KU LEUVEN

FACULTY OF SCIENCE

Assignments

Advanced Analytics for a Big Data World
Group 7

Gianina CRISTIAN
Daniel GIL SANCHEZ
Chanukya Patnaik MANIPATRUNI
Donald VAN MARCKE
Hotze VELDHUIS

Supervisor: Prof. Dr. Seppe vanden Broucke

Academic year 2018-2019

Contents

1 Assignment 1: Paper review

Paper summary

A simple approach to Ordinal Classification by Frank and Hall presents a method that enables standard classification algorithms to make use of ordering information in class attributes.

First the data is transformed from a k-class ordinal problem to a $k-1$ binary class problem. Training starts by deriving one new dataset for each of the $k-1$ binary class attributes. In the next step the classification algorithm is applied to generate a model for each of the datasets. To predict the class value of an unseen instance the probabilities of the k original ordinal classes is estimated using the $k-1$ models. The class with maximum probability is assigned to the unseen instance.

By applying the proposed method in conjunction with a decision tree learner on 29 artificial and real-world datasets it is shown that the ordinal classification method frequently outperforms the naive approach, which treats the class values as an unordered set.

Application setting

The ordinal classification method is applicable in conjunction with any base learner that can output class probability estimates. The technique of generating binary attributes to replace an ordinary attribute can also be applied to the attributes making up the input space.

The method proposed by Frank and Hall is applicable in many fields. In many real life problems items or objects are compared or ranked in order to make the most appropriate choice for a specific goal. Examples include collaborative filtering, customer classification in marketing research, stock trading support systems, where one wants to predict whether to buy, keep or sell a stock.

Collaborative filtering is a method of making automatic predictions about the interests of a user by collecting information from many users. For example, a collaborative filtering recommendation system for books could make predictions about which book a user should like given a partial list of that user's book ratings. These predictions use information gleaned from many users, for example amazon.com where users rate books between one and five stars.

Review

An advantage of the method proposed by Frank and Hall is the increase in accuracy for ordinal prediction problems compared to methods where one assumes the class values are unordered. Frank and Hall performed experiments on 29 artificial and real-world datasets with the C4.5 decision tree learner. Results show there is very strong evidence that the ordinal classification procedure frequently improves performance compared to the naive approach. Furthermore there is strong evidence that the ordinal classification procedure becomes even more useful when the number of classes increases.

Compared to special-purpose algorithms for ordinal classification the method discussed in this paper has the advantage that it can be applied without any modification to the underlying learning scheme.

However, it has been suggested, that this method causes some weaknesses:

- **Cases where standard one per class encoding (C4.5-1PC) outperforms ordinal classification (C4.5-ORD):** While analyzing the results for the three-bin discretization, they show that there is relatively little to be gained by exploiting the ordering information in the class. Compared to C4.5, C4.5-ORD is significantly more accurate on 15 datasets, and significantly worse on none. However, C4.5-ORD does not perform markedly better than C4.5-1PC: it is significantly more accurate on 10 datasets, and significantly worse on seven. It is interesting to see that the one-per-class encoding outperforms plain C4.5 on these three-class datasets. Further, if the significance of the individual differences is not taken into account and ignoring the smaller version of each pair of datasets from the same domain, C4.5-ORD wins against C4.5 on 20 datasets, and looses on four. This difference is statistically significant according to a two-sided sign test: the corresponding p-value is 0.0008. However, compared to C4.5-1PC, the win/loss-ratio for C4.5- ORD is 15/10, with a corresponding p-value of 0.21. Thus there is only very weak evidence that the ordinal classification method improves on the standard one-per-class encoding.

- **Ordering information becomes more useful as the no. of classes increases:** As we increase the discretization, for example, for the ten-bin case we expect a more significant difference, in particular when compared to the one-per-class method. The difference in performance between C4.5-ORD and C4.5 remains high but increases only slightly when moving from five to ten bins. C4.5-ORD outperforms C4.5 on all but seven of the datasets. It is significantly more accurate on 22 datasets, and significantly less accurate on only two. Compared to C4.5-1PC, C4.5-ORD is significantly more accurate on 25 datasets, and significantly worse on two. This is a marked improvement over the five-bin case and suggests that ordering information becomes more useful as the number of classes increases.

Expansion

Some expansions of the simple ordinal classification can be found below.

- Given that the performance the ordinal classification is largely dependent on the number of classes in the problem i.e., performance of ordinal classification increases (when compared to tradition classification methods such as one per class encoding) as the number of classes increases. It would interesting to find viable solutions for problems with fewer number of classes.

- Ordinal regression methods based on SVM could improve the performance. However, it is often prone to high computational complexity and might be ignoring global information. It would worthwhile to enhance this and find an optimal solution which decreases the complexity eventually enhancing the performance. Kernel Discriminant analysis using a rank constraint could be a viable solution.

- More recently, a cascade classification technique encompassing a decision tree classifier and a model tree algorithm was also proposed [1]. A new reduction technique could also be used allowing to solve the problem of ordinal classification using a single binary classifier.

2 Assignment 2: Predictive modelling

Introduction

A Latin-American telco provider is interested in predicting customer churn, which occurs when a customer stops doing business with the company. A dataset with 5000 instances is provided to train a prediction model that enables the company to identify which costumers are more likely to leave in terms of 38 features that were extracted by the end of October 2013. The status of the customer, i.e., whether the customer leave or not the company, is collected by the end of December 2013.

The goal of this assignment is to obtain a prediction model, based on machine learning algorithms, that allows the company to identify with high confidence which costumers are more likely to leave, in an action period of one month.

In the following, a exploratory analysis is presented with the intention of identifying patterns in the dataset. Then, the preprocessing of the dataset is described and the models used in the prediction are shown.

Exploratory Analysis

The dataset has two types of variables, the majority of them, 33, are numerical measures and the remaining are categorical. Among the categorical variables lies the target variable, Churn, which is a dummy variable where 1 represents a costumer that left the company in December 2013 and 0 represents a costumer that is still in the company.

Prior to explore patterns in the customers, a partition of the training set was made with the purpose of having data to test the prediction of each model. This was performed by randomly selecting 80% of the customers in the dataset to train the models and 20% to test them. As a result, the training set has 4000 customers and the test set has 1000 observations.

It is important to mention that another dataset, composed of 2033 customers, was given to test the prediction of the model. However, it was not possible to use it in the validation of the models because the response variable was not available. The intention of this dataset was to make comparisons of the quality of the predictions between different groups.

Table 1 shows univariate descriptive statistics of the training set. The labels of the variables are usually self-explanatory but when it is not the case the explanation of them is given in the correspondent interpretation. Note that from now on the training set is the dataset with 4000 customers. The test set is not going to be used in this section nor the following but only in the evaluation of the models.

In this dataset, 28.75% of the customers left the company in December 2013, indicating that up-sampling may be needed to get better predictions. **FIN_STATE** is a dummy variable that indicates the financial state parameter of a customer, but only 4.48% of them have this information. What is important to mention about this variable is the number of missing values: 2906 customers did not have information, that is over 72% of them. For this reason, this variable was not used in the training process.

In regard to continuous variables, what is important to notice is the difference in scale between of all them. For instance, while the average number of complaints

received in the past month is 0.034, the average number of bytes consumed is 600 millions. This indicated that a transformation of the continuous variables, such as standardization, was a good idea and actually necessary for any feature importance methodology.

Variable	Mean/ Frequency in %	Std. Deviation	Min/Max
CHURN	28.75		
FIN_STATE	4.48		
PREPAID	22.65		
COMPLAINT_1MONTH	0.034	0.26	0/4
COMPLAINT_1WEEK	0.14	0.56	0/8
COMPLAINT_2WEEKS	0.063	0.36	0/7
COMPLAINT_3MONTHS	0.38	0.96	0/8
COMPLAINT_6MONTHS	0.71	1.32	0/12
COUNT_CONNECTIONS_3MONTH	48.19	49.45	0/236
COUNT_OFFNET_CALLS_1WEEK	17.86	22.44	0/266
COUNT_ONNET_CALLS_1WEEK	29.68	42.89	0/867
COUNT_PAYMENT_DELAYS_1YEAR	2.53	2.73	0/18
COUNT_PAYMENT_DELAYS_CURRENT	0.12	0.47	0/6
COUNT_SMS_INC_OFFNET_1MONTH	14.33	36.17	0/628
COUNT_SMS_INC_OFFNET_WKD_1MONTH	3.42	11.16	0/211
COUNT_SMS_INC_ONNET_1MONTH	83.17	321.03	0/19178
COUNT_SMS_INC_ONNET_6MONTH	467.75	1447.01	0/70946
COUNT_SMS_INC_ONNET_WKD_1MONTH	20.59	116.45	0/7121
COUNT_SMS_OUT_OFFNET_1MONTH	16.08	35.49	0/586
COUNT_SMS_OUT_OFFNET_6MONTH	87.93	157.15	0/2422
COUNT_SMS_OUT_OFFNET_WKD_1MONTH	3.54	10.67	0/214
COUNT_SMS_OUT_ONNET_1MONTH	25.17	62.25	0/1220
COUNT_SMS_OUT_ONNET_WKD_1MONTH	5.93	19.17	0/486
DAYS_PAYMENT_DELAYS_1YEAR	37.84	102.32	0/1235
DAYS_PAYMENT_DELAYS_CURRENT	-5.35	27.47	$-101/0$
AVG_DATA_1MONTH	6×10^8	1.8×10^9	$0/3.8 \times 10^{10}$
AVG_DATA_3MONTH	2.6×10^{12}	1.6×10^{14}	$0/9.9 \times 10^8$
AVG_MINUTES_INC_OFFNET_1MONTH	63.11	108.93	0/99.97
AVG_MINUTES_INC_ONNET_1MONTH	76.81	153.8	$0/99.9'$
AVG_MINUTES_OUT_OFFNET_1MONTH	59.34	93.89	0/99.88
AVG_MINUTES_OUT_ONNET_1MONTH	80.06	148.03	0/99.92
CLV	19735.06	15115.13	$-12257.42/225838.8$
MINUTES_INC_OFFNET_WKD_1MONTH	26.21	44.66	0/99.77
MINUTES_INC_ONNET_WKD_1MONTH	29.26	46.33	0/645.47
MINUTES_OUT_OFFNET_WKD_1MONTH	24.4	33.1	0/671.8
MINUTES_OUT_ONNET_WKD_1MONTH	30.67	46.73	0/650.3

Table 1: Univariate Descriptive statistics

With respect to multivariate descriptive statistics, a principal components analysis

was performed on the standardized continuous variables. 20 principal components retained 90% of the variability, meaning that there was a lot of variation in the dataset and it was not possible to reduce further the dimensionality of the input space without sacrificing the quality of the representation in a lower dimension space. In the sake of illustration, Figure 1 presents a representation of the variables in the first two principal components, which explained over 23% of the total variability. Each arrow was coloured taking into the account the contribution of the corresponding variable with respect to the components, also represented with the length of the vector.

This plot suggests that those variables that measure the number of SMS messages sent and received over different time frames are highly correlated (located in the first quadrant) as well as those variables that measure the consumption of calls in different time frames (located in the fourth quadrant). Since the angle between these groups of vectors is around 90 degrees, it can be concluded that the consumption of SMS is not correlated with the consumption of Calls. On the other hand, it is better not to give any conclusion about those variables that measure the number of complaints received during different time frames and the payment behaviour of the consumer, because their contributions in this lower dimensional space were not large.

Overall, from this principal components analysis can be concluded that some of the variables were highly correlated and therefore it is very likely that only a few of them are important in the training process. This was verified in each model fitted and presented later in this document.

Finally, with respect to the relationship between the independent variables and the response, an analysis is presented in the next section after the preprocessing of the dataset was performed.

Preprocessing

As it was mentioned in the previous section, the first preprocessing step was to make a partition of the training set into a smaller training set with 80% of the customers and a test set with the remain 20%. The following steps were conducted only on the training set and then the same preprocess was applied to both test datasets.

Basically the preprocessing involves imputing missing values, transforming and creating new variables, encoding categorical variables and any kind of sampling methodology to deal with the imbalance of the target variable, if there is indeed imbalance. In this case, four variables have missing values: **FIN_STATE** (mentioned before), **COUNT_CONNECTIONS_3MONTH**, **AVG_DATA_1MONTH**, **AVG_DATA_3MONTH**. These last three variables are related to the number of data connections and the average bytes of data used per connection. Of course if the customer does not have a data subscription, it will not have any value. This is the so-called structural missing value and instead of keeping them as *NAs* in the dataset, they were replaced by zeros. In this way no noise is introduced in the dataset and the interpretation of those zeros make sense.

Regarding the transformation and creation of new variables, the date when the customer joined the telco provider was used to create the number of days the customer has been with the company until the end of October 2013. Additionally, all continuous variables were standardized so they have zero mean and unit variance. On the other hand, no encoding was needed in the categorical variables because they only have

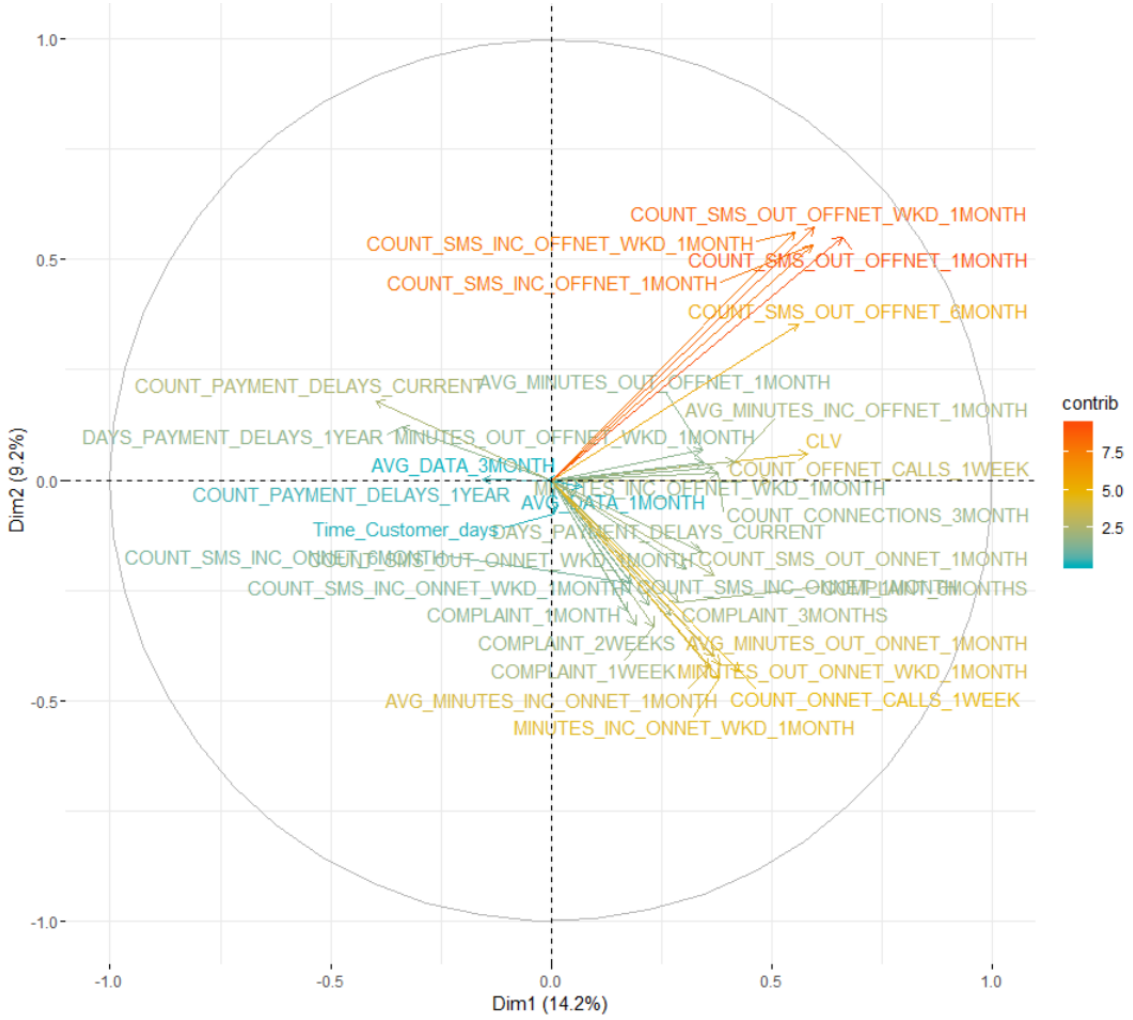


Figure 1: Principal Components

two categories, either zero or one. If there had been any categorical variable with more categories, hot encoding would have been used.

Finally, since only 28% of the customers left the company in December 2013, up-sampling was performed to match the proportion of successes and failures in this variable. This was conducted by creating random copies of those customers that left the company until the proportion of successes was 50%.

In regard to both test sets, the structural missing values were replaced by zeros, the variable number of days the customer has been in the company was created and all continuous variables were transformed by removing the corresponding mean and dividing by the standard deviation from the training set.

Now, it is useful to check the relationship between the explanatory variables and the target variable. Figure 2 shows the density of each variable grouped by the response variable, blue for customers that are still in the company and pink for customers that left. In this case, a variable is considered important if the density curves of both kinds of customers are different in terms of kurtosis (height) and skewness (placement).

The first thing that can be noticed is the skewness in the distribution of almost all variables. The majority of measures of the customers lies in the lower range of values

regardless their situation with respect to the company. Now, focusing in the height of the densities, it is easy to see that variables related to the duration of the calls, first row of plots, present differences in the peak for each kind of customer. Also the peak of the density of the customer lifetime value (**CLV**) for customers that are still in the company is highest than the other category. Of special interest is also the extreme values of variables related to the number of days the customer paid late, last row of plots, because usually these values correspond to customers who left the company. Finally, it is wise to mention that these variables are likely to be important to predict the response variable, but there could be hidden patters in the data, which are not possible to visualize, that can impact the way some variables interact with the decision of the customer.

Training process

Different models were considered to predict whether a customer leaves the company or not. In this section, a description of each model trained is given together with some performance measures. `R` was the software used to train these models, using the package `caret` as an umbrella over all single packages that developed each algorithm. The motivation to use `caret` as the single tool to train different models was that it allows the user to set some characteristics of the training process easily instead of programming them by hand. Things such:

- Using cross-validation.
- Tune hyper-parameters for optimal performance
- Choose an optimal model based on a given metric

In this case, this advantage was exploited by training each model using three times repeated 10-fold cross-validation scheme, using different hyper-parameters, which are mentioned in each model, and predicting the probability of a customer leaving the company. The metric used to choose the best possible model was the area under the ROC curve.

