\sum_{\ell=1}^{K} \exp(x^T \beta_\ell)}$$

for $k \in \{1,2,\ldots,K\}$.

Recall that

$$\sum_{k=1}^{K} p_k(x;\theta) = 1$$

What might this differentiable function look like?

$$p_k(x;\theta) = \frac{\exp(x^T\beta_k)}{\sum_{\ell=1}^{K}\exp(x^T\beta_\ell)}$$

For $K = 2$ classes, let class 1 be associated with response $y = 1$ and let class 2 be associated with response $y = 0$, and define

$$p_1(x;\theta) = p(x;\beta)$$

$$p_2(x;\theta) = 1 - p(x;\beta)$$

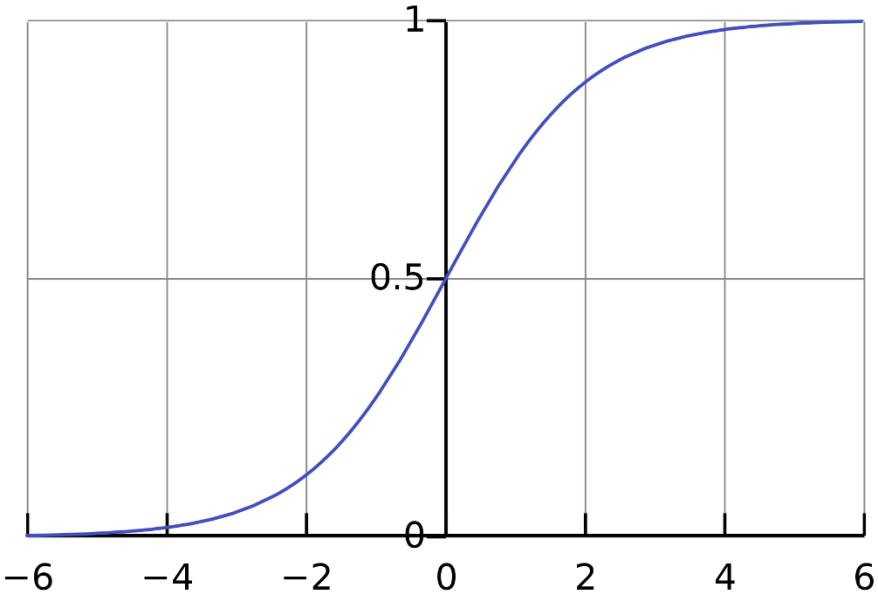
with

$$p(x;\beta) = \frac{\exp(x^T\beta)}{1 + \exp(x^T\beta)}$$

What might this differentiable function look like?

We have found that this function looks like

$$\text{logistic}(u) = \frac{\exp(u)}{1 + \exp(u)}$$



Negative values of u that are distant from 0 will lead to the logistic function close to 0, whereas positive values of u that are distant from 0 will lead to the logistic function close to 1.

https://en.wikipedia.org/wiki/Logistic_function.

How do we infer the best class?

Assume that we have a set $\{1,2,\ldots,K\}$ of labels for K classes, and have estimated the set of model parameters θ as

$$\hat{\theta} = \{\hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_K\}$$

where $\hat{\beta}_k \in \mathbb{R}^{p+1}$ is an estimate of parameter vector β_k for class k.

Given a new observation X, we can predict the class as

$$\hat{Y}(X) = \underset{k \in \{1,2,\ldots,K\}}{\operatorname{argmax}} p_k(X; \hat{\theta})$$

Even more important is that logistic regression gives the probability of each class, and not simply the predicted class label as for discriminant analysis or for indicator response regression.

We just need to figure out how to fit these model parameters θ.

Start with $K = 2$, and then extend to $K > 2$

We will begin by deriving the logistic regression classifier for $K = 2$ classes to learn the parameter set $\theta = \beta$ for $\beta \in \mathbb{R}^{p+1}$.

Based on the framework for $K = 2$, we will formulate the classifier for $K > 2$ classes, which is also known as multinomial regression, to learn the parameter set $\theta = \{\beta_1, \beta_2, \ldots, \beta_K\}$ for $\beta_k \in \mathbb{R}^{p+1}$.

The formulation for $K > 2$ follows almost directly from $K = 2$.

The Bernoulli distribution

The **Bernoulli distribution** describes the possible results of a random variable that can take on $K = 2$ categories, with each of the 2 categories having a specific probability.

Specifically, let y be an indicator variable determining whether an observation derives from the first of 2 categories and let p denote the probability the observation derives from the first category.

The distribution for observation x is then

$$f(x; p) = p^y(1 - p)^{1-y}$$

If observation x is from the first category, then $f(x; p) = p$, and if it is from the second category, then $f(x; p) = 1 - p$.

Computing the likelihood with $K = 2$ of the parameters β

Recall for $K = 2$ classes, we had class 1 associated with response $y = 1$ and class 2 be associated with response $y = 0$, and defined

$$p_1(x; \theta) = p(x; \beta)$$

$$p_2(x; \theta) = 1 - p(x; \beta)$$

with

$$p(x; \beta) = \frac{\exp(x^T \beta)}{1 + \exp(x^T \beta)}$$

We can write the likelihood of parameter vector β as the likelihood under a Bernoulli distribution

$$L(\beta) = \prod_{i=1}^{N} p(x_i; \beta)^{y_i} [1 - p(x_i; \beta)]^{1 - y_i}$$

Computing the likelihood with $K = 2$ of the parameters β

We seek β that maximizes $L(\beta)$,

$$L(\beta) = \prod_{i=1}^{N} p(x_i; \beta)^{y_i} [1 - p(x_i; \beta)]^{1-y_i}$$

which is the same as identifying β that maximizes the log likelihood

$$\begin{aligned} \ell(\beta) &= \log L(\beta) \\ &= \sum_{i=1}^{N} (y_i \log p(x_i; \beta) + (1 - y_i) \log[1 - p(x_i; \beta)]) \end{aligned}$$

Computing the likelihood with $K = 2$ of the parameters β

The log likelihood of β is then

$$\ell(\beta) = \sum_{i=1}^{N} (y_i \log p(x_i;\beta) + (1-y_i)\log[1-p(x_i;\beta)])$$

$$= \sum_{i=1}^{N} \left(y_i x_i^T \beta - y_i \log[1+\exp(x_i^T\beta)] - (1-y_i)\log[1+\exp(x_i^T\beta)]\right)$$

$$= \sum_{i=1}^{N} \left(y_i x_i^T \beta - \log[1+\exp(x_i^T\beta)]\right)$$

$$= \sum_{i=1}^{N} \left(y_i \left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j \right) - \log\left[1 + \exp\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j \right) \right] \right)$$

Fitting the model parameters β

We can employ (batch, stochastic, mini-batch) gradient descent to fit the model parameters β to training data (no closed-form solution).

Because $\ell(\beta)$ is concave, $-\ell(\beta)$ is convex and has a unique global minimum.

Therefore, we can estimate β that fit training dataset (x_i, y_i), $i = 1,2,\dots,N$, for each $y_i \in \{0,1\}$ by minimizing the cost function

$$J(\beta) = -\ell(\beta) = -\sum_{i=1}^{N} (y_i \log p(x_i;\beta) + (1-y_i)\log[1-p(x_i;\beta)])$$

Recall that $\beta \in \mathbb{R}^{p+1}$.

Finding β with batch gradient descent ($K = 2$)

Fix learning rate α

1. Randomly initialize $\beta^T = [\beta_0, \beta_1, \dots, \beta_p]$.

2. Update jth parameter $j \in \{0,1, \dots, p\}$ as

$$\beta_j \coloneqq \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta)$$

$$= \beta_j + \alpha \frac{\partial}{\partial \beta_j} \ell(\beta)$$

3. Repeat step 2 until convergence.

To implement batch gradient descent for logistic regression with $K = 2$ classes, we need to compute the partial derivative of the log likelihood function $\ell(\beta)$ with respect to each parameter β_j.

