Data Mining 2 Project

DMC 2016

Presented by

Axel Rantil, ID: 1494446

Mahendra Padi, ID: 1444424

Aleksandar Tomasovic, ID: 1495916

Sureshvarma Danthuluri, ID: 1491118

Chengqi Li, ID: 1490148

Alejandro Riera (Guest student)

Thomas Kara, ID: 1486608

Submitted to

Data and Web Science Group

Prof. Dr. Heiko Paulheim

Robert Meusel

University of Mannheim

May 22, 2016

Contents

Chapter 1

Introduction

1.1 Motivation

Returns have been a major cost driver for online distributors for many years. Approximately, 50 percent of the overall orders are sent back by the customers. This aspect has become even more important due to the new consumer policy introduced by the European Union in 2014. Therefore, a low return ratio becomes all the more a competitive advantage in online trading. The task of the Data Mining Cup 2016 is the prediction of the return rates of a fashion distributor which sells articles of particular sizes and colors. Therefor, historical data from January 2014 to September 2015 is provided to build a model which enables a good prediction of return rates. In some cases articles are returned to the fashion distributor for various reasons. Especially the influence of discounted items and vouchers on return rates should be considered in more detail.

1.2 Structure of dataset

The data set provided by this years Data Mining Cup consists of real anonymized fashion data given in structured text files. The training file 'orders-train.txt' contains about 2.33 million order positions (January 2014 to September 2015) whereas the test set 'orders-class.txt' has approx. 340 000 order positions (October 2015 to December 2015). Furthermore, ASCII is used as character set and missing values are present which are coded using the string 'NA'. The feature 'returnQuantity' represents the target attribute and indicates the number of returned items per order position. It is not present in the test data. See also table A.1 in appendix, which shows all 15 attributes at a glance.

Chapter 2

Preprocessing

2.1 Data discovery

After getting a good sense of the data, the team started discovering more detailed properties which we will present in this chapter. Firstly,we tried to deal with the missing values of the dataset. Several approaches for predicting the missing values was done but we later realized that all the examples with missing values also had a specific label. We then tried to find more patterns in data that would reveal the labels value. This group of examples, which we internally called give-aways, was:

- Quantity and Price = 0 ->returnQuantity = 0
- Missing values in productGroup and rrp ->returnQuantity = 0

These give-aways were filtered out of the dataset and predicted their corresponding label in the testset.
The next step was to try to extract additional information from the data. This was done through a brainstorm of potential interesting data. We quickly realized that a lot of information could be extracted by joining attributes such as customerId and orderId.

2.2 Attribute/Feature generation

In order to help prediction, we generated a number of new features. These features were about color, size, article, customer etc. What's more, since some customers appear in both train and test dataset, we generated *color_returned_times*, *color_bought_times*, *color_returned_ratio*, *size_returned_times*, *size_bought_times*, *size_returned_ratio*, *customer_sum_quantities*, *customer_sum_returns*, *customer_return_ratio*

and *customer_cluster* for these known customers.
Furthermore, we created a correlation matrix to see how relevant a feature is in contrast to *has_return*. See also appendix, table A.2 containing the new attributes for prediction and table A.3 for the correlation matrix[1].
In order to allow the team to work independently we generated an unique ID for every row in both the train and test data sets. That would allow every team member to export their generated features in a csv format including these ids', necessary to join all the features together.
Much of these feature creation and also join management, splitting in new VS known customers was done by a set of SQL queries[2].

2.3 Data Transformation

The features in our dataset have different data types: polynominal, binominal, real and integer. But in RapidMiner, some operators cannot handle polynominal or binominal attributes. In addition, we also used spark and R to classify the data which can just handle the numerical attributes. Thus we converted these types of features to numerical, e.g. *customerID paymentmethod*.
Based on our findings we decided to split the problem in between new customers and known customers, in order to be able to exploit some of the historical data available.
Our ruby and sql process was able to convert a given training and testing dataset into five different datasets:

- **train_known_customers:** a train dataset for the known customers split
- **test_known_customers:** subset of test dataset that contains only known customers. The customer-history related features included here are taken from the train known customers (don't use forbidden information!)
- **train_new_customers:** a train dataset for the known customers
- **test_new_customers:** subset of the given test dataset that contains only new customers
- **test_giveaways:** subset of given test dataset that contains anomalies and trivial examples, prediction is always 0 returnquantity.