Logistic Regression

This is probably the simplest, yet powerful, interpretable, model. By definition it does no have hyperparameters to tune, but it can be protected against overfitting and variable selection by using any kind of regularization. In this case, we consider Lasso (L1), Ridge (L2) and the combination of both, elastic net. A main effects model was trained with no interactions nor higher order effects, so it can be a disadvantage with respect to other models.

To tune the hyperparameters of each regularization a grid search was computed between the mixing percentage, *alpha*, and the regularization parameter, *lambda*. For Lasso regularization alpha was set to one by definition, and lambda was defined as a sequence between 0 and 2, increasing by 0.5. In a similar fashion, for Ridge regularization, alpha was set to zero by definition, and lambda was defined as a sequence

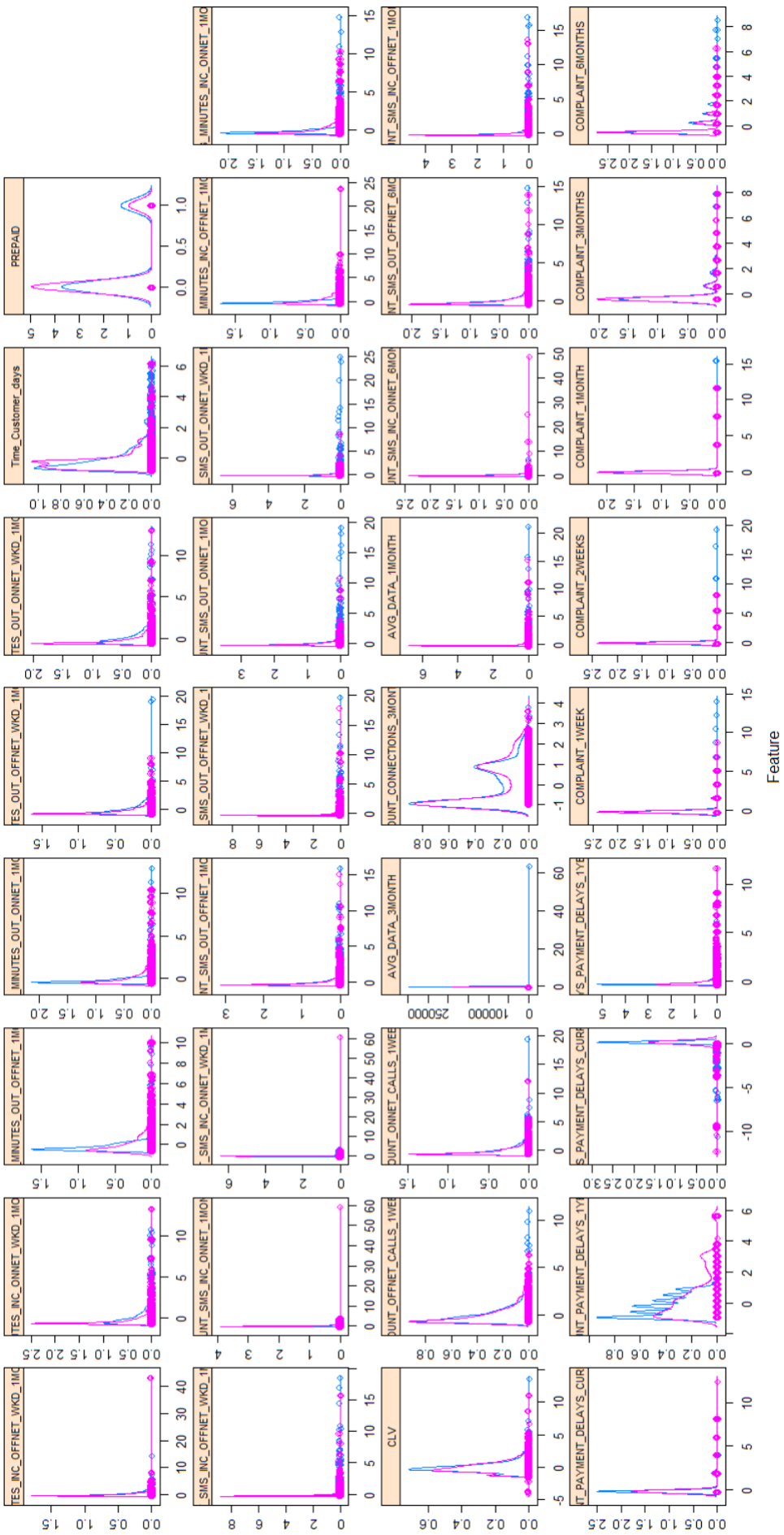


Figure 2: Density of each variable grouped by the target variable (blue: still in the company, pink: left the company)

between 0 and 2, increasing by 0.5. On the other hand, for the elastic net alpha was defined as a sequence between 0 and 1, increasing by 0.1 and lambda was a sequence of ten numbers between 0 and 0.15. As a result, the best hyperparameter selected based on the ROC curve was lambda equal to zero for Lasso and Ridge, and for the elastic net, alpha was one and lambda was 0.00015, which is really close to Lasso.

As it was mentioned before, one of the advantages of logistic regression is the interpretability of its parameters. Table 2 shows the estimated coefficients of each variable for each model considered. This table shows that the difference in the estimation of the parameters between models is not large, specially if a comparison between the logit, lasso and elastic net is of interest. This was an expected result because the lambda chosen in those regularizations are zero or close to zero. On the other hand, some of the Ridge coefficients were shrunk towards zero, as is the case of **COUNT_PAYMENT_DELAYS_1YEAR**.

In addition, those parameters that were significant in the Logit model are blue coloured taking as a reference a significance level of 5%. In this way it is easy to see which variables are important in the model, and the correspondent sign illustrates whether the variable increases or decreases the probability of leaving the company. As an illustration, variables such as **DAYS_PAYMENT_DELAYS_1YEAR** or **AVG_MINUETS_INC_OFFNET_1MONTH** increases the probability of leaving the company, whereas **COUNT_OFFNET_CALLS_1WEEK** or **COUNT_SMS_INC_ONNET_WKD_1MONTH** decreases this probability.

Variable	Logit	Lasso	Ridge	Elastic Net
Intercept	−0.199	−0.196	−0.157	−0.196
COUNT_PAYMENT_DELAYS_CURRENT	−0.506	−0.502	−0.286	−0.5
COUNT_PAYMENT_DELAYS_1YEAR	0.354	0.352	0.367	0.352
DAYS_PAYMENT_DELAYS_CURRENT	0.207	0.203	0.114	0.202
DAYS_PAYMENT_DELAYS_1YEAR	0.984	0.975	0.557	0.972
COMPLAINT_1WEEK	−0.017	−0.016	−0.02	−0.015
COMPLAINT_2WEEKS	0.012	0.01	0.016	0.009
COMPLAINT_1MONTH	0.115	0.114	0.089	0.114
COMPLAINT_3MONTHS	−0.033	−0.033	−0.037	−0.032
COMPLAINT_6MONTHS	−0.113	−0.113	−0.1	−0.113
CLV	−0.162	−0.159	−0.086	−0.158
COUNT_OFFNET_CALLS_1WEEK	−0.363	−0.361	−0.213	−0.36
COUNT_ONNET_CALLS_1WEEK	−0.362	−0.363	−0.294	−0.363
AVG_DATA_3MONTH	−0.142	−0.047	−0.032	−0.042
COUNT_CONNECTIONS_3MONTH	0.15	0.148	0.121	0.147
AVG_DATA_1MONTH	−0.072	−0.071	−0.062	−0.071
COUNT_SMS_INC_ONNET_6MONTH	−0.011	−0.01	−0.015	−0.01
COUNT_SMS_OUT_OFFNET_6MONTH	−0.03	−0.03	−0.033	−0.03
COUNT_SMS_INC_OFFNET_1MONTH	0.079	0.07	0.03	0.067
COUNT_SMS_INC_OFFNET_WKD_1MONTH	−0.18	−0.167	−0.105	−0.163
COUNT_SMS_INC_ONNET_1MONTH	−1.335	−1.158	−0.08	−1.099
COUNT_SMS_INC_ONNET_WKD_1MONTH	1.594	1.375	0.095	1.302
COUNT_SMS_OUT_OFFNET_1MONTH	−0.019	−0.01	0.016	−0.007
COUNT_SMS_OUT_OFFNET_WKD_1MONTH	0.142	0.129	0.068	0.124
COUNT_SMS_OUT_ONNET_1MONTH	0.206	0.162	−0.063	0.147
COUNT_SMS_OUT_ONNET_WKD_1MONTH	−0.442	−0.393	−0.136	−0.376
AVG_MINUTES_INC_OFFNET_1MONTH	1.108	1.099	0.7	1.096
AVG_MINUTES_INC_ONNET_1MONTH	0.551	0.545	0.404	0.543
MINUTES_INC_OFFNET_WKD_1MONTH	−0.714	−0.708	−0.452	−0.706
MINUTES_INC_ONNET_WKD_1MONTH	−0.176	−0.173	−0.146	−0.172
AVG_MINUTES_OUT_OFFNET_1MONTH	0.758	0.753	0.541	0.751
AVG_MINUTES_OUT_ONNET_1MONTH	0.5	0.497	0.394	0.496
MINUTES_OUT_OFFNET_WKD_1MONTH	−0.676	−0.671	−0.493	−0.67
MINUTES_OUT_ONNET_WKD_1MONTH	−0.344	−0.341	−0.247	−0.34
TIME_CUSTOMER_DAYS	−0.278	−0.277	−0.242	−0.277
PREPAID	−0.266	−0.266	−0.278	−0.266

Table 2: Estimated coefficients Logit models

Now, in terms of evaluation of the predictive performance of these models in new datasets, different measures were used to compare them with special emphasis in the area under the ROC curve (AUC). Table 3 shows the confusion matrix for each model using the best threshold, which was selected by maximizing the sum of sensitivity and specificity, and other performance measures. The performance of the Logit, Lasso and Elastic net models are quite similar: the area under the curve is the same, the true

positive rate (sensitivity) and true negative rate (specificity) is almost the same, the accuracy is also the same. On the other hand, Ridge model outperforms the prediction of customers that left the company, 82.8%, but the performance of the prediction of customers that are still is the company is lower, 67.2%. Having these results, it was important to judge what performance measure was the best in this context. Since the goal of the exercise is to correctly predict those customers that are likely to leave to company, special focus should be given to the sensitivity as it is the case of the Ridge model, even though it is going to flag some customers that are not going to leave the company. Figure 3 shows the corresponding ROC curves.

Model	AUC	Thres.	TP	FP	TN	FN	Acc.	Prec.	Sens.	Spec.
Logit	0.83	0.448	227	187	517	69	0.744	0.548	0.767	0.734
Lasso	0.83	0.443	229	189	515	67	0.744	0.548	0.774	0.732
Ridge	0.829	0.42	245	231	473	51	0.718	0.515	0.828	0.672
Elastic net	0.83	0.445	229	189	515	67	0.744	0.548	0.774	0.732

Table 3: Logit models. Confusion matrix and performance measures

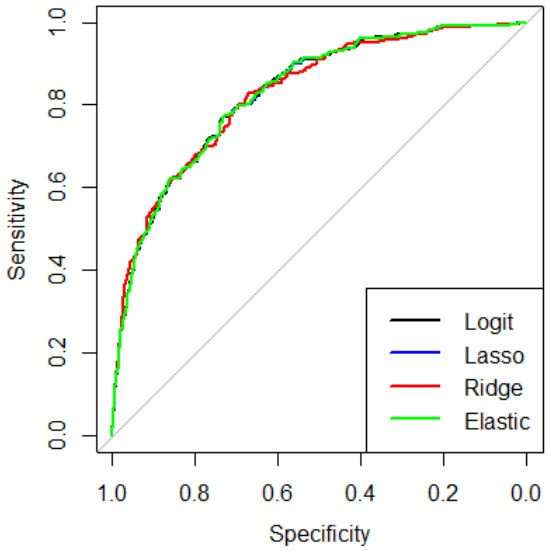


Figure 3: Logit models. ROC curve

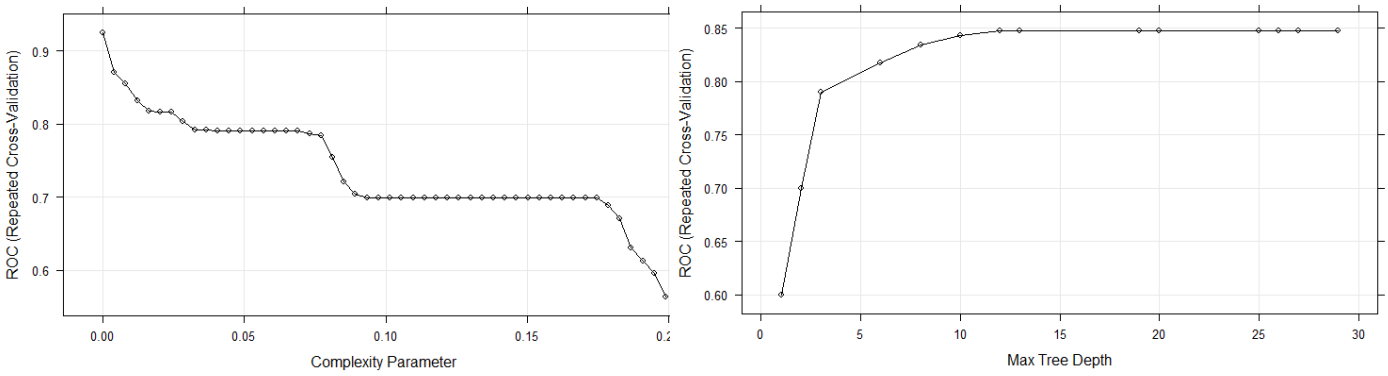
Decision tree

One of the advantages of this kind of model with respect to logistic regression is the capacity to discover interactions between variables. Its interpretability is also straightforward because it creates a sequence of "good" questions that allows the company to classify with confidence whether a customer leaves or not. However, one important disadvantage is that it usually gives overfitted solutions. For this reason, generalization had to be taken into account in the training process.

Two different methods were used to train a decision tree inside the `caret` package: `rpart`, which has one hyperparameter called *complexity parameter*, and `rpart2`, which

also depends on one hyperparameter called *Max Tree Depth*. In the first method, any split that does not decrease the lack of fit by a factor of the complexity parameter is not attempted. Therefore, this parameter is used to improve computing time by pruning those splits that are not worthwhile. The larger the complexity parameter is, the more pruned the decision tree will be. In this case, 50 different complexity parameters were used, in a range between 0 and 0.20, and the selection of the best hyperparameter was based on the ROC curve curve using 10-fold cross-validation repeated three times as it was mentioned before.

In the second method, the depth of the tree, defined as the number of nodes between the root node and any leaf node, is controlled by setting a maximum value. This is another way to protect the model from overfitting and usually works pretty well. In this case, 15 different depths were considered, from 1 to 29 and the selection of the best depth was based on the same criteria mentioned before. Figure 4 shows the ROC for each method using different hyperparameters. Figure 4a shows that when the complexity parameter increases the ROC deteriorates, implying that the best hyperparameter is zero. Figure 4b shows that the ROC is the same in a tree with a depth of 29 and a depth of 12. Pruning the tree further decreases the performance of the tree, so the best *Max Depth* hyperparameter is 12.



(a) rpart (b) rpart2

Figure 4: Tuning hyperparameters for decision trees based on area under ROC curve

In terms of evaluation of the predictive performance of these models in new datasets, different measures were used to compare them with special emphasis in the area under the ROC curve (AUC). Table 4 shows the confusion matrix for each model using the best threshold, as it was presented before. The AUC of both methods is the same and it is better than the best result obtained from the logistic model. The threshold that maximizes the sum of the true positive rate and true negative rate is 0.207 and 0.269 for `rpart` and `rpart2` respectively. The accuracy, precision and specificity of the first method are better than the second method but the sensitivity is better for the later. Putting this results in context, the second method was chosen because the true positive rate was larger and the company is interested in predicting with accuracy those customers that are more likely to leave. Furthermore, since the final tree was pruned, it is easiest to deploy and maintain. Figure 5 shows the corresponding ROC curves.

Model	AUC	Thres.	TP	FP	TN	FN	Acc.	Prec.	Sens.	Spec.
rpart	0.863	0.207	243	124	580	53	0.823	0.662	0.821	0.824
rpart2	0.863	0.269	254	162	542	42	0.796	0.611	0.858	0.77

Table 4: Decision tree. Confusion matrix and performance measures

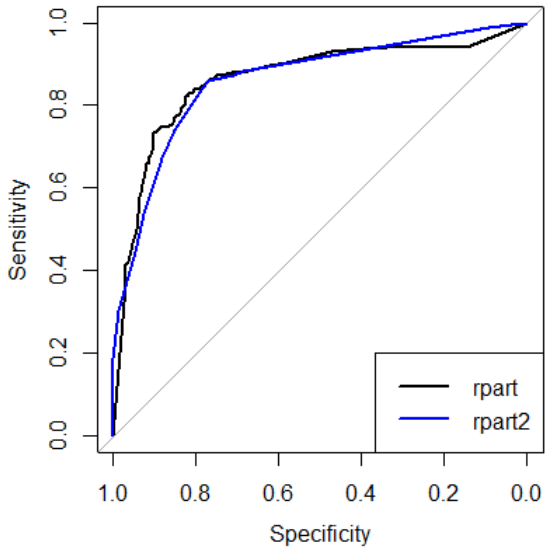


Figure 5: Decision tree. ROC curve

In addition to the performance, it was also of interest to know which variables were important in the training process. To see this behaviour, `caret` package has an in-built function that conducts ROC curve analysis on each predictor. This analysis consists in applying a series of cutoffs to the predictor data to predict the class. Then the AUC is computed and normalized to be between 0 and 100. The larger the value the more important the predictor is in the model. Figure 6 shows the score associated to each predictor, where it can be seen that only 17 variables (out of 35) were important in the training process. These results make this model even more useful because with a little more than half of the variables, good predictions can be obtained.

The top 5 of variables that were important for this model is: **AVG_MINUTES_OUT_OFFNET_1MONTH**, **MINUTES_INC_OFFNET_WKD_1MONTH**, **AVG_MINUTES_INC_OFFNET_1MONTH**, **MINUTES_INC_ONNET_WKD_1MONTH** and **AVG_MINUTES_INC_ONNET_1MONTH**. All related to the average duration of the calls and total number of minutes in incoming calls, from any kind of subscriber (onnet or offnet), received in any day of the week over the last month. This conclusion is consequent with those findings in the descriptive section, where different heights in the densities of each variable were found for each kind of customer. On the other hand, variables that were also considered important in the descriptive section, such as Customer Lifetime Value (CLV) or any variable associated with complaints in the past, were not important in this model.

