Data Science project
Fall semester 2020/21

Comparative classification of online shoppers' purchasing intention

Project members: Guillaume Billes
Gaëlle Jakubowski
Christian Nestroy
Rodolfo Rubino

Contents

1. Project goal

Nowadays, online shopping is becoming more and more important and if we consider also the last months with the pandemic situation it will increase in the near future.

Taking in consideration this situation and the chosen dataset, our main goal was to find out how to predict the purchasing intention in online shopping sessions best. This means that we wanted to do a binary classification.

We also set the goal to gain insights into the relations of the variables, i.e. to perform inference. Taking in account our main research question and the main attributes we are using for the analysis, we counted on the shopping success to be dependent on special dates during the year.

Secondly, we wanted to find out if the shopping success and the region are correlated to each other in order to improve the marketing team research, so they will be able to target their efforts in a specific region or not.

2. Characteristics of the dataset

The dataset chosen named “online_shoppers_intention”, describes the users’ characteristics of their online shopping session: the goal being to understand which factors can impact a transaction. It contains 18 columns including 17 predictors and one response which is the variable “Revenue”. The dataset is quite large with 12,330 rows. Each row represents an online shopping session of a particular user. The sessions were selected such that each session belongs to a different user and that the sessions are uniformly distributed across a calendar year in order to avoid any tendency to a specific marketing campaign or user group. Below the figure 1 offers the description of the dataset.

Name	Data Type	Description
Revenue	logi	Indicating if the client purchased or not
Browser	int	Browser of the user
Month	chr	Month of the visit
OperatingSystems	int	OS of the user
Region	int	Where the user was connected during the session
TrafficType	int	Traffic source by which the visitor has arrived at the Web site (e.g., banner, SMS, direct)
VisitorType	chr	Visitor type as ‘‘New Visitor,’’ ‘‘Returning Visitor,’’ and ‘‘Other’’
Weekend	logi	Indicating whether the date of the visit is weekend

Administrative	int	Number of pages visited by the visitor about account management
Administrative_ Duration	num	Time (in seconds) spent by the visitor on account management related pages
BounceRates	num	Average bounce rate value of the pages visited
ExitRates	num	Average exit rate value of the pages visited by the visitor
Informational	int	Number of pages visited by the visitor about Web site, communication and address information of the shopping site
Informational_ Duration	num	Total amount of time (in seconds) spent by the visitor on informational pages
PageValues	num	Average page value visited
ProductRelated	int	Number of pages visited about product related pages
ProductRelated _Duration	num	Total amount of time (in seconds) spent by the visitor on product related pages
SpecialDay	num	Closeness of the site visiting time to a special day

Figure 1: Description of variables present in the dataset

The next step after taking a look at the dataset is exploratory data analysis in order to be familiar with data types and variables and so get a better understanding of their distributions and relations..

This first analysis allows us to detect missing or noisy values. However, the dataset chosen was already cleaned. We did not encounter any missing values or outliers.

The “scatter plot” allows us to visualize the correlation between features. In our dataset, we have a mix of quantitative and qualitative data. The scatter plot is only viable for quantitative ones. In our case, we have quite many quantitative variables, so it is a little hard to read. But we can see that ProductRelated and ProductRelated_Duration have a strong linear correlation, similarly BounceRates and ExitRates. These strongly correlated variables can be considered redundant and should therefore not appear together in the selected variable subset.

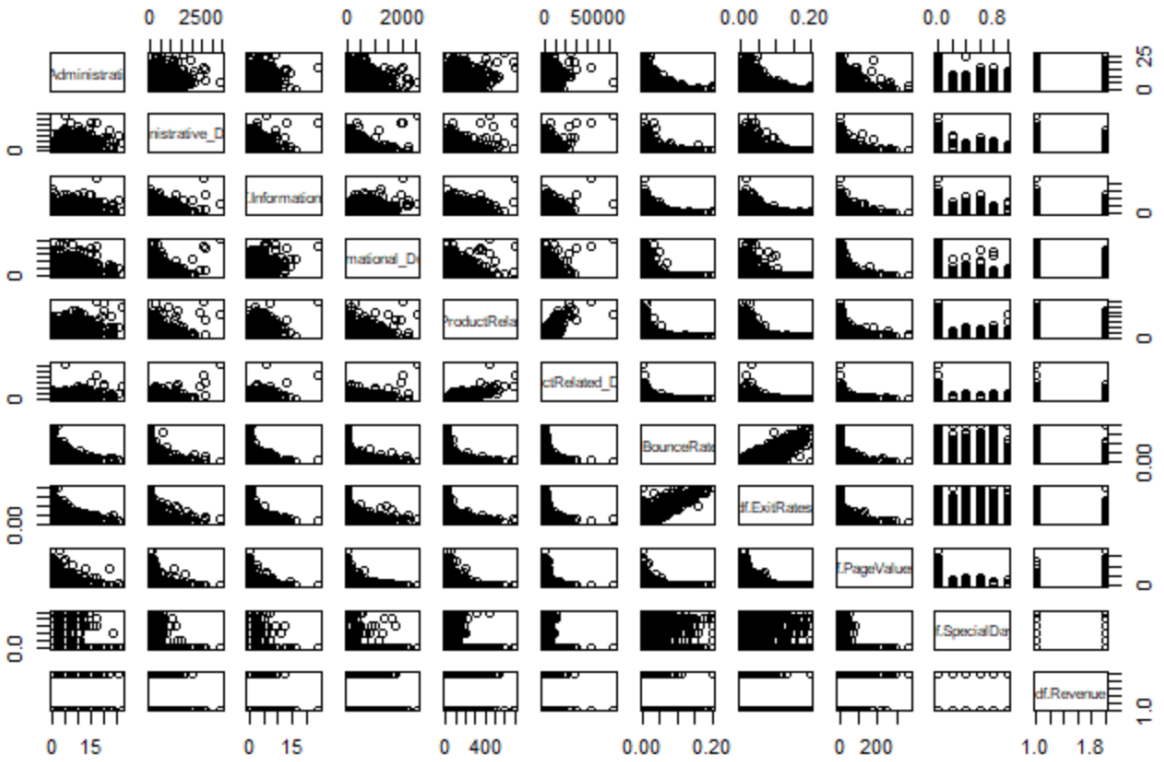

Figure 2: Scatter plot of the quantitative variables

3. Methodology

The global methodology we used in this project corresponds to the well known CRoss Industry Standard Process for Data Mining (cf. Wirth and Hipp 2000, p. 33) shown in Figure 3.