[1] RapidMiner set-up: Return quantity true correlation = 0, false correlation = 1.

[2] PostgreSQL as backend, orchestrated by several ruby scripts that helped automating the whole process, code accessible here https://github.com/ariera/dmc2016.

One model is built learning from *train known customers* and used to predict *test known customers*. The same is true for new customers. And for the giveaways we always predict 0. In the end we append all the predictions together to calculate acurracy, confusion matrix, and absolute error.

2.4 Sampling

For training a classifier in an appropriate time, we created samples for training dataset. We followed two different strategies for sampling.
While exploring the data we discovered there was some indications that seasonality could be related to some of the features, specially the label. For that reason we decided to create a sample containing the same months of the test dataset but taken from the year before.
An second sampling was also implemented, in which the goal was to preserve the whole customer history. This would have the benefit that historical data of a given customer could be accurately extracted. It would also preserve order integrity, i.e. if an order appears in the sample it appears on its whole, which we also judged necessary if features such as *different_sizes* or *different_colors* needed to be implemented.
The evidence that our sampling was successful were given by applying Naive Bayes and decision tree on the sample and whole training set. The accuracy for both were approximately the same.

Chapter 3

Data Mining

3.1 Applying Models

3.1.1 RapidMiner

Since we divided our main train data set into known_customer, new_customer and giveaways datasets based on the customer behavior, we decided to build separate models for each datasets. For quick experimentations, samples of the different datasets were built.
We appended the predictions of these models in order to calculate the overall accuracy and absolute error. Further, we decided to do binomial classification, creating a new label has_return with possible values true and false.
For classification, we applied a variety of machine learning methods: K-NN, Decision Trees (standard, Random Forest), Neural nets, Nave Bayes, SVM, Logistic regression. with respect to the model we did the preprocessing like Generate new attribute (has_return), data transformation (Numerical to polynominal) and feature selection, etc. We trained all the models by tuning different parameters to get the better accuracy in order to find the best model.
At first we applied Decision trees because it can easily handle feature intersections and non-parametric. We do not have to worry about outliers or weather data is linearly separable. Major problem with Decision trees is they easily overfit. So, we tried for ensemble method like Random Forests from 10 to 150 trees, observing an increase in accuracy. As well random forest is fast and scalable. And we do not have to tune bunch of parameters like we do for SVM. And we tried logistic regression because it has a lot of advantages over SVM and Decision tree. No need to worry about features being correlated and have nice probabilistic interpretation (Unlike Decision tree or SVM).
We selected the best features by calculating its correlation to the label and applied

Nave Bayes. It gave us better results over other classifiers. And then we started a very long manual process of adding and removing one feature at a time to measure its impact. On every iteration we applied different ways of discretizing the new feature (when it applied) and kept the best. Whenever we got an increase in performance we would try again other classifiers to compare how the changes affected them. Naive Bayes kept yielding the best results, with some other sometimes pairing with it: NBTree, AdaBoost of RandomTree or RandomForest.
This feature selection process made us realize some overfitting problems we had. Some of the features related to the known_customers had a very high correlation with the label. For example color_return_ratio has a correlation with the label of 0.8 in the train dataset, but only 0.25 on the train dataset. More over, this feature tended to dominate all the trees that we trained, which were splitting their nodes based only in this color_return_ratio feature, drastically dropping our accurracies.
Manual work and exploration allowed us to detect this sort of problems. In some cases we managed to keep the feature the way it was, in some others we needed to apply discretization to sensible values, and in some others we just had to drop the feature completely.
From the results, we understood that better data often beats better algorithms, and designing good features goes a long way. Since our data is huge we have very slit changes in our accuracy. No matter which classifier we are using. Since we care about accuracy, we tried different algorithms.

3.1.2 Spark MLLib

When dealing with large datasets such as this one (over 3Mi examples in total), and then from those generate additional features (over 60 total attributes) high time and space complexity naturally follows for the models. Sampling of examples and feature reduction of the data ought to be done for testing and evaluating different models. However, when building the final model, as much data as possible ought to be used. That motivates the use of Apache Spark.
Spark is a framework that leverages multi-core power in-memory. By loading the data into Resilient Distributed Datasets (RDDs), computing can be done on clusters with a MapReduce workflow in a distributed fashion. Spark also provides the package Spark DataFrame, for opening, dealing and modifiyng data as well as Spark MLLib, the machine learning library for Spark.
After generating all the additional data in other processes, it was ready to be loaded into Spark. First, we had to load the data from CSV into a DataFrame (which consists of RDDs of Rows of data and a schema). The schema had to be specified manually which took a great deal of time. The details of the implementation process of Spark can be found in appendix, section B.1. Moreover, we have the

implementation stored in a Github repository[1].