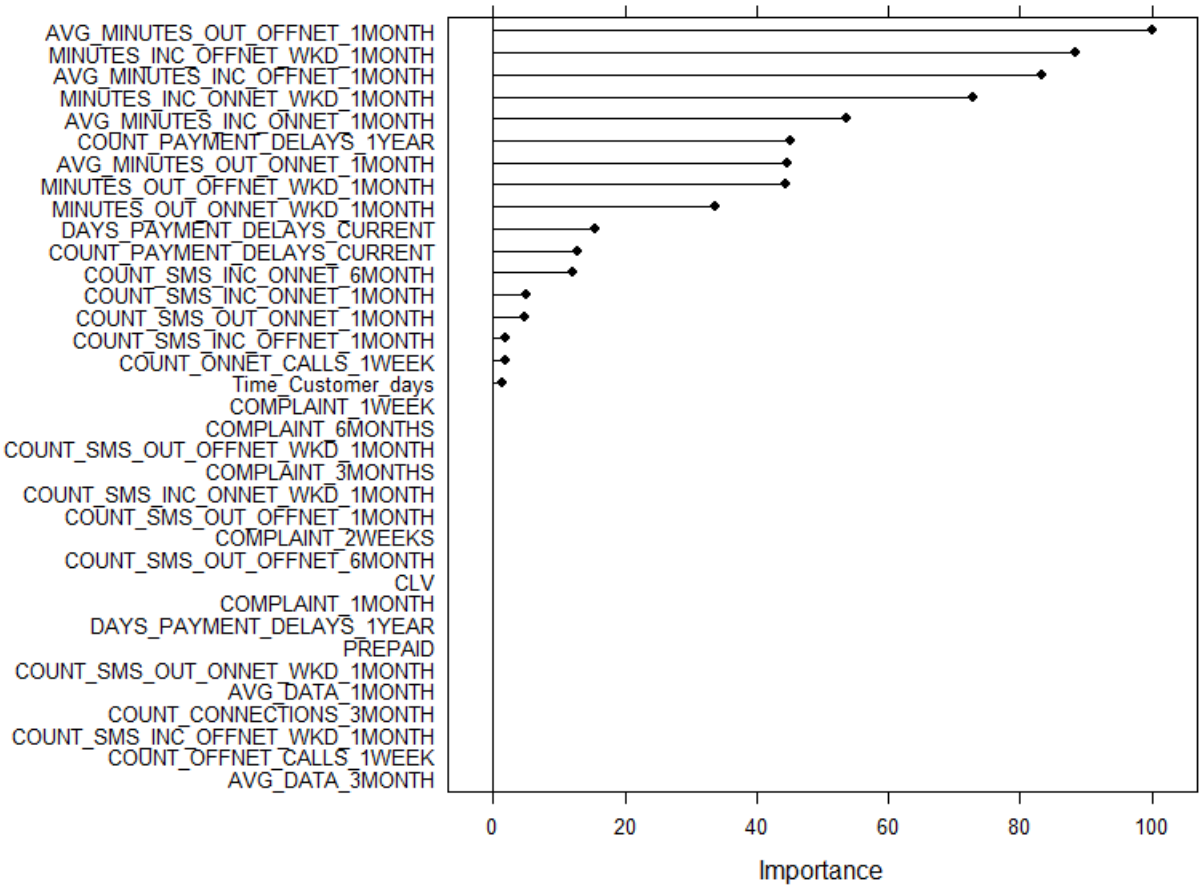


Figure 6: `rpart2` Variable importance

Now, in the sake of illustration, the final pruned decision tree is presented in Figure 7. Note that even though the plot does not look as well because of the length of the variable names, it can easily be interpreted. For instance, if the number of payment arrears over the past year is larger than 1.5, the customer will then be directly classified as a customer that leaves the company. On the other hand, if the number of payment arrears over the past year is smaller than 1.5, but the average duration of incoming and outgoing calls in the last month is larger than 0.34 and 0.51 minutes, respectively, the customer will be classified as customer that churns. This is a usual behaviour of those customers that want to spend all the credit they have, before leaving the company.

Random Forest

So far it has been shown that the decision tree outperforms the logistic regression model in terms of prediction of the response variable in the test set. The next question is whether it is possible to somehow combine different models to obtain better prediction performance. To tackle this problem, ensemble models were created under two different scenarios: bagging and boosting. The former consists in obtaining a large number of bootstrapped replicas of the training set, and training the same kind of model on all of these replicas. Then, the individual classifiers are combined by taking

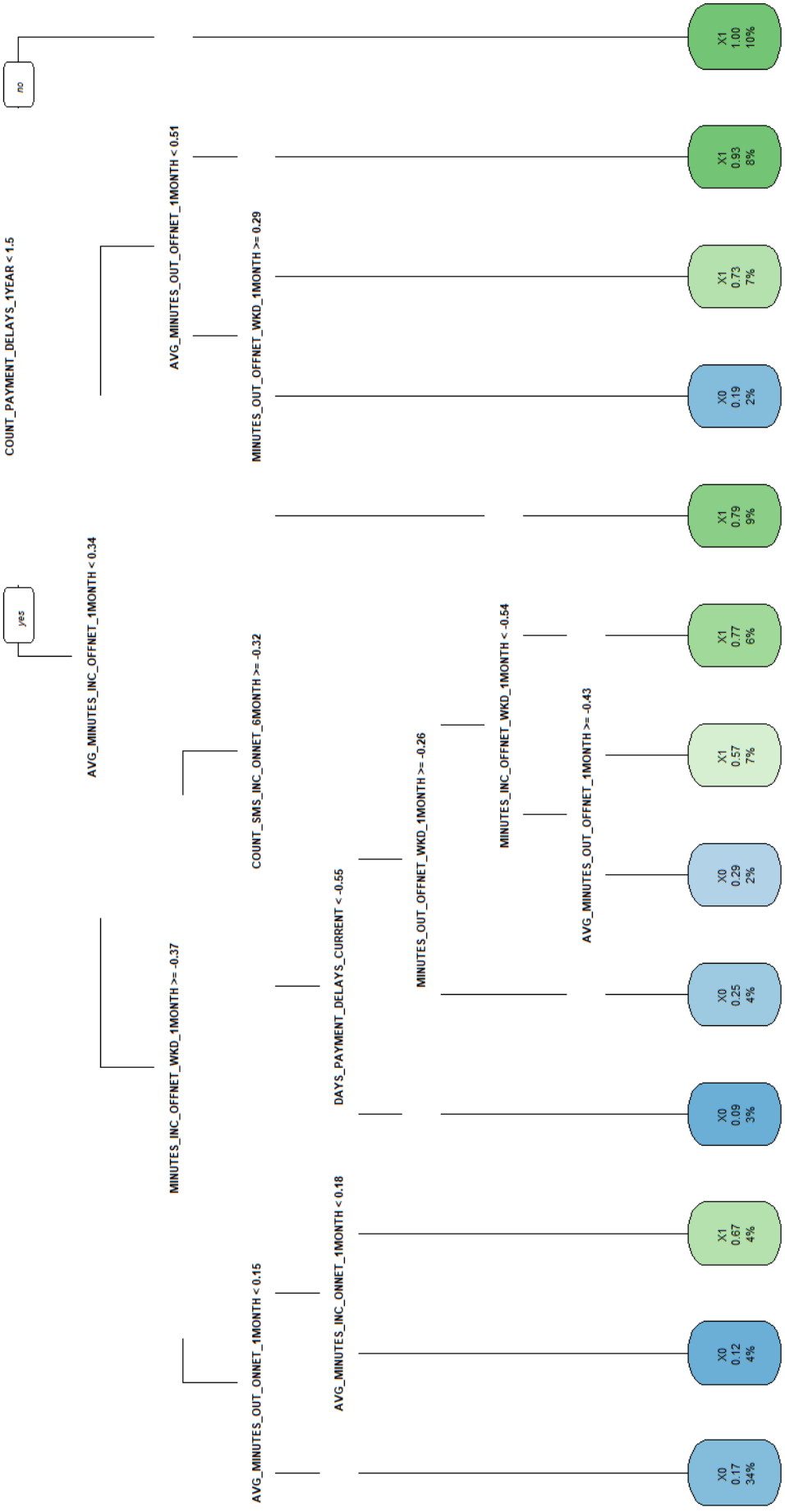


Figure 7: Pruned decision tree

a simple majority vote of their decisions. In a similar way, boosting creates different individual classifiers which are then combined by majority vote. The difference is that it is not applied on bootstrapped replicas but each classifier is added sequentially with the aim of correct the mistakes the current classifier may have.

The most powerful technique under the umbrella of bagging methods is called Random Forest. This consists in training a decision tree in each of the bootstraps replicas obtained from the training set. One special characteristic of this algorithm is that in each split in each classifier, the selection of the candidate attributes is conducted randomly. This means that there are two sources of randomness in the algorithm: one for the selection of bootstrap samples and other for the selection of variables in each split. As a consequence, it is a more powerful technique than a decision tree alone, in terms of prediction, and the problem of overfitting and pruning is put away. On the other hand, its main disadvantage is that it is not possible to interpret the final output because it is a combination of different trees.

To use this algorithm in `caret` package, `rf` method was used. The number of variables to select in each split was the only hyperparameter that needed to be tuned. In regard to the number of bootstrapped replicas and trees to train, the package set this number to 500. Although it is not clear how many trees should be fitted to obtain a good solution, empirical results has shown that 500 is enough. To select the optimal number of variables chosen in each split, 10 different possible values were used in a range between 2 and 35, and the selection of the best hyperparameter was based on the ROC curve using 10-fold cross-validation repeated three times as it was mentioned before.

Figure 8 shows the AUC for each random forest trained with different number of variables selected in each split. Using only two variables gave an AUC of almost 0.9915 in the training set, which was already better than any model fitted before. This area under the ROC curve was further improved when using five or nine predictors in each split. After this point, increasing the number of variables in each split decreases the performance up to a point that using all variables leads to worse performance that only using two of them. For this reason, the final number of variables to consider in each split was 5. A second random forest was also fitted by using the rule of thumb that the optimal number of variables used in each split is the square root of the number of features in the dataset, which in this case is 5.916. In the following output this model is labelled as *rf_thumb*.

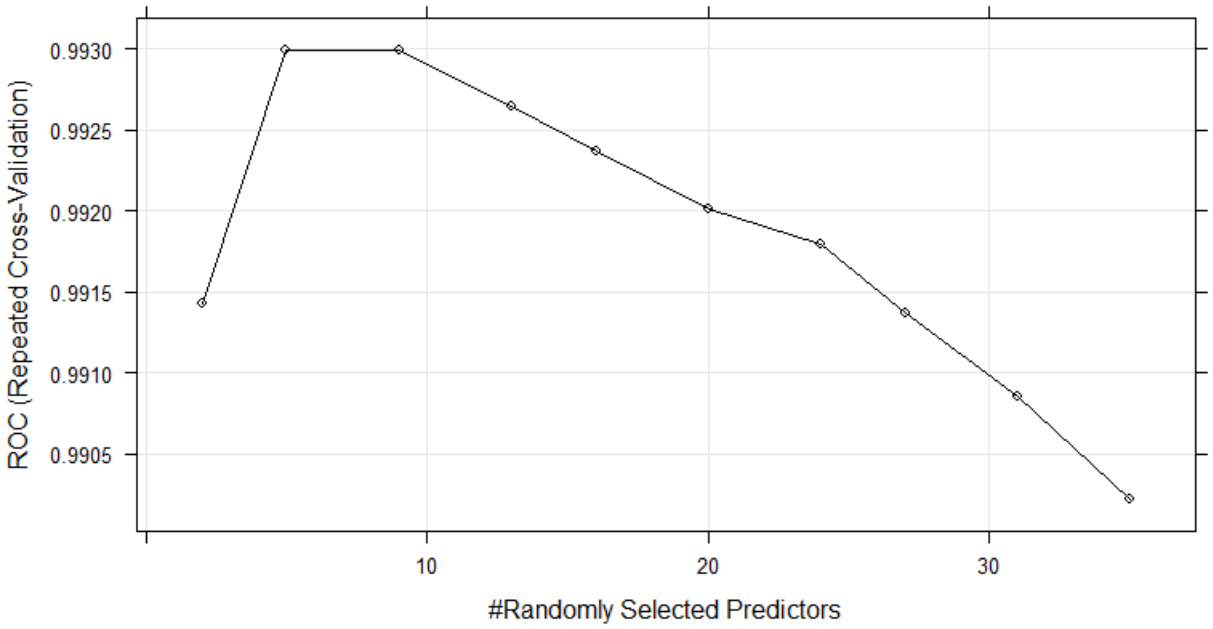


Figure 8: Tuning hyperparameter for Random Forest based on AUC

It is important to mention that in general ensemble models take more time to train because a large number of models are trained and different hyperparameters are also used to get an optimal solution. In this case, the training process of the first random forest took almost 20 minutes, which compared to any of the previous models presents an increase of processing time in orders of magnitude. Parallel computing was also used to train these models because the way this algorithm works allows the processor to use it: different trees can be trained simultaneously.

Now, to evaluate the predictive performance of these models in new datasets, different measures were used to compare them with special emphasis in the AUC, as it has been shown before. Table 5 shows the confusion matrix for each model using the best threshold. The AUC of both models was practically the same, differences appeared after third decimal, and it was better than the best models obtained before. The threshold that maximizes the sum of the true positive rate and true negative rate is 0.347 for the random forest that uses 5 variables in each split and 0.355 for the random forest that uses the rule of thumb. The second random forest was better that the other one in every single performance measure. This model has the ability to correctly predict 87.5% of customers who left the company and 88.5% of customers that are still in the company. Furthermore, the improvement of this model with respect to the best decision tree is due to the prediction of those customers that are still in the company because this model predicted correctly 81 customers more. Figure 9 shows the corresponding ROC curves.

Model	AUC	Thres.	TP	FP	TN	FN	Acc.	Prec.	Sens.	Spec.
rf	0.944	0.347	257	83	621	39	0.878	0.756	0.868	0.882
rf_thumb	0.946	0.355	259	81	623	37	0.882	0.762	0.875	0.885

Table 5: Random forest. Confusion matrix and performance measures

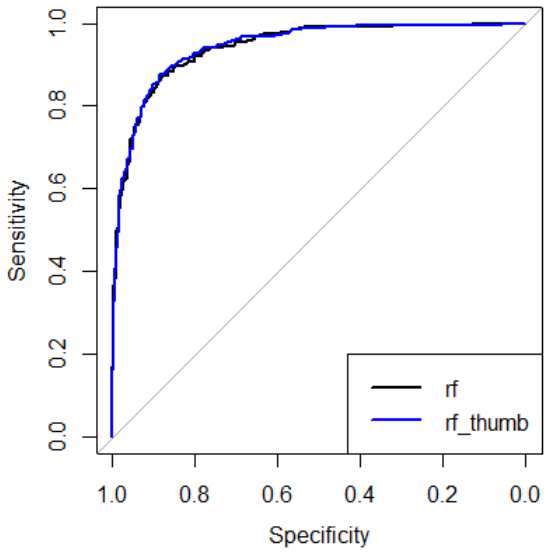


Figure 9: Random forest. ROC curve

Having these results, it was also of interest to know which variables were important in the training process and even more interesting to know what variables helped the model to predict with higher accuracy those customers that are still in the company. To do so, ROC curve analysis was conducted in each predictor as it was explained before. Figure 10 shows the score associated to each predictor.

The first difference that can be notice is the fact that now all variables seem to be important in the training but one, **COMPLAINT_1MONTH**. The top 10 of the variables more important here is similar to the top 10 of the decision tree. On the other hand, variables such as the number of days that the customer has been with the company (**Time_Customer_days**) and Customer Lifetime Value (**CLV**) are now more important than before.

These differences are somewhat expected though, because the fact that 500 different decision trees are now used, increases the probability of having more variables in the final solution. Unfortunately, there is no way to know how each of these variables influences the final decision and this is one of the main drawbacks of this technique. However, given that now it is possible to know what variables are important to the classification of customers, another analysis can be performed with special focus on them.

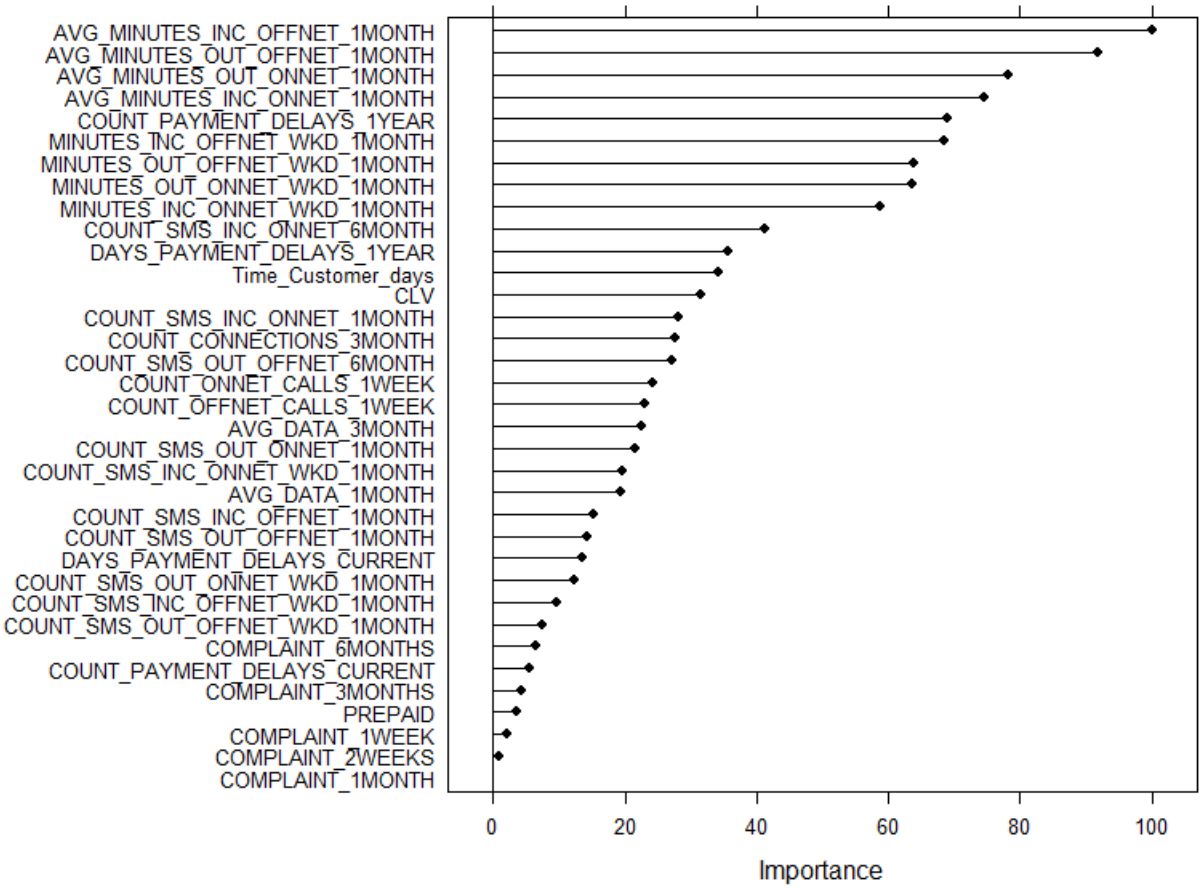


Figure 10: *rf_thumb* Variable importance

Adaboost

In the previous section ensemble models and the difference between bagging and boosting were mentioned. In this section, the idea of boosting is exploited using the Adaboost algorithm. This consists in training a "weak" learner, as a decision tree, over a number of rounds. In the beginning, every observation has the same weight and a decision tree (or several decision trees) is trained. In the second round, those observations that were misclassified are given a heavier weight and the classifier is trained again. This process continues until a good solution is obtained.

