Understanding Evalaution Metrics

Introduction: Building The Logistic Model

To show the use of evaluation metrics, I need a classification model. So, let's build one using logistic regression.

```r
# 0. Path to access the data
trainDataPath<- 'https://raw.githubusercontent.com/duttashi/datasets/master/breastcancer_training.csv'
testDataPath<-'https://raw.githubusercontent.com/duttashi/datasets/master/breastcancer_test.csv'

# 1. Import dataset
trainData <- read.csv(trainDataPath)
testData <- read.csv(testDataPath)

# 2. Build Logistic Model
logitmod <- glm(Class ~ Cl.thickness + Cell.size + Cell.shape, family = "binomial", data=trainData)

# 3. Predict on testData
pred <- predict(logitmod, newdata = testData, type = "response")

# 4. If p > .5, then Class is 1 else 0
y_pred <- ifelse(pred > 0.5, 1, 0)
y_act <- testData$Class

# 5. Accuracy
mean(y_pred == y_act)  # 94%
```

```
## [1] 0.9411765
```

The computed accuracy from the above model turned out to be 94%, which sounds pretty good. But, it doesn't reveal much information about how well the model actually did in predicting the 1's and 0's independently.

Nor does it say how well it would have performed with a different prediction probability cutoff.

Let's have a deeper look into this, starting with the confusion matrix.

Evaluation Metrics Simplified

1. The Confusion Matrix

The caret package provides the awesome confusionMatrix function for this. It takes in the predicted and actual values.

And to avoid confusion, always specify the positive argument.

Otherwise, it is possible for '0' to be taken as 'positive' or the 'event', and will cause a big mistake which may go unnoticed.

I want 'malignant' Class to be 'positive' so, I have set positive="1" below.

```r
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```r
caret::confusionMatrix(y_pred, y_act, positive="1", mode="everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 122   1
##          1  11  70
##
##                Accuracy : 0.9412
##                  95% CI : (0.8995, 0.9692)
##     No Information Rate : 0.652
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8745
##  Mcnemar's Test P-Value : 0.009375
##
##             Sensitivity : 0.9859
##             Specificity : 0.9173
##          Pos Pred Value : 0.8642
##          Neg Pred Value : 0.9919
##               Precision : 0.8642
##                  Recall : 0.9859
##                      F1 : 0.9211
##              Prevalence : 0.3480
##          Detection Rate : 0.3431
##    Detection Prevalence : 0.3971
##       Balanced Accuracy : 0.9516
##
##        'Positive' Class : 1
##
```

In the above output, the table in the first 4 lines of the output is the confusion matrix.

The remaining part of the output shows a bunch of more valuable evaluation metrics. Let's break down what they are.

1.1. How to interpret caret's confusionMatrix?

First, let's focus on the first 4 lines of the above output.

Confusion Matrix and Statistics

```
          Reference
Prediction   0   1
         0 122   1
         1  11  70
```

The rows in the confusion matrix are the count of predicted 0's and 1's (from y_pred), while, the columns are the actuals (from y_act).

So, you have 122 out of 133 benign instances predicted as benign and 70 out of 71 malignant instances predicted as malignant. This is good.

Secondly, look at the 1 in top-right of the table. This means the model predicted 1 instance as benign which was actually positive.