Gradient of the cost function for $K = 2$

For $j = 0$, we have

$$\frac{\partial}{\partial \beta_0}\ell(\beta) = \sum_{i=1}^{N} y_i \frac{\partial}{\partial \beta_0}\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)$$

$$-\sum_{i=1}^{N} \frac{\partial}{\partial \beta_0} \log\left[1 + \exp\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)\right]$$

$$= \sum_{i=1}^{N}\left[y_i - \frac{\exp(x_i^T\beta)}{1 + \exp(x_i^T\beta)}\right]$$

$$= \sum_{i=1}^{N}[y_i - p(x_i;\beta)]$$

Gradient of the cost function for $K = 2$

For $j \in \{1,2,\ldots,p\}$, we have

$$\frac{\partial}{\partial \beta_j}\ell(\beta) = \sum_{i=1}^{N} y_i \frac{\partial}{\partial \beta_j}\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)$$

$$- \sum_{i=1}^{N} \frac{\partial}{\partial \beta_j} \log\left[1 + \exp\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)\right]$$

$$= \sum_{i=1}^{N}\left[y_i x_{ij} - \frac{x_{ij}\exp(x_i^T\beta)}{1+\exp(x_i^T\beta)}\right]$$

$$= \sum_{i=1}^{N} x_{ij}[y_i - p(x_i;\beta)]$$

Gradient of the cost function for $K = 2$

Define the vector of probabilities

$$\mathbf{p} = \begin{bmatrix} p(x_1; \beta) \\ p(x_2; \beta) \\ \vdots \\ p(x_N; \beta) \end{bmatrix}$$

and the indicator vector

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Gradient of the cost function for $K = 2$

The gradient vector is

$$\frac{\partial}{\partial \beta}\ell(\beta) = \begin{bmatrix} \sum_{i=1}^{N}[y_i - p(x_i;\beta)] \\ \sum_{i=1}^{N} x_{i1}[y_i - p(x_i;\beta)] \\ \vdots \\ \sum_{i=1}^{N} x_{ip}[y_i - p(x_i;\beta)] \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_{11} & x_{21} & \cdots & x_{N1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1p} & x_{2p} & \cdots & x_{Np} \end{bmatrix} \begin{bmatrix} y_1 - p(x_1;\beta) \\ y_2 - p(x_2;\beta) \\ \vdots \\ y_N - p(x_N;\beta) \end{bmatrix}$$

$$= \mathbf{X}^T[\mathbf{y} - \mathbf{p}]$$

Finding β with batch gradient descent (vector, $K = 2$)

Fix learning rate α

1. Randomly initialize $\beta^T = [\beta_0, \beta_1, \ldots, \beta_p]$.

2. Update parameter vector $\beta \in \mathbb{R}^{p+1}$ as

$$\beta \coloneqq \beta + \alpha \mathbf{X}^T [\mathbf{y} - \mathbf{p}]$$

3. Repeat step 2 until convergence.

Note: the vector

$$\mathbf{p} = \begin{bmatrix} p(x_1; \beta) \\ p(x_2; \beta) \\ \vdots \\ p(x_N; \beta) \end{bmatrix}$$

can change on each iteration for step 2.

Finding β with batch gradient descent (elements, $K = 2$)

Fix learning rate α

1. Randomly initialize $\beta^T = [\beta_0, \beta_1, \dots, \beta_p]$.

2. Update jth parameter $j \in \{0,1, \dots, p\}$ as

$$\beta_j := \begin{cases} \beta_0 + \alpha \sum_{i=1}^{N} [y_i - p(x_i; \beta)] & \text{if } j = 0 \\ \beta_j + \alpha \sum_{i=1}^{N} x_{ij} [y_i - p(x_i; \beta)] & \text{if } j = 1,2, \dots, p \end{cases}$$

3. Repeat step 2 until convergence.

The categorical distribution

The **categorical distribution** (or **multinoulli distribution**) is an extension of the Bernoulli distribution to $K \geq 2$ categories.

Specifically, let y_k, $k = 1, 2, \ldots, K$ be an indicator variable determining whether an observation derives from category k and let p_k denote the probability the observation derives from category k.

The distribution of x is then

$$f(x; p_1, p_2, \ldots, p_K) = \prod_{k=1}^{K} p_k^{y_k} = p_1^{y_1} p_2^{y_2} \cdots p_K^{y_K}$$

If observation x is from category k, then

$$f(x; p_1, p_2, \ldots, p_K) = p_k$$

Computing the likelihood of the parameters θ

Consider again the general case of K classes, and let y_{ik} be an indicator variable with $y_{ik} = 1$ if $y_i = k$ and 0 otherwise.

Assuming all N training observations are independent, the likelihood of the parameter set θ is

$$L(\theta) = \prod_{i=1}^{N} \prod_{k=1}^{K} p_k(x_i; \theta)^{y_{ik}}$$

We seek θ that maximizes $L(\theta)$, which is the same as identifying θ that maximizes the log likelihood

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log p_k(x_i; \theta)$$

Fitting the model parameters θ

We can employ (batch, stochastic, mini-batch) gradient descent to fit the model parameters θ to training data (no closed-form solution).

Because $\ell(\theta)$ is concave, $-\ell(\theta)$ is convex and has a unique global minimum.

Therefore, we can estimate θ that fit training dataset (x_i, y_i), $i = 1,2,\dots,N$, for each $y_i \in \{1,2,\dots,K\}$ by minimizing the cost function

$$J(\theta) = -\ell(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log p_k(x_i;\theta)$$

Recall that $\beta_k \in \mathbb{R}^{p+1}$ with $\beta_k^T = [\beta_{k0} \quad \beta_{k1} \quad \cdots \quad \beta_{kp}]$ and that θ is defined as the collection $\theta = \{\beta_1, \beta_2, \dots, \beta_K\}$

Finding θ with batch gradient descent

Fix learning rate α

1. Randomly initialize $\theta = \{\beta_1, \beta_2, \dots, \beta_K\}$.
2. For class $k \in \{1,2, \dots, K\}$, update jth parameter $j \in \{0,1, \dots, p\}$ as

$$\beta_{kj} \coloneqq \beta_{kj} - \alpha \frac{\partial}{\partial \beta_{kj}} J(\theta)$$

$$= \beta_{kj} + \alpha \frac{\partial}{\partial \beta_{kj}} \ell(\theta)$$

3. Update θ as $\theta = \{\beta_1, \beta_2, \dots, \beta_K\}$.
4. Repeat steps 2 and 3 until convergence.

Computing the gradient of the cost function

Skipping the derivation, for $j \in \{1,2,\ldots,p\}$, we have

$$\frac{\partial}{\partial \beta_{kj}} \ell(\theta) = \sum_{i=1}^{N} x_{ij}[y_{ik} - p_k(x_i;\theta)]$$

Define the vector of probabilities

$$\mathbf{p}_k = \begin{bmatrix} p_k(x_1;\theta) \\ p_k(x_2;\theta) \\ \vdots \\ p_k(x_N;\theta) \end{bmatrix}$$

and the indicator vector (recall as kth column of first classifier)

$$\mathbf{y}_k = \begin{bmatrix} y_{1k} \\ y_{2k} \\ \vdots \\ y_{Nk} \end{bmatrix}$$

Computing the gradient of the cost function

The gradient vector for class $k \in \{1,2,\ldots,K\}$ is

$$\frac{\partial}{\partial \beta_k}\ell(\theta) = \begin{bmatrix} \sum_{i=1}^{N}[y_{ik} - p_k(x_i;\theta)] \\ \sum_{i=1}^{N} x_{i1}[y_{ik} - p_k(x_i;\theta)] \\ \vdots \\ \sum_{i=1}^{N} x_{ip}[y_{ik} - p_k(x_i;\theta)] \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_{11} & x_{21} & \cdots & x_{N1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1p} & x_{2p} & \cdots & x_{Np} \end{bmatrix} \begin{bmatrix} y_{1k} - p_k(x_1;\theta) \\ y_{2k} - p_k(x_2;\theta) \\ \vdots \\ y_{Nk} - p_k(x_N;\theta) \end{bmatrix}$$

$$= \mathbf{X}^T[\mathbf{y}_k - \mathbf{p}_k]$$

Finding θ with batch gradient descent (vector notation)

Fix learning rate α

1. Randomly initialize $\theta = \{\beta_1, \beta_2, \ldots, \beta_K\}$.
2. For class $k \in \{1,2, \ldots, K\}$, update parameter vector $\beta_k \in \mathbb{R}^{p+1}$ as

$$\beta_k := \beta_k + \alpha \mathbf{X}^T [\mathbf{y}_k - \mathbf{p}_k]$$

3. Update θ as $\theta = \{\beta_1, \beta_2, \ldots, \beta_K\}$.
4. Repeat steps 2 and 3 until convergence.

Note: the vector

$$\mathbf{p}_k = \begin{bmatrix} p_k(x_1; \theta) \\ p_k(x_2; \theta) \\ \vdots \\ p_k(x_N; \theta) \end{bmatrix}$$

can change on each iteration for step 2.