To use this algorithm in `caret` package, `adaboost` method was used. This method has two hyperparameters that need to be tuned: The number of trees and the algorithm to use, either *Real adaboost* or *Adaboost.M1*. To select the optimal combination of hyperparameters, a sequence from 50 to 500 increasing by 50 was used in the number of trees, and each possible number was used to train both algorithms. The selection of the best combination of hyperparameters was based on the ROC curve using 10-fold cross-validation repeated three times as it was mentioned before.

Figure 11 shows the AUC in the training set for each combination of hyperparameters. The first thing that can be noticed is the difference in performance between both algorithms, being *Adaboost.M1* the one that produces better results. The performance

of this algorithm slightly increases as the number of trees considered in the training increases as well. As a consequence, the final model is the one that has 500 trees and uses *Adaboost.M1* algorithm.

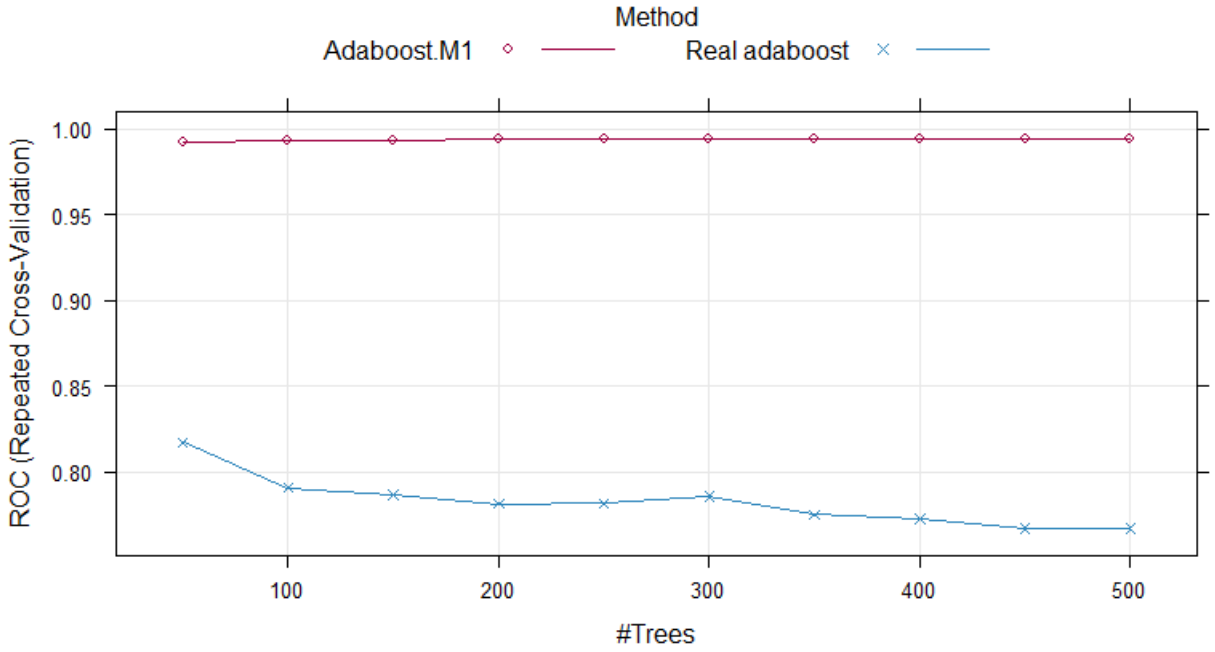


Figure 11: Tuning hyperparameters for Adaboost based on AUC

In regard to the processing time required to train this model, it is important to mention that it took around 14 hours to complete. There are a couple of reasons that justify this increment in time: the first one is how the algorithm works, because it is only possible to update instances' weights after fitting the classifier. As a consequence, the second reason is that parallel computed cannot be used in the same way as it happened in random forest. Here, parallel computing was only used to run repeated cross-validation in each core.

Now, to evaluate the predictive performance of this model in new datasets, different measures were used to compare it with previous models with special emphasis in the AUC, as it has been done before. Table 6 shows the confusion matrix for the model using the best threshold. The AUC is the larger obtained so far, being 0.951, although it is pretty close to the result obtained by the best random forest solution. The threshold that maximizes the sum of the true positive rate and true negative rate was 0.423. In regard to predictive performances, this model presents the best specificity when compared to any of the models shown before. 93.3% of the customers that are still in the company are predicted as such and 81.8% of the customers that left the company are correctly classified as such. Compared to the best solution obtained so far, *rf_thumb*, the potential of this model is the classification of those customers that are still in the company. Figure 12 shows the corresponding ROC curve.

Model	AUC	Thres.	TP	FP	TN	FN	Acc.	Prec.	Sens.	Spec.
Adaboost	0.951	0.423	242	47	657	54	0.899	0.837	0.818	0.933

Table 6: Adaboost. Confusion matrix and performance measures

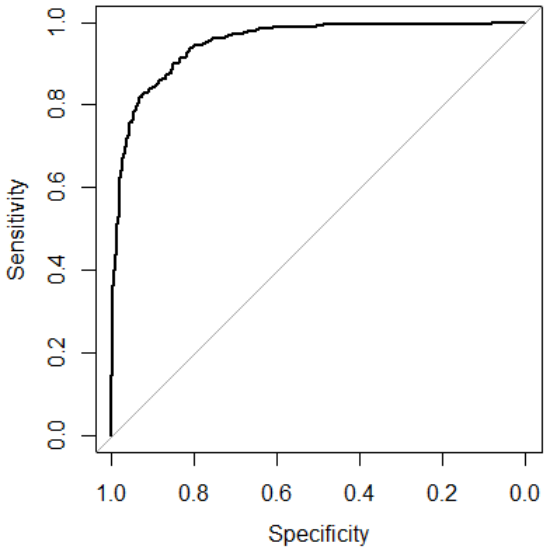


Figure 12: Adaboost. ROC curve

As before it was of interest to know which variables were important in the training process. Figure 13 shows the score associated to each predictor after the ROC curve analysis was conducted. The top 10 of the most important variables is a little bit different compared to previous models. In this case, those variables associated to the total consumption of minutes, either incoming or outgoing, made in the weekends in the last month were the most important ones. In contrast to any of the previous models trained, variables associated to SMS were important here. Additionally, variables as the number of days that the customer has been with the company or customer lifetime value were not important here.

As it was mentioned before, there is no way to know how each of these variables influences the final classification, but this results are important in the sense that variables associated to SMS consumption could lead to a better classification of customers that are still in the company.

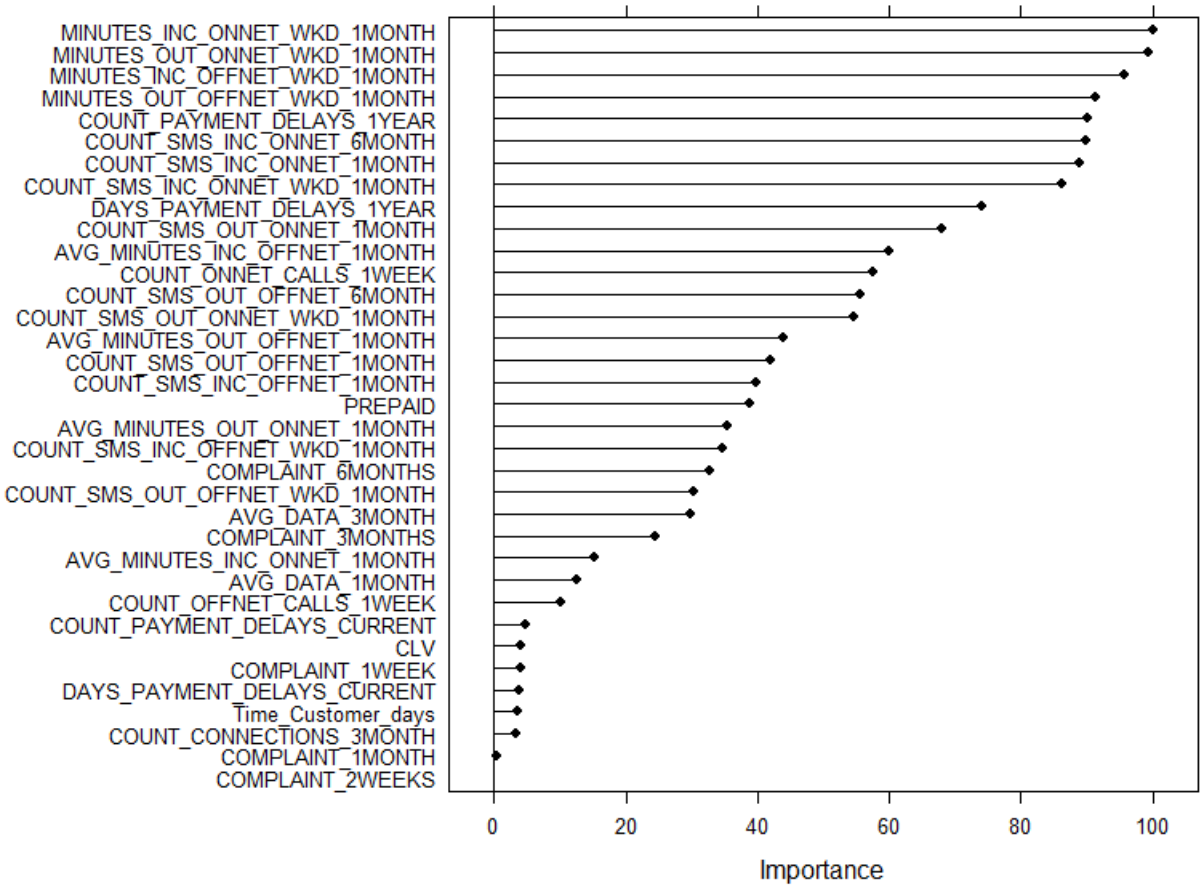


Figure 13: *Adaboost* Variable importance

Extreme Gradient Boosting

Another algorithm under the umbrella of boosting models is the so-called Extreme Gradient Boosting. This is natural extension of the framework in which Adaboost was developed, where a exponential loss function is optimized. In this case, different loss functions can be chosen in the training process, producing better results that any other technique. However, one of the big flaws is again the risk of obtaining overfitted solutions. For this reason, several hyperparameters need to be tuned to improve generalization.

To use this algorithm in `caret` package, `xgbDART` method was used. In contrast to any other method shown before, this method has nine hyperparameters that need to be tuned: number of boosting iterations, max tree depth, shrinkage, minimum loss reduction, subsample percentage, subsample ration of columns, fraction of trees dropped, probability of skipping drop-out and minimum sum of instance weight. Therefore, in order to get an optimal solution a grid search was conducted by varying one of the hyperparameter and leaving the remaining ones fixed. The selection of the best combination of hyperparameters was based on the area under the ROC curve using 10-fold cross-validation repeated three times as it was mentioned before.

Figure 14 shows the AUC for a small sample of the grid search considered in the training process. The first thing that can be noticed is that no matter what combination of hyperparameters is consider in the training, a max depth of 5 always produces better

results. In addition to this, any other combination of hyperparameters results in similar performance in the training set. Since the search of the optimal model is done by the AUC, not visually, it is concluded that the best possible combination of hyperparameters is shown in the plot located in the upper-right corner.

Regarding processing time required to train this model, it is important to notice that given the complexity of this algorithm, a small grid search of hyperparameters was conducted. The total processing time was around 9 hours but better results could have been achieved if the grid search had been more exhaustive. As it was mentioned in Adaboost, parallel computing was used to run repeated cross-validation in different cores but it was not possible to split more the work inside the algorithm because one iteration depends on the iteration before.

With respect to the evaluation of the predictive performance of this model in new datasets, different measures were used to compare it with previous models with special emphasis in the AUC, as it has been done before. Table 7 shows the confusion matrix for the model using the best threshold. The AUC is a little bit smaller than the Adaboost algorithm and a little bit larger than random forest, 0.948. The threshold that maximizes the sum of the true positive rate and true negative rate is 0.108. In regard to predictive performance, this model presents the best sensitivity when compared to any of the models shown before, including Adaboost. 92.2% of the customers that left the company are correctly classified as such. Until now the best sensitivity was produced by the random forest, *rf_thumb*, which presents a sensitivity of 87.5%. Figure 15 shows the corresponding ROC curve.

Model	AUC	Thres.	TP	FP	TN	FN	Acc.	Prec.	Sens.	Spec.
XGB	0.948	0.108	273	119	585	23	0.858	0.696	0.922	0.831

Table 7: Extreme Gradient Boosting. Confusion matrix and performance measures

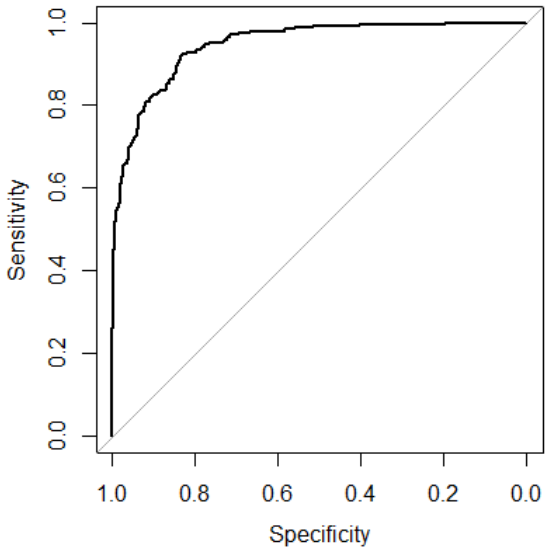


Figure 15: Extreme Gradient Boosting. ROC curve

As before it was mentioned before, it was of interest to know which variables were

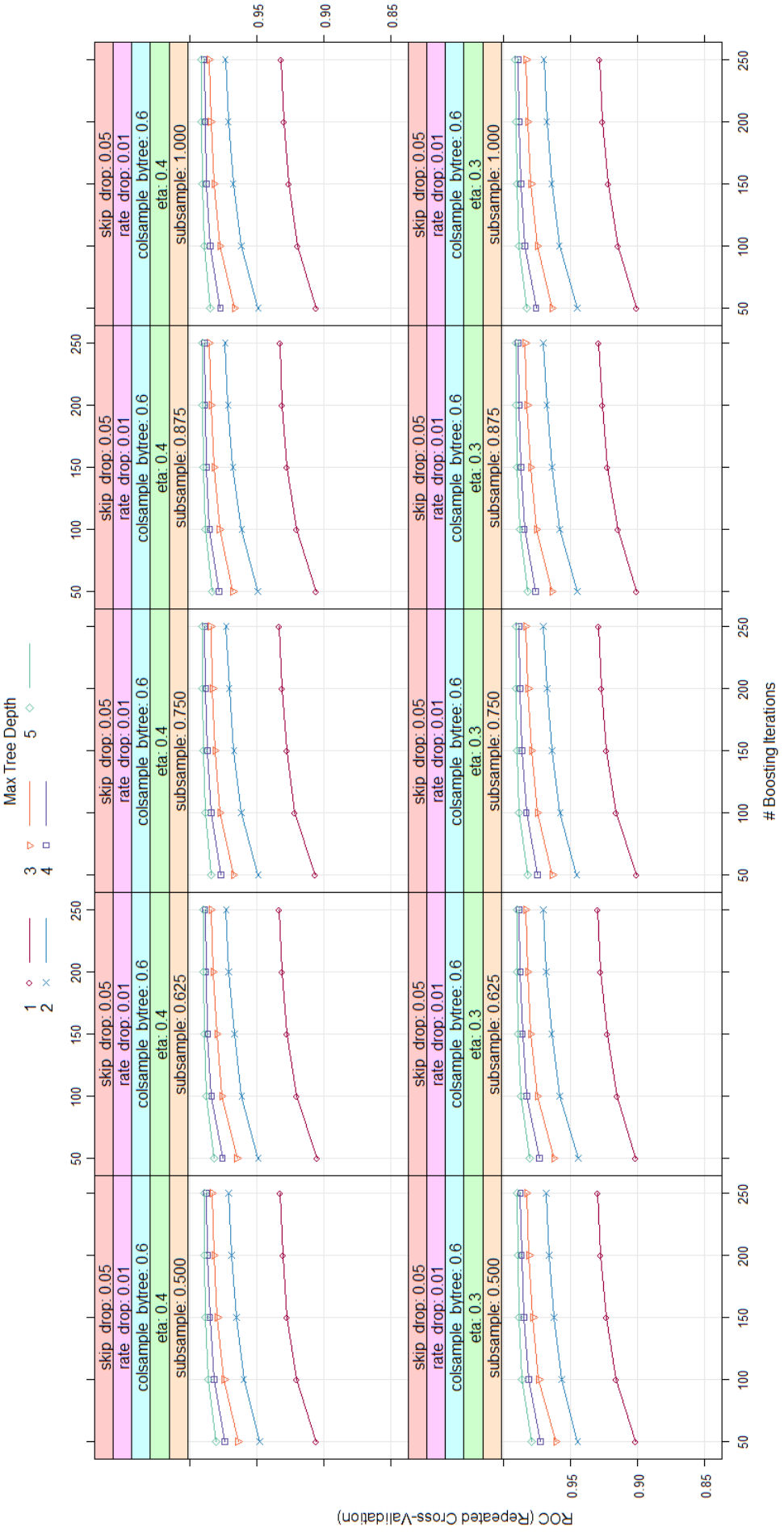


Figure 14: Tuning hyperparameters for Extreme Gradient Boosting based on AUC

important in the training process and even more in this model because of its predictive performance. Figure 16 shows the score associated to each predictor after the ROC curve analysis was conducted. The top 10 of the most important variables is similar to the top 10 from the random forest. All variables related to the average duration of the calls and total number of minutes in incoming and outgoing calls, regardless the kind of subscriber and the day in the week, over the last month were important. Also the number of payment arrears over the past year and the number of days the customer has been in the company were important to predict whether a customer will leave the company or not. Unfortunately, there is no way to know with certainty how each of these variables influences the final decision but it is possible to use them later in another analysis to characterize the profile of those customers that left the company.