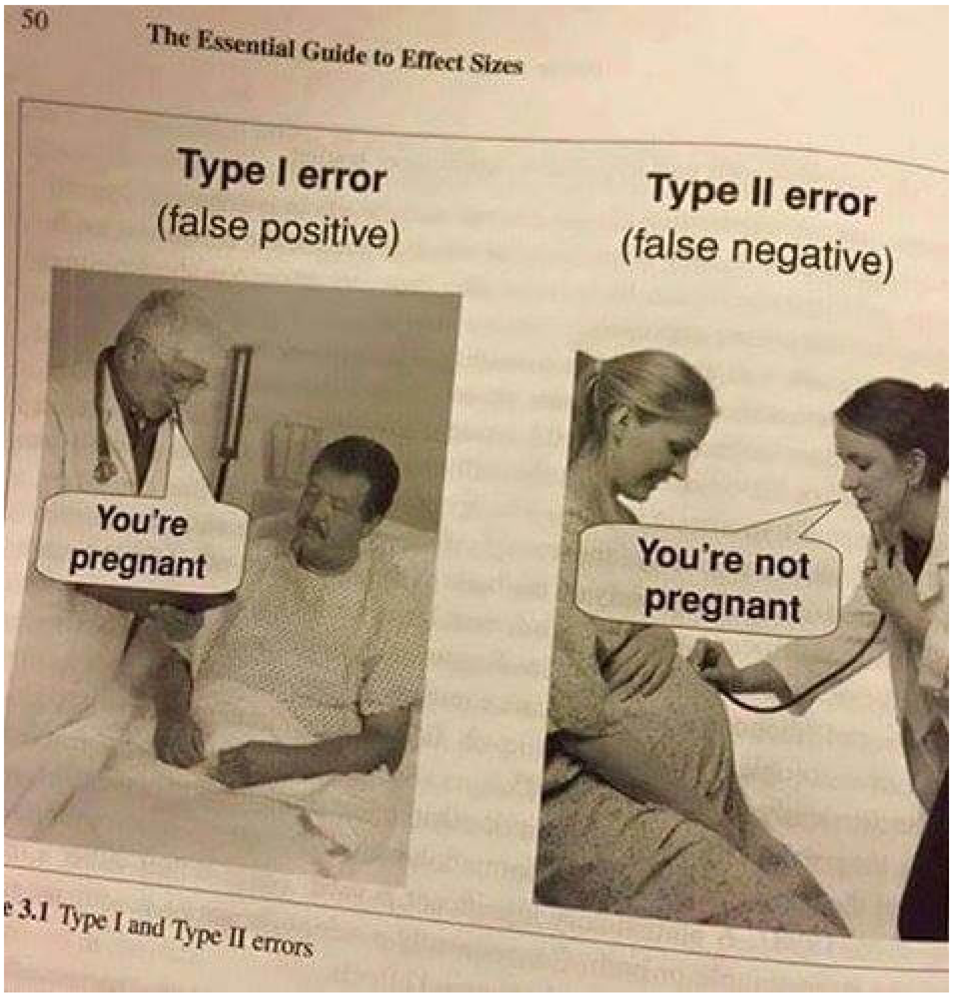


Figure 1: Figure: False Positive

This is a classic case of 'False Negative' or Type II error. You want to avoid this at all costs, because, it says the patient is healthy when he is actually carrying malignant cells.

Also, the model predicted 11 instances as 'Malignant' when the patient was actually 'Benign'. This is called 'False Positive' or Type I error. This condition should also be avoided but in this case is not as dangerous as Type II error.

1.2. What is Sensitivity, Specificity and Detection Rate?

Sensitivity is the percentage of actual 1's that were correctly predicted. It shows what percentage of 1's were covered by the model.

The total number of 1's is 71 out of which 70 was correctly predicted. So, sensitivity is 70/71 = 98.59%

Sensitivity matters more when classifying the 1's correctly is more important than classifying the 0's. Just like what we need here in `BreastCancer` case, where you don't want to miss out any malignant to be classified as 'benign'.

Likewise, Specificity is the proportion of actual 0's that were correctly predicted. So in this case, it is 122 / (122+11) = 91.73%.

Specificity matters more when classifying the 0's correctly is more important than classifying the 1's.

Maximizing specificity is more relevant in cases like spam detection, where you strictly don't want genuine messages (0's) to end up in spam (1's).

Detection rate is the proportion of the whole sample where the events were detected correctly. So, it is 70 / 204 = 34.31%.

You can see further explanation of all the metrics in this wiki link. https://en.wikipedia.org/wiki/Confusion_matrix

1.3. What is Precision, Recall and F1 Score?

Another great way to know the goodness of the model is using the Precision, Recall and the F1 Score.

The approach here is to find what percentage of the model's positive (1's) predictions are accurate. This is nothing but Precision.

Let's suppose you have a model with high precision, I also want to know what percentage of ALL 1's were covered. This can be captured using Sensitivity.

But in this context, it is known as Recall. Just because, it is customary to call them together as 'Precision and Recall'.

A high precision score gives more confidence to the model's capability to classify 1's. Combining this with Recall gives an idea of how many of the total 1's it was able to cover.