Finding θ with batch gradient descent (elementwise)

Fix learning rate α

1. Randomly initialize $\theta = \{\beta_1, \beta_2, \dots, \beta_K\}$.

2. For class $k \in \{1,2, \dots, K\}$, update jth parameter $j \in \{0,1, \dots, p\}$ as

$$\beta_{kj} := \begin{cases} \beta_{k0} + \alpha \sum_{i=1}^{N} [y_{ik} - p_k(x_i; \theta)] & \text{if } j = 0 \\ \beta_{kj} + \alpha \sum_{i=1}^{N} x_{ij} [y_{ik} - p_k(x_i; \theta)] & \text{if } j = 1,2, \dots, p \end{cases}$$

3. Update θ as $\theta = \{\beta_1, \beta_2, \dots, \beta_K\}$.

4. Repeat steps 2 and 3 until convergence.

Error rates for parametric-based classifiers

Data contains $K = 11$ classes with $p = 10$ features, of which 3 of the features account for 90% of the variance (as computed by PCA).

Logistic (multinomial) regression performs slightly better than other methods, potentially due to slight model violations for LDA and QDA, with linear regression exhibiting masking.

Technique	Error Rates	
	Training	Test
Linear regression	0.48	0.67
Linear discriminant analysis	0.32	0.56
Quadratic discriminant analysis	0.01	0.53
Logistic regression	0.22	0.51

Adapted from Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Non-parametric classification

LDA, QDA, and logistic (multinomial) regression are all parametric methods, in that they make an assumption about the form of the decision boundaries separating classes.

Specifically, LDA and logistic regression assume a linear decision boundary, whereas QDA assumes a quadratic decision boundary.

In addition, LDA and QDA make Gaussian assumptions.

If the assumptions of these methods are correct, then they should perform well; but if they are incorrect, then they may perform poorly.

We therefore may not wish to make assumptions about the particular form of the decision boundary.

k-nearest neighbors classification

We will consider a simple non-parametric classification method, known as ***k*-nearest neighbors classification** (**kNN classification**).

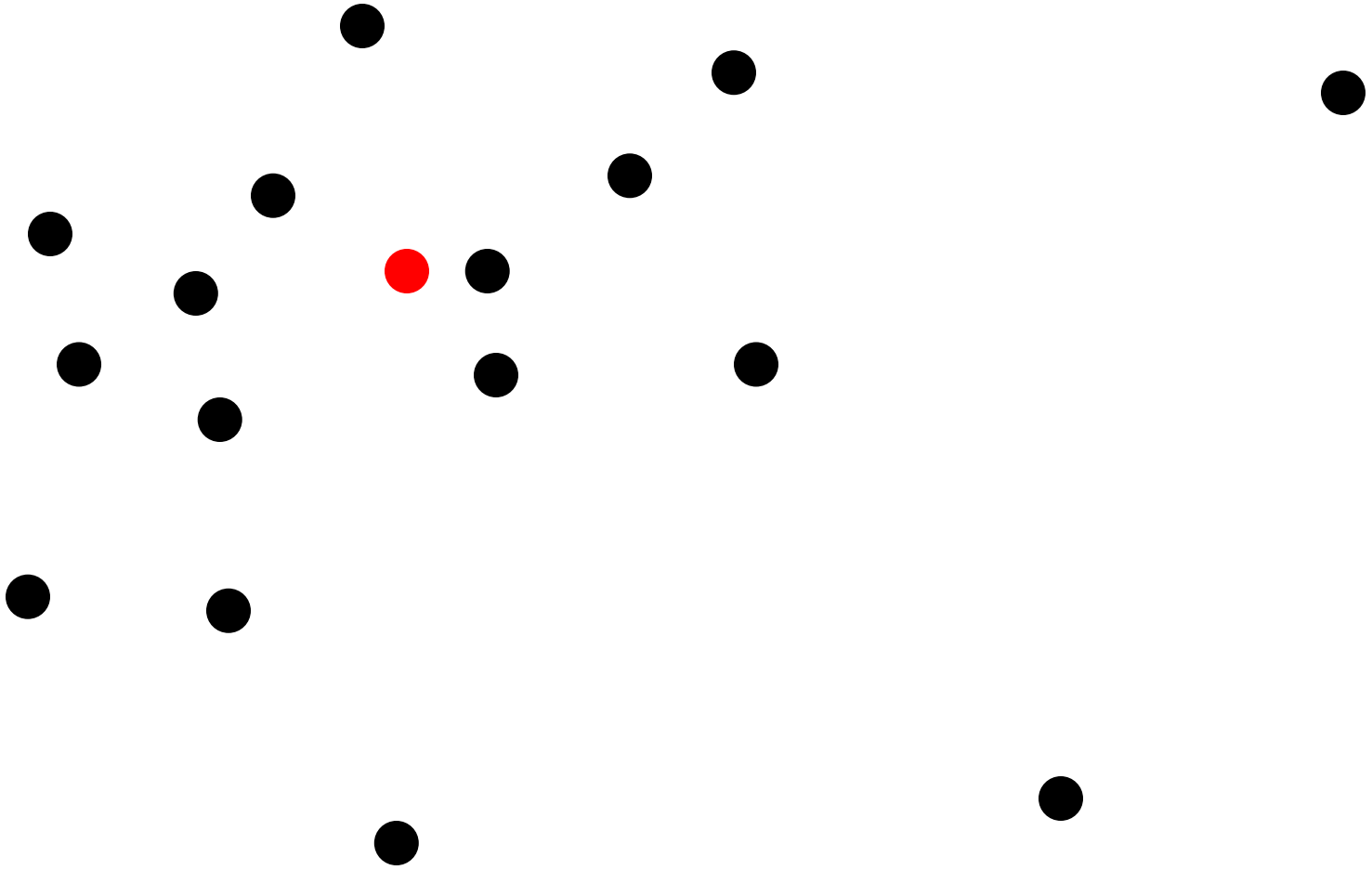
We start by choosing a positive integer k, $k = 1,2, ...$, which is a parameter of kNN classification, even though we say that kNN classification is non-parametric.

The parameter k represents the number of training observations that will be used to predict the output class y for the input X of a test observation.

How do we choose these k training observations used to make this prediction?

