```
Prior probabilities of groups:
    No    Yes
0.9667 0.0333

Group means:
      balance studentYes
No   803.9438  0.2914037
Yes 1747.8217  0.3813814

> qda_pred_def_1 <- predict(qda_fit_def_1)
> names(qda_pred_def_1)
[1] "class"     "posterior"

> table(qda_pred_def_1$class, Default$default)

        No  Yes
  No  9637  244
  Yes   30   89
> mean(qda_pred_def_1$class == Default$default)
[1] 0.9726
> mean(qda_pred_def_1$class != Default$default)
[1] 0.0274
```

Naive Bayes

- The naive Bayes classifier takes a different tack for estimating $f_1(x), ..., f_K(x)$ in

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

- Instead of assuming that these functions belong to a particular family of distributions (e.g. multivariate normal), we instead make a single assumption:

Within the kth class, the p predictors are independent.

That is, for $k = 1, ..., K$, $x = (x_1, \cdots, x_p)$

$f_k(x) = f_{k1}(x_1) \times f_{k2}(x_2) \times \cdots \times f_{kp}(x_p)$

where f_{kj} is the density function of the jth predictor among observations in the kth class.

- Why is this assumption so powerful?
- Essentially, estimating a p-dimensional density function is challenging because we must consider not only the marginal distribution of each predictor, that is, the distribution of each predictor on its own, but also the joint distribution of the predictors, that is, the association between the different predictors.
- But by assuming that the p covariates are independent within each class, we completely eliminate the need to worry about the association between the p predictors, because we have simply assumed that there is no association between the predictors!
- Do we really believe the naive Bayes assumption that the p covariates are independent within each class?
- In most settings, we do not. But even though this modeling assumption is made for convenience, it often leads to pretty decent results, especially in settings where n is not large enough relative to p for us to effectively estimate the joint distribution of the predictors within each class.
- Once we have made the naive Bayes assumption, we can plug

$$f_k(x) = f_{k1}(x_1) \times f_{k2}(x_2) \times \ldots \times f_{kp}(x_p),$$

into

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

to obtain an expression for the posterior probability.

- To estimate the one-dimensional density function f_{kj} using training data $x_{1j}, ..., x_{nj}$, we have a few options.
  - If X_j is quantitative, then we can assume that $X_j|Y = k \sim N(\mu_{jk}, \sigma^2_{jk})$. In other words, we assume that within each class, the jth predictor is drawn from a (univariate) normal distribution.

  - If X_j is quantitative, then another option is to use a non-parametric estimate for f_{kj}. A very simple way to do this is by making a histogram for the observations of the jth predictor within each class. Then we can estimate $f_{kj}(x_j)$ as the fraction of the training observations in the kth class that belong to the same histogram bin as x_j.

  - If X_j is qualitative, then we can simply count the proportion of training observations for the jth predictor corresponding to each class.
    For instance, suppose that $X_j \in \{1, 2, 3\}$, and we have 100 observations in the kth class. Suppose that the jth predictor takes on values of 1, 2, and 3 in 32, 55, and 13 of those observations, respectively. Then we can estimate f_{kj} as

$$\hat{f}_{kj}(x_j) = \begin{cases} 0.32 & \text{if } x_j = 1 \\ 0.55 & \text{if } x_j = 2 \\ 0.13 & \text{if } x_j = 3 \end{cases}$$

Performing Naive Bayes in R

Now we will fit a naive Bayes model to the Default data in order to predict whether or not an individual will default on the basis of credit card balance and student status.

Naive Bayes is implemented in R using the naiveBayes() function, which is part of the e1071 library. The syntax is identical to that of lda() and qda().

By default, this implementation of the naive Bayes classifier models each quantitative feature using a Gaussian distribution.

```
> library(ISLR)
> attach(Default)
> names(Default)
[1] "default" "student" "balance" "income"
> library(e1071)
> nb_fit <- naiveBayes(default ~ balance + student, data = Default)
> nb_fit

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
    No    Yes
0.9667 0.0333

Conditional probabilities:
     balance
Y           [,1]     [,2]
  No   803.9438 456.4762
  Yes 1747.8217 341.2668

     student
Y            No       Yes
  No  0.7085963 0.2914037
  Yes 0.6186186 0.3813814
```

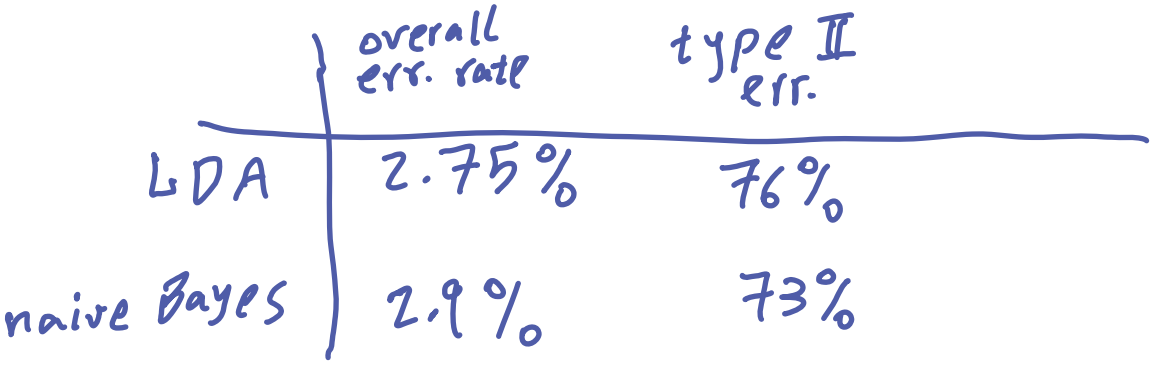
The output contains the estimated mean and standard deviation for each variable in each class.

```
> mean(balance[default == 'No'])
[1] 803.9438
> sd(balance[default == 'No'])
[1] 456.4762
```