A good model should have a good precision as well as a high recall. So ideally, I want to have a measure that combines both these aspects in one single metric – the F1 Score.

F1 Score = (2 * Precision * Recall) / (Precision + Recall)

These three metrics can be computed using the `InformationValue` package. But you need to convert the factors to numeric for the functions to work as intended.

```
#install.packages("InformationValue")
#load the package
library(InformationValue)
```

```
##
## Attaching package: 'InformationValue'

## The following objects are masked from 'package:caret':
##
##     confusionMatrix, precision, sensitivity, specificity
```

Now the factors are converted to numeric. Let's compute the precision, recall and the F1 Score.

```
actual <- as.factor(as.character(y_act))
pred <- as.factor(as.character(y_pred))
recall_value<-recall(actual, pred)
recall_value
```

```
## [1] 0.9918699
```

```
#> 0.999918699
```

```
actual <- as.numeric(as.character(y_act))
pred <- as.numeric(as.character(y_pred))
precision_value<- precision(actual, pred)
precision_value
```

```
## [1] 0.8641975
```

```
#> 0.8641975
```

```
# F-measure is 2 * precision * recall / (precision + recall) is
Fmeasure <- 2 * precision_value * recall_value / (precision_value + recall_value)
Fmeasure
```

```
## [1] 0.9236427
```

We get an F1 score of 92 percent. That is pretty good.

1.4. What is Cohen's Kappa?

Kappa is similar to Accuracy score, but it takes into account the accuracy that would have happened anyway through random predictions.

Kappa = (Observed Accuracy - Expected Accuracy) / (1 - Expected Accuracy)

Cohen's kappa is shown as an output of caret's confusionMatrix function.

2. What is KS Statistic and How to interpret KS Chart?

The KS Statistic and the KS Chart (discussed next) are used to make decisions like: How many customers to target for a marketing campaign? or How many customers should we pay for to show ads etc.

So how to compute the Kolmogorov-Smirnov statistic?

Step 1: Once the prediction probability scores are obtained, the observations are sorted by decreasing order of probability scores. This way, you can expect the rows at the top to be classified as 1 while rows at the bottom to be 0's.

Step 2: All observations are then split into 10 equal sized buckets (bins).

Step 3: Then, KS statistic is the maximum difference between the cumulative percentage of responders or 1's (cumulative true positive rate) and cumulative percentage of non-responders or 0's (cumulative false positive rate).

The significance of KS statistic is, it helps to understand, what portion of the population should be targeted to get the highest response rate (1's).

The KS statistic can be computed using the `ks_stat` function in InformationValue package. By setting the `returnKSTable = T`, you can retrieve the table that contains the detailed decile level splits.

```
#library(InformationValue)
ks_stat(y_act, y_pred)
```

```
## [1] 0.8891
```

```
ks_stat(y_act, y_pred, returnKSTable = T)
```