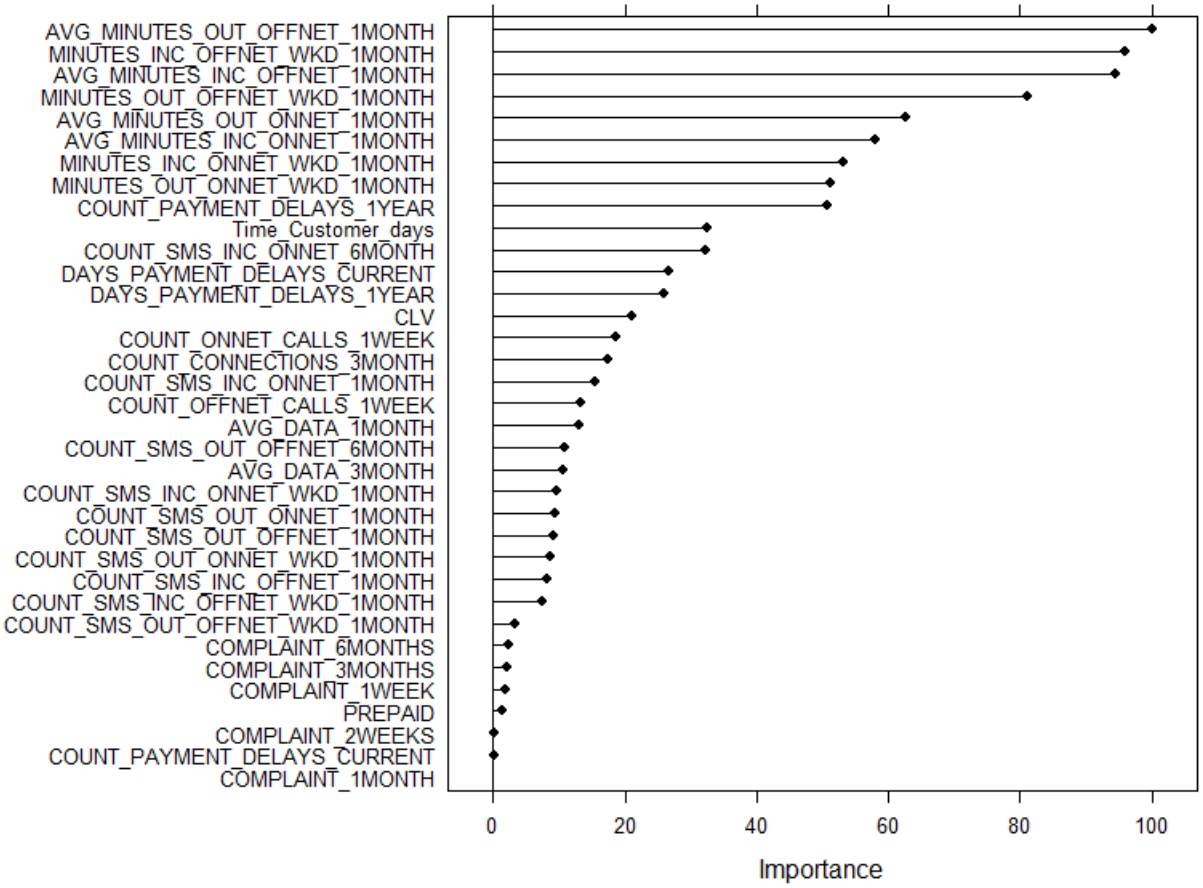


Figure 16: Extreme Gradient Boosting. Variable importance

Neural Network

As a final attempt to obtain a model that predicts with confidence whether a customer leaves the company or not, a neural network was trained. This is a complete different model that is very powerful in classification and regression which allows to estimate with precision the target variable. Its architecture can be extended to any kind of problem by manipulating the number of hidden layers, the number of hidden units and the

activation function in each of this neurons. However, its power in real life settings comes with a cost that is related to non-convex optimization problems. This means that it suffers a lot of local minimum solutions which in the end may affect the overall performance of the model. Overfitting is another problem that usually arises when the network is too complex for a given dataset, so regularization has to be taken into account in the training process.

To use this algorithm in `caret` package, `nnet` method was used. This method has two hyperparameters that needed to be tuned: the number of hidden neurons and the regularization parameter. It is important to note that this architecture only considers one hidden layer, but this can be further extended using other methods inside the package. Note that the activation function used in the output neuron is the logistic function. To select the optimal combination of hyperparameter, a grid search was created by using a sequence between 1 and 99 for the number of neurons and a sequence between 0 and 0.1 for the regularization parameter. The selection of the best combination was based on the area under the ROC curve using 10-fold cross-validation repeated three times as it was mentioned before.

Figure 17 shows the AUC in the training set for each combination of hyperparameters. The first thing that can be noticed is that the AUC increases as the number of neuron increases up to a point, regardless the regularization parameter. When the net has more than 27 hidden neurons there is a drop in the AUC and the model is not useful anymore. As a consequence the best model is the one that has 27 hidden neurons and a regularization constant equal to 0.1.

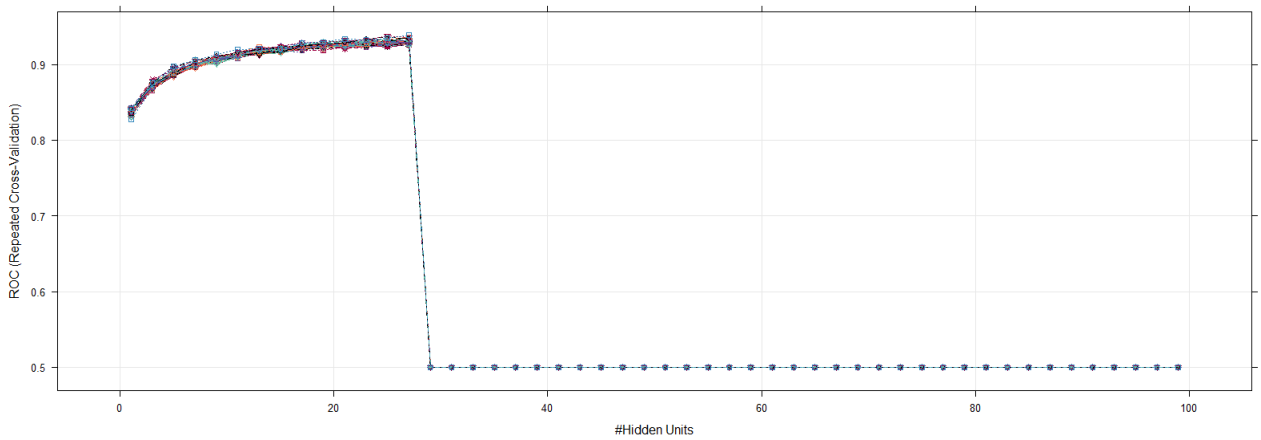


Figure 17: Tuning hyperparameters for Neural net based on AUC

Now, to evaluate the predictive performance of this model in new datasets, different measures were used to compare it with previous models with special emphasis in the AUC, as it has been shown before. Table 8 shows the confusion matrix for the model using the best threshold. The AUC is lower than it was expected, 0.862, and it is similar to the AUC obtained with the decision tree. The threshold that maximizes the sum of the true positive rate and true negative rate is 0.201. With respect to predictive performance, it is not as good as any other model considered before, particularly the sensitivity where it is only 80.4%. Figure 18 shows the corresponding ROC curve.

Model	AUC	Thres.	TP	FP	TN	FN	Acc.	Prec.	Sens.	Spec.
NN	0.862	0.201	238	159	545	58	0.783	0.599	0.804	0.774

Table 8: Neural network. Confusion matrix and performance measures

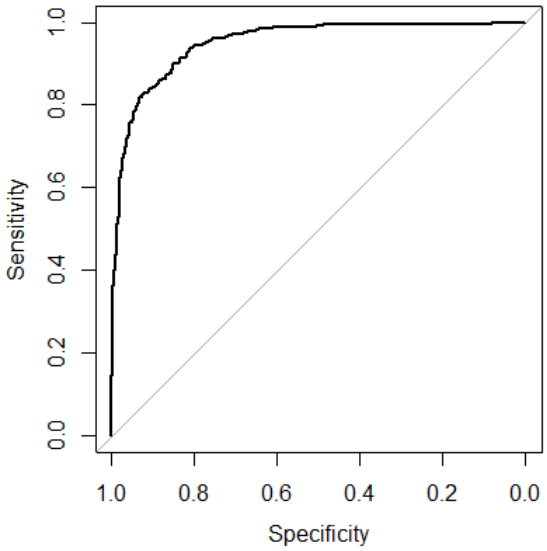


Figure 18: Neural network. ROC curve

In regard to the importance of the variables in the training process of the neural network, Figure 19 shows the score associated to each predictor after the ROC curve analysis was conducted. Apparently, all variables were important to the model with special focus in those related to the consumption of minutes and SMS incoming and outgoing in any day of the week in the last month. However, this results may not be as important because of the predictive performance of the neural net.

What it is important to mention though, is that building a neural network is a complete journey by itself, because different aspects have to be taken into account in the training process. Here, only the number of hidden neurons were taken into account but several other characteristics can be changed, such as the selection of cost function, the number of hidden layer or the selection of activation functions. If all of these aspects had been taken into account in the training process, the results would have improved a lot.

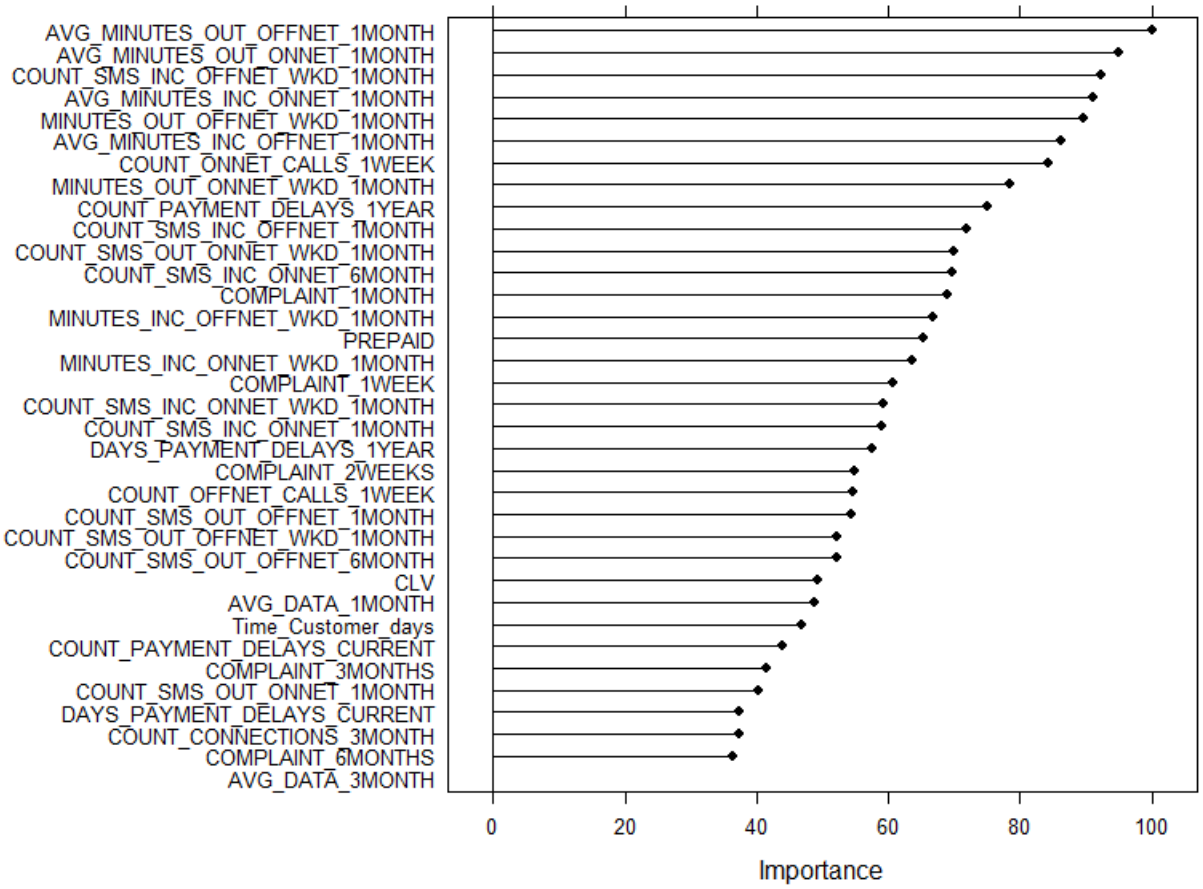


Figure 19: Neural network. Variable importance

Discussion of results on a test set and conclusion

Until this point, the evaluation of the performance of each model has been done in terms of the test set that was randomly selected from the training set in the beginning of the exercise, prior to the preprocessing. In this section, the predictive performance of some of these models is tested in the additional test set that was available to compare the results with other teams.

Table 9 shows the area under the ROC curve for each model on the test set taken from the initial training set (first column), the AUC on 50% of the test set that was public to all teams (second column), and the private AUC, which corresponds to the performance measure in the remain 50% on this test set. Only few models trained in this section were tested in the leaderboard. This is the reason why there are missing values in the second and third column.

Model	AUC	Public AUC	Hidden AUC
Adaboost	0.951	0.925	0.9374
XGB	0.948	0.937	–
rf_thumb	0.946	0.924	–
rf	0.944	–	–
rpart	0.863	–	–
rpart2	0.863	–	–
NN	0.862	0.843	–
Logit	0.83	–	–
Lasso	0.83	–	–
Elastic net	0.83	–	–
Ridge	0.829	–	–

Table 9: Performance measure of each model in different test sets

This table is ordered decreasingly by the AUC of the first column, so it is easy to see that ensemble models were the most powerful. In particular, boosting techniques outperformed any other model in both test sets, although the difference with respect to bagging techniques is not significant. On the other hand, the performance of those models that are easy to interpret, such logistic regression and decision trees, was not as good. In terms of performance with respect to the second test set, only ensemble methods and the neural network were tried. What is important to note is the fact that the private AUC did not decrease but increased when compared with the public AUC. Other teams got higher AUC values,

Regarding the comparison with other teams, some of them had AUC values of 0.98 and 0.99, whereas this team had a value around 0.93. This difference can be explained by the use of the variable **FIN_STATE** while training the models. As it was mentioned in the beginning, this variable was left out of the analysis because most of the customers did not have any value in it. Making a contingency table between this variable and the response shows that all customers that are still in the company also have missing values in **FIN_STATE**, so there is a strong correlation between this covariate and the response. This is why this variable leads to a model that perfectly predicts when a customer will churn. What it is important to judge is the reason of this missing values and its relationship with the response. Otherwise no model would be needed to predict customers that leave the company because this could be predicted by just having a look at **FIN_STATE**.

3 Assignment 3: Sentiment Analysis

Introduction

This assignment focuses on a non-static (streaming) data set consisting of book reviews on Amazon being received in real time. The main objective of this assignment is to built a model based on textual data that is able to predict in real time the number of stars given in a review. Three different pieces of work are needed to accomplish this task successfully: First, build a historical dataset by setting up a streaming environment using `Spark` so data can be received on local machines. Second, employ text mining techniques in the data to create features that can be used in a model to make the corresponding predictions. Third, develop a model that predicts the number of stars and use it to predict new titles in a streaming setup. In the following, these steps are described with detail.

Data Streaming

The data is obtained from a streaming server seppe.net (port 7778) using `Pyspark` and `Jupiter` notebook. The script provided creates a folder every 10 seconds on the local machine with the corresponding reviews that are received in that lapse of time. The majority of the generated folders do not contain a review (rather, a few reviews per hour are received), so the initial task was to collect the actual review files, convert them into a readable format (i.e. `.csv`) and merge them into a static data frame which is to be used to train and test the model. This process was automatized using an `R` script which deletes the empty folders and gathers the target files to generate the data frame (see Appendix section). Since each team member collected data independently, all reviews were merged into a single csv and duplicates were removed. The first 5 rows of the final dataset are shown as follows:

	X	book_title	review_title	review_user	book_id	review_id	timestamp	review_text	review_score
0	1	A Gentleman in Moscow: A Novel	Russian aristocracy following the Russian Revo...	Kansabelle	0143110438	R2UFCQ9WES7VFH	1555241537	A great read. In addition to the plot, charact...	4
1	2	A Gentleman in Moscow: A Novel	Knowing nothing ahead of time just walk into t...	D.P. McHenry	0143110438	R24B1HA9J9I99G	1555241542	Great story, well told. Characters were well d...	4
2	3	Pet Sematary: A Novel	One of King's finest, most frightening, & earl...	Gordon Hoffman	198211598X	R1P137WFADSBYR	1555241649	Only the second novel written and published un...	4
3	4	Less (Winner of the Pulitzer Prize): A Novel	Not my favorite	R. Zocher	0316316121	R35533AKR5CBNS	1555242044	This book is t what I expected. Story is a lit...	4
4	5	Supermarket	AMAZING, BOBBY	D. Mahoney	1982127139	R1D6LXSDR0CRMN	1555242169	As someone who hates reading and never really ...	4

Figure 20: Dataset Preview

Offline Training

Data Pre-processing

A number of 8559 reviews were collected over the span of several weeks by all team members. From this set of reviews, 4320 duplicates were removed prior to the analysis as multiple instances of training examples will bias the classifier and prediction accuracy will be inaccurately reported. The total count after removing duplicates is 4239 reviews. Table 10 shows the frequency of each rating class, where it can be noted that in the existing sample, the majority of reviews (3236) have a score of 5 out of 5 stars. At this point, one review did not have a score, so it was removed from the analysis. As a result, the final dataset has 4238 reviews.

Review Score	Count
5	3236
4	598
3	187
2	110
1	108

Table 10: Review Frequencies

The first challenge encountered was about reading the data properly in `Python` by using `Pyspark` library. In turns out that the default options of `SQLContext` reading function, could not deal with some reviews that had commas or line breaks. For this reason, some of the arguments of the read function were tweaked. Figure 21 shows how the arguments were setup to get the reviews in a proper order.

```
from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)
data = sqlContext.read.format('com.databricks.spark.csv').options(header='true',
                                                                  inferschema='true',
                                                                  quote = '"',
                                                                  escape = '"',
                                                                 multiline = 'true',
                                                                 ignoreTrailingWhiteSpace = 'true').load('Data\\data.csv')
```

Figure 21: Arguments of `SQLcontext` reading function

In order to be able to analyse both title and review data simultaneously, `review_title` and `review_text` were concatenated into a single column called `text`. The rationale behind this decision is that not just the review text, but also the review title are informative when predicting the rating and thus can be used in the analysis. Figure 22 illustrates how this new column looks in the dataset.