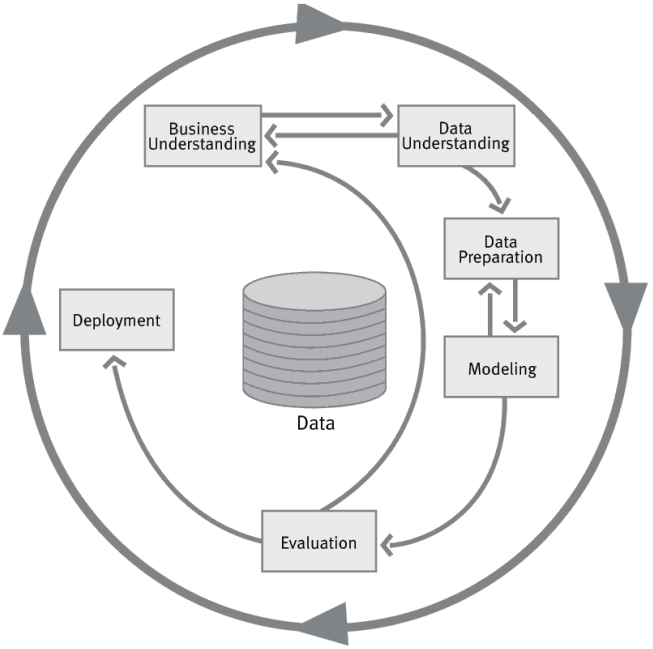

Figure 3: CRISP-DM framework serving as global methodology of our project

Phase Business Understanding comprised doing research about the meaning of the variables and setting our project goals according to the needs of the online shopping business.

Phase Data Understanding included plotting the distribution of single variables and relationship of multiple variables and determining the most relevant variables by feature selection.

Phase Data Preparation included performing one-hot encoding of the categorical variables, scaling the data to a standard normal distribution and splitting the data in training, validation and test set.

Phase Modeling included training different machine learning models and tuning their hyperparameters by grid search.

Phase Evaluation included estimating the performance of the models by applying them to the validation respectively test set.

Phase Deployment is out of the project scope. It could mean to train a productive model to predict online shopping success in real time during a session.

This procedure is depicted in more detail in figure 4.

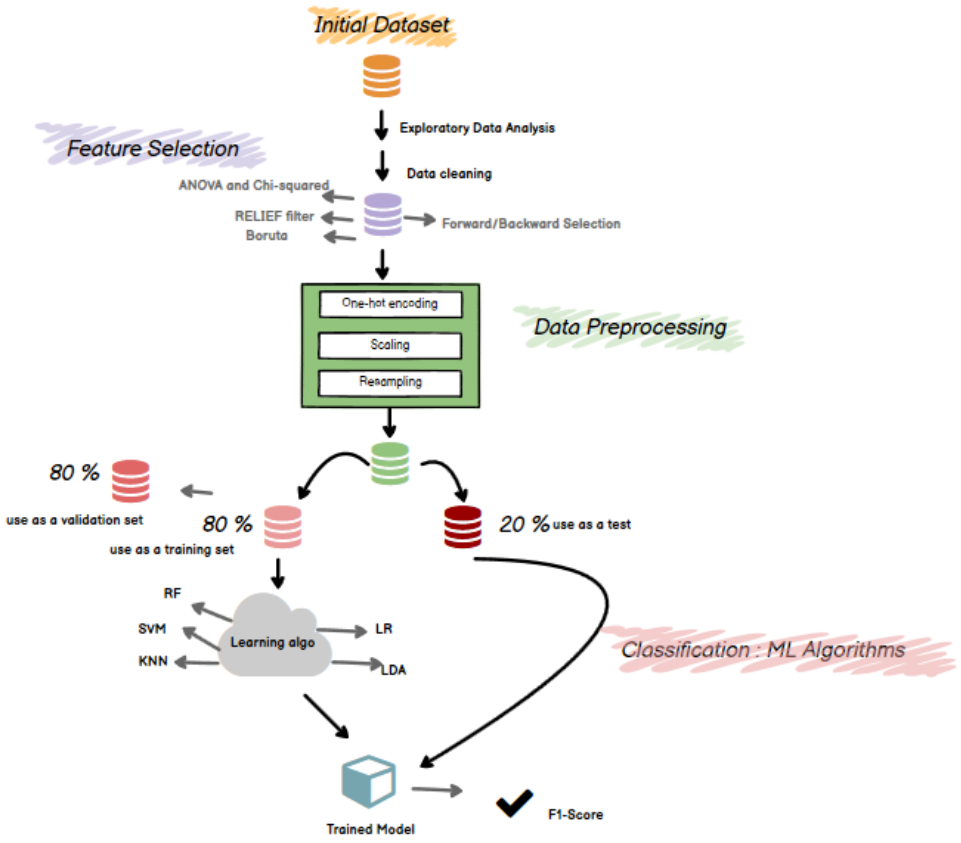

Figure 4: Detailed methodology of our project

4. Data preprocessing

a. One-hot encoding

In this step, in order to work with different models later, we will create dummy variables for each of our categorical variables (Month, VisitorType…).

To do it we use the dummyVars function from the caret package which will transform all characters and factors columns. Therefore we need to previously transform all our categorical columns into factors.

```
df$TrafficType = as.factor(df$TrafficType)
df$VisitorType = as.factor(df$VisitorType)
df$Browser = as.factor(df$Browser)
df$Region = as.factor(df$Region)
df$OperatingSystems = as.factor(df$OperatingSystems)
df$Month = as.factor(df$Month)
df$Weekend = as.factor(df$Weekend)
```

Figure 5: Converting data types

Then we just use `dmy = dummyVars(~ . , data = df)` which will create for us the dummy variables. To avoid any problem with Revenue (which is a categorical column for the function) we save it and then restore it to our dataframe.

b. Scaling

In this step the numerical variables are scaled consistently according to the standard normal distribution. This ensures that variables with a larger range of values are not automatically weighted stronger than variables with a smaller range of values. Figure 6 shows the histogram of variable PageValues before and figure 7 after the standard scaling.