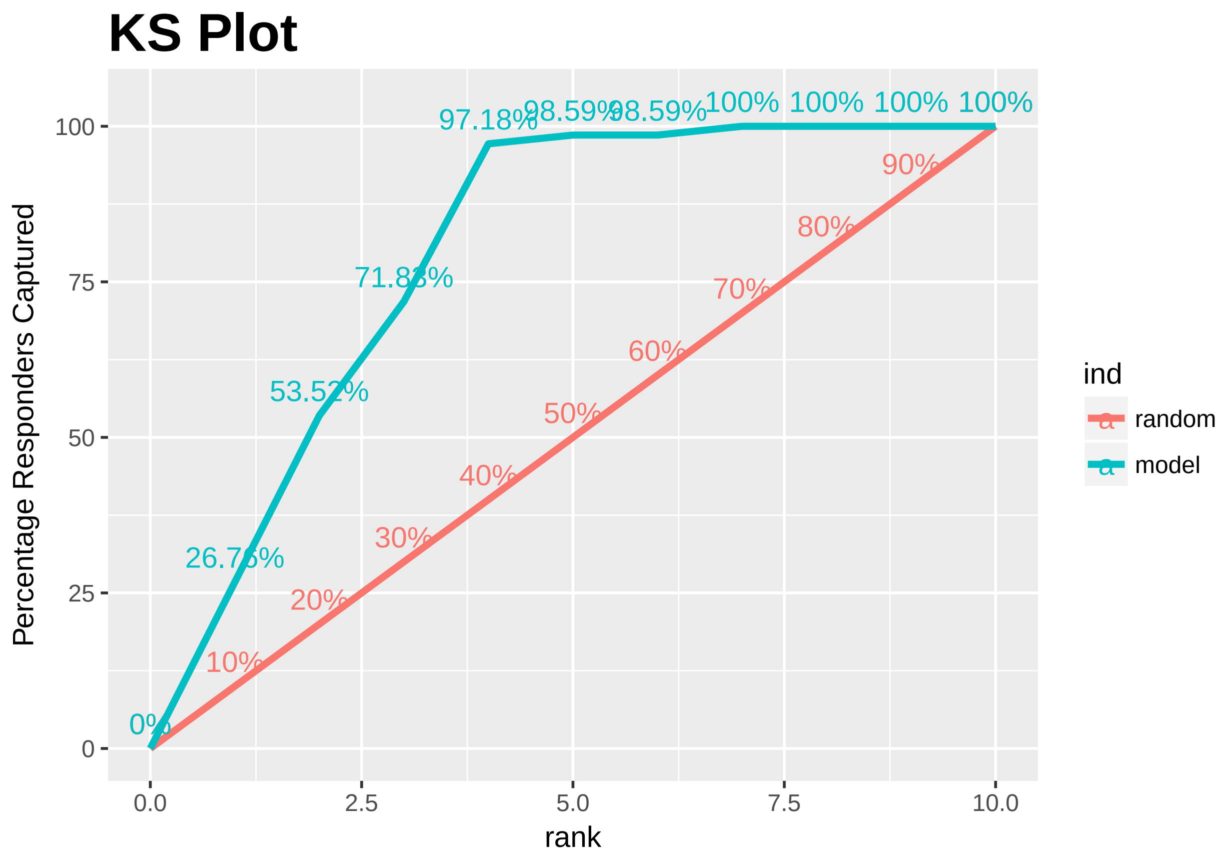
```
##    rank total_pop non_responders responders expected_responders_by_random
## 1     1        20              1         19                      6.960784
## 2     2        20              1         19                      6.960784
## 3     3        20              7         13                      6.960784
## 4     4        20              2         18                      6.960784
## 5     5        20             19          1                      6.960784
## 6     6        20             20          0                      6.960784
## 7     7        20             19          1                      6.960784
## 8     8        20             20          0                      6.960784
## 9     9        20             20          0                      6.960784
## 10   10        24             24          0                      8.352941
##    perc_responders perc_non_responders cum_perc_responders
## 1       0.26760563         0.007518797           0.2676056
## 2       0.26760563         0.007518797           0.5352113
## 3       0.18309859         0.052631579           0.7183099
## 4       0.25352113         0.015037594           0.9718310
## 5       0.01408451         0.142857143           0.9859155
## 6       0.00000000         0.150375940           0.9859155
## 7       0.01408451         0.142857143           1.0000000
## 8       0.00000000         0.150375940           1.0000000
## 9       0.00000000         0.150375940           1.0000000
## 10      0.00000000         0.180451128           1.0000000
##    cum_perc_non_responders difference
## 1              0.007518797  0.2600868
## 2              0.015037594  0.5201737
## 3              0.067669173  0.6506407
## 4              0.082706767  0.8891242
## 5              0.225563910  0.7603516
## 6              0.375939850  0.6099756
## 7              0.518796992  0.4812030
## 8              0.669172932  0.3308271
## 9              0.819548872  0.1804511
## 10             1.000000000  0.0000000
```

3. How to plot Kolmogorov Smirnov Chart in R?

The KS Chart is particularly useful in marketing campaigns and ads click predictions where you want to know the right population size to target to get the maximum response rate.

The KS chart below shows how this might look like. The length of the vertical dashed red line indicates the KS Statistic.

```
ks_plot(y_act, y_pred)
```



By targeting the top 40% of the population (point it touches the X-axis), the model is able to cover 97.18% of responders (1's).

The KS chart and statistic that is widely used in credit scoring scenarios and for selecting the optimal population size of target users for marketing campaigns.