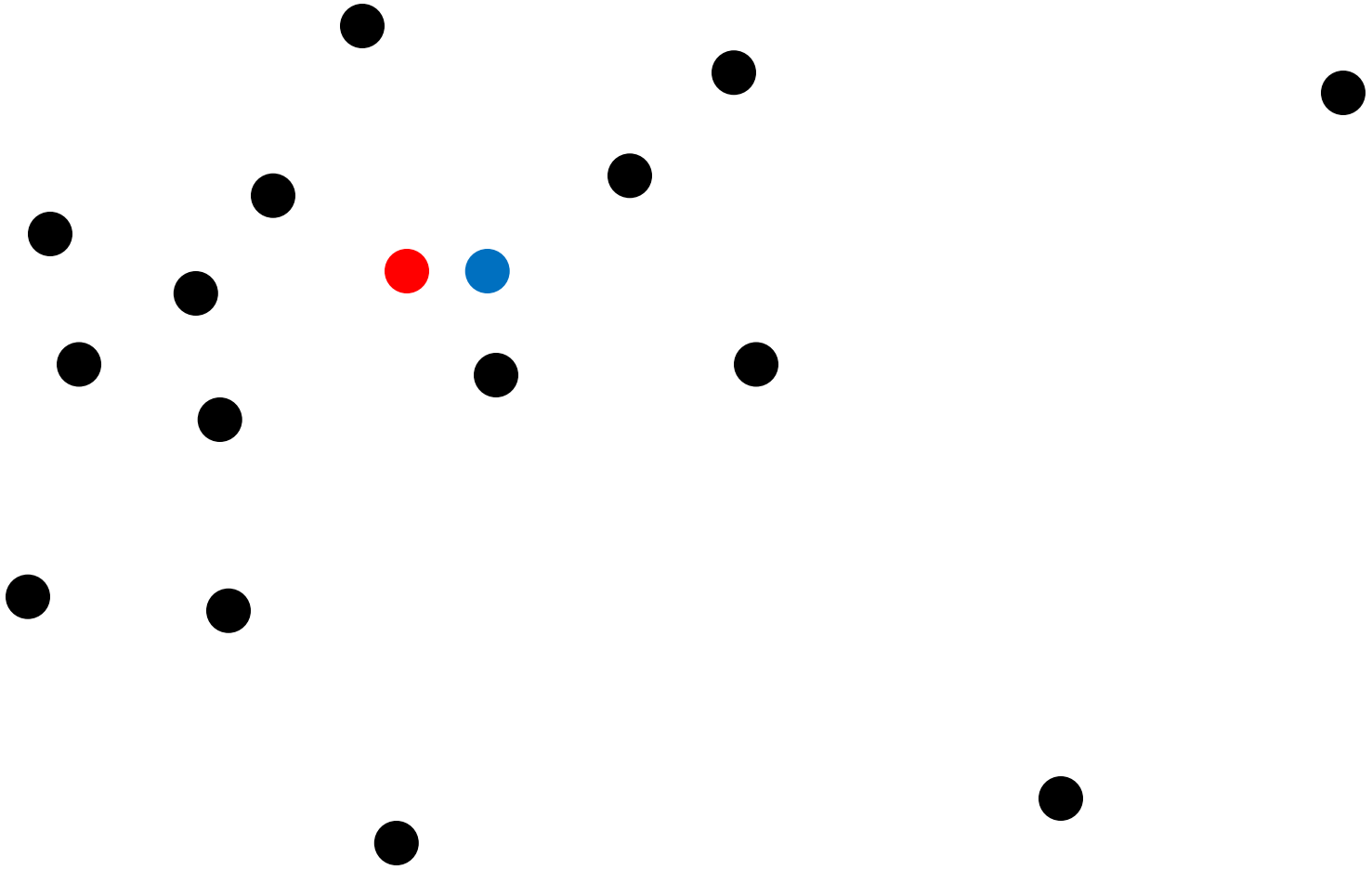
Neighborhood of the test observation

Let $\mathcal{N}(X)$ denote the set of k training observations (blue) that are the closest to test input X (red).



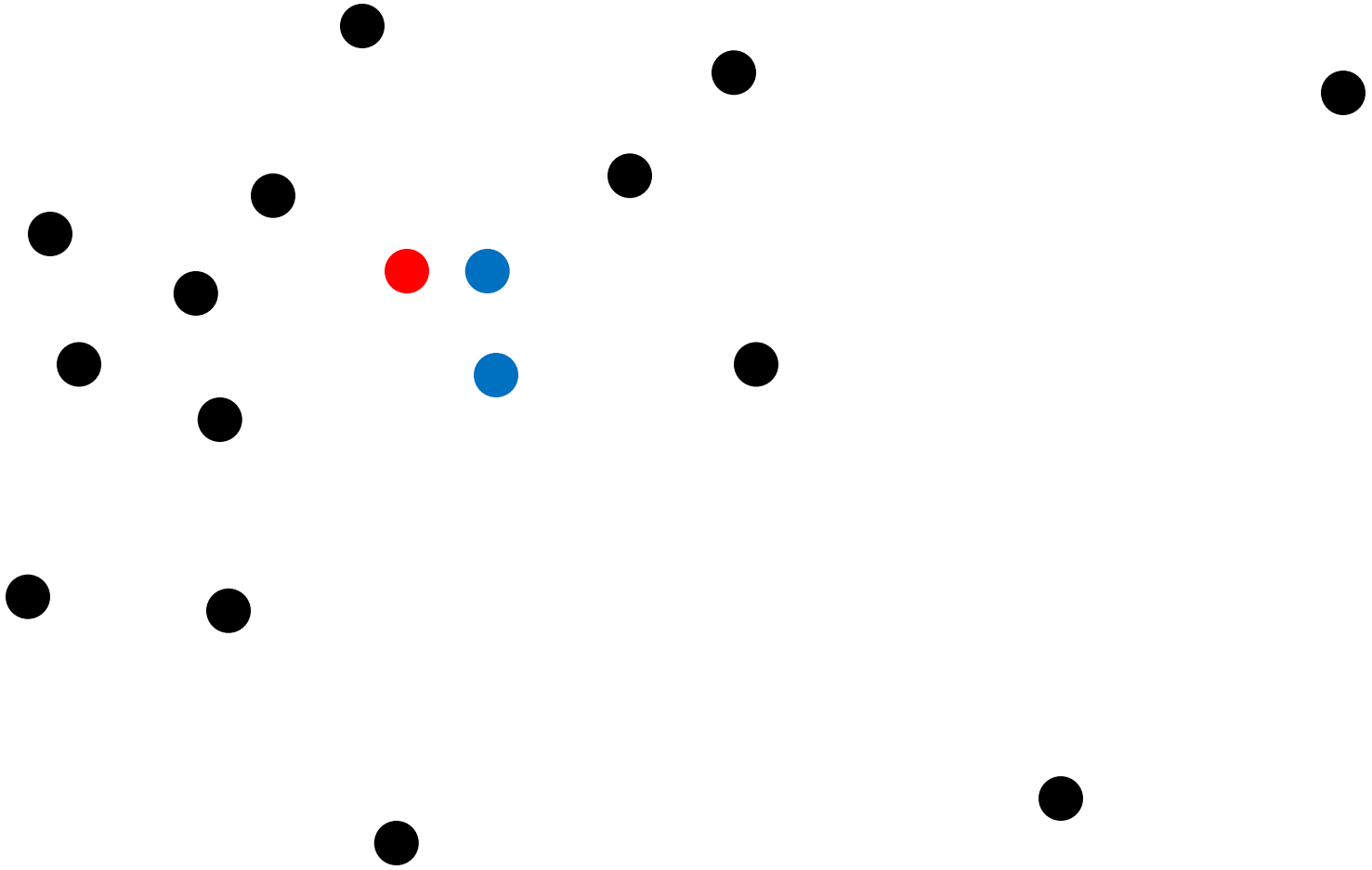
Neighborhood of the test observation ($k = 1$)

Let $\mathcal{N}(X)$ denote the set of k training observations (blue) that are the closest to test input X (red).



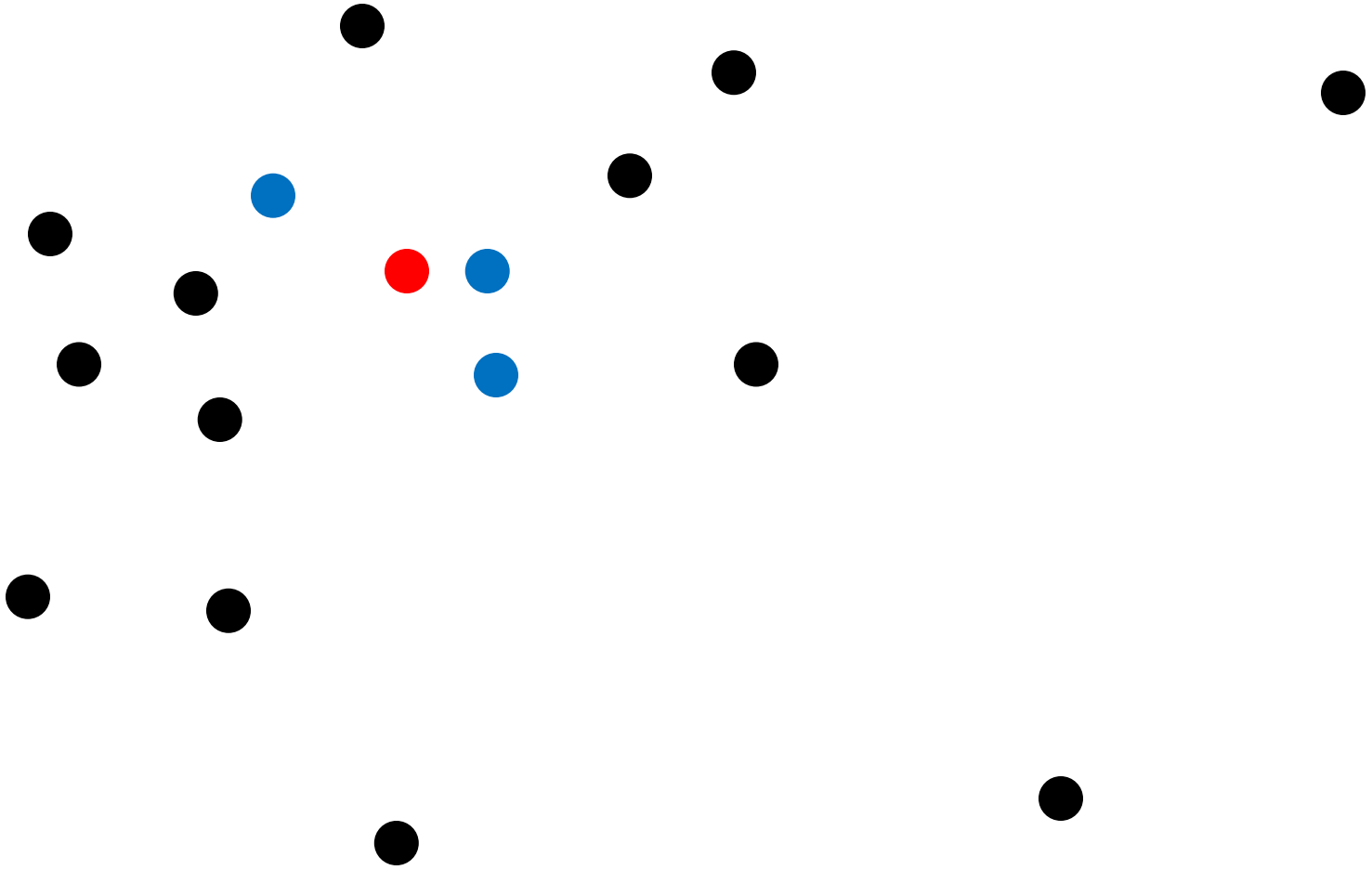
Neighborhood of the test observation ($k = 2$)