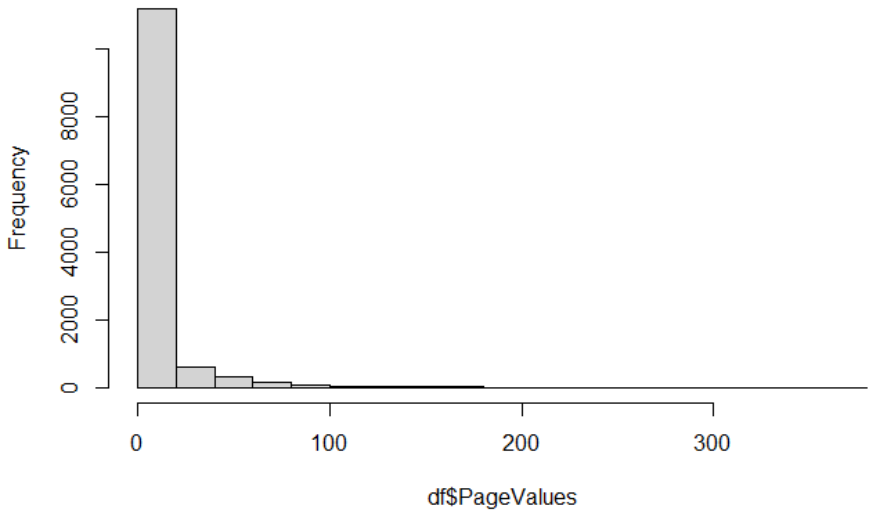

Figure 6: Histogram of PageValues before scaling

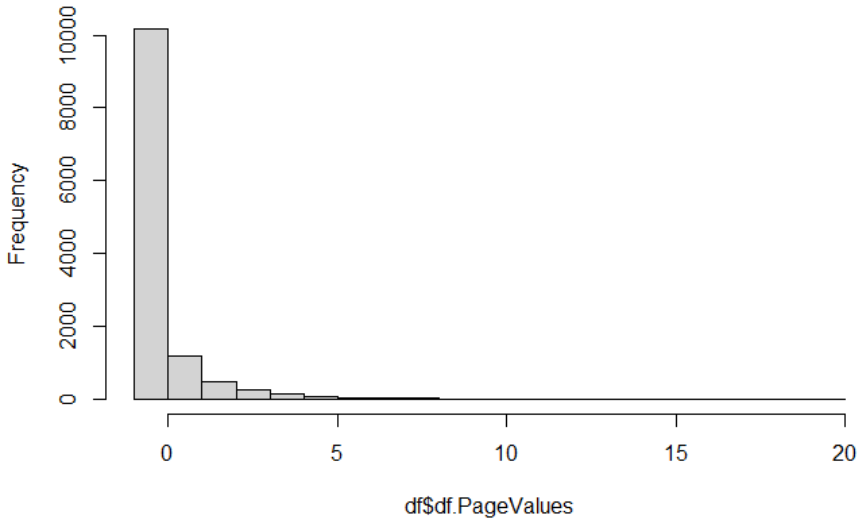

Figure 7: Histogram of PageValues after scaling

c. Resampling

As regards data splitting, it has been decided that in order to perform the training the data should be splitted with the validation set method:

- 64% as training set
- 16% as validation set
- 20% as test set

The validation set serves to estimate the performance of different hyperparameter configurations. Since it underestimates the true error on unseen data, we use the performance on an extra test set as a comparison criterion.

5. Feature selection

Feature selection is a process used in machine learning in order to support the learning task. Indeed, useless features pollute the learning algorithm. Variable selection is a process that allows you to "select" a subset of variables considered by the process to be relevant.There are three types of feature selection: Wrapper Methods, Filter Methods and Embedded methods.

a. Wrapper methods

In our dataset the number of rows is highly bigger than the number of predictors, then it is allowed to use methods using least squares (even if we are in a classification problem and least square is a standard approach in regression analysis) on the reduced set of variables like Best Subset Selection and Stepwise Selection (including Forward and Backward).We have a classification problem so it was not appropriate to use the methods seen in class linked with regression. Therefore, we used forward/backward stepwise variable selection for classification using LDA as specified classification method and selecting by estimated classification performance measure. The resulting performance measure for these models

are estimated (by cross-validation). For the function used only the quantitative variables are taken into account.

a) Forward stepwise selection

This strategy starts from an empty set. The variables are added one by one. At each iteration, the optimal variable according to the performance of a certain classifier (we use LDA) is added. The process stops either when there is no more variable to add, or when additional variables do not improve the classifier performance by a certain percentage. Once a variable has been added, the method cannot remove it.

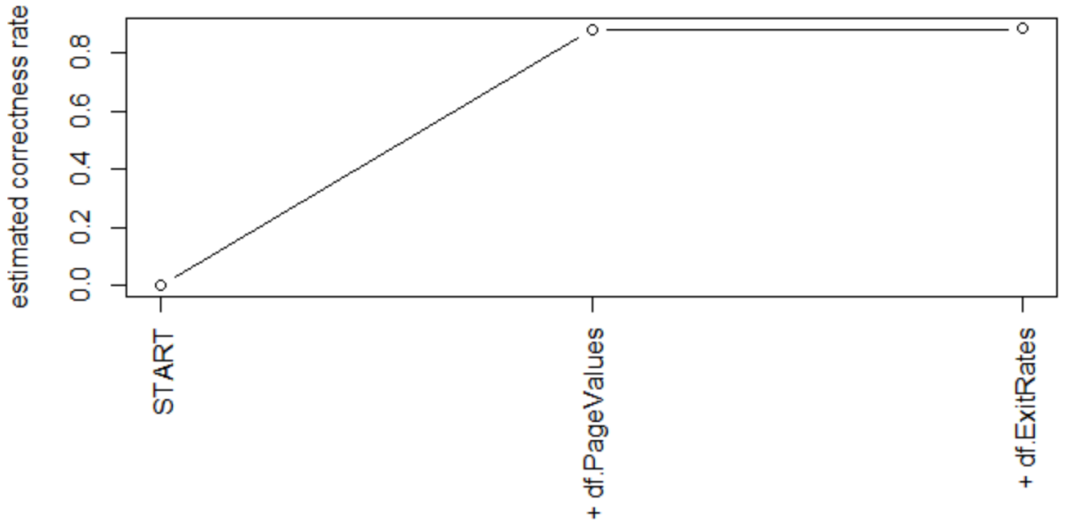

Figure 8: Result of forward stepwise selection

For this case the percentage of improvement has been changed in order to get more than one variable. Here, the “+“ means the variables added in the model step by step. Here only PageValues and ExitRates increase are selected as the best feature.

b) Backward stepwise selection

This strategy starts from the initial set of variables. At each iteration, one variable is removed from the set. This variable is such that its removal gives the best subset according to the performance criterion. Once the variable is removed, it is impossible to reintegrate it.