3.1.3 R implementation

According to mentioned reasons in section 3.1.2 we tried spark and additionally the sci-kit learn library provided by python. In order to load the respective training and test set, additional libraries like panda or lumpy can be used. Unfortunately, it turned out that the data loading process requires too much of free memory space.
Another alternative has been R. In comparison to the sci-kit learn package it can easily be installed on Windows environments and has an open-source GUI, which is called rattle. With some manually implemented code lines, we were able to apply, on the whole training set, classification algorithms like Naive Bayes, K-NN and Decision Trees. See also the implementation of R in appendix, figure B.1 and B.2 It turned out that decision tree is so far the most accurate algorithm with 64,7% accuracy.
Furthermore, we assumed to increase the accuracy of the baseline learners with Random Forest for Regression, which takes the basic concept of decision trees but with a large number of trees. The algorithm creates for sets of features growing decision trees. Besides that process, it obtains importance values for the features and depicts the course of the error rate for the number of trees. See also the the appendix figure B.3 and B.4 of the feature importance values for known and new customers. Nevertheless, the decision tree algorithm yielded better results than Random Forest.
We expected that the attributes, used by decision tree are having the most significant importance for predicting the return quantity, all other attributes are negligible. Therefore, we decided to start over with Naive Bayes in Rapidminer to identify more promising features.

3.2 Evaluation

In order to evaluate the models we used cross validation and confusion matrices to check whether the trained model is predicting the test data well enough or not. We defined our baseline by creating a Naive Bayes model with 10 fold cross validation on the training set and measured against the test dataset achieving a 63.77% accuracy on binomial classification. For the baseline evaluation we used 10 fold cross validation with a sample dataset, to achieve an unbiased estimation of the model performance. Furthermore, by the use of confusion matrices for two classes (Positive and Negative) the absolute error was calculated. Even though the training error

[1] Spark repository link: https://github.com/ariera/dmc2016-hm-spark.

was low, the validation error was high. Therefore, we can argue that the model is overfitted.

3.3 Results

Tool	Learning method	Accuracy (known/new)	Abs. Err.	Dataset
RM	KNN (K= 3)	60.51% (57,49%/59.49%)	95638	whole
RM	KNN (K= 25)	65.60% (65.29%/63.10%)	83378	whole
RM	Naive Bayes	67.00% (66.67%/64.67%)	80001	whole
RM	W-AdaBoost(REPTree)	61.31% (64.21%/51.69%)	93696	whole
RM	AdaBoost Decision Tree	65.15% (65.02%/62.23%)	84453	whole
RM	Decision Tree	65.15% (65.02%/62.23%)	84453	whole
RM	Stacking(NB+DT: NB)	67.63% (67.16%/65.66%)	78483	whole
R	Decision Tree	63,7 %	87545	whole
R	AdaBoosting, stumps	63 %	89679	whole
R	Random Forest	65 %	84829	whole
Spark	Random Forest	61 %	-	known

Table 3.1: Results

Note that final accurracy is always higher than the aggregate of new and known accurracies. The reason is that we also have to count the giveaways, a dataset of around 7k examples that get an accuracy of 100%, not explicitly reflected on the table.
The best results are achieved using stacking of Naive Bayes and two different Decision Trees and using Naive Bayes again to make the final prediction.
The goal of using Spark was to leverage it's power to compute a model for big data-sets, the complete data set in this case. However, specifying the schema had to be done manually and took a great deal of time. After realizing that Spark was performing worse than other learners, that approach was later abandoned due to time-restraints. Worth mentioning is that the Spark-model tried to predict all 6 labels (returnQuantity=0,1,2,3,4,5).

3.4 Lessons learned

For most of the parts of the project we were applying a waterfall kind of process. First, we explored the data until we gathered all evidence that we could find. Based on our findings, we defined a set of all the features that could be created. Once we

had them all we started writing our first R and Spark scripts and also trained our first models in RapidMiner. As a result it was not until the end of the 4th week that for the first time we were getting some results, putting to test our hypothesis and facing all sorts of different problems related to the technologies we have chosen or with the features that we designed.