Let $\mathcal{N}(X)$ denote the set of k training observations (blue) that are the closest to test input X (red).



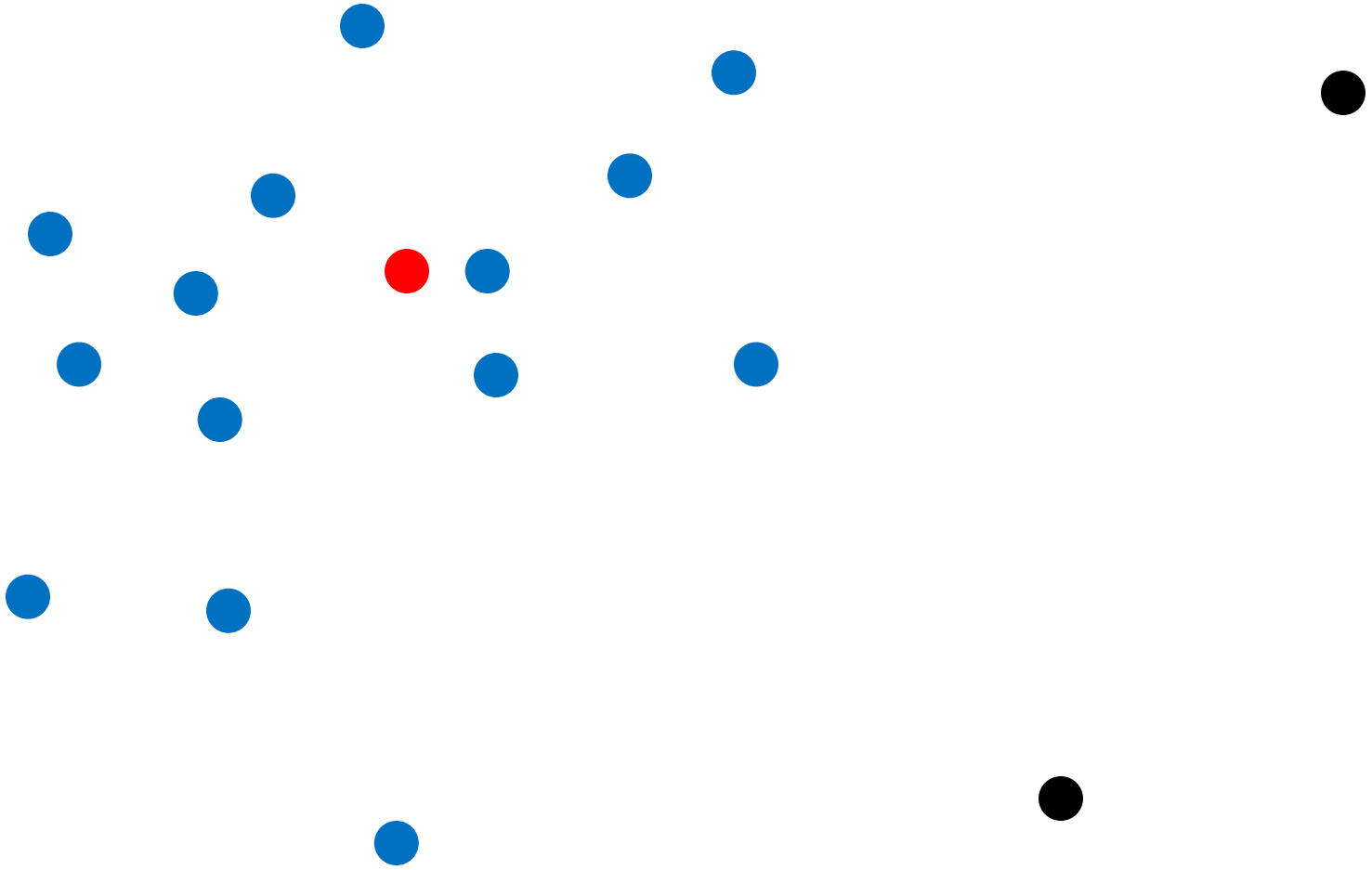
Neighborhood of the test observation ($k = 3$)

Let $\mathcal{N}(X)$ denote the set of k training observations (blue) that are the closest to test input X (red).



Neighborhood of the test observation ($k = 14$)

Let $\mathcal{N}(X)$ denote the set of k training observations (blue) that are the closest to test input X (red).



Predicting the output Y of test input X

Assuming parameter k, we will estimate $\hat{Y} \in \{1,2,\ldots,K\}$ as

$$\hat{Y}(X) = \underset{y \in \{1,2,\ldots,K\}}{\operatorname{argmax}} P(Y = y|X)$$

where

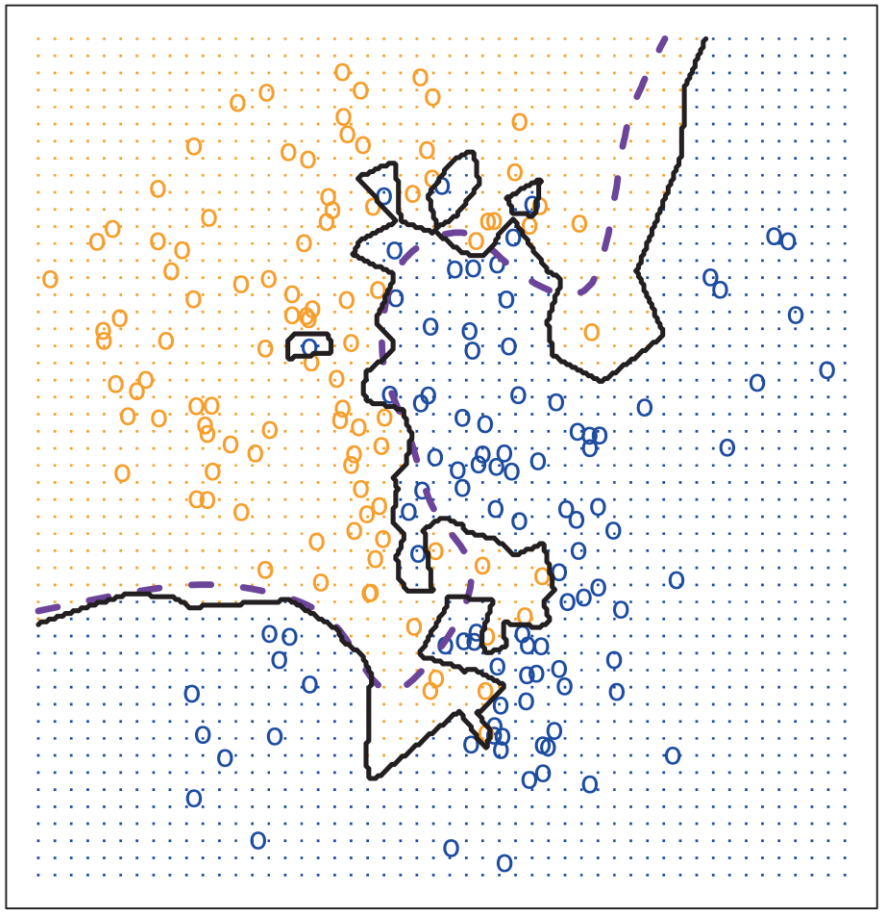
$$P(Y = y|X) = \frac{1}{k} \sum_{x_i \in \mathcal{N}(X)} I(y_i = y)$$

Therefore, we predict the class Y of test input X, based on the sample proportion of the k closest training observations.

