classification

Ata COLAK

2023-07-28

Classification

Introduction

This report focuses on applying classification algorithms using R to analyze and predict the direction of cryptocurrency movements. The dataset used for this report is the "Sustainable Society Indices 2016 Dataset."

The report is divided in five sections, which explain how to: Import the necessary libraries and dataset. Create a train-test split for model training and testing. Use decision trees and analyse accuracy, precision, recall, and F-measure. Use logistic regression and analyse accuracy, precision, recall, and F-measure. Use the k-nearest neighbors algorithm and accuracy, precision, recall, and F-measure. The last section analyses the difference between each algorithm, and discusses on the results.

Each part includes code snippets and explanations for the corresponding steps.

The dataset used for this report is "btcusd_change_2022". The dataset can be accessed using the link below:

https://github.com/gagolews/teaching-data/blob/master/marek/btcusd_change_2022.csv

Part 1 - Importing the Libraries and Dataset

In the first example below, the csv file and libraries are loaded and read by my device, and heads of the dataset and the library are printed to show that they have been read successfully.

```
library(tree)
library(rpart)
library(class)
library(rpart.plot)
Smarket <- read.csv(paste0("https://raw.githubusercontent.com/gagolews/teaching-data/master/marek/btcus
comment.char="#")

head(Smarket)
```

```
##         Date    Close      Change Dir        Lag1        Lag2       Lag3
## 1 2022-01-07 41557.90 -1603.02734 dec  -408.07422 -2328.57031  -560.5430
## 2 2022-01-08 41733.94   176.03906 inc -1603.02734  -408.07422 -2328.5703
## 3 2022-01-09 41911.60   177.66016 inc   176.03906 -1603.02734  -408.0742
## 4 2022-01-10 41821.26   -90.33984 dec   177.66016   176.03906 -1603.0273
## 5 2022-01-11 42735.86   914.59375 inc   -90.33984   177.66016   176.0391
## 6 2022-01-12 43949.10  1213.24609 inc   914.59375   -90.33984   177.6602
##         Lag4       Lag5
## 1  -887.1016  -341.5938
## 2  -560.5430  -887.1016
## 3 -2328.5703  -560.5430
## 4  -408.0742 -2328.5703
## 5 -1603.0273  -408.0742
## 6   176.0391 -1603.0273
```

```
head(tree)
```

```
##
## 1 function (formula, data, weights, subset, na.action = na.pass,
## 2     control = tree.control(nobs, ...), method = "recursive.partition",
## 3     split = c("deviance", "gini"), model = FALSE, x = FALSE,
## 4     y = TRUE, wts = TRUE, ...)
## 5 {
## 6     if (is.data.frame(model)) {
```

```
head(rpart)
```

```
##
## 1 function (formula, data, weights, subset, na.action = na.rpart,
## 2     method, model = FALSE, x = FALSE, y = TRUE, parms, control,
## 3     cost, ...)
## 4 {
## 5     Call <- match.call()
## 6     if (is.data.frame(model)) {
```

```
head(class)
```

```
##
## 1 .Primitive("class")
```

Part 2 - Creation of Train-Test Split

Below is the way I created a train-test split, where I divide the dataset into two parts - %60 and %40. The %60 of data will be used to train our model to predict the Dir variable, and the remaining %40 of the data will be used to test our predictions. I used round() to take into considerations all values, otherwise one would not be in the dataset, therefore accurate percentage of data in testing dataset is %40.11142061 (144 (objects of Smarket_test) / 359 (total objects) * 100). I used rpart library to build the classification tree for this example, as rpart is one of the most powerful machine learning libraries for this report.

```r
# Creation of the train and test dataframe
train_i <- round(0.6 * nrow(Smarket))
Smarket_train <- Smarket[1:train_i,]
Smarket_test <- Smarket[(train_i+1):nrow(Smarket),]
tree_model <- rpart(Dir ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5, data = Smarket_train)
tree_predictions <- predict(tree_model, Smarket_test, type = "class")
```

Part 3 - Using Decision Trees and Analysing the Performance

I started the first section by defining what true/false positive/negative values will be for this model, where True Positive means the tree predicted a positive answer correctly, True Negative means the tree predicted a negative answer correctly, False Positive means the tree predicted a positive answer incorrectly, and False Negative means the tree predicted a negative answer incorrectly. The values would be summed up using sum() function and added to their matching list.