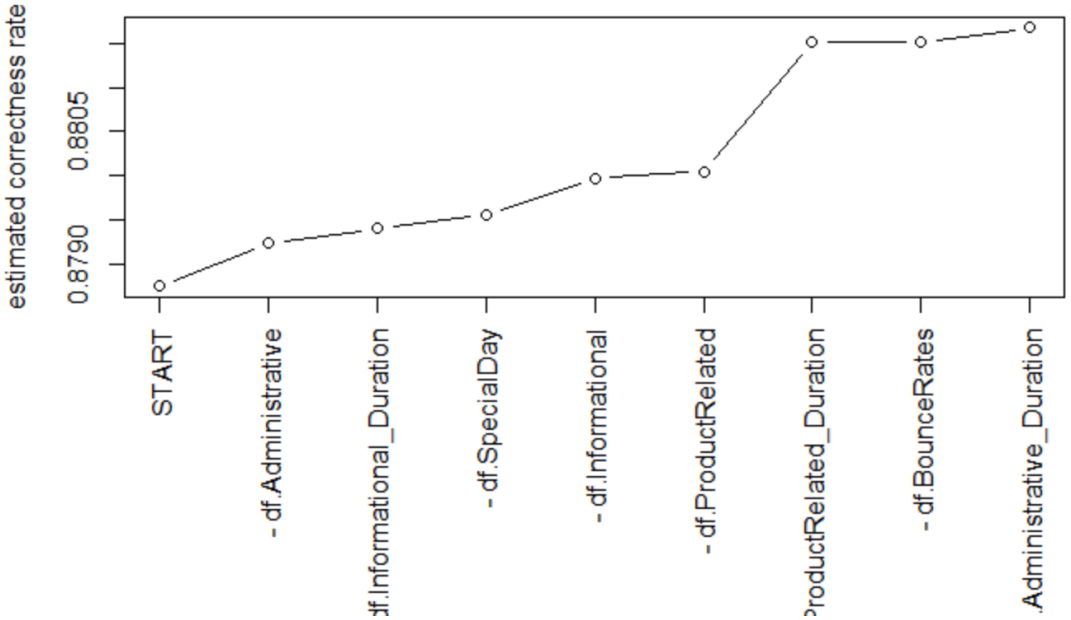

Figure 9: Result of backward stepwise selection

Here, the "-" means that the variables are removed from the model step by step. We get the same variables selected as in the previous method.

b. Boruta

Another way to easily detect the relation of predictors and the class variable is the algorithm Boruta. Boruta gives us a first idea of the feature selection. The goal of this algorithm is to assess the importance of the variables. Boruata creates copies of predictors and shuffles them. These new variables are called "shadow variables".

The approach trains a random forest and assesses feature importance by measuring the MDA (Mean Decrease Accuracy).

In each run, the Z score is also computed. It checks whether the feature has a higher Z score than the maximum Z score of its shadow features. If it is the case the algorithm marks it is an important feature. Step by step the algorithm rejects useless features. The algorithm stops either when all features get confirmed or rejected or it reaches a specified limit of random forest runs.

The output of this algorithm is this kind of graph with boxplots showing for each predictor the distribution of the data. The color gives us the information whether the predictor has been confirmed (green) or rejected (red) or a tentative (yellow), shadow predictor (blue).
In figure 10 only the most important features are represented.

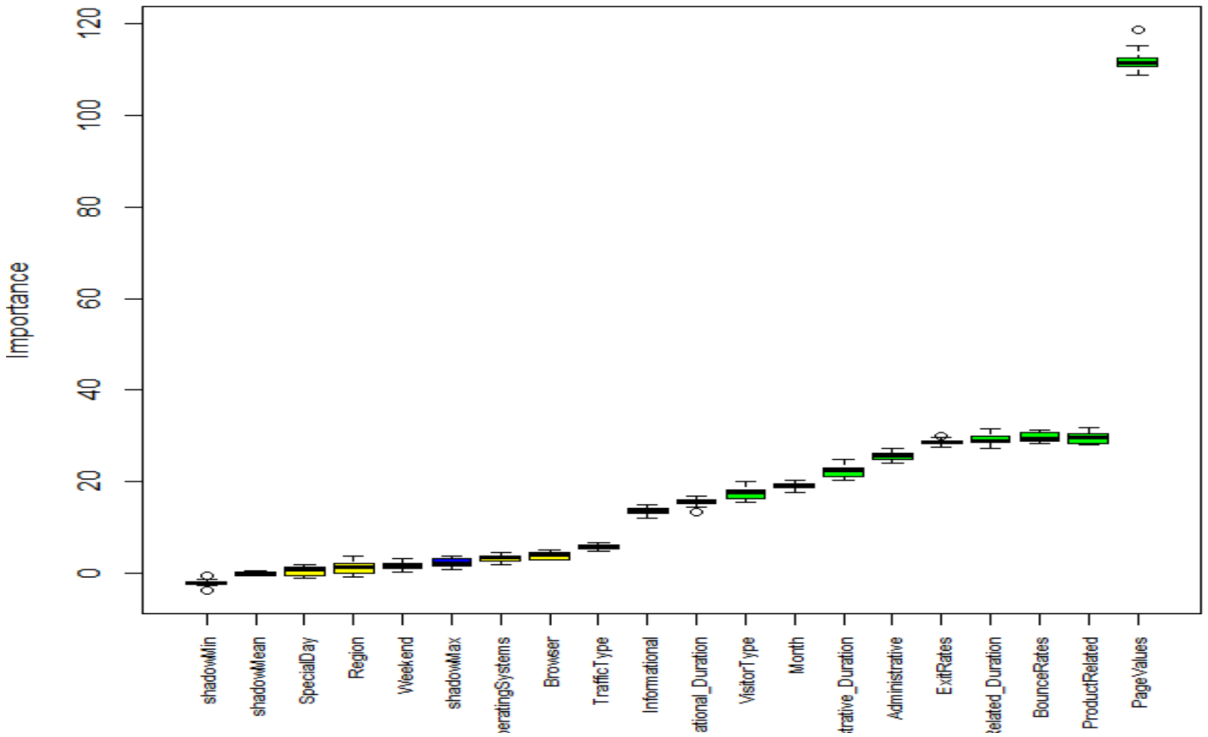

Figure 10: Boruta plot

Here we can deduce in green the features having a relationship with the target: PagesValues, ProductRelated, BouceRates, ProductRelatedDuration, ExitRates, Administrative, AdministrativeDuration, Month, VisitorType.

c. ANOVA and Chi-squared filter

In order to assess the importance of the variables, we performed the chi-square test for the influence of the categorical variables on the class variable and the so-called ANOVA ("analysis of variance", basically a modified F-test) for the influence of the numerical variables on the class variable.

The chi-squared filter is a statistical test which helps us to test the null hypothesis which is "there is no relation between two variables". This can so help us to determine if two variables are correlated or not. A chi-square test for independence compares two variables in a contingency table to see if they are related. In a more general sense, it tests to see whether distributions of categorical variables differ from each other. A very small chi square test statistic means that your observed data fits your expected data extremely well meaning that there is a relationship, when on the other side a large chi square test statistic means that the data does not fit very well. In other words, there is not a relationship.

In the following table the results of these analyses are shown:

Categorical variable	**chi-square value**
Browser	27.72

Month	384.94
OperatingSystems	75.03
Region	9.25
TrafficType	373.14
VisitorType	135.25
Weekend	10.39

Numerical variable	**F value in ANOVA test**
Administrative	242.59
Administrative_Duration	108.93
BounceRates	286.38
ExitRates	552.29
Informational	112.75
Informational_Duration	61.31
PageValues	3949.26
ProductRelated	317.85
ProductRelated_Duration	293.03
SpecialDay	84.08

At both tests higher values indicate a stronger relationship between the predictor and the class variable. Needless to say, that chi-square values may not be compared with F values. The analysis reveals that Month and TrafficType show strong relations with the class variables and PageValues seems to be the most important numerical variable by far. This finding is in line with the paper of Sakar et al. (2019) who also published this dataset. Furthermore, this can be confirmed with a box-whisker-plot of PageValues showing salient differences in the quantiles of the distribution, grouped by sessions with and without shopping success.