This process not only turned out to be slow but also helped hide and postpone problems that should have been addressed earlier.

In the last weeks of the project we switched to a more iterative process, with short loop cycles were, one at a time, we would establish our hypothesis, developed the features necessary, put them to the test, try with different models and applying different preprocessing steps when needed and based on the performance change produce we would then either keep them, drop them, or come up with more hypothesis to be tested.

This second iterative approach proved to be much faster, reliable and to yield better results as iterations were getting completed.

A good amount of time was invested on learning new technologies and tools such as R and Spark and its associated libraries on the second half of the project. We could have improved our scripts and tested many more models if we would have started working on them from day one.

Having a good representative sample was useful for it allowed to run test faster and still having similar measures to the original train set. However maintaining a sample meant that every time a feature was created it needed to be created for at least three different datasets. Since every team member was creating its own set of features there was a lot of manual work involved in all the process of creating, sharing and merging features together that costed a lot of time. The decision to write some script to automate the whole process, from feature creation to dataset splitting in new/known/giveaways, was a big time saver in the long run.

3.5 Future Work

While applying models, it turned out that the training set is overfitted. For instance, the training error was low and the validation error high. That means, we had a generalization issue. In the scope of this data mining project we were running out of time to increase the accuracy by optimizing the dataset. For instance, a significant decrease in complexity of the training set might yield, in contrast to our results, a higher accuracy, since the *unseen* test dataset would be predicted by generalized training data.

It was not until the end that we started applying a more iterative process where hypothesis were translated into features that were put to the test immediately and

explored. We are still using some of the original features for which there are many possibilities on how to be merged and transformed that we didn't have the time to try.
Some ideas were not developed: building a model to predict whether an order contains or not a return and adding this prediction as a new feature.
Some other ideas deserve more work before completely discarding them: we still believe that customer segmentation (being able to accurately separate final customers from retailers) can have a positive impact when done correctly. The same holds true for productgroup, sizecode and articles.
As reflected in the results stacking was yielding the best accurracies, but there is much work left to be done in this regard: more models should be added, perhaps more trees with different configuration parameters. KNN25 was not included due to time constraints although it was showing good results and some models are still missing such as SVMs or Rule Inductions.

Appendix A

Structure of dataset and feature generation

Attribute Name	Type
returnQuantity	Integer
orderID	Polynominal
orderDate	Date
articleID	Polynominal
colorCode	Polynominal
sizeCode	Polynominal
ProductGroup	Polynominal
quantity	Integer
rrp	Integer
price	Integer
voucherAmount	Integer
voucherID	Polynominal
customerID	Polynominal
deviceID	Polynominal
paymentMethod	Polynominal

Table A.1: Structure of dataset

Attribute Name	type	Description
id		Give each row a unique number
price_per_item	real	Represents the amount the customer paid for only 1 item of the given article without taking the voucher into consideration
price_to_rrp_ratio	real	Indicates if the article was sold above (> 1.0) or below (< 1.0) the recommended retail price
usual_price_ratio	real	Description of the relation between article single price and article average price.Bigger than 1 if this article in this order being sold above the article average price
color_ral_group	polynominal	The first number of color code
has_voucher	binominal	Represents whether the customer has a voucher
article_average_price	real	The average price of each item in the dataset
article_cheapest_price	real	The smallest single price of each item in dataset
article_most_expensive_price	real	The biggest single price of each item in dataset
article_number_of_different_prices	integer	With how many different single price has each article been sold
total_order_price	real	The total price of each order subtracts the value of voucher
different_sizes	integer	The number of different sizes in each order
sizes	polynominal	A list of the different sizes ordered for this article in this order
different_colors	integer	The number of different colors in each order
colors	polynominal	A list of the different colors ordered for this article in this order
color_returned_times	integer	How many times has this particular customer returned this particular color
color_bought_times	integer	How many times has this particular customer bought this particular color