As with kNN regression, k can be chosen through cross validation (CV), and kNN classification will suffer from the curse of dimensionality in the same way that kNN regression does.

Example classification with $p = 2$ features ($k = 1$)

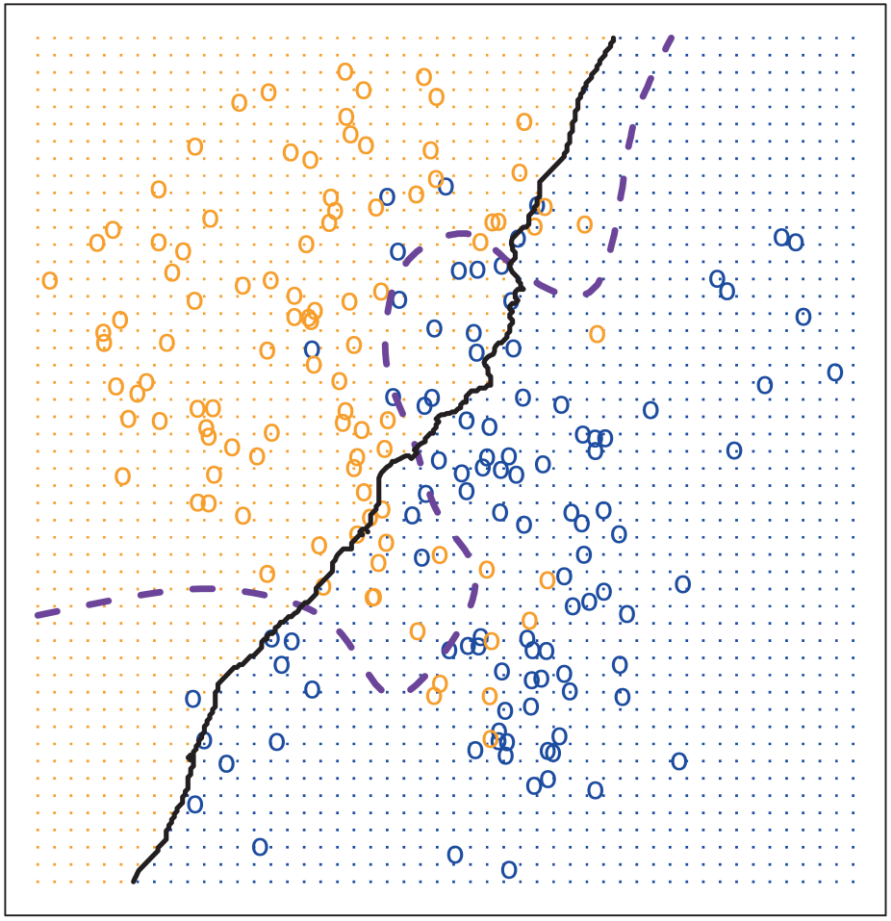
kNN with $k = 1$ overfits the data, leading to high variance.



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Example classification with $p = 2$ features ($k = 100$)

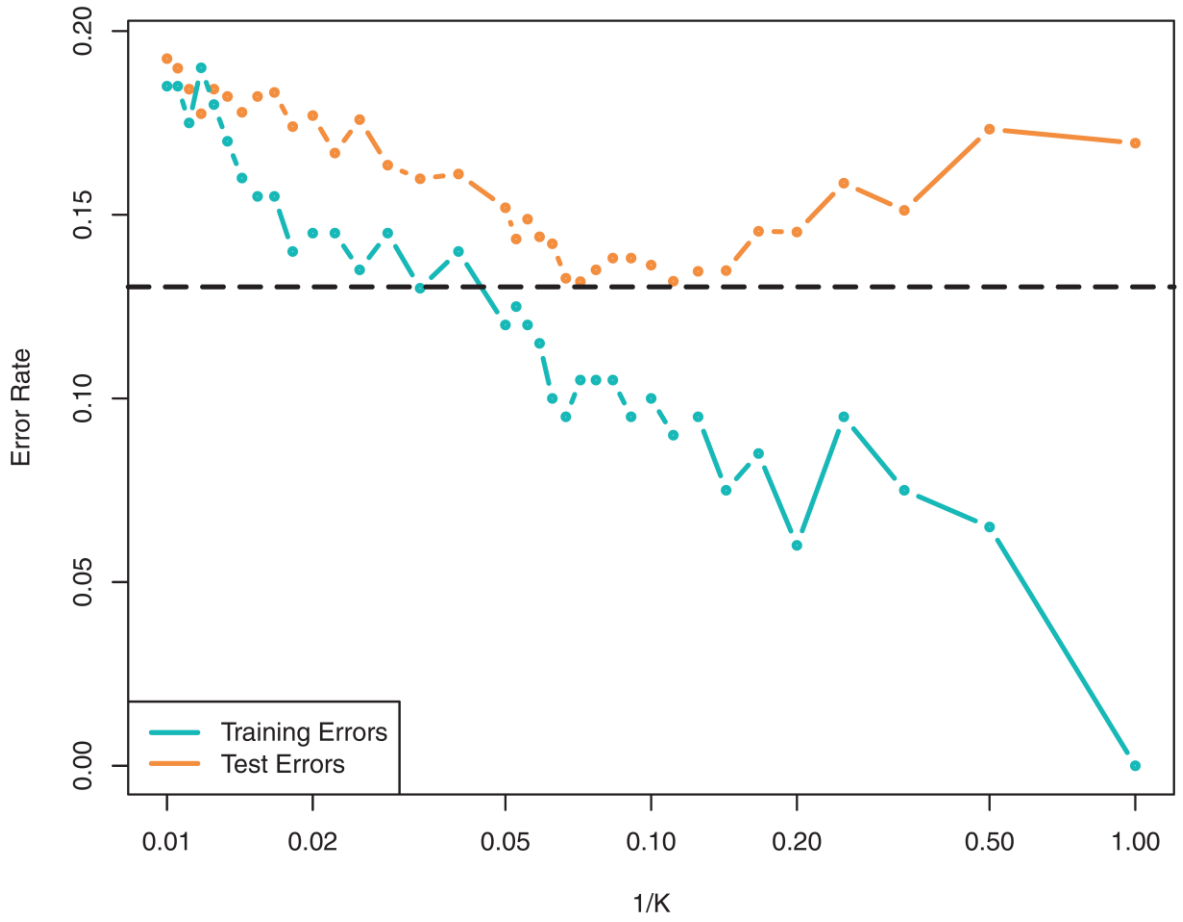
kNN with $k = 100$ underfits the data, leading to high bias.



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Example test error for classification with $p = 2$ features

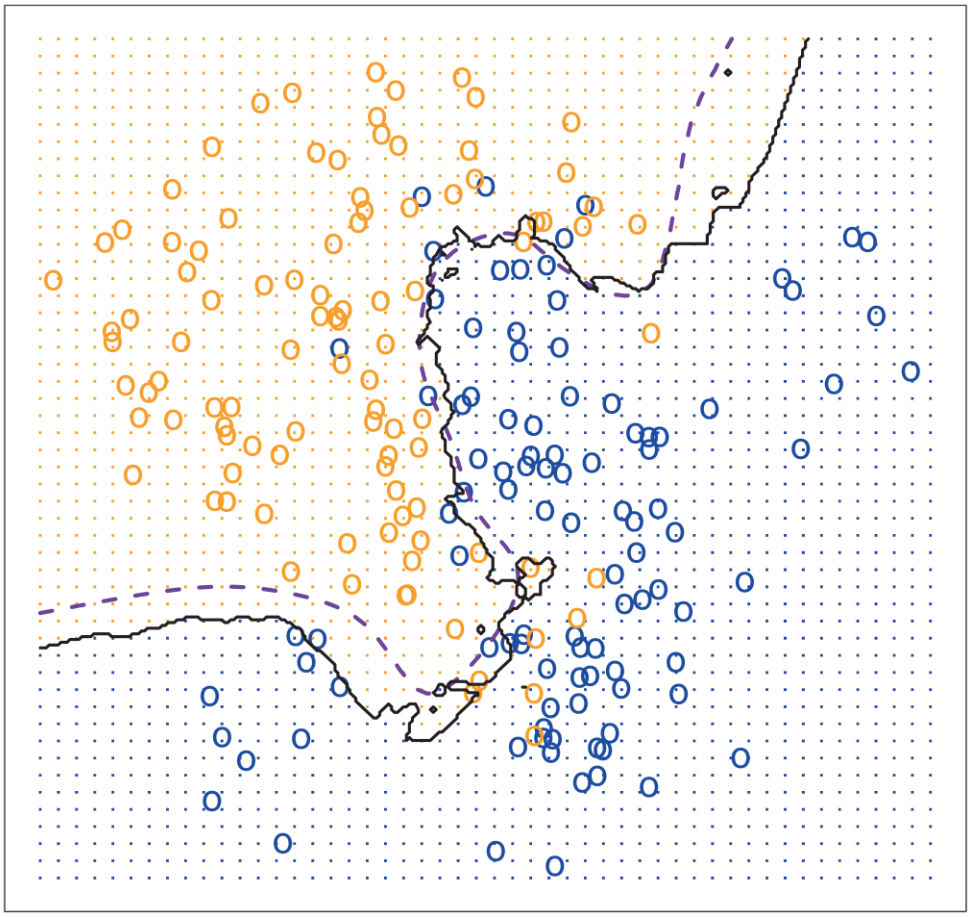
Test error is smallest for intermediate values of k, with values close to $k = 10$ as optimal for trading off bias and variance.