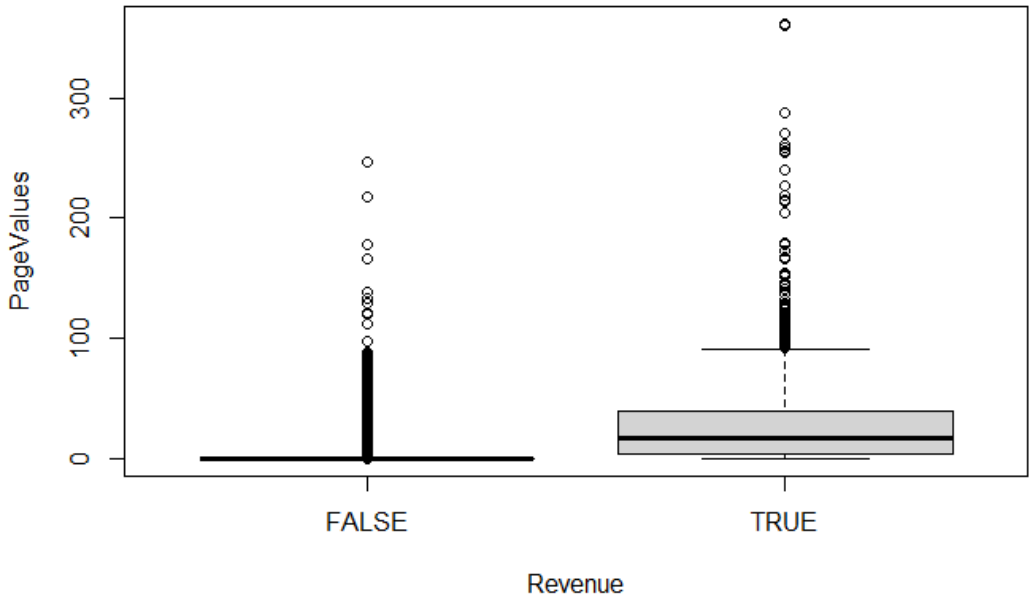

Figure 11: Grouped box-whisker plots of PageValues

In our project proposal we expected that SpecialDay and VisitorType would be the most important predictors. We clearly see that this assumption does not hold. We also wanted to provide guidance on whether advertising should be started in specific regions. However, the chi-square test shows that the influence of Region on the class variable is negligible. The following diagram shows that the distribution of Region is close to identical with both values of the class variable.

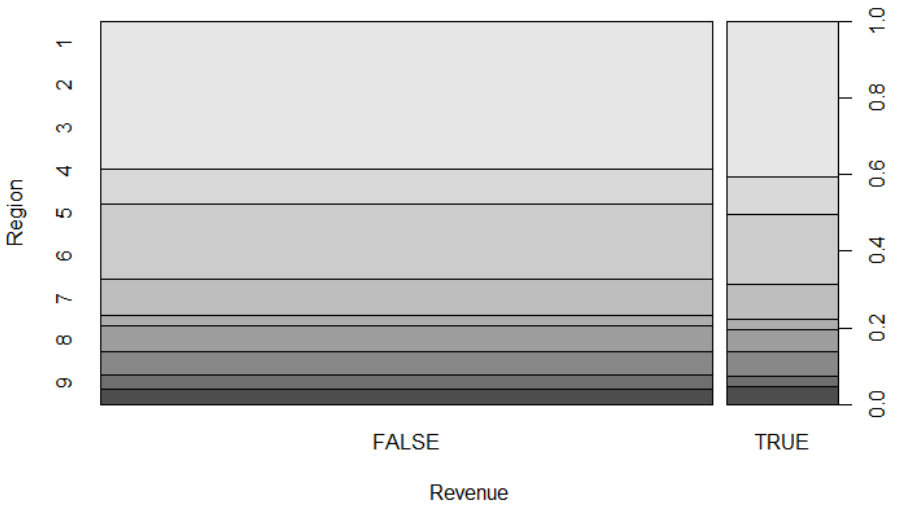

Figure 12: Grouped distribution plot of Region

d. Relief filter

Additionally to the ANOVA respectively Chi-squared filter feature selections, we applied a variant of an established algorithm called RReliefF. This algorithm is readily utilizable in R using the FSelector package. It is able to handle variables of every type: qualitative and quantitative. The basic idea is to search for each observation the nearest (based on Euclidean distance) observation from the same class (near-hit) and from the other class (near-miss). The notion is that a relevant feature leads to a small distance between the observation and its near-hit and a large distance to its near-miss. A thorough explanation of this algorithm is out of the scope of this report. The reader is referred to the papers of Kira, Rendell 1992 and Robnik-Sikonja, Kononenko 1997. The results of RReliefF for our dataset were rather disappointing because they did not match the results of the other feature selection techniques at all.

6. Classification methods

a. Random forest

Basis for the random forest is the understanding of decision trees. The respective decision tree algorithm (ID3, C4.5, CART, etc.) constructs a kind of flowchart. Each inner node symbolizes a test with respect to the feature that can optimally distinguish the objects. A possible criterion to determine which feature and threshold value to choose for the split is the gain ratio which quantifies how much a feature reduces the uncertainty in the classification. In CART each split is required to be binary in order to preserve a good comprehensibility of the tree. The following branches represent the possible outcomes of this decision. The outer nodes (leaves) show the resulting class division.

In C4.5 and CART (cf. Quinlan 1986) the decision tree is pruned in a postprocessing step. In the feature selection it is assumed that all features not appearing in the tree are irrelevant and the features used should form the subset (cf. Han et al. 2012, p. 105). Each path to an outer node corresponds to a feature set. The union of all nodes represents the selected feature subset (cf. Dash and Liu 1997, p. 141).

A single decision tree is prone to overfitting the training set even though it is pruned. That is why we make use of a so-called bagging approach. Here, bootstrapped samples are drawn from the training set and multiple decision trees are trained with these samples. When applied to the test set, each decision tree in the forest performs a classification for each observation. These predictions are aggregated by a majority vote of all the trees. By doing so, this technique reduces the variance in the classifier. The following code snippet x shows the training of the random forest.

```r
#Random Forest

f1_rf_val = hash()
ntree_grid = c(20,30,50,80,120,160,200,500,600)
for (number_of_trees in ntree_grid){
  rf = randomForest(y ~ . , data = df_train_reduced, ntree = number_of_trees, importance = TRUE)
  rf_pred = predict(rf, df_val[,1:3])
  f1_rf_val[number_of_trees] = F1_Score(df_val[,4], rf_pred)
}
f1_rf_val = as.list(f1_rf_val)
best_ntree = as.numeric(names(which.max(f1_rf_val)))
rf = randomForest(y ~ . , data = df_train_reduced, ntree = best_ntree, importance = TRUE)
rf_pred = predict(rf, df_test_reduced[,1:3])
f1_rf = F1_Score(y_true, rf_pred)
```

Figure 13: Training of random forest

We assumed that the number of trees that are trained for the random forest constitute the most important hyperparameter. Thus, we tuned this hyperparameter using a grid search with our validation dataset. 500 decision trees was found to be a good choice showing that the bagging approach is necessary to improve generalization.

b. Support vector machine

A slightly more complicated machine learning method is the support vector machine. It separates classes by a hyperplane like shown in figure 14. Normally, there is an infinite number of hyperplanes that correctly classify the training observations. The SVM selects the hyperplane that has the largest margin M ≥ 0 according to the Euclidean metric to the nearest observations of all classes. It is assumed that this model generalizes best, since there is the most space available to correctly classify unknown observations. The position vectors to the observations that are closest to the hyperplane are called support vectors.