4. How to Interpret ROC Curve?

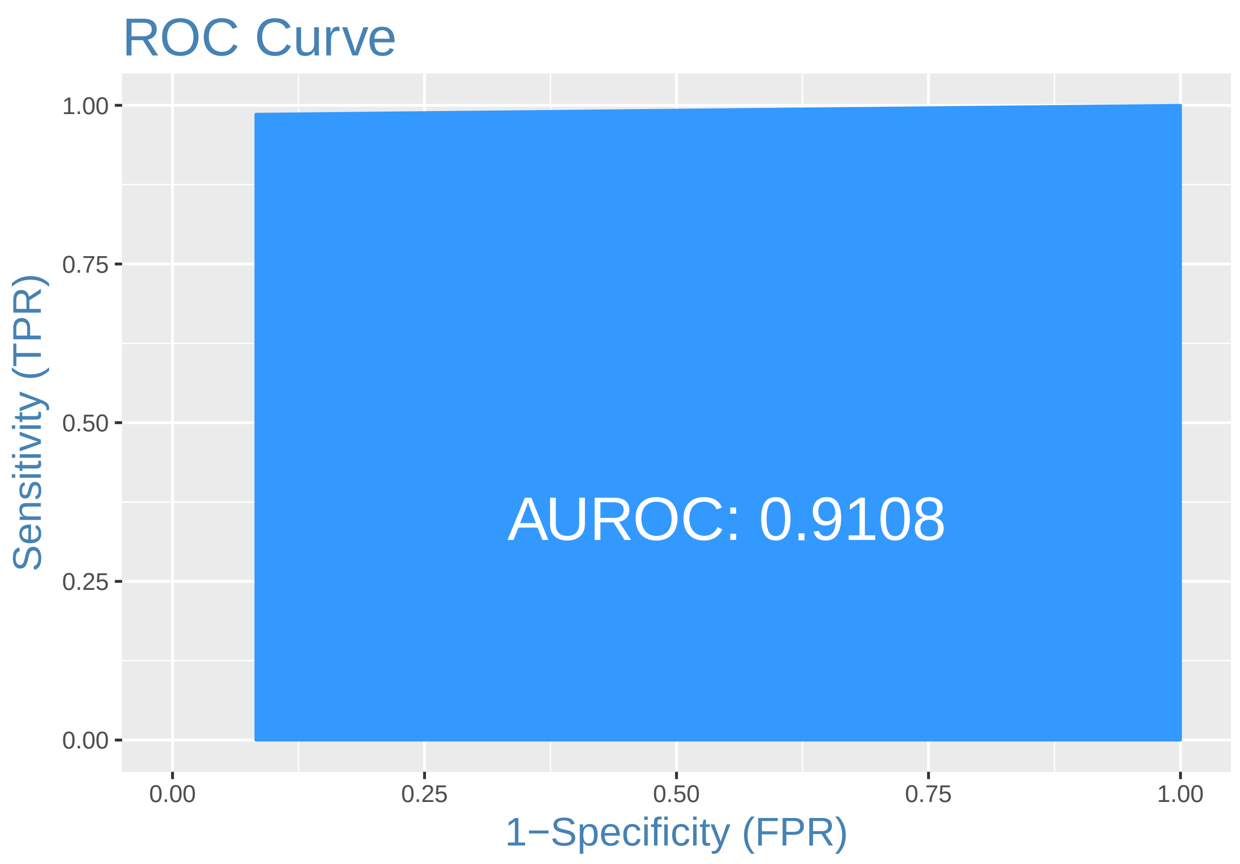
Often, choosing the best model is sort of a balance between predicting the one's accurately or the zeroes accurately. In other words sensitivity and specificity.

But it would be great to have something that captures both these aspects in one single metric.

This is nicely captured by the 'Receiver Operating Characteristics' curve, also called as the ROC curve. In fact, the area under the ROC curve can be used as an evaluation metric to compare the efficacy of the models.

Let's plot the curve and the area using the `plotROC` and `AUROC` functions from `InformationValue` package.

```
InformationValue::plotROC(y_act, pred)
```



```r
InformationValue::AUROC(y_act, pred)
```

```
## [1] 0.9108334
```

The actuals are contained in y_act and the predictions are contained in pred. You need to pass it in the same order to get the curve right.

The area under the ROC curve is also shown. But how to interpret this plot?