	X	book_title	review_title	review_user	book_id	review_id	timestamp	review_text	review_score	text
0	1	A Gentleman in Moscow: A Novel	Russian aristocracy following the Russian Revolution	Kansabelle	0143110438	R2UFCQ9WES7VFH	1555241537	A great read. In addition to the plot, characters, hidden treasure etc. I enjoyed the philosophical question: do Russians destroy more of what they create more than other cultures? Even if you stick to the characters-great fun.	4	Russian aristocracy following the Russian Revolution A great read. In addition to the plot, characters, hidden treasure etc. I enjoyed the philosophical question: do Russians destroy more of what they create more than other cultures? Even if you stick to the characters-great fun.
1	2	A Gentleman in Moscow: A Novel	Knowing nothing ahead of time just walk into the story without expectations and become charmed.	D.P. McHenry	0143110438	R24B1HA9J9I99G	1555241542	Great story, well told. Characters were well developed. Ending was perfect!	4	Knowing nothing ahead of time just walk into the story without expectations and become charmed. Great story, well told. Characters were well developed. Ending was perfect!

Figure 22: Dataset with Merged Review Title and Text

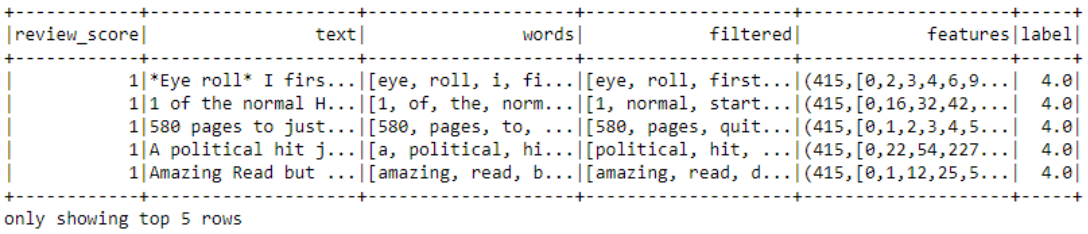
Now, for the analysis only the `review_score` and `text` columns were retained, and the other columns were removed from the dataset. This dataset was then split into a training set (80%) and a test set (20%) by randomly selecting instances for each subset. This was done proportionally to the rating distribution across the sample, i.e., the proportion of reviews in each class was maintained after the data was split. The resulting number of reviews are 3333 for the training set and 905 for the test set.

Next, a pre-processing pipeline was created using `RegexTokenizer`, `StopWordsRemover`, `CountVectorizer` and `StringIndexer`. `RegexTokenizer` was used to split sentences into words or tokens. `StopWordRemover` was used to remove commonly used words that have no meaning on their own and are likely to appear in most sentences, such as *and*, *if*, *the*, *or*, *because*, among others. `CountVectorizer` was used to convert the collection of reviews to vectors of token counts, also known as bag of words approach. Finally, `StringIndexer` was used to encode the target variable from a string column to a column of indices. Table 11 shows how this encoding was performed.

Original Encoding	New Encoding	Count Training	Count Test
5	0.0	2543	692
4	1.0	466	132
3	2.0	146	41
2	3.0	90	22
1	4.0	88	18

Table 11: Encoding for Rating and Frequencies in Training/Test Datasets

This pre-processing pipeline was fitted in the training set and then applied to the test set. The result for 5 data points is shown in the following table. The sentences have been split into words (”words” column) and the stop words have been removed (”filtered” column), whilst the ”features” column describes the obtained featurization. Finally, ”label” is the encoded column of `review_score`.



review_score	text	words	filtered	features	label
1	*Eye roll* I firs...	[eye, roll, i, fi...	[eye, roll, first...	(415,[0,2,3,4,6,9...	4.0
1	1 of the normal H...	[1, of, the, norm...	[1, normal, start...	(415,[0,16,32,42,...	4.0
1	580 pages to just...	[580, pages, to, ...	[580, pages, quit...	(415,[0,1,2,3,4,5...	4.0
1	A political hit j...	[a, political, hi...	[political, hit, ...	(415,[0,22,54,227...	4.0
1	Amazing Read but ...	[amazing, read, b...	[amazing, read, d...	(415,[0,1,12,25,5...	4.0

only showing top 5 rows

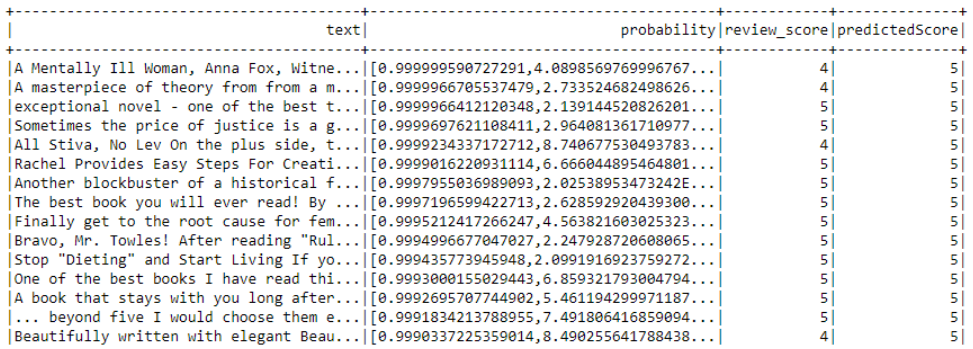
Figure 23: Bag of Words Featurization - Training set

Modelling

Multinomial Logistic Regression Using Count Vectors

In order to predict probabilities of an outcome having more than 2 classes (in this case, a rating of 1, 2, 3, 4 or 5 stars), a multinomial logistic regression model based on count vectors (bag of words approach) was used. Since this kind of model can be penalized by using a regularization constant and also modified by changing the loss function (elasticNet), a grid search via 5-fold cross-validation was performed. The regularization constant was tuned in the range of 0.01 to 0.5, and the elastic net parameter was tuned in the range of 0 to 1, where 0 represents ridge and 1 lasso. As a result, the model that provides the best performance uses a rigde loss function and has a regularization constant equals to 0.01.

The obtained probabilies and predicted ratings in the test set are shown in Figure 24. When choosing 15 reviews at random, the model correctly predicts the rating for 11. In fact, the prediction accuracy of this model on the test set is 0.721.



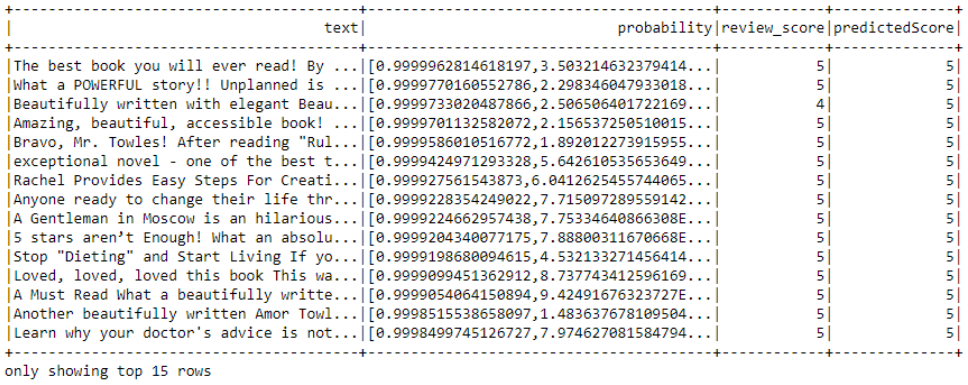
text	probability	review_score	predictedScore
A Mentally Ill Woman, Anna Fox, Witne...	[0.999999590727291,4.0898569769996767...	4	5
A masterpiece of theory from from a m...	[0.9999966705537479,2.733524682498626...	4	5
exceptional novel - one of the best t...	[0.9999966412120348,2.139144520826201...	5	5
Sometimes the price of justice is a g...	[0.9999697621108411,2.964081361710977...	5	5
All Stiva, No Lev On the plus side, t...	[0.9999234337172712,8.740677530493783...	4	5
Rachel Provides Easy Steps For Creati...	[0.9999016220931114,6.666044895464801...	5	5
Another blockbuster of a historical f...	[0.9997955036989093,2.02538953473242E...	5	5
The best book you will ever read! By ...	[0.9997196599422713,2.628592920439300...	5	5
Finally get to the root cause for fem...	[0.9995212417266247,4.563821603025323...	5	5
Bravo, Mr. Towles! After reading "Rul...	[0.9994996677047027,2.247928720608065...	5	5
Stop "Dieting" and Start Living If yo...	[0.9994357739459448,2.0991916923759272...	5	5
One of the best books I have read thi...	[0.9993000155029443,6.859321793004794...	5	5
A book that stays with you long after...	[0.9992695707744902,5.461194299971187...	5	5
... beyond five I would choose them e...	[0.9991834213788955,7.491806416859094...	5	5
Beautifully written with elegant Beau...	[0.9990337225359014,8.490255641788438...	4	5

Figure 24: Multinomial Logistic Regression - Results

Naive Bayes Classifier

The Naive Bayes classifier is a simple probabilistic classifier which uses Bayes' theorem. The classifier assumes feature independence, which means that the probability that a couple of words appear simultaneously in a review is equal to the product of the probability that each word appears in a review. In this case, this model was used by setting the smoothing parameter to one, with the purpose of avoiding numerical problems when the probability of a word in the test set is zero, because it did not appear in

the training set. The prediction accuracy of this model on the test set is 0.732. Figure 25 shows 15 random reviews and the model correctly predicts the rating in 14 of them.



text	probability	review_score	predictedScore
The best book you will ever read! By ...	[0.9999962814618197,3.503214632379414...	5	5
What a POWERFUL story!! Unplanned is ...	[0.9999770160552786,2.298346047933018...	5	5
Beautifully written with elegant Beau...	[0.9999733020487866,2.506506401722169...	4	5
Amazing, beautiful, accessible book! ...	[0.9999701132582072,2.156537250510015...	5	5
Bravo, Mr. Towles! After reading "Rul...	[0.9999586010516772,1.892012273915955...	5	5
exceptional novel - one of the best t...	[0.9999424971293328,5.642610535653649...	5	5
Rachel Provides Easy Steps For Creati...	[0.999927561543873,6.0412625455744065...	5	5
Anyone ready to change their life thr...	[0.9999228354249022,7.715097289559142...	5	5
A Gentleman in Moscow is an hilarious...	[0.9999224662957438,7.753346408663 08E...	5	5
5 stars aren't Enough! What an absolu...	[0.9999204340077175,7.888003116706 68E...	5	5
Stop "Dieting" and Start Living If yo...	[0.9999198680094615,4.532133271456414...	5	5
Loved, loved, loved this book This wa...	[0.9999099451362912,8.737743412596169...	5	5
A Must Read What a beautifully writte...	[0.9999054064150894,9.42491676323727E...	5	5
Another beautifully written Amor Towl...	[0.9998515538658097,1.483637678109504...	5	5
Learn why your doctor's advice is not...	[0.9998499745126727,7.974627081584794...	5	5

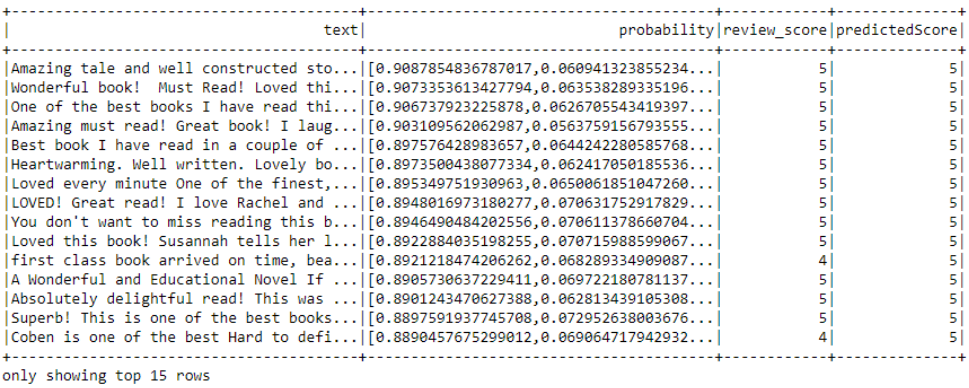
only showing top 15 rows

Figure 25: Naive Bayes - Results

Random Forest

Although it is known that Random Forests is not the best choice in problems with high-dimensional sparse data, it was considered to compare its accuracy with previous models. Since this kind of models depends on the number of trees and the maximum depth to get an optimal solution, a grid search via 5-fold cross-validation was performed. The number of trees was tuned in the range of 100 to 500, and the maximum depth was tuned in the range of 4 to 20. As a result, the model that provides the best performance has 500 trees and a maximum depth of 20.

The obtained probabilities and predicted ratings in the test set are shown in Figure 26. When choosing 15 reviews at random, the model correctly predicts the rating for 14. In fact, the prediction accuracy of this model on the test set is 0.67



text	probability	review_score	predictedScore
Amazing tale and well constructed sto...	[0.9087854836787017,0.060941323855234...	5	5
Wonderful book! Must Read! Loved thi...	[0.9073353613427794,0.063538289335196...	5	5
One of the best books I have read thi...	[0.906737923225878,0.0626705543419397...	5	5
Amazing must read! Great book! I laug...	[0.903109562062987,0.0563759156793555...	5	5
Best book I have read in a couple of ...	[0.897576428983657,0.0644242280585768...	5	5
Heartwarming. Well written. Lovely bo...	[0.8973500438077334,0.062417050185536...	5	5
Loved every minute One of the finest,...	[0.895349751930963,0.0650061851047260...	5	5
LOVED! Great read! I love Rachel and ...	[0.8948016973180277,0.070631752917829...	5	5
You don't want to miss reading this b...	[0.8946490484202556,0.070611378660704...	5	5
Loved this book! Susannah tells her l...	[0.8922884035198255,0.070715988599067...	5	5
first class book arrived on time, bea...	[0.8921218474206262,0.068289334909087...	4	5
A Wonderful and Educational Novel If ...	[0.8905730637229411,0.069722180781137...	5	5
Absolutely delightful read! This was ...	[0.8901243470627388,0.062813439105308...	5	5
Superb! This is one of the best books...	[0.8897591937745708,0.072952638003676...	5	5
Coben is one of the best Hard to defi...	[0.8890457675299012,0.069064717942932...	4	5

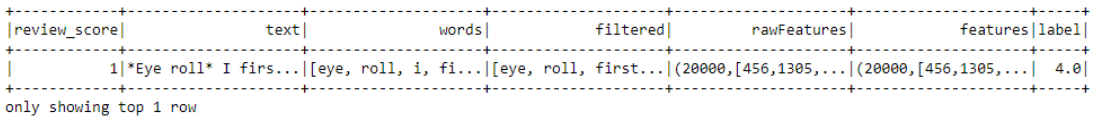
only showing top 15 rows

Figure 26: Random Forest - Results

TF-IDF: Multinomial Logistic Regression, Naive Bayes Classifier, Random Forest

A TF-IDF pipeline was created as a featurization alternative to the initial pipeline which employed `CountVectorizer`. The new pipeline preserves the `RegexTokenizer`, `StopWordsRemover`

and `StringIndexer`, as it was mentioned before, and additionally uses `HashingTF` function to make the feature extraction and `IDF` to remove sparse terms. In contrast to Bag of Words approach, the TF-IDF methodology employs a weighting factor to determine the importance of words. These weights are proportional to the number of appearances of a word in the document. Figure 27 shows the featurization results for one data point in the training dataset.



review_score	text	words	filtered	rawFeatures	features	label
1	*Eye roll* I firs...	[eye, roll, i, fi...	[eye, roll, first...	(20000,[456,1305,...	(20000,[456,1305,...	4.0

only showing top 1 row

Figure 27: TF-IDF Featurization - Training

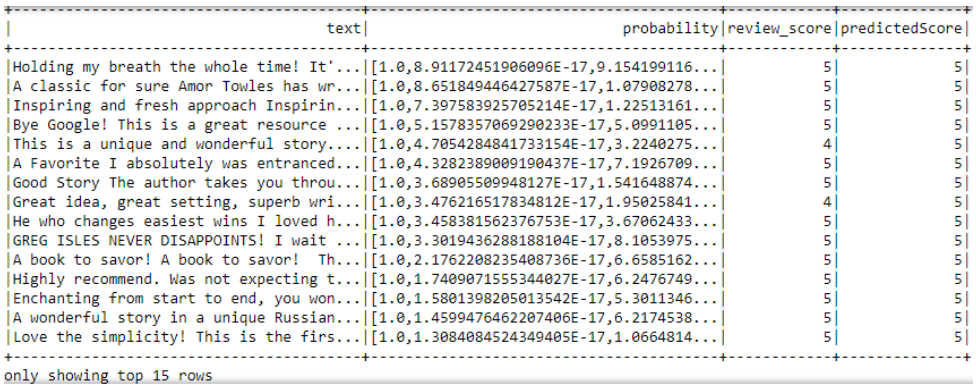
As before, three different models were trained using 5-fold crossvalidation to tune their corresponding parameters. In the multinomial logistic model, the same scheme used to tune the parameters in the previous section was used. As a result the model that provides the best performance has a regularization constant equals to 0.01 and a elasticNet parameter equals to 0.2. The corresponding prediction accuracy is 0.732, a little bit better than before. An example of the obtained probabilities and predicted ratings in the test set are shown in Figure 28.

text	probability	review_score	predictedScore
exceptional novel - one of the best t...	[0.9999622696793031,2.733527368071582...	5	5
The best book you will ever read! By ...	[0.9999581685305785,1.678374488334822...	5	5
... beyond five I would choose them e...	[0.9999160145116898,1.548068141767122...	5	5
An engaging (exciting, even) memoir/m...	[0.9997986893320138,2.072211483093863...	5	5
Creative, delicious, and the only cak...	[0.9997223337672339,2.508246223168706...	5	5
Anyone ready to change their life thr...	[0.9997033191053903,2.576619774236201...	5	5
Empowering and Uplifting Let me start...	[0.9996897334926627,1.104777746090736...	4	5
Rachel Provides Easy Steps For Creati...	[0.9996679695897795,1.281634338760052...	5	5
Finally get to the root cause for fem...	[0.9995400540191969,3.154363279797863...	5	5
I mean, it's a classic now, right? Th...	[0.9992489063952487,6.484301444080188...	5	5
Superb Having loved Amor Towles' firs...	[0.9992274282438185,6.931465476376522...	5	5
From the depths of brutality and insa...	[0.9992024405432739,5.738916687801269...	5	5
WOW! A must read... I read the last ...	[0.9988457024707325,8.98408364729792E...	5	5
A delightful tale of love and ingenui...	[0.9987566034701539,6.03481881191926E...	5	5
Now isn't it time you step into the E...	[0.9986074668866405,1.754205373886014...	5	5

only showing top 15 rows

Figure 28: Multinomial Logistic Regression TF-IDF - Results

With respect to Naive Bayes classifier, the smoothing parameter was also set to one in this case. The prediction accuracy on the test set is 0.71, a little bit worse than before. An example of the obtained probabilities and predicted ratings in the test set are shown in Figure 29.