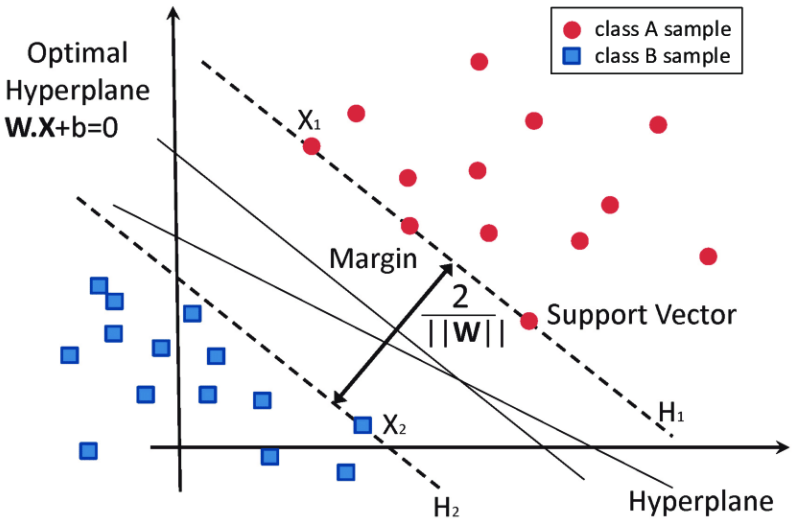

Figure 14: Principle of SVM

```
#SVM

f1_svm_val = hash()
gamma_grid = c(0.1,0.33,0.5,0.8)
for (gamma in gamma_grid){
  SVC = svm(y ~ . , data = df_train_reduced, kernel = "radial", gamma = gamma)
  svm_pred = predict(SVC, df_val[,1:3])
  f1_svm_val[gamma] = F1_Score(df_val[,4], svm_pred)
}
f1_svm_val = as.list(f1_svm_val)
best_gamma = as.numeric(names(which.max(f1_svm_val)))
SVC = svm(y ~ . , data = df_train_reduced, kernel = "radial", gamma = best_gamma)
svm_pred = predict(SVC, df_test_reduced[,1:3])
f1_svm = F1_Score(y_true, svm_pred)
```

Figure 15: Training of SVM

In the code snippet x above the training and optimization of the support vector machine is shown. We consider “gamma” the most important hyperparameter. Intuitively, “gamma” defines how far the influence of a single training observation reaches, with low values meaning “far” and high values meaning “close”. When gamma is very small, the model has too low capacity and cannot capture the complexity characteristics of the data. The region of influence of any selected support vector would include the whole training set (cf. Sci-kit learn). Hence, we would encounter an underfitting model. The value of gamma selected was 0.5, balancing the bias-variance trade-off best.

c. K-nearest neighbors

The k-Nearest neighbors (KNN) is a standard classification algorithm in machine learning.It is called non-parametric since the only minimum required parameter is the value of k. A small value for k will give a big importance to the noise when a big value will be computationally more expensive and lose the specificities of the dataset.

The way this model work is pretty simple, it is based on a distance, mostly euclidean distance or Mahalanobis (based on covariance matrix calculation and can avoid problem if the data are not standardized which is required for euclidean) that can be passed as parameter to the model. Then, to classify an individual, we determine its distance to all the other individuals using its predictors values and then choose the k nearest individuals. The attributed value for our individual will be the y of the nearest neighbors majority.

```
#KNN

x_train = df_train_reduced[,1:3]
y_train = df_train_reduced[,4]
x_test = df_test_reduced[,1:3]
f1_knn_val = hash()
K_grid = c(1,3,4,8,10,30,35,40,50,75,150)
for (K in K_grid){
  knn_pred = knn(x_train, x_test, y_train, k = K)
  f1_knn_val[K] = F1_Score(y_true, knn_pred)
}
f1_knn_val = as.list(f1_knn_val)
best_K = as.numeric(names(which.max(f1_knn_val)))
knn_pred = knn(x_train, x_test, y_train, k = best_K)
f1_knn = F1_Score(y_true, knn_pred)
```

Figure 16: Training of KNN

A major strength of kNN is that it is easy to implement and comprehend. The backlash of this method is that it is a basic non-parametric model. Moreover, you have to choose k carefully, because if I only have 10 individuals with a Revenue having true and 30 having false and if my k is higher than 20, I will always have a false as Revenue classification.

d. Logistic regression

The multiple logistic regression is used to predict the probability of a purchase based on multiple predictor variables chosen by the feature selection. The value computed by a linear regression is fitted in the [0,1] interval by a sigmoid (logistic) function in order to represent a valid probability. We tried different cutoff values to make a classification based on this probability. The hyperparameter grid is visible in figure 17.

```
#Log_Reg

f1_logreg_val = hash()
logreg = glm(y ~ . , data = df_train_reduced, family = binomial(link = "logit"))
y_prob = predict(logreg, df_test_reduced[,1:3], type = 'response')
y_true_logreg = as.integer(y_true)
cutoff_grid = c(0.3, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4, 0.45)
for (cutoff in cutoff_grid){
  y_pred = ifelse(y_prob > cutoff, 2, 1)
  f1_logreg_val[cutoff] = F1_Score(y_true_logreg, y_pred)
}
f1_logreg_val = as.list(f1_logreg_val)
best_cutoff = as.numeric(names(which.max(f1_logreg_val)))
y_pred = ifelse(y_prob > best_cutoff, 2, 1)
f1_logreg = F1_Score(y_true_logreg, y_pred)
```

Figure 17: Training of logistic regression

We also performed a regularized LASSO version of the logistic regression by using the L1 penalty in the objective function of the method. The L1 penalty is able to force some weights to zero. Hence, variables may be discarded.

e. Linear discriminant analysis

Linear discriminant analysis is a supervised classification technique very similar to logistic regression. It is used in models where categorical outputs are found and it also allows for both multi-class classification and binary classification.

```
Call:
lda(y ~ ., data = df_train_reduced)

Prior probabilities of groups:
    FALSE      TRUE
0.8451946 0.1548054

Group means:
      x.PageValues x.ExitRates x.ProductRelated_Duration
FALSE   -0.2122351  0.08988822                -0.07347409
TRUE     1.1811435 -0.48752616                 0.34927134

Coefficients of linear discriminants:
                                LD1
x.PageValues               1.0458096
x.ExitRates               -0.2239624
x.ProductRelated_Duration  0.2341902
```

Figure 18: LDA summary

PageValues has by far the largest difference between the group means of positive and negative examples, followed by ExitRates and ProductRelated_Duration. The only striking discrepancy compared to logistic regression and the findings of feature selection is the magnitude of influence of ExitRates and ProductRelated_Duration. Here ExitRates is given a smaller influence on the class variable than ProductRelated_Duration. The reason for that could be a smaller standard deviation of the positive and negative groups when groups were splitted based on ProductRelated_Duration compared to the other predictors. This enables a better separability of the groups.