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Example classification with $p = 2$ features ($k = 10$)

kNN with $k = 10$ fits the data well, as it made a tradeoff between bias and variance.

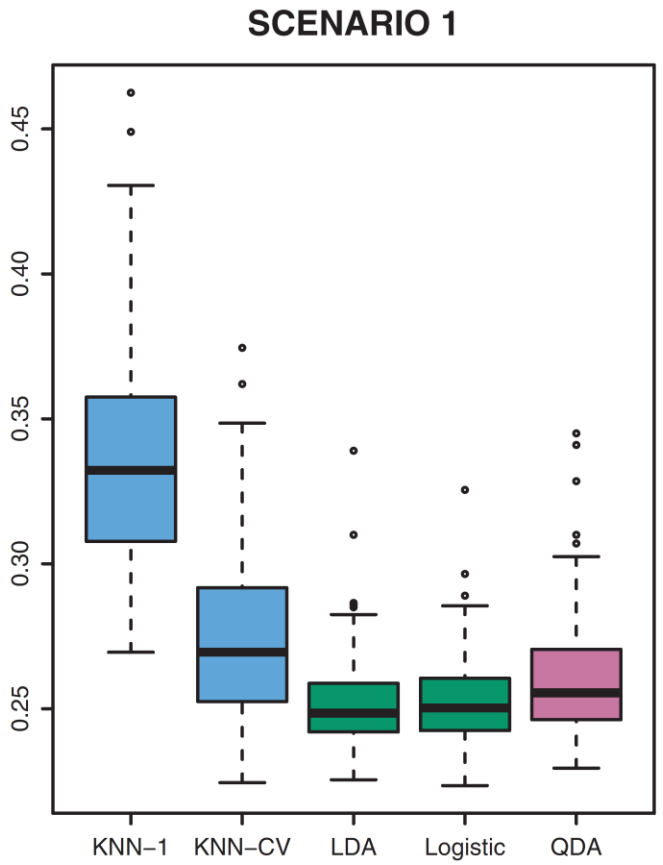


James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Test errors of kNN, LDA, QDA, and logistic regression

Scenario 1 (linear): $K = 2$ classes with $p = 2$ features. Observations within each class were uncorrelated and normally-distributed with differences in means between the 2 classes.

LDA performs best as it satisfies the assumptions of how the data were generated, with logistic regression performing almost as well.

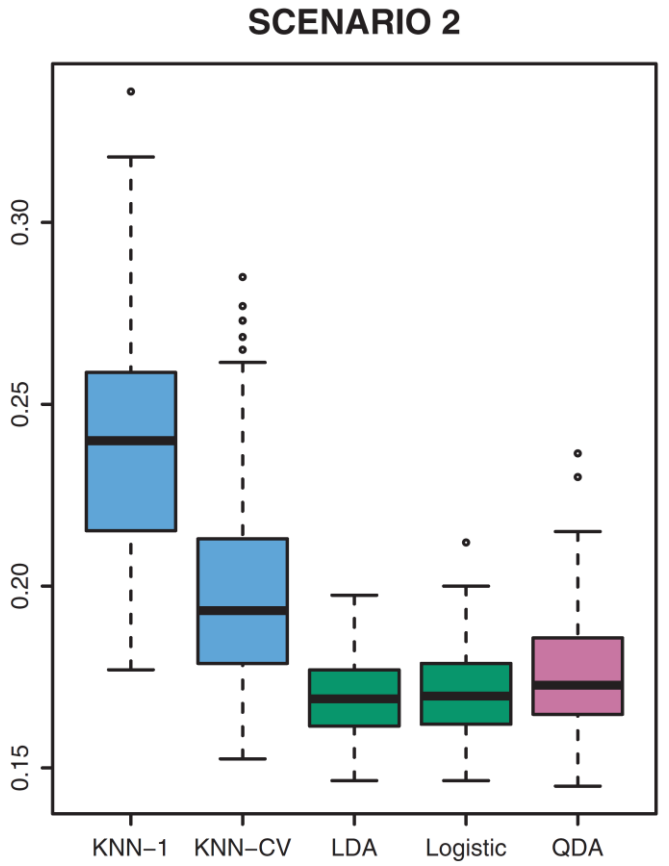


James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Test errors of kNN, LDA, QDA, and logistic regression

Scenario 2 (linear): $K = 2$ classes with $p = 2$ features. Observations within each class were uncorrelated and normally-distributed with differences in means between the 2 classes and the features had correlation of -0.05.

LDA performs best as it satisfies the assumptions of how the data were generated, with logistic regression performing almost as well.

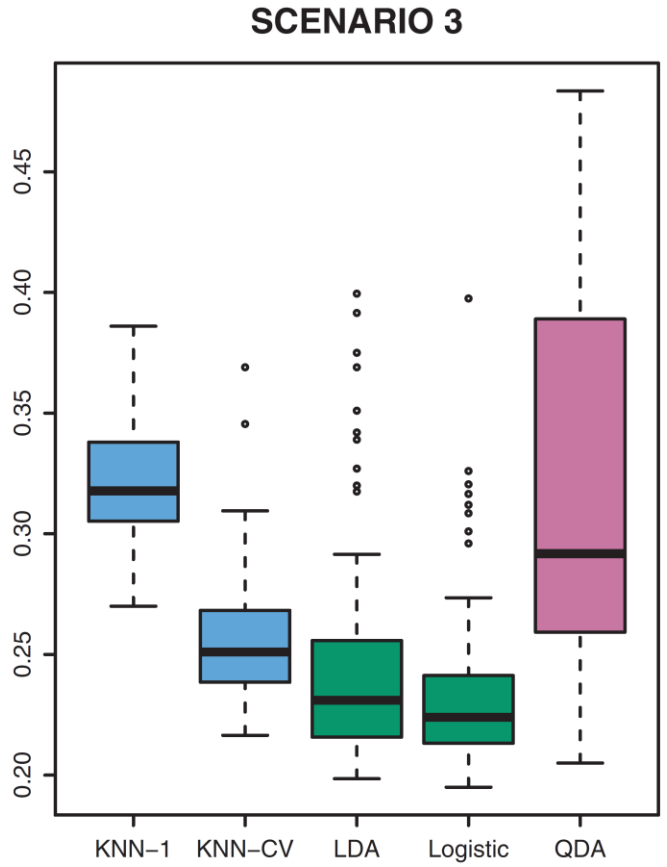


James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Test errors of kNN, LDA, QDA, and logistic regression

Scenario 3 (linear): $K = 2$ classes with $p = 2$ features. Observations were drawn from a non-Gaussian distribution.

Logistic regression performs better than LDA because of the violation of the Gaussian assumption, but they both perform better than QDA due to the decision boundary still being linear.