color_returned_ratio	real	Percentage of how much this particular customer returns this particular color he has bought
size_returned_times	integer	How many times has this particular customer returned this particular size
size_bought_times	integer	How many times has this particular customer bought this particular size
size_returned_ratio	real	The percentage of how much this particular customer returned this particular size he has bought
customer_sum_quantities	integer	The total number of articles this particular customer has bought
customer_sum_returns	integer	The total number of articles this particular customer has returned
customer_return_ratio	real	The proportion of how much this particular customer returned articles he has bought
day_in_month	polynominal	Which day is the order date in this month
month_of_year	polynominal	Which month is the order in this year
day_of_week	polynominal	Which day is the order date in this week
quarter	polynominal	Which quarter is the order in this year
year_and_month	polynominal	The year and month of this order date
end_of_month	binominal	indicates whether the order was submited in the last 7 days of the month
is_productgroup_with_low_return_rate	binominal	indicates if the product group belongs to those that historically have been returned less often
is_productgroup_with_high_return_rate	binominal	indicates if the product group belongs to those that historically have been returned more often

product_size_cluster	polynominal	Grouping train and test datasets by productgroup and sizecode we calculated the aggregates values of count, average of rrp that we later use for K-Means to cluster in 7 different groups
gender	polynominal	if *product_kind* is *bra* then gender is *female* otherwise *unknown*
product_kind	polynominal	again clustering by productgroup and size, but this time using field knowledge to create 4 different categories: top, bottom, bra and unkown
subtotal_order_price	real	the sum of all the prices present in the order
n_articles_in_order	integer	how many different articles does this order contain
voucher_to_order_ratio	real	voucheramount divided by subtotal_order_price
n_times_article_appears_in_order	integer	how many times this particular article appears in the order
customer_cluster	polynominal	grouping train and test datasets by cusomerid and calculating the aggregated several aggregated values (average quantity of his orders, or average price of his orders) to cluster in 2 different types (final client and retail) of customers using K-Means
final_client_with_many_colors	binomial	will be true if the customer belongs to the final_client cluster and this order contains the same article more than with different colors. False otherwise
final_client_with_many_sizes	binomial	will be true if the customer belongs to the final_client cluster and this order contains the same article more than with different sizes. False otherwise

retail_with_many_sizes	binomial	will be true if the customer belongs to the retail cluster and this order contains the same article more than with different sizes. False otherwise
retail_with_many_colors	binomial	will be true if the customer belongs to the retail cluster and this order contains the same article more than with different colors. False otherwise
NewSizeCode	polynominal	3 different values. They depends on different segment.
customer_count_article	integer	How many different articles this customer bought
customer_count_color	integer	How many different colors of articles which this customer has bought
customer_count_device	integer	How many different device has this customer used to buy articles
customer_count_order	integer	How many different orders did this customer have
customer_count_orderdate	integer	How many different dates did this customer make orders
customer_count_paymentmethod	integer	How many different payment method did this customer have
customer_count_productgroup	integer	How many different product group of articles which this customer has bought
customer_count_sizecode	integer	How many different size code of articles which this customer has bought
customer_count_voucher	integer	How many different vouchers did this customer use
customer_sum_price	real	Total price of articles which this customer has bought
customer_sum_rrp	real	Total rrp of articles which this customer has bought
article_count_customer	integer	How many different customers bought this article
article_count_color	integer	How many different colors of this article
article_count_device	integer	How many different device used to buy this article
article_count_order	integer	How many different orders had this article

article_count_orderdate	integer	How many different dates did this article be bought
article_count_paymentmethod	integer	How many different payment method used to buy this article
article_count_sizecode	integer	How many different size code of this article
article_count_voucher	integer	How many different vouchers used to by this article
article_sum_quantity	integer	The total number of this article
article_sum_return	integer	The total return quantity of this article
article_sum_ rrp	real	The total rrp of this article
NewProductGroup	polynominal	3 different values. "High" means customers always keep the articles from this product group;"Medium" means in this product group,the number of articles which return quantity = 0 and return quantity = 1 are the almost the same; "Low" means customers always return the articles from this product group.