7. Results and discussion

ML method	Accuracy
Random Forest	0.8828
Support Vector Machine	0.8946
Logistic Regression	0.8799
Logistic Regression with LASSO	0.8881
k-Nearest Neighbor	0.8966
Linear Discriminant Analysis	0.8751

We see that all machine learning techniques achieve a quite similar accuracy. Moreover, tuning of hyperparameters did not lead to better performance than with the default parameters suggested by the packages in R. This understanding is valid for all the machine learning approaches. That suggests that further improvements in the classifier performance might rather be possible by further data preprocessing and feature selection. It becomes clear too, that with just three predictors used we do not suffer from the “curse of dimensionality”. This is why non-parametric methods such as random forest and k-nearest neighbors do not perform worse (due to potential overfitting) than parametric methods such as logistic regression or linear discriminant analysis. Contrary to expectations, LASSO regularization has no beneficial effect on the performance by reducing potential overfitting.

To conclude, the online shop which provided the dataset should focus on making its website convenient to browse because we found out that people who spend more time looking at products are more likely to make a purchase. Another finding was that there are web pages with high ExitRates that are often visited prior to cancelling the transaction. Thus, the online shop could also try to prevent users from doing so by giving some nice recommendations on those web pages with high ExitRates. A limitation of our work might be the rather simple resampling strategy of a global holdout set. Since we perform hyperparameter tuning for almost every method, a usual cross validation would not have been viable. Instead, one would be required to perform nested cross validation. Moreover, the hyperparameter tuning could be extended to more than one hyperparameter for each method.

References

Dash, Manoranjan; Liu, Huan (1997): Feature selection for classification. In: Intelligent data analysis 1 (3), S. 131–156.

Han, Jiawei; Kamber, Micheline; Pei, Jian (2012): Data mining. Concepts and techniques. 3. ed. Amsterdam: Elsevier/Morgan Kaufmann (The Morgan Kaufmann series in data management systems).

Quinlan, John Ross (1986): Induction of Decision Trees. In: Machine Learning 1 (1), S. 81–106. DOI: 10.1023/A:1022643204877.

Sakar, C.O. et al. (2019): Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and LSTM recurrent neural networks. In: Neural Comput & Applic 31, S. 6893–6908. https://doi.org/10.1007/s00521-018-3523-0

Sci-kit learn: RBF SVM parameters, available at https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

Wirth, Rüdiger; Hipp, Jochen (2000): CRISP-DM: Towards a standard process model for data mining. In: Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining. London, UK: Springer-Verlag.

Appendix

Main script for preprocessing, resampling, model training and evaluation

#Import and data type conversion

```
df = read.csv("online_shoppers_intention.csv")
set.seed(71)

df$TrafficType = as.factor(df$TrafficType)
df$VisitorType = as.factor(df$VisitorType)
df$Browser = as.factor(df$Browser)
df$Region = as.factor(df$Region)
df$OperatingSystems = as.factor(df$OperatingSystems)
df$Month = as.factor(df$Month)
df$Weekend = as.factor(df$Weekend)
df$Revenue = as.factor(df$Revenue)
y = df$Revenue
```

#One-hot encoding of categorical variables

```
dmy = dummyVars(~ . , data = df)
df = data.frame(predict(dmy, newdata = df))
df$Revenue.FALSE = NULL
df$Revenue.TRUE = NULL
df$Revenue = y

num_features = data.frame(df$Administrative, df$Administrative_Duration, df$Informational, df$Informational_Duration,
                          df$ProductRelated, df$ProductRelated_Duration, df$BounceRates, df$ExitRates, df$PageValues, df$SpecialDay)
num_features_scaled = data.frame(scale(num_features))
drops = c('Administrative', 'Administrative_Duration', 'Informational', 'Informational_Duration',
                          'ProductRelated', 'ProductRelated_Duration', 'BounceRates', 'ExitRates', 'PageValues', 'SpecialDay')
df_cat = df[,!(names(df) %in% drops)]
df_cat$ID = seq.int(nrow(df))
num_features_scaled$ID = seq.int(nrow(num_features_scaled))
df = join(df_cat, num_features_scaled, by = "ID")
```

#80/20 train-test-split

```
df = df[sample(nrow(df)),]
size_train = 0.8*nrow(df)
start_test = size_train + 1
df_train = df[1:size_train,]
df_test = df[start_test:12330,]
```

#Extract selected variables

```
df_train_reduced = data.frame(x = data.frame(df_train$df.PageValues, df_train$df.ExitRates, df_train$df.ProductRelated_Duration), y = df_train$Revenue)
df_test_reduced = data.frame(x = data.frame(df_test$df.PageValues, df_test$df.ExitRates, df_test$df.ProductRelated_Duration), y = df_test$Revenue)
names(df_train_reduced) = c("x.PageValues", "x.ExitRates", "x.ProductRelated_Duration", "y")
names(df_test_reduced) = c("x.PageValues", "x.ExitRates", "x.ProductRelated_Duration", "y")
y_true = df_test_reduced[,ncol(df_test_reduced)]
```

#Extract validation set

```
size_train = 0.8*nrow(df_train_reduced)
start_val = size_train + 1
df_val = df_train_reduced[start_val:9864,]
df_train_reduced = df_train_reduced[1:size_train,]
```

#Random Forest

```
f1_rf_val = hash()
ntree_grid = c(20,30,50,80,120,160,200,500,600)
for (number_of_trees in ntree_grid){
  rf = randomForest(y ~ . , data = df_train_reduced, ntree = number_of_trees, importance = TRUE)
  rf_pred = predict(rf, df_val[,1:3])
  f1_rf_val[number_of_trees] = F1_Score(df_val[,4], rf_pred)
}
f1_rf_val = as.list(f1_rf_val)
best_ntree = as.numeric(names(which.max(f1_rf_val)))
rf = randomForest(y ~ . , data = df_train_reduced, ntree = best_ntree, importance = TRUE)
rf_pred = predict(rf, df_test_reduced[,1:3])
f1_rf = F1_Score(y_true, rf_pred)
```