Interpreting the ROC plot is very different from a regular line plot. Because, though there is an X and a Y-axis, you don't read it as: for an X value of 0.25, the Y value is .9.

Instead, what we have here is a line that traces the probability cutoff from 1 at the bottom-left to 0 in the top right.

This is a way of analyzing how the sensitivity and specificity perform for the full range of probability cutoffs, that is from 0 to 1.

Ideally, if you have a perfect model, all the events will have a probability score of 1 and all non-events will have a score of 0. For such a model, the area under the ROC will be a perfect 1.

So, if we trace the curve from bottom left, the value of probability cutoff decreases from 1 towards 0. If you have a good model, more of the real events should be predicted as events, resulting in high sensitivity and low FPR. In that case, the curve will rise steeply covering a large area before reaching the top-right.

Therefore, the larger the area under the ROC curve, the better is your model.

The ROC curve is the only metric that measures how well the model does for different values of prediction probability cutoffs. The `optimalCutoff` function from `InformationValue` can be used to know what cutoff gives the best sensitivity, specificity or both.

5. What is Gini Coefficient?

Gini Coefficient is an indicator of how well the model outperforms random predictions. It can be computed from the area under the ROC curve using the following formula:

Gini Coefficient = (2 * AUROC) - 1

6. Concordant-Discordant Ratio

In an ideal model, the probability scores of all true 1's should be greater than the probability scores of ALL true 0's. Such a model is said to be perfectly concordant and this phenomenon can be measured by Concordance and Discordance.

So how to calculate Concordance?

Let's consider the following 4 observation's actual class and predicted probability scores.

Patient No True Class Probability Score P1 1 0.9 P2 0 0.42 P3 1 0.30 P4 1 0.80 From the above 4 observations, there are 3 possible pairs of 1's and 0's. That is, P1-P2, P3-P2 and P4-P2.

A pair is said to be concordant if the probability score of True 1 is greater than the probability score of True 0.

P1-P2 => 0.9 > 0.42 => Concordant! P3-P2 => 0.3 < 0.42 => Discordant! P4-P2 => 0.8 > 0.42 => Concordant!

Out of the 3 pairs, only 2 are concordant. So, the concordance is 2/3 = 0.66 and discordance is 1 - 0.66 = 0.33.

In simpler words, we take all possible combinations of true events and non-events. Concordance is the percentage of pairs, where true event's probability scores are greater than the scores of true non-events.

For a perfect model, this will be 100%. So, the higher the concordance, the better is the quality of the model. This can be computed using the `Concordance` function in InformationValue package.

```
InformationValue::Concordance(y_act, pred)
```

```
## $Concordance
## [1] 0.9043736
##
## $Discordance
## [1] 0.09562639
##
## $Tied
## [1] -5.551115e-17
##
## $Pairs
## [1] 9443
```

7. Root Mean Square Error (RMSE)

RMSE is the most popular evaluation metric used in regression problems. It follows an assumption that error are unbiased and follow a normal distribution. Here are the key points to consider on RMSE.

- The RMSE is the square root of the variance of the residuals.
- It indicates the absolute fit of the model to the data–how close the observed data points are to the model's predicted values.
- Whereas R-squared is a relative measure of fit, RMSE is an absolute measure of fit.

- Lower values of RMSE indicate better fit. RMSE is a good measure of how accurately the model predicts the response, and is the most important criterion for fit if the main purpose of the model is prediction.

The best measure of model fit depends on the researcher's objectives, and more than one are often useful.

```r
# Function that returns Root Mean Squared Error
rmse <- function(error)
{
    sqrt(mean(error^2))
}
# Example data
actual <- c(4, 6, 9, 10, 4, 6, 4, 7, 8, 7)
predicted <- c(5, 6, 8, 10, 4, 8, 4, 9, 8, 9)

# Calculate error
error <- actual - predicted

# Example of invocation of functions
rmse(error)
```

```
## [1] 1.183216
```

8. Cross Validation (CV)

8.1 Understanding CV

In Machine Learning, Cross-validation is a resampling method used for model evaluation to avoid testing a model on the same dataset on which it was trained. This is a common mistake, especially that a separate testing dataset is not always available. The concept of cross validation is actually simple: Instead of using the whole dataset to train and then test on same data, we could randomly divide our data into training and testing datasets.