text	probability	review_score	predictedScore
Holding my breath the whole time! It'...	[1.0,8.91172451906096E-17,9.154199116...	5	5
A classic for sure Amor Towles has wr...	[1.0,8.651849446427587E-17,1.07908278...	5	5
Inspiring and fresh approach Inspirin...	[1.0,7.397583925705214E-17,1.22513161...	5	5
Bye Google! This is a great resource ...	[1.0,5.1578357069290233E-17,5.0991105...	5	5
This is a unique and wonderful story....	[1.0,4.70542848417331 54E-17,3.2240275...	4	5
A Favorite I absolutely was entranced...	[1.0,4.3282389009190437E-17,7.1926709...	5	5
Good Story The author takes you throu...	[1.0,3.68905509948127E-17,1.541648874...	5	5
Great idea, great setting, superb wri...	[1.0,3.476216517834812E-17,1.95025841...	4	5
He who changes easiest wins I loved h...	[1.0,3.458381562376753E-17,3.67062433...	5	5
GREG ISLES NEVER DISAPPOINTS! I wait ...	[1.0,3.3019436288188104E-17,8.1053975...	5	5
A book to savor! A book to savor! Th...	[1.0,2.1762208235408736E-17,6.6585162...	5	5
Highly recommend. Was not expecting t...	[1.0,1.7409071555344027E-17,6.2476749...	5	5
Enchanting from start to end, you won...	[1.0,1.5801398205013542E-17,5.3011346...	5	5
A wonderful story in a unique Russian...	[1.0,1.4599476462207406E-17,6.2174538...	5	5
Love the simplicity! This is the firs...	[1.0,1.3084084524349405E-17,1.0664814...	5	5

only showing top 15 rows

Figure 29: Naive Bayes TF-IDF - Results

In regard to the random forest model, the same scheme used to tune the parameters in the previous section was used. As a result the model that provides the best performance has 500 trees and a maximum depth of 20. The corresponding prediction accuracy is 0.665, a little bit worse than before. An example of the obtained probabilities and predicted ratings in the test set are shown in Figure 30.

text	probability	review_score	predictedScore
Amazing must read! Great book! I laug...	[0.8394878420017327,0.098090323020065...	5	5
Wonderful book! Must Read! Loved thi...	[0.8350624376964675,0.103200703017782...	5	5
Amazing tale and well constructed sto...	[0.8340589625245075,0.101695406571931...	5	5
One of the best books I have read thi...	[0.8300513716897474,0.105026777976361...	5	5
Loved, loved, loved this book This wa...	[0.8277551885868162,0.106696788690534...	5	5
Loved this book! Loved this book! It'...	[0.8261263650670962,0.105873147661166...	5	5
5 stars aren't Enough! What an absolu...	[0.8257218220105224,0.103640884757110...	5	5
A Wonderful and Educational Novel If ...	[0.8257086126975356,0.107933883931365...	5	5
Loved this book! Susannah tells her l...	[0.8255632632667492,0.106719056498556...	5	5
first class book arrived on time, bea...	[0.8247208295268611,0.106099841718886...	4	5
Great book! This has been the best bo...	[0.8244945746446438,0.102499994001282...	5	5
Awesome Novel - Not to be Missed Best...	[0.8239044216974601,0.106662693652428...	5	5
Excellent well written book about an ...	[0.8237734723093199,0.109302444791946...	5	5
Heartwarming. Well written. Lovely bo...	[0.8230847563416945,0.106904419365415...	5	5
LOVED! Great read! I love Rachel and ...	[0.822945875873666,0.1083256060359850...	5	5

only showing top 15 rows

Figure 30: Random Forest TF-IDF - Results

Word2Vec: Random Forest and Logistic Regression

Subsequently, a Word2Vec pipeline was created as another option of featurization. This methodology consists in a two-layer neural network that produces word embeddings, taking into account the context of words. As before, the new pipeline preserves the `RegexTokenizer`, `StopWordsRemover` and `StringIndexer`, as it was mentioned before, and additionally uses `Word2Vec` function to make the feature extraction. Figure 31 shows the featurization results for one data point in the training dataset.

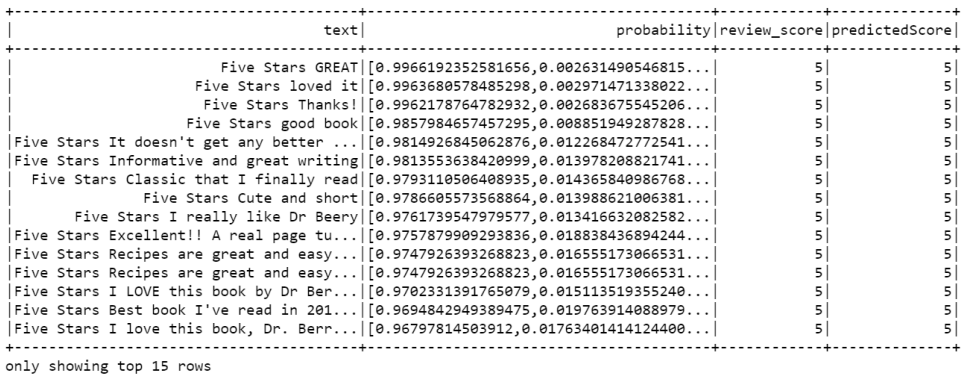
review_score	text	words	filtered	features	label
1	*Eye roll* I first saw...	[eye, roll, i, first, ...	[eye, roll, first, saw...	[0.12171184380861873,-...	4.0

only showing top 1 row

Figure 31: Word2Vec Featurization - Training

As before, two different models were trained using 5-fold crossvalidation to tune their corresponding parameters. It is important to mention that Naive Bayes Classifier could not be used here because the implementation of the algorithm does not support negative features.

In the multinomial logistic model, the same scheme used to tune the parameters in the previous section was used. As a result the model that provides the best performance uses a ridge loss function and has a regularization constant equals to 0.01. The corresponding prediction accuracy is 0.663, the worst result so far. An example of the obtained probabilities and predicted ratings in the test set are shown in Figure 32.

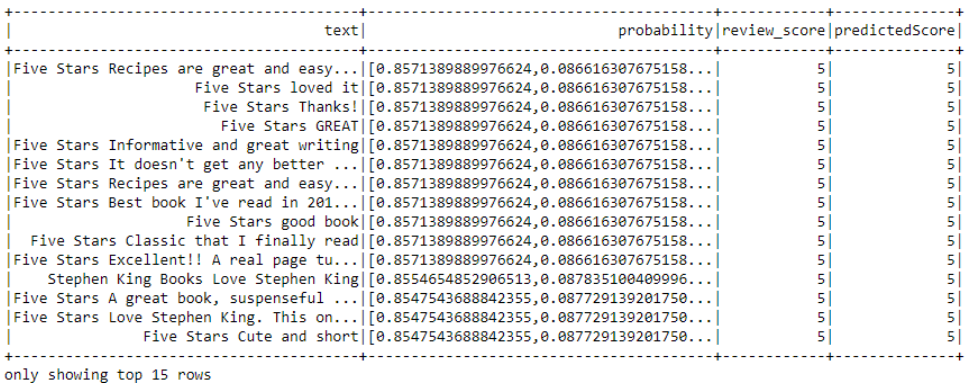


text	probability	review_score	predictedScore
Five Stars GREAT	[0.9966192352581656,0.002631490546815...	5	5
Five Stars loved it	[0.9963680578485298,0.002971471338022...	5	5
Five Stars Thanks!	[0.9962178764782932,0.002683675545206...	5	5
Five Stars good book	[0.9857984657457295,0.008851949287828...	5	5
Five Stars It doesn't get any better ...	[0.9814926845062876,0.012268472772541...	5	5
Five Stars Informative and great writing	[0.9813553638420999,0.013978208821741...	5	5
Five Stars Classic that I finally read	[0.9793110506408935,0.014365840986768...	5	5
Five Stars Cute and short	[0.9786605573568864,0.013988621006381...	5	5
Five Stars I really like Dr Beery	[0.9761739547979577,0.013416632082582...	5	5
Five Stars Excellent!! A real page tu...	[0.9757879909293836,0.018838436894244...	5	5
Five Stars Recipes are great and easy...	[0.9747926393268823,0.016555173066531...	5	5
Five Stars Recipes are great and easy...	[0.9747926393268823,0.016555173066531...	5	5
Five Stars I LOVE this book by Dr Ber...	[0.9702331391765079,0.015113519355240...	5	5
Five Stars Best book I've read in 201...	[0.9694842949389475,0.019763914088979...	5	5
Five Stars I love this book, Dr. Berr...	[0.96797814503912,0.01763401414124400...	5	5

only showing top 15 rows

Figure 32: Multinomial Logistic Regresssion W2V - Results

With respect to the random forest model, the same scheme used to tune the parameters in the previous section was used. As a result the model that provides the best performance has 100 trees and a maximum depth of 4. The corresponding prediction accuracy is 0.663, a little bit worse than before. An example of the obtained probabilities and predicted ratings in the test set are shown in Figure 33.



text	probability	review_score	predictedScore
Five Stars Recipes are great and easy...	[0.8571389889976624,0.086616307675158...	5	5
Five Stars loved it	[0.8571389889976624,0.086616307675158...	5	5
Five Stars Thanks!	[0.8571389889976624,0.086616307675158...	5	5
Five Stars GREAT	[0.8571389889976624,0.086616307675158...	5	5
Five Stars Informative and great writing	[0.8571389889976624,0.086616307675158...	5	5
Five Stars It doesn't get any better ...	[0.8571389889976624,0.086616307675158...	5	5
Five Stars Recipes are great and easy...	[0.8571389889976624,0.086616307675158...	5	5
Five Stars Best book I've read in 201...	[0.8571389889976624,0.086616307675158...	5	5
Five Stars good book	[0.8571389889976624,0.086616307675158...	5	5
Five Stars Classic that I finally read	[0.8571389889976624,0.086616307675158...	5	5
Five Stars Excellent!! A real page tu...	[0.8571389889976624,0.086616307675158...	5	5
Stephen King Books Love Stephen King	[0.8554654852906513,0.087835100409996...	5	5
Five Stars A great book, suspenseful ...	[0.8547543688842355,0.087729139201750...	5	5
Five Stars Love Stephen King. This on...	[0.8547543688842355,0.087729139201750...	5	5
Five Stars Cute and short	[0.8547543688842355,0.087729139201750...	5	5

only showing top 15 rows

Figure 33: Random Forest W2V - Results

Model Comparison

According to the predictive performance of the tuned models on the test set, the best models are the Naive Bayes using a Bag of Words approach, and the Multinomial Logistic Regression using TF-IDF featurization. The ”worst-performing” models in terms of accuracy are the ones that used Word2Vec featurization. Table 12 summarizes

these results. As a consequence, Naive Bayes using a Bag of words approach was chosen to make online predictions.

Featurization	Model	Parameters	Accuracy
Bag of Words	Multinomial	Regul = 0.01 elasticNet = 0	0.721
Bag of Words	Naive Bayes	Smooth = 1	**0.732**
Bag of Words	Random Forest	N_trees = 500 MaxDepth = 20	0.67
TF-IDF	Multinomial	Regul = 0.01 elasticNet = 0.2	**0.732**
TF-IDF	Naive Bayes	Smooth = 1	0.71
TF-IDF	Random Forest	N_trees = 500 MaxDepth = 20	0.665
Word2Vec	Multinomial	Regul = 0.01 elasticNet = 0	0.663
Word2Vec	Random Forest	N_trees = 100 MaxDepth = 4	0.663

Table 12: Model Comparison

Online Prediction

Once the final model was chosen, it was time to deploy it in a streaming way. To do so, the whole pipeline was exported in a folder, so it can be used to make predictions in the reviews that come from the server. Figure 34 shows the piece of code used to export the pipeline.

```
pipeline_nb_final = Pipeline(stages=[regexTokenizer, stopwordsRemover, countVectors, label_stringIdx, nb, labelConverter])
pipeline_nb_export = pipeline_nb_final.fit(trainingData)
```

```
# Export pipeline
pipeline_nb_export.save('pipeline_nb')
```

Figure 34: Exporting Pipeline

The first line sets the whole pipeline in the following order:

1. Transform sentences to tokens
2. Remove stopwords
3. Feature extraction using Bag of Words approach
4. Transform the target variable from string to index
5. Predict the label using the Naive Bayes classifier
6. Transform back the prediction from index to the corresponding label

In the second line and the third line, this pipeline was applied to the training set and saved in the working directory. By defining the pipeline in this way, it is certain that the prediction of a new review will follow the same process used to fit the model.

Additionally, a new script was developed with the purpose of repeating this whole process when new data come through the server (In the appendix the whole code is presented). This process is divided in different steps:

1. Read the data in `json` format
2. Transform from `json` to a `dataframe`
3. Concatenate the title of the review and the review in a new column called `text`
4. Select only the `review_score` and the `text` from the data frame
5. Fit the pipeline defined above
6. Print results

As a result, it is now possible to obtain real time predictions based on the title of the review and the review itself. Figure 35 shows an screenshot of three different reviews that came online and its corresponding prediction. The first prediction was wrong because the predicted score was 4, and the true value was 5. The second and the third review were correctly classified by predicting a score of 5 stars.

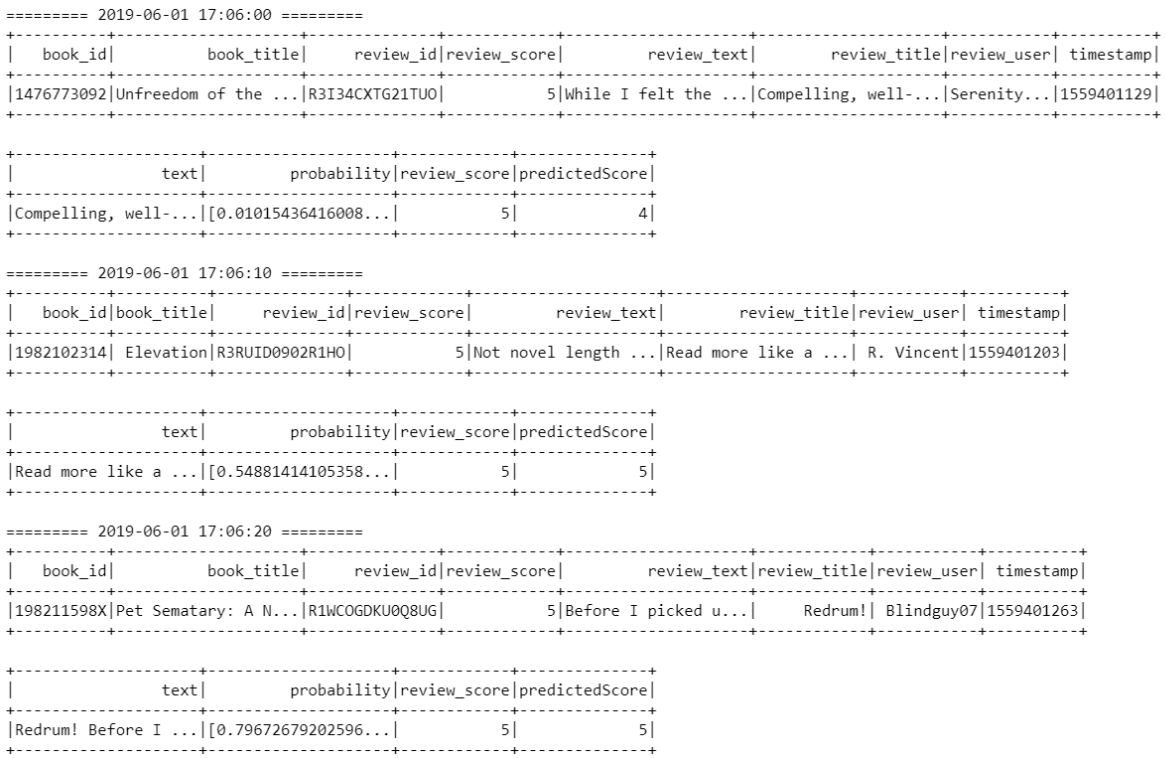


Figure 35: Online prediction

Conclusion

The aim of this research project was threefold: to successfully stream Amazon book review data live from a server, to train a model off-line which predicts one out of 5 possible review ratings and lastly, to implement the model in an online context, predicting the rating as the review data is coming in. In the first step of the analysis, we used a script which deleted the empty folders, merged individual review files into a single .csv file containing column (variable) names and removed duplicates. In the next step, we looked at three kinds of featurization, namely BagOfWords, TF-IDF and Word2Vec and employed three kinds of machine learning models: Multinomial Logistic Regression, Naive Bayes and RandomForest using train-test validation and cross-validation. In terms of prediction accuracy, the best models were Naive Bayes with BagOfWords featurization (0.732) and Multinomial Logistic Regression with TF-IDF featurization (0.732). In the final step, we saved the BagOfWords pipeline together with the chosen Naive Bayes model. The incoming review data is processed as it comes in, from the standard .json format to a data frame containing the key variables ”text” and ”review score”. We then fit the saved pipeline and model to the processed incoming reviews and obtain real-time predictions. As we gathered from the output, the review ratings are predicted live with a seemingly satisfactory accuracy.