After all values are assigned to their matching lists, remaining of this report was only writing the formula correctly.

The formula for accuracy is true positive + true negative / (true positive + true negative + false positive + false negative). Therefore, I have calculated accuracy by first adding true values together, and dividing them by all values, as TP+TN+FP+FN includes all values.

The formula of precision is: True Positive / (True Positive + False Positive),

The formula of recall is: True Positive / (True Positive + False Negative), and lastly,

The formula for F measure is (2* Precision * Recall) / (Precision + Recall).

I have printed the results using cat().

```r
true_positive <- sum(tree_predictions == "inc" & Smarket_test$Dir == "inc")
true_negative <- sum(tree_predictions == "dec" & Smarket_test$Dir == "dec")
false_positive <- sum(tree_predictions == "inc" & Smarket_test$Dir == "dec")
false_negative <- sum(tree_predictions == "dec" & Smarket_test$Dir == "inc")


accuracy <- (true_positive + true_negative) / nrow(Smarket_test)
precision <- true_positive / (true_positive + false_positive)
recall <- true_positive / (true_positive + false_negative)
fmeasure <- (2 * precision * recall) / (precision + recall)


cat("Accuracy:", accuracy)
```

```
## Accuracy: 0.5763889
```

```r
cat("\nPrecision:", precision)
```

```
##
## Precision: 0.5434783
```

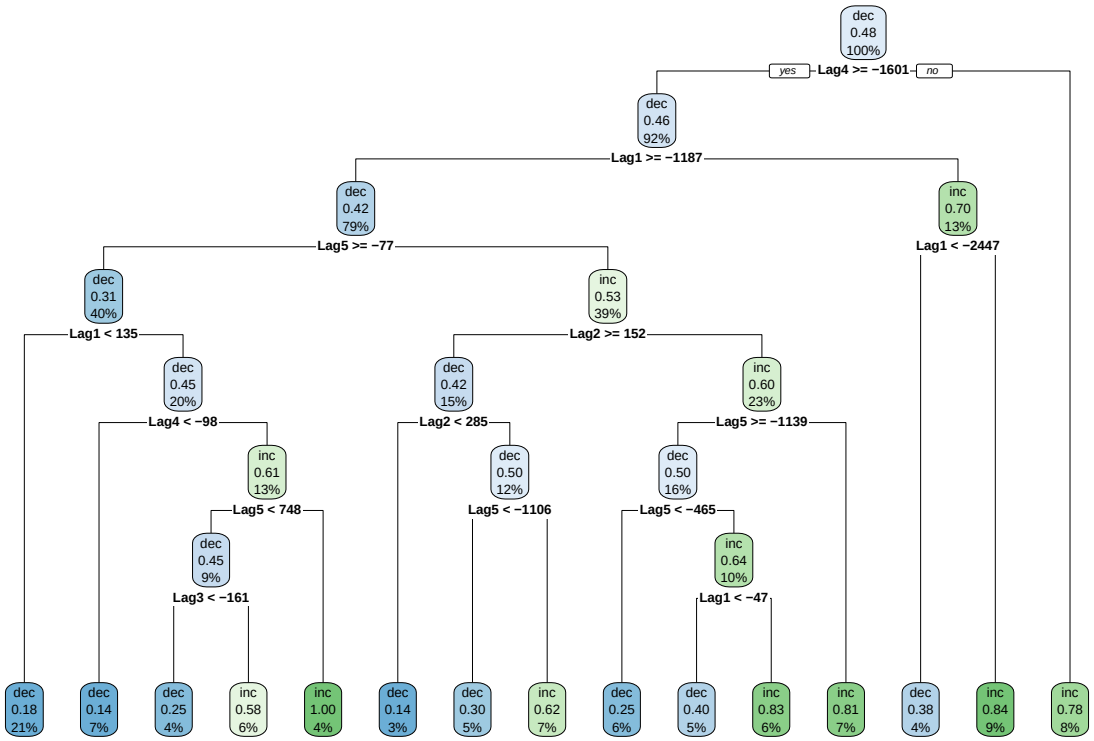
```
cat("\nRecall:", recall)
```

```
##
## Recall: 0.3846154
```

```
cat("\nF-measure:", fmeasure)
```

```
##
## F-measure: 0.4504505
```

```
rpart.plot(tree_model)
```



> After observing the decision tree, we can see that our model has shown the splitting rules which determine the resulting prediction. The plot helps us understand the structure and logic behind the decision tree model, and how it makes predictions depending on different features, as we can directly analyze the plot in order to see the way decision tree has been set.