There are several types of cross validation methods (LOOCV – Leave-one-out cross validation, the holdout method, k-fold cross validation). Here, I'm going to discuss the K-Fold cross validation method. K-Fold basically consists of the below steps:

- Randomly split the data into k subsets, also called folds.
- Fit the model on the training data (or k-1 folds).
- Use the remaining part of the data as test set to validate the model. (Usually, in this step the accuracy or test error of the model is measured).
- Repeat the procedure k times.

8.2 Cross Validation with R: the caret package

There are many R packages that provide functions for performing different flavors of CV. In my opinion, one of the best implementation of these ideas is available in the caret package by Max Kuhn (see Kuhn and Johnson 2013).

8.2.1 K-fold Cross Validation

The k-fold cross validation method involves splitting the dataset into k-subsets. For each subset is held out while the model is trained on all other subsets. This process is completed until accuracy is determine for each instance in the dataset, and an overall accuracy estimate is provided.

It is a robust method for estimating accuracy, and the size of k and tune the amount of bias in the estimate, with popular values set to 3, 5, 7 and 10.

The following example uses 10-fold cross validation to estimate Naive Bayes on the iris dataset.

```
# load the library
library(caret)
# load the iris dataset
data(iris)
# define training control
train_control <- trainControl(method="cv", number=10)
# train the model
model <- train(Species~., data=iris, trControl=train_control, method="nb")
```

```
## Loading required package: MASS
```

```
# summarize results
print(model)
```

```
## Naive Bayes
##
## 150 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.9466667  0.92
##    TRUE      0.9600000  0.94
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE
##  and adjust = 1.
```

8.2.2 Repeated k-fold Cross Validation

The process of splitting the data into k-folds can be repeated a number of times, this is called Repeated k-fold Cross Validation. The final model accuracy is taken as the mean from the number of repeats.

The following example uses 10-fold cross validation with 3 repeats to estimate Naive Bayes on the iris dataset.

```
# load the library
library(caret)
# load the iris dataset
data(iris)
# define training control
train_control <- trainControl(method="repeatedcv", number=10, repeats=3)
# train the model
model <- train(Species~., data=iris, trControl=train_control, method="nb")
```

```r
# summarize results
print(model)
```

```
## Naive Bayes
##
## 150 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.9555556  0.9333333
##    TRUE      0.9555556  0.9333333
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were fL = 0, usekernel = FALSE
##  and adjust = 1.
```

8.2.3 Leave One Out Cross Validation

In Leave One Out Cross Validation (LOOCV), a data instance is left out and a model constructed on all other data instances in the training set. This is repeated for all data instances.

The following example demonstrates LOOCV to estimate Naive Bayes on the iris dataset.

Leave One Out Cross Validation in R

```r
# load the library
library(caret)
# load the iris dataset
data(iris)
# define training control
train_control <- trainControl(method="LOOCV")
# train the model
model <- train(Species~., data=iris, trControl=train_control, method="nb")
# summarize results
print(model)
```

```
## Naive Bayes
##
## 150 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 149, 149, 149, 149, 149, 149, ...
## Resampling results across tuning parameters:
```

```
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.9533333  0.93
##    TRUE      0.9600000  0.94
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE
##  and adjust = 1.
```

8.2.4 Bootstrap

Bootstrap resampling involves taking random samples from the dataset (with re-selection) against which to evaluate the model. In aggregate, the results provide an indication of the variance of the models performance. Typically, large number of resampling iterations are performed (thousands or tends of thousands).

The following example uses a bootstrap with 10 resamples to prepare a Naive Bayes model.

```r
library(caret)
# load the iris dataset
data(iris)
# define training control
train_control <- trainControl(method="boot", number=100)
# train the model
model <- train(Species~., data=iris, trControl=train_control, method="nb")
# summarize results
print(model)
```

```
## Naive Bayes
##
## 150 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Bootstrapped (100 reps)
## Summary of sample sizes: 150, 150, 150, 150, 150, 150, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.9532509  0.9291499
##    TRUE      0.9533356  0.9292901
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE
##  and adjust = 1.
```

9. Conclusion

Clearly, just the accuracy score is not enough to judge the performance of the models. One or a combination of the following evaluation metrics is typically required to do the job.