Appendix - R and Python Code

R code to merge reviews

```r
# Set directory
setwd(...)

# Delete empty folders
files <- list.files(full.names = T)
for (i in 1:length(files)) {
  subfiles <- list.files(files[i])
  if (length(subfiles) == 1) {
    unlink(files[i], recursive = T)
  }
}

#----
# Concatenate reviews in a single data frame
# This was done by each team member
library(jsonlite)
library(parallel)

# Import data
cl <- makeCluster(detectCores() - 1)
json_files <- list.files(path="...\\StreamingData\\Unstructured\", recursive = T, pattern = "part*", full.names = T)
json_list <- parLapply(cl,json_files,function(x) rjson::fromJSON(file=x,method = "R"))
stopCluster(cl)

# Create dataframe
df <- data.frame(matrix(unlist(json_list), nrow = length(json_list), byrow=T))

# Rename columns
names(df) <- c("book_title", "review_title", "review_user", "book_id", "review_id",
"timestamp", "review_text", "review_score")

# Transform to character
df$book_title <- as.character(df$book_title)
df$review_title <- as.character(df$review_title)
df$review_user <- as.character(df$review_user)
df$book_id <- as.character(df$book_id)
df$review_id <- as.character(df$review_id)
df$timestamp <- as.character(df$timestamp)
df$review_text <- as.character(df$review_text)
df$review_score <- as.integer(df$review_score)

# Remove duplicates
df2 <- df[!duplicated(df$review_text),]

# Export csv file
write.csv(df2, file = "...\\Streaming\\Data\\bookdata.csv")

# Import data check
datacheck <- read.table("...\\Streaming\\Data\\bookdata.csv", header = TRUE, sep = ",") # OK

#----
# Merge data of each team member into a single csv file
setwd("...\\Data")

files <- list.files()

file1 <- read.csv(files[1],sep = ",")
file2 <- read.csv(files[2],sep = ",")
file3 <- read.csv(files[3],sep = ",")

file_all <- rbind(file1, file2, file3)
file_all$X <- seq(1:nrow(file_all))

# Delete duplicated reviews
file_all2 <- file_all[!duplicated(file_all[,-1]),]

# Export csv file
write.csv(file_all2, "data.csv", row.names = F)
```

Python code to pre-process the data and train different models

```python
# Import reviews
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
data = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true', quote = '"', escape = '"'
      , multiline = 'true', ignoreTrailingWhiteSpace = 'true').load('.../Data/data.csv')

data.show(5)

# Frequency table
from pyspark.sql.functions import col
```

```
data.groupBy("review_score").count().orderBy(col("count").desc()).show()

# Remove observation where review_text is null
data = data.na.drop(subset=["review_text"])

# Concatenate book_title, review_title and review_text into a single column
from pyspark.sql import functions as ff
data = data.withColumn('text', ff.concat(ff.col('review_title'), ff.lit(' '), ff.col('review_text')))
data.show(1, truncate = False)

drop_list = ['X', 'book_title', 'review_title', 'review_user', 'book_id', 'review_id',
             'timestamp', 'review_text']
data = data.select([column for column in data.columns if column not in drop_list])

#########################################################################################################
# PREPROCESSING PIPELINE
from pyspark.ml.feature import RegexTokenizer, StopWordsRemover, CountVectorizer
from pyspark.ml.classification import LogisticRegression
import nltk

# Split the data in training ang test set
# set seed for reproducibility
(trainingData, testData) = data.randomSplit([0.8, 0.2], seed = 12345)
print("Training Dataset Count: " + str(trainingData.count()))
print("Test Dataset Count: " + str(testData.count()))

# regular expression tokenizer: To split sentences into words
regexTokenizer = RegexTokenizer(inputCol="text", outputCol="words", pattern="\\W")

# Remove stop words
stopwordList = nltk.corpus.stopwords.words('english')
stopwordsRemover = StopWordsRemover(inputCol="words", outputCol="filtered", stopWords=stopwordList)

# bag of words count
countVectors = CountVectorizer(inputCol="filtered", outputCol="features", vocabSize=20000, minDF=50)

# Define pipeline
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler

# Recoding target variable
label_stringIdx = StringIndexer(inputCol = "review_score", outputCol = "label")
labels_stars = label_stringIdx.fit(trainingData).labels # Save this levels to be able later to transform back

# Create pipeline
pipeline = Pipeline(stages=[regexTokenizer, stopwordsRemover, countVectors, label_stringIdx])

# Fit the pipeline to training data.
pipelineFit = pipeline.fit(trainingData)
dataset = pipelineFit.transform(trainingData)

# Fit the pipeline to test data.
test_dataset = pipelineFit.transform(testData)

#########################################################################################################
# OFFLINE TRAINING
# Logistic regression using count vector features
from pyspark.ml.feature import IndexToString

# Transform back from index to original coding
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedScore", labels = labels_stars)

lr = LogisticRegression(maxIter=20, regParam=0, elasticNetParam=0)

lrModel = lr.fit(dataset) # Fit model
predictions = lrModel.transform(test_dataset) # Predict
predictions = labelConverter.transform(predictions) # Transform labels

predictions.filter(predictions['prediction'] == 0).select("text","probability","review_score","predictedScore").orderBy("
    probability", ascending=False).show(n = 20, truncate = 45)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
evaluator.evaluate(predictions) # This is the accuracy

# Crossvalidation
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

lr = LogisticRegression(maxIter=20, regParam=0, elasticNetParam=0)

# Create ParamGrid for Cross Validation
paramGrid = (ParamGridBuilder()
             .addGrid(lr.regParam, [0.01, 0.05, 0.1, 0.2, 0.3, 0.5]) # regularization parameter
             .addGrid(lr.elasticNetParam, [0.0, 0.2, 0.4, 0.6, 0.8, 1]) # Elastic Net Parameter (Ridge = 0)
#             .addGrid(model.maxIter, [10, 20, 50]) #Number of iterations
#             .addGrid(idf.numFeatures, [10, 100, 1000]) # Number of features
             .build())

# Create 5-fold CrossValidator
cv = CrossValidator(estimator=lr, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=5)

cvModel = cv.fit(dataset)

predictions = cvModel.transform(test_dataset)
predictions = labelConverter.transform(predictions) # Transform labels

```

```
# Evaluate best model
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", labelCol="label")
evaluator.evaluate(predictions)

best_model = cvModel.bestModel
best_model

best_reg_param = best_model._java_obj.getRegParam()
best_elasticnet_param = best_model._java_obj.getElasticNetParam()
print(best_reg_param);print(best_elasticnet_param)

predictions.filter(predictions['prediction'] == 0).select("text","probability","review_score","predictedScore").orderBy("
    probability", ascending=False).show(n = 20, truncate = 40)

# Naive Bayes
from pyspark.ml.classification import NaiveBayes

nb = NaiveBayes(smoothing=1)

model = nb.fit(dataset)
predictions = model.transform(test_dataset)
predictions = labelConverter.transform(predictions) # Transform labels

predictions.filter(predictions['prediction'] == 0).select("text","probability","review_score","predictedScore").orderBy("
    probability", ascending=False).show(n = 15, truncate = 40)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
evaluator.evaluate(predictions)

# Saving complete pipeline for online prediction
pipeline_nb_final = Pipeline(stages=[regexTokenizer, stopwordsRemover, countVectors, label_stringIdx, nb, labelConverter])
pipeline_nb_export = pipeline_nb_final.fit(trainingData)

# Export pipeline
pipeline_nb_export.save('pipeline_nb')

# Random Forest
rf = RandomForestClassifier(labelCol="label", \
                            featuresCol="features", \
                            numTrees = 100, \
                            maxDepth = 4)

# Create ParamGrid for Cross Validation
paramGrid = (ParamGridBuilder()
             .addGrid(rf.numTrees, [100, 200, 500]) # regularization parameter
             .addGrid(rf.maxDepth, [4, 10, 20]) # Elastic Net Parameter (Ridge = 0)
#            .addGrid(model.maxIter, [10, 20, 50]) #Number of iterations
#            .addGrid(idf.numFeatures, [10, 100, 1000]) # Number of features
             .build())

# Create 5-fold CrossValidator
cv = CrossValidator(estimator=rf, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=5)

cvModel = cv.fit(dataset)

predictions = cvModel.transform(test_dataset)
predictions = labelConverter.transform(predictions) # Transform labels

# Evaluate best model
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", labelCol="label")
evaluator.evaluate(predictions)

best_model = cvModel.bestModel
best_model

best_numTrees = best_model.getNumTrees
best_maxDepth = best_model.getOrDefault('maxDepth')
print(best_numTrees);print(best_maxDepth)

predictions.filter(predictions['prediction'] == 0).select("text","probability","review_score","predictedScore").orderBy("
    probability", ascending=False).show(n = 15, truncate = 40)

#########
# TF-IDF Features
from pyspark.ml.feature import HashingTF, IDF

hashingTF = HashingTF(inputCol="filtered", outputCol="rawFeatures", numFeatures=20000)
idf = IDF(inputCol="rawFeatures", outputCol="features", minDocFreq=5) #minDocFreq: remove sparse terms

pipeline_tfidf = Pipeline(stages=[regexTokenizer, stopwordsRemover, hashingTF, idf, label_stringIdx])

# applying pipeline to training data
pipelineFit_tfidf = pipeline_tfidf.fit(trainingData)
dataset_tfidf = pipelineFit_tfidf.transform(trainingData)

# applying pipeline to test data
test_dataset_tfidf = pipelineFit_tfidf.transform(testData)

# After this point, the same syntax is used as in the previous models but replacing dataset for dataset_tfidf and test_dataset
    fo test_dataset_tfidf

#########
# Word2Vec Features
from pyspark.ml.feature import Word2Vec
```

```
197
198 w2v = Word2Vec(vectorSize=3, minCount=0, inputCol="filtered", outputCol="features")
199
200 pipeline_w2v = Pipeline(stages=[regexTokenizer, stopwordsRemover, w2v, label_stringIdx])
201
202 # applying pipeline to training data
203 pipelineFit_w2v = pipeline_w2v.fit(trainingData)
204 dataset_w2v = pipelineFit_w2v.transform(trainingData)
205
206 # applying pipeline to test data
207 test_dataset_w2v = pipelineFit_w2v.transform(testData)
208
209 # After this point, the same syntax is used as in the previous models but replacing dataset for dataset_w2v and test_dataset
      fo test_dataset_w2v
210
211 ##########################################################################################################
212 # ONLINE PREDICTION
213 from IPython.core.display import display, HTML
214 display(HTML("<style>.container { width:95% !important; }</style>"))
215
216 from threading import Thread
217
218 class StreamingThread(Thread):
219     def __init__(self, ssc):
220         Thread.__init__(self)
221         self.ssc = ssc
222     def run(self):
223         ssc.start()
224         ssc.awaitTermination()
225     def stop(self):
226         print('----- Stopping... this may take a few seconds -----')
227         self.ssc.stop(stopSparkContext=False, stopGracefully=True)
228
229 from pyspark.streaming import StreamingContext
230 from pyspark.sql import Row
231 from pyspark.sql.functions import udf, struct
232 from pyspark.sql.types import IntegerType
233 from pyspark.sql import functions as ff
234 from pyspark.ml import Pipeline, PipelineModel
235
236 globals()['models_loaded'] = False
237
238 def process(time, rdd):
239     if rdd.isEmpty():
240         return
241     print("========= %s =========" % str(time))
242
243     # Convert to data frame
244     df = spark.read.json(rdd)
245     df.show()
246
247     # concatenate review_title and review_text
248     df_withpreds = df.withColumn('text', ff.concat(ff.col('review_title'), ff.lit(' '), ff.col('review_text')))
249
250     drop_list = ['book_title', 'review_title', 'review_user', 'book_id', 'review_id',
251              'timestamp', 'review_text']
252     df_withpreds = df_withpreds.select([column for column in df_withpreds.columns if column not in drop_list])
253
254     # Load in the model if not yet loaded:
255     if not globals()['models_loaded']:
256         # load in your models here
257         globals()['my_model'] = PipelineModel.load('pipeline_nb') # Replace this with:    [...].load('my_logistic_regression')
258         globals()['models_loaded'] = True
259
260     # Predict using the model:
261     df_result = globals()['my_model'].transform(df_withpreds)
262     df_result = df_result.select("text","probability","review_score","predictedScore")
263     df_result.show()
264
265 ssc = StreamingContext(sc, 10)
266
267 lines = ssc.socketTextStream("seppe.net", 7778)
268 lines.foreachRDD(process)
269
270 ssc_t = StreamingThread(ssc)
271 ssc_t.start()
```

4 Assignment 4: Neo4j/Gephi Assignment

Introduction

This assignment focuses on a database consisting of political mandates in Belgium. Politicians in Belgium are required to list all mandates they have at public and private organizations. The main objectives of this assignment are to explore the graph using Neo4j (Cypher) and Gephi, find interesting patterns and visualize findings in an appealing manner. We start with a simple descriptive analysis of the data set regarding politicians with paid mandates. Next we investigate weather or not pillarisation is still present in the Belgium.

Paid Political Mandates

The Big winner of the 2014 election in Flanders was the NVA (nieuwe vlaamse Alliantie). The big loser was Vlaams Belang who's voters decided to vote for the more politically correct NVA[1] . The data we have shows the mandates, both paid and unpaid, of the members of the different political parties in public and privates organisations. We have data for two years 2016 and 2017. Since there was no new election in between these two years we assume there won't be a great deal of change between the two years. In this part of the analysis we look at the descriptive side of the data. First, we would want to show what party has the most members with paid mandates. We expect he two largest parties (NVA, PS) to be the ones with the most paid mandates since they were the winners of the election in 2014.

Distribution of paid mandates

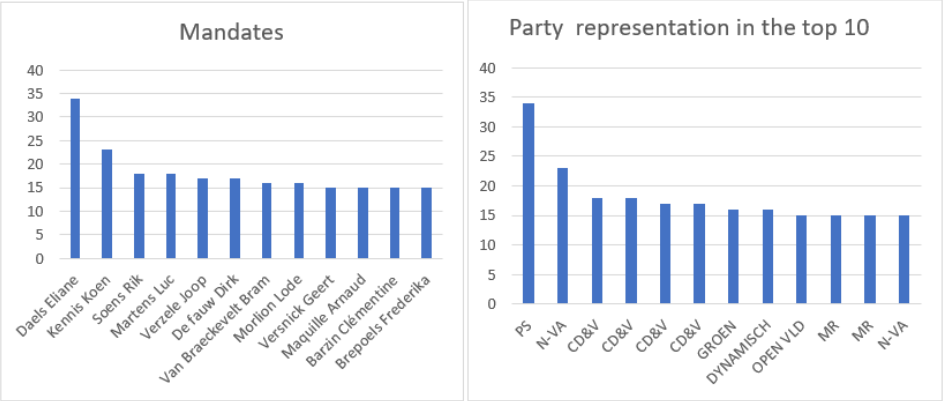


Figure 36: Distribution of paid political mandates. Both graphs show the same data except that on the left hand side we show the name of the politician while on the right hand side we show the political party the politician represents.

In figure 36 on can see that the PS (parti-socialiste) is home of the politician with the most paid mandates. We should not be surprised by that since they are the biggest

[1]After the 2019 election NVA is still the most popular party in Flanders (24.8%) while the big loser of 2014 Vlaams Belang has become the second most popular party in Flanders (18,5%).

party in Wallonia. It is however interesting that Eliane Deaels has the time to work for 34 paid mandates. Furthermore, we find that she is the only one of her party in the top 10 of politicians with the most mandates. This might indicate she is rather an exception then a rule. Now interestingly, CD&V[2] (Christian democrat), has 4 of the 10 politicians with the most mandates among its ranks while being the third biggest federal party (measured in seats in the federal government). At the same time CD&V was second most popular party in Flanders with around 20% of the votes.

In figure37, one can see that the distribution of paid mandates for all Belgian political party's is left skewed. The distribution of paid mandates follows a power law (note that the graph only shows the 10 parties with the most mandates). Again, we notice something odd. CD&V the second largest Flemish party has the most paid mandates on total (3776). That is 2032 more then the actual biggest Flemish party in Flanders NVA. While in Wallonia the difference between PS and MR (number 1 and 2) is far smaller. Furthermore, to outsiders it might be interesting to see that the socialist party in Flanders, the SP.A, has about half the amount of paid mandates of its Waloon counterpart.

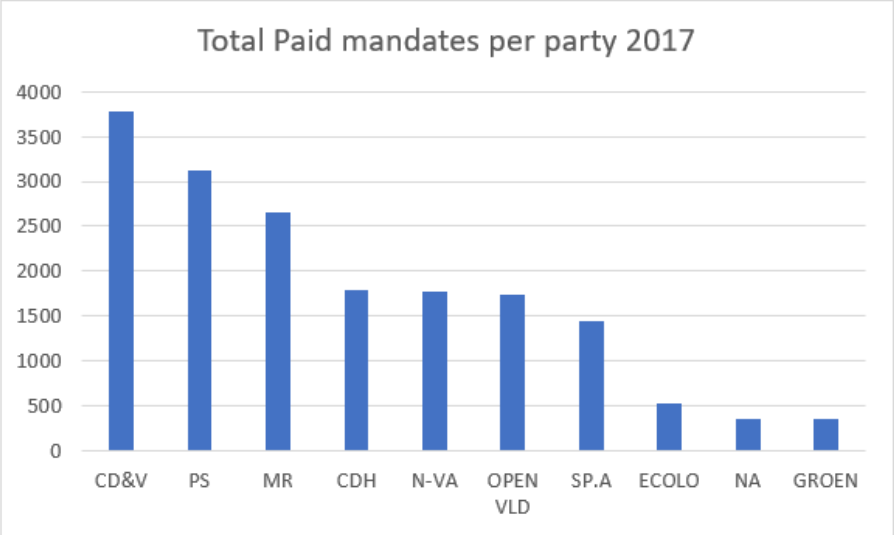


Figure 37: Sum of paid political mandates per party. Only the top 10 is shown.

[2]In the 2019 election CD&V got down to 15% of the vote.

Pillarisation in Belgian Politics

Pillarisation (Dutch: *verzuiling*) is a consociational social arrangement between different religious and social groups [2]. The best-known examples of this have historically occurred in the Netherlands and Belgium. Each pillar may have their own social institutions and social organizations. These may include their own newspapers, broadcasting organisations, political parties, trade unions and farmers' associations, banks, schools, hospitals, universities, scouting organisations, sports clubs and other organizations. Such segregation means that many people have little or no personal contact with people from another pillar. In both Flanders and Wallonia, societies were pillarised.

Belgium had three pillars: the catholic pillar, the liberal pillar and the social-democratic pillar. As of today pillarisation is not present any more in daily life, but remnants of pillasization are still present. For example in health insurance (*Christelijke Mutualiteit*, *Liberale Mutualiteit* and the *Socialistische Mutualiteit*) or political parties: Open Vld for the liberal pillar sp.a for the social-democratic pillar and CD&V for the catholic pillar.

We will analyze if pillarisation still is a factor regarding political mandates in 2017. The focus will be on the 5 largest parties in the Flamish parliament as of the elections from 2014: N-VA (42 seats), CD&V (27 seats), Open Vld (19 seats), sp.a (18 seats) and Groen (9 seats). In total these parties have 117 out of 124 seats in Flemish parliament. As mentioned before CD&V (catholic pillar), Open Vld (liberal pillar) and sp.a (social-democratic pillar) all are parties which are associated with one of the three pillars. The largest party N-VA (founded in 2001) is a Flemish nationalist, conservative party which is not associated with any pillar. Groen (founded in 1982) is a progressive, green party which is also not associated with any pillar.

Pillarisation part 1

Process in Neo4j/Gephi

For our first analysis we will analyze the mandates for catholic, liberal and social-democratic organisations. After importing the graph in Neo4j we created a new label to distinguish christian ("CHR"), liberal ("LIB") and social-democratic ("SOC") organisations. We do this by using the following keywords for organisations: *Christelijk* (christian) and *Katholiek* (catholic) for the catholic pillar, *Liberale* and *Liberaal* (liberal) for the Liberal pillar and *Socialistisch* (