Table A.2: Generated features

Attribute	Correlation
paymentmethod	0.105470561
customer_cluster	0.090018736
is_productgroup_with_low_return_rate	0.080383734
product_kind	0.042566697
voucher_to_order_price_ratio	0.037990521
month	0.012316106
quarter	0.012228714
day_of_week	0.006597699
day	0.003100932
quantity	0.002902907
size_bought_times	0.000872224
voucherid	0.000775691
colors	0.000476307
orderid	0.000171615
productgroup	-0.000525649
end_of_month	-0.00170978
gender	-0.002580398
has_voucher	-0.004389022
different_colors	-0.004684831
deviceid	-0.005668462
article_number_of_different_prices	-0.0084446
customer_sum_quantities	-0.012770158
id	-0.013206544
orderdate	-0.013806721
voucheramount	-0.014230521
sizecode	-0.016040599
year	-0.018810426
customerid	-0.019274438
colorcode	-0.021363589
color_ral_group	-0.02159452
year_and_month	-0.022763972
price_to_rrp_ratio	-0.028457037

Table A.3: Correlation matrix, part I

Attribute	Correlation
color_bought_times	-0.028472563
articleid	-0.040545128
usual_price_ratio	-0.065758061
article_cheapest_price	-0.094955647
n_times_article_appears_in_order	-0.096376617
is_productgroup_with_high_return_rate	-0.099759142
sizes	-0.106285624
size_returned_times	-0.123022178
customer_sum_returns	-0.123737562
article_average_price	-0.128442513
rrp	-0.13415642
articlemost_expensive_price	-0.134752417
n_articles_in_order	-0.138697812
price	-0.14187157
price_per_item	-0.142902517
different_sizes	-0.17461434
total_order_price	-0.187261212
subtotal_order_price	-0.187341471
color_returned_times	-0.224447314
customer_return_ratio	-0.461646163
size_returned_ratio	-0.577450805
color_returned_ratio	-0.815288043
returnquantity	-0.995992554

Table A.4: Correlation matrix, part II

Appendix B

Implementation of Spark and R

B.1 Spark implementation

In order to run Random Forest in Spark, all inputs had to be numerical. So the next step was to index all the nominal values using StringIndexer before fitting a model.

```
val knownCustTestSchema = StructType(
  Array(
    StructField("colorcode" , StringType, true),
    StructField("deviceid" , StringType, true),
    ...
   )
  val knownTestLoad = sqlContext
  .read
  .format("com.databricks.spark.csv")
  .option("header", "true")
  .option("delimiter", ";")
  .schema(knownCustTestSchema)
  .load("path/to/file.csv")
)
```

In order to run Random Forest, all inputs had to be numerical. So the next step was to index all the nominal values using StringIndexer.

```
val voucheridIndexer = new StringIndexer()
.setInputCol("voucherid" )
.setOutputCol("voucheridIndexed");

val voucheridIndexed = voucheridIndexer.fit(knownTrain).transform(knownTrain)
```

This was done for all attributes (and the same indexer was used for both the training and the test to ensure they were classified based upon the same value). The next step was to create RDD's of Vectors. After this was done, a ChiSqSelector was used to generate a subset of

the features in the following manner.

```
  val selector = new ChiSqSelector()
 .setNumTopFeatures(10)
 .setFeaturesCol("features")
 .setLabelCol("returnquantity")
 .setOutputCol("selectedFeatures")

  val list = selector.fit(knownTr).selectedFeatures
```

The list defined here above was then used to select the same features for the test data.

Thereafter an RDD of LabeledPoint was createted as an input for RandomForest.

```
val labeledKnownTr = chiSQselected
.map(row => LabeledPoint(row.getDouble(1), row(2).asInstanceOf[Vector]))

val knownCategoricalFeaturesInfo = Map[Int, Int](
        7 -> 2
        8 -> 3
        # ..
)
    val numClasses = 2
    val numTrees = 500
    val featureSubsetStrategy = "sqrt"
    val impurity = "gini"
    val maxDepth = 5
    val maxBins = 100

val model = RandomForest.trainClassifier(
    labeledKnownTr,
    numClasses,
    knownCategoricalFeaturesInfo,
    numTrees,
    featureSubsetStrategy,
    impurity,
    maxDepth,
    maxBins
   )
```

The testdata was then generated in a similar fashion as the training data and applied to the model.

```
val labelAndPreds = labeledKnownTe.map { point =>
```

```
  val prediction = model.predict(point.features)
  (point.label, prediction)
}

val testErr = labelAndPreds
  .filter(r => r.\_1 != r.\_2)
  .count.toDouble / labeledKnownTe.count()

println("Test␣Error␣=␣" + testErr)
```