Decision tree model has: Accuracy of approximately 58%, Precision of approximately 54%, Recall of approximately 38%, and F-measure results of 45%.

According to these results of predictions, it seems that decision tree model has low performance.

Part 4 - Using Logistic Regression and Analysing the Performance

Now I will use Logistic Regression to train the model. Overall structure of the model is very similar to Part 3, but I have added a new column to Smarket_train list with binary values, as I did not want to change the actual non-binary data for the next report.

After creating the model, I print out the coefficients.

I made predictions on test data using logical regression model, and afterwards I also added a log_class_predictions variable to convert probabilities to class predictions.

The rest of the code is the same as Part 3, where I calculate the results and print them.

```
Smarket_train$Dir_log <- ifelse(Smarket_train$Dir == "inc", 1, 0)
log_model <- glm(Dir_log ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5, data = Smarket_train, family = binomial)
cat("Coefficients:\n")
```

```
## Coefficients:
```

```
print(coef(log_model))
```

```
##   (Intercept)          Lag1          Lag2          Lag3          Lag4
## -1.029752e-01 -3.466060e-05 -5.554738e-05  2.187743e-06 -1.767008e-04
##          Lag5
## -7.401616e-05
```

```
log_predictions <- predict(log_model, Smarket_test, type = "response")
log_class_predictions <- ifelse(log_predictions > 0.5, "inc", "dec")

log_true_positive <- sum(log_class_predictions == "inc" & Smarket_test$Dir == "inc")
log_true_negative <- sum(log_class_predictions == "dec" & Smarket_test$Dir == "dec")
log_false_positive <- sum(log_class_predictions == "inc" & Smarket_test$Dir == "dec")
log_false_negative <- sum(log_class_predictions == "dec" & Smarket_test$Dir == "inc")

log_accuracy <- (log_true_positive + log_true_negative) / nrow(Smarket_test)
log_precision <- log_true_positive / (log_true_positive + log_false_positive)
log_recall <- log_true_positive / (log_true_positive + log_false_negative)
log_fmeasure <- (2 * log_precision * log_recall) / (log_precision + log_recall)

cat("\nAccuracy:", log_accuracy)
```

```
##
## Accuracy: 0.5763889
```

```
cat("\nPrecision:", log_precision)
```

```
##
## Precision: 0.6111111
```

```r
cat("\nRecall:", log_recall)
```

```
##
## Recall: 0.1692308
```

```r
cat("\nF-measure:", log_fmeasure)
```

```
##
## F-measure: 0.2650602
```

The logistic regression model has: Accuracy of approximately 58%, Precision of approximately 61%(+7% from decision tree), Recall of approximately 17% (-21% from decision tree), and F-measure results of 27%. (-18% from decision tree)

According to these results of predictions, it seems that logistic regression model has 7% higher precision compared to the decision tree model, but we can observe that recall has dropped down by 21%, and F-measure has also dropped down by 18%.

According to the results, we can say that although this model has higher precision, the recall and F-measure results have dropped drastically.

Part 5 - Using K-Nearest Neighbors Algorithm and Analysing the Performance

I start by fitting the k-nearest neighbors model into our dataset. KNN compared to other algorithms has a simpler and more user-friendly syntax, therefore making the code self explanatory.