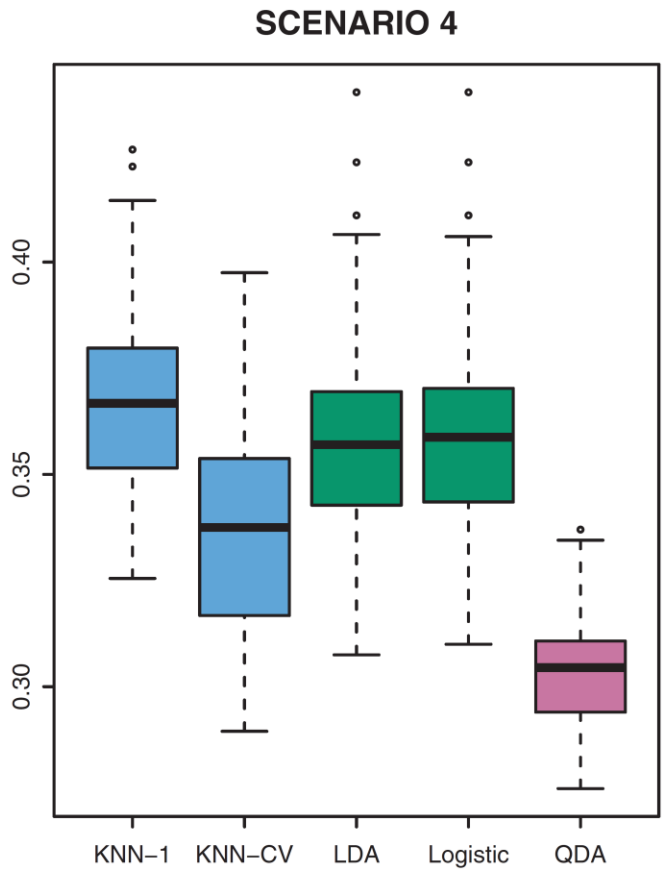


James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Test errors of kNN, LDA, QDA, and logistic regression

Scenario 4 (non-linear): $K = 2$ classes with $p = 2$ features. Observations within each class were normally-distributed with correlation between features of 0.05 in the first class and of -0.05 in the second class.

Here the classes have different covariance matrices and therefore have a non-linear decision boundary, yet still satisfy the normality assumption, and therefore QDA performs best.

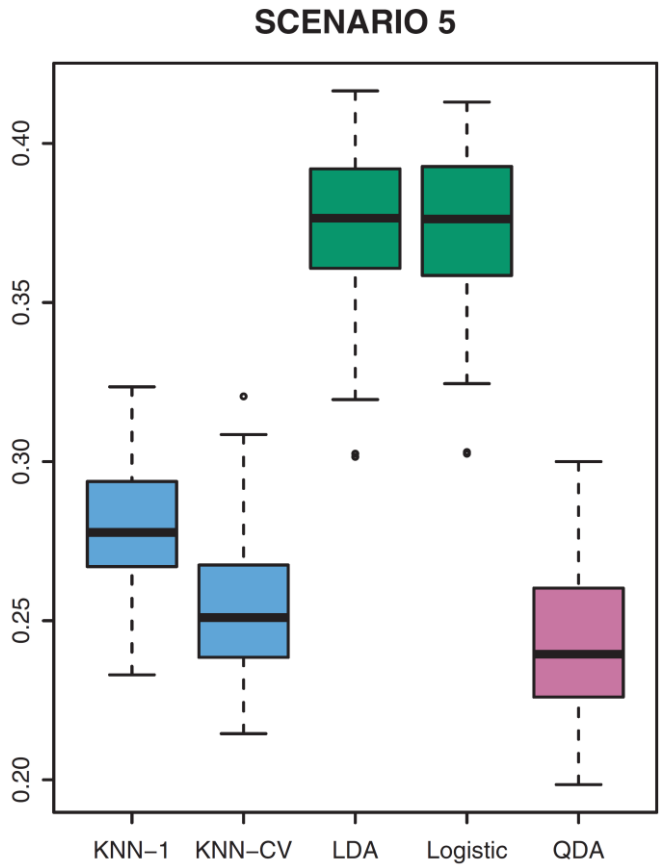


James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Test errors of KNN, LDA, QDA, and logistic regression

Scenario 5 (non-linear): $k = 2$ classes with $p = 2$ features. Observations within each class were normally-distributed with uncorrelated features, with the responses sampled from the logistic function with X_1^2, X_2^2, and $X_1 X_2$ as features, leading to a quadratic decision boundary.

QDA performs well because of the quadratic decision boundary, which it employs in its assumption.

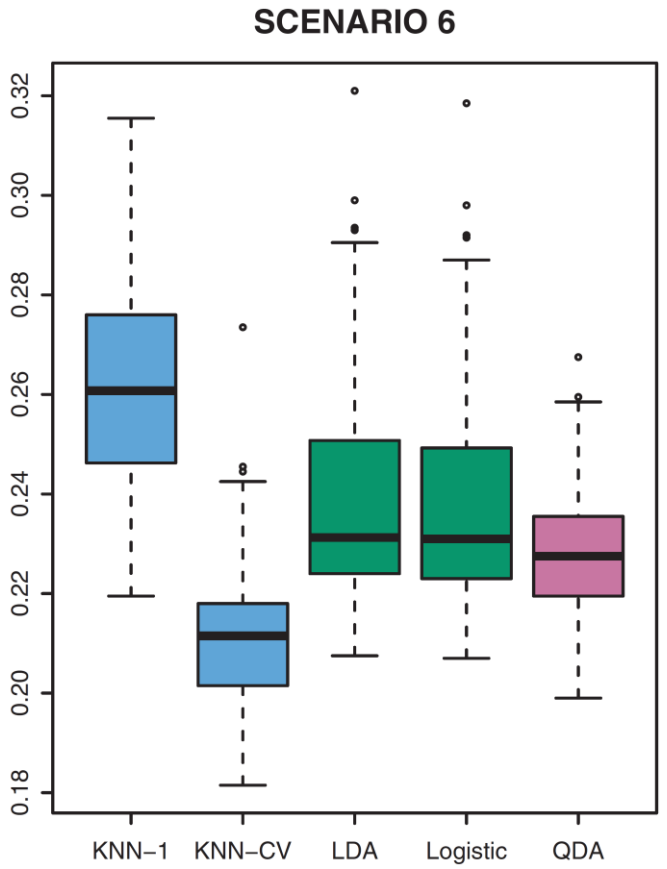


James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Test errors of KNN, LDA, QDA, and logistic regression

Scenario 6 (non-linear): $k = 2$ classes with $p = 2$ features. Observations within each class were normally-distributed with uncorrelated features, with the responses sampled from a non-linear function leading to a more complicated decision boundary than a quadratic one.

LDA, QDA, and logistic regression perform poorly, and instead the non-parametric k-nearest neighbor (kNN) classifier with k chosen through cross validation (CV) performs best.



James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

What about parameter overfitting?

We can use the same regularization techniques (ridge regression, lasso, elastic net, trend filtering, fused lasso, group lasso) to penalize the log likelihood function.

We will consider the ridge regression penalty, as it is differentiable.

The ridge-penalized logistic regression (with $K = 2$ classes) cost function is

$$J(\beta, \lambda) = -\ell(\beta) + \lambda \sum_{j=1}^{p} \beta_j^2$$

Recall that we do not penalize the intercept (β_0), and for ridge regression to be effective we must standardize the features.

Computing the gradient of $J(\beta, \lambda)$

For $k = 0$, we have

$$\frac{\partial}{\partial \beta_0} J(\beta, \lambda) = -\frac{\partial}{\partial \beta_0} \ell(\beta) + \lambda \frac{\partial}{\partial \beta_0} \sum_{j=1}^{p} \beta_j^2 = -\frac{\partial}{\partial \beta_0} \ell(\beta)$$

$$= -\sum_{i=1}^{N} [y_i - p(x_i; \beta)]$$

and for $k \in \{1,2, \ldots, p\}$, we have

$$\frac{\partial}{\partial \beta_k} J(\beta, \lambda) = -\frac{\partial}{\partial \beta_k} \ell(\beta) + \lambda \frac{\partial}{\partial \beta_k} \sum_{j=1}^{p} \beta_j^2$$

$$= -\sum_{i=1}^{N} x_{ik} [y_i - p(x_i; \beta)] + 2\lambda \beta_k$$

Finding β with ridge batch gradient descent ($K = 2$)

Fix learning rate α

1. Randomly initialize $\beta^T = [\beta_0, \beta_1, \dots, \beta_p]$.

2. Update jth parameter $j \in \{0,1, \dots, p\}$ as

$$\beta_j := \begin{cases} \beta_0 + \alpha \sum_{i=1}^{N} [y_i - p(x_i; \beta)] & \text{if } j = 0 \\ \beta_j + \alpha \left(\sum_{i=1}^{N} x_{ij} [y_i - p(x_i; \beta)] - 2\lambda\beta_j \right) & \text{if } j = 1,2, \dots, p \end{cases}$$

3. Repeat step 2 until convergence.