B.2 R implementation

```
# building the model on train...
library(rpart)
set.seed(crv$seed)
crs$rpart <- rpart(returnquantity ~ .,
    data=crs$dataset[, c(crs$input, crs$target)],
    method="class",
    parms=list(split="information"),
    control=rpart.control(usesurrogate=0,
        maxsurrogate=0))
```

Figure B.1: Train model - Decision Tree with R

```
crs$testset <- read.csv("C:/Users/thomas/OneDrive/hdminers/all_features_splited_v3/dm2_sample_

crs$pr <- predict(crs$rpart, newdata=crs$testset[,c(crs$input, crs$target)], type="class")
# Generate confusion matrix
table(crs$testset[,c(crs$input, crs$target)]$returnquantity, crs$pr,
      useNA="ifany",
      dnn=c("Ist", "Vorausgesagt"))

# Generate the confusion matrix showing proportions.
pcme <- function(actual, cl)
{
per <- pcme(crs$testset[,c(crs$input, crs$target)]$returnquantity, crs$pr)
round(per, 2)

# Calculate the overall error percentage.
cat(100*round(1-sum(diag(per), na.rm=TRUE), 2))

# Calculate the averaged class error percentage.
cat(100*round(mean(per[,"Error"], na.rm=TRUE), 2))
```

Figure B.2: Prediction of accuracy with R

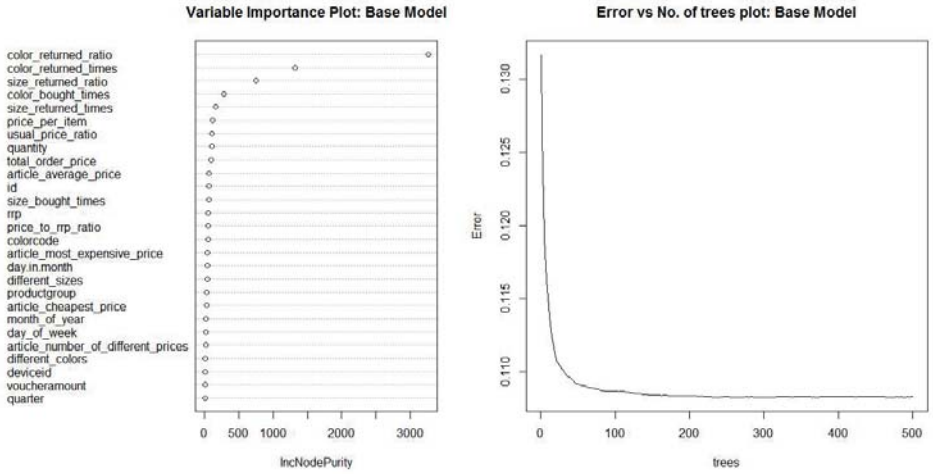

Figure B.3: Known customer feature importance

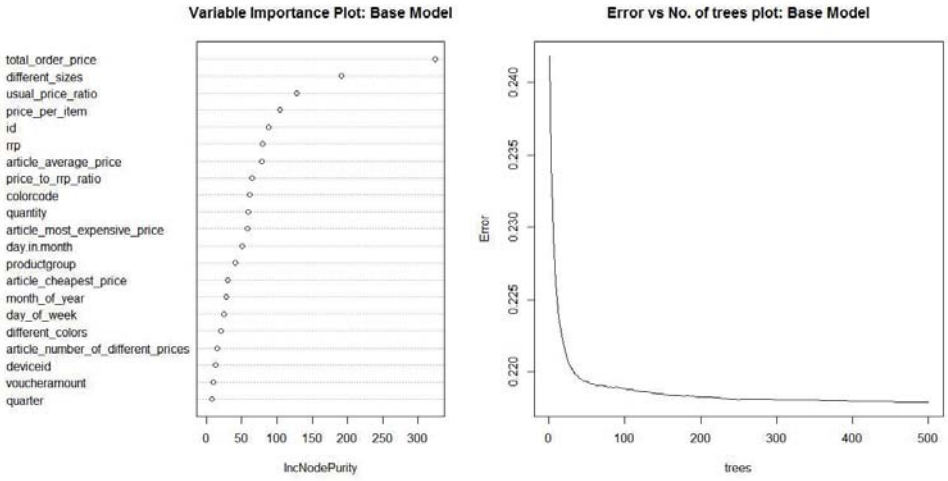

Figure B.4: New customer feature importance