Afterwards, I do the calculations for accuracy, precision, recall and F-measure the same way I did for both part 3 and 4, and I print the results.

```r
knn_model <- knn(
train = Smarket_train[, c("Lag1", "Lag2", "Lag3", "Lag4", "Lag5")],
test = Smarket_test[, c("Lag1", "Lag2", "Lag3", "Lag4", "Lag5")],
cl = Smarket_train$Dir,
k = 3)

# Calculate performance metrics for k-nearest neighbors model
knn_true_positive <- sum(knn_model == "inc" & Smarket_test$Dir == "inc")
knn_true_negative <- sum(knn_model == "dec" & Smarket_test$Dir == "dec")
knn_false_positive <- sum(knn_model == "inc" & Smarket_test$Dir == "dec")
knn_false_negative <- sum(knn_model == "dec" & Smarket_test$Dir == "inc")

knn_accuracy <- (knn_true_positive + knn_true_negative) / nrow(Smarket_test)
knn_precision <- knn_true_positive / (knn_true_positive + knn_false_positive)
knn_recall <- knn_true_positive / (knn_true_positive + knn_false_negative)
knn_fmeasure <- (2 * knn_precision * knn_recall) / (knn_precision + knn_recall)

cat("Accuracy:", knn_accuracy)
```

```
## Accuracy: 0.4652778
```

```
cat("\nPrecision:", knn_precision)
```

```
##
## Precision: 0.4090909
```

```
cat("\nRecall:", knn_recall)
```

```
##
## Recall: 0.4153846
```

```
cat("\nF-measure:", knn_fmeasure)
```

```
##
## F-measure: 0.4122137
```

The K-nearest-neighbour algorithm model has: Accuracy of approximately 46%, (-12% from logistic regression), Precision of approximately 40%(-21% from logistic regression), Recall of approximately 42% (+25% from logistic regression), and F-measure results of 42%. (+14% from logistic regression)

According to these results of predictions, it seems that logistic regression model lower performances in both accuracy and precision compared to the logistic regression model, but recall and f-measure results are higher.

According to the results, we can say that although this model has lower accuracy and precision, the recall and f-measure results are higher than the previous model.

Part 6 - Summarizing and Analysing the Results

To summarize the results and analyse the difference between the three algorithms, the dataframe below can be inspected.

```
results <- data.frame(
  Algorithm = c("K-Nearest Neighbors", "Logistic Regression", "Decision Trees"),
  Accuracy = c(knn_accuracy, log_accuracy, accuracy),
  Precision = c(knn_precision, log_precision, precision),
  Recall = c(knn_recall, log_recall, recall),
  F_measure = c(knn_fmeasure, log_fmeasure, fmeasure))

results
```

```
##             Algorithm  Accuracy Precision    Recall F_measure
## 1 K-Nearest Neighbors 0.4652778 0.4090909 0.4153846 0.4122137
## 2 Logistic Regression 0.5763889 0.6111111 0.1692308 0.2650602
## 3      Decision Trees 0.5763889 0.5434783 0.3846154 0.4504505
```

As it can be observed in the dataframe, both logistic regression and decision tree methods have an accuracy of approximately 58%, while KNN algorithm has an accuracy as low as %47. The precision of logistic regression is highest with 61%, compared to decision trees with %54, and

KNN with %41. The recall value however, is the highest in KNN with 42%, with decision trees having a recall percentage of %38, and logistic regression method having a recall percentage as low as 17%. While comparing the F-Measure results, we can see that decision trees had the highest percentage with 45%, and KNN had a close percentage of 41%, whilst logistic regression had a value as low as 27%.

In the context of this report, accuracy is relevant as a general measure of correctness in prediction of the tool. Prediction is also relevant when it comes to analysing the tool's ability to identify positive movements correctly. With a high precision, the tool would be reliable in recognizing false positives. With high recall, the tool would be able to capture higher numbers of positive price movements.

Overall, all results are important in the context of this report, but a program with high prediction would be the best option, as decreasing the risk of false positives would increase confidence within cryptocurrency traders significantly.

Conclusion

In conclusion, this report implemented and compared three different classification algorithms for cryptocurrency analysis: decision trees, logistic regression, and the k-nearest neighbors algorithm.

In the context of a cryptocurrency price analysis tool, the goal is to identify positive price movements accurately, therefore precision becomes an important metric to prioritize. High precision reduces the risk of false positives, which can be costly for traders. Therefore based on the results, the logistic regression algorithm would be the best option among the three algorithms.