#Support Vector Machine

```
f1_svm_val = hash()
gamma_grid = c(0.1,0.33,0.5,0.8)
for (gamma in gamma_grid){
  SVC = svm(y ~ . , data = df_train_reduced, kernel = "radial", gamma = gamma)
  svm_pred = predict(SVC, df_val[,1:3])
  f1_svm_val[gamma] = F1_Score(df_val[,4], svm_pred)
}
f1_svm_val = as.list(f1_svm_val)
best_gamma = as.numeric(names(which.max(f1_svm_val)))
SVC = svm(y ~ . , data = df_train_reduced, kernel = "radial", gamma = best_gamma)
svm_pred = predict(SVC, df_test_reduced[,1:3])
f1_svm = F1_Score(y_true, svm_pred)
```

#Logistic Regression

```
f1_logreg_val = hash()
logreg = glm(y ~ . , data = df_train_reduced, family = binomial(link = "logit"))
y_prob = predict(logreg, df_test_reduced[,1:3], type = 'response')
y_true_logreg = as.integer(y_true)
cutoff_grid = c(0.3, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4, 0.45)
for (cutoff in cutoff_grid){
  y_pred = ifelse(y_prob > cutoff, 2, 1)
  f1_logreg_val[cutoff] = F1_Score(y_true_logreg, y_pred)
}
f1_logreg_val = as.list(f1_logreg_val)
best_cutoff = as.numeric(names(which.max(f1_logreg_val)))
y_pred = ifelse(y_prob > best_cutoff, 2, 1)
f1_logreg = F1_Score(y_true_logreg, y_pred)
```

#Logistic Regression with LASSO regularization

```
lambdas = 10^seq(0, -3, by = -.1)
logreg_lasso_cv = cv.glmnet(as.matrix(df_train_reduced[,1:3]), df_train_reduced[,4], family = "binomial", lambda = lambdas)
opt_lambda = logreg_lasso_cv$lambda.min
logreg_lasso = glmnet(as.matrix(df_train_reduced[,1:3]), df_train_reduced[,4], family = "binomial", lambda = opt_lambda)

f1_logreg_lasso_val = hash()
y_prob = predict(logreg_lasso, as.matrix(df_test_reduced[,1:3]), type = 'response')
y_true_logreg = as.integer(y_true)
cutoff_grid = c(0.3, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4, 0.45)
for (cutoff in cutoff_grid){
  y_pred = ifelse(y_prob > cutoff, 2, 1)
  f1_logreg_lasso_val[cutoff] = F1_Score(y_true_logreg, y_pred)
}
f1_logreg_lasso_val = as.list(f1_logreg_lasso_val)
best_cutoff = as.numeric(names(which.max(f1_logreg_lasso_val)))
y_pred_lasso = ifelse(y_prob > best_cutoff, 2, 1)
f1_logreg_lasso = F1_Score(y_true_logreg, y_pred_lasso)
```

#k-Nearest Neighbors

```
x_train = df_train_reduced[,1:3]
y_train = df_train_reduced[,4]
x_test = df_test_reduced[,1:3]
f1_knn_val = hash()
K_grid = c(1,3,4,8,10,30,35,40,50,75,150)
for (K in K_grid){
  knn_pred = knn(x_train, x_test, y_train, k = K)
  f1_knn_val[K] = F1_Score(y_true, knn_pred)
}
f1_knn_val = as.list(f1_knn_val)
best_K = as.numeric(names(which.max(f1_knn_val)))
knn_pred = knn(x_train, x_test, y_train, k = best_K)
f1_knn = F1_Score(y_true, knn_pred)
```

#Linear Discriminant Analysis

```
lda.fit = lda(y ~ . , data = df_train_reduced)
lda.fit
lda.pred = predict(lda.fit, newdata = df_test_reduced)
lda.class = lda.pred$class
f1_lda = F1_Score(y_true, lda.class)
f1_lda
```

#Storing results

```
f1_scores = hash()
f1_scores["RF"] = f1_rf
f1_scores["SVM"] = f1_svm
f1_scores["Log Reg"] = f1_logreg
f1_scores["Log Reg LASSO"] = f1_logreg_lasso
f1_scores["KNN"] = f1_knn
f1_scores["LDA"] = f1_lda
accuracies = hash()
accuracies["RF"] = Accuracy(rf_pred, y_true)
accuracies["SVM"] = Accuracy(svm_pred, y_true)
accuracies["Log Reg"] = Accuracy(y_pred, y_true_logreg)
accuracies["Log Reg LASSO"] = Accuracy(y_pred_lasso, y_true_logreg)
accuracies["KNN"] = Accuracy(knn_pred, y_true)
accuracies["LDA"] = Accuracy(lda.class, y_true)
f1_scores
accuracies
```

Auxiliary script for feature selection

#Import and data type conversion

```
df = read.csv("online_shoppers_intention.csv")
set.seed(71)

df$TrafficType = as.factor(df$TrafficType)
df$VisitorType = as.factor(df$VisitorType)
df$Browser = as.factor(df$Browser)
df$Region = as.factor(df$Region)
df$OperatingSystems = as.factor(df$OperatingSystems)
df$Month = as.factor(df$Month)
df$Weekend = as.factor(df$Weekend)
df$Revenue = as.factor(df$Revenue)
```

#Chi-squared filter for categorical variables

```
cat_features = data.frame(df$Month, df$OperatingSystems, df$Browser, df$Region, df$TrafficType, df$VisitorType, df$Weekend)
chi_squares = hash()
i = 1
for (feature in cat_features){
  chi_sq = chisq.test(x = feature, y = df$Revenue)
  chi_squares[[colnames(cat_features[i])]] = chi_sq$statistic
  i = i + 1
}
chi_squares
```

#ANOVA filter for numerical variables

```
num_features = data.frame(df$Administrative, df$Administrative_Duration, df$Informational, df$Informational_Duration, df$ProductRelated, df$ProductRelated_Duration, df$BounceRates, df$ExitRates, df$PageValues, df$SpecialDay, df$Revenue)
F_values = hash()
j = 1
for (feature in num_features[1:10]){
  res.aov = aov(feature ~ df.Revenue, data = num_features)
  F_values[[colnames(num_features[j])]] = summary.aov(res.aov)[[1]][["F value"]][[1]]
  j = j + 1
}
F_values
```

#Relief filter

```
weights = relief(Revenue ~ ., df[1:12330,], neighbours.count = 5, sample.size = 20)
print(weights)
```

#Stepwise forward and backward feature selection

```
fs_lda_forward = stepclass(num_features, df$Revenue, "lda", direction = "forward", improvement = 0.01)
fs_lda_backward = stepclass(num_features, df$Revenue, "lda", direction = "backward")
plot(fs_lda_forward)
plot(fs_lda_backward)
```

Required packages in R

caret
class
dplyr
e1071
FSelector
ggplot2
glmnet
klaR
lattice
MASS
MLMetrics
plyr
randomForest