The predict() function is straightforward.

```
> nb_class <- predict(nb_fit, Default)
> table(nb_class, default)
        default
nb_class   No  Yes
     No  9621  244
     Yes   46   89
> mean(nb_class == default)
[1] 0.971
```

We predict a default if the posterior probability of a default, that is, $P(Y = \text{default}|X = x)$, exceeds 0.5.



The predict() function can also generate estimates of the probability that each observation belongs to a particular class. Just as with LDA, we can easily adjust the probability threshold for predicting a default.

```
> nb_preds <- predict(nb_fit, Default, type = "raw")
> nb_preds[1:5, ]
            No          Yes
[1,] 0.9995250 4.750089e-04
[2,] 0.9985377 1.462322e-03
[3,] 0.9932451 6.754851e-03
[4,] 0.9999179 8.212027e-05
[5,] 0.9992442 7.558238e-04
> nb_class <- rep("No", 10000)
> nb_class[nb_preds[,2] >= 0.2]  = "Yes"
> table(nb_class, default)
        default
nb_class   No  Yes
     No  9339  130
     Yes  328  203
> mean(nb_class == default)
[1] 0.9542
```

	overall err. rate	type II err.
LDA	3.73%	41%
naire Bayes	4.58%	39%

K-Nearest Neighbors (KNN)

- Recall that KNN takes a completely different approach from the classifiers seen in this chapter.
- In order to make a prediction for an observation $X = x$, the training observations that are closest to x are identified. Then X is assigned to the class to which the plurality of these observations belong.
- Hence KNN is a completely non-parametric approach: no assumptions are made about the shape of the decision boundary.
- We make the following observations about KNN:
  - Because KNN is completely non-parametric, we can expect this approach to dominate LDA and logistic regression when the decision boundary is highly non-linear, provided that n is very large and p is small.
  - In order to provide accurate classification, KNN requires a lot of observations relative to the number of predictors, that is, n much larger than p.
  - In settings where the decision boundary is non-linear but n is only modest, or p is not very small, then QDA may be preferred to KNN.
  - Unlike logistic regression, KNN does not tell us which predictors are important.

Performing K-Nearest Neighbors in R

Now we will perform KNN on the Default data in order to predict whether or not an individual will default on the basis of credit card balance, income, and student status.

We perform KNN using the knn() function, which is part of the class library. Rather than a two-step approach in which we first fit the model and then we use the model to make predictions, knn() forms predictions using a single command. The function requires four inputs:

1. A matrix containing the predictors associated with the training data.
2. A matrix containing the predictors associated with the data for which we wish to make predictions.
3. A vector containing the class labels for the training observations.
4. A value for K, the number of nearest neighbors to be used by the classifier.

We use the cbind() function, short for column bind, to bind the balance, income, and student variables together into two matrices, one for the training set and the other for the test set.

Note: Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters.

A good way to handle this problem is to standardize the data so that all variables are given a mean of zero and a standard deviation of one. Then all variables will be on a comparable scale. The scale() function does just this.

```
> library(ISLR)
> attach(Default)
> names(Default)
[1] "default" "student" "balance" "income"
> library(class)
> train <- 1:9000
> train_X <- scale(cbind(balance, income, student)[train, ] )
> test_X <- scale(cbind(balance,income, student)[-train, ] )
> train_default <- default[train]
> test_default <- default[-train]
```

Now the knn() function can be used to predict the default variable for the test data.

```
> set.seed(1)
> knn_pred <- knn(train_X, test_X, train_default, k = 1)
> table(knn_pred, test_default)
        test_default
knn_pred  No Yes
     No  944  25
     Yes  20  11
> mean(knn_pred == test_default)
[1] 0.955
```

Below, we repeat the analysis using $K = 12$.

```
> set.seed(1)
> knn_pred <- knn(train_X, test_X, train_default, k = 12)
> table(knn_pred, test_default)
        test_default
knn_pred  No Yes
     No  961  26
     Yes   3  10
> mean(knn_pred == test_default)
[1] 0.971
```

The results have improved slightly. But increasing K further turns out to provide no further improvements.

An Analytical Comparison of Classification Methods

We now perform an analytical (or mathematical) comparison of LDA, QDA, naive Bayes, and logistic regression.

- We consider these approaches in a setting with K classes, so that we assign an observation to the class that maximizes $\Pr(Y = k|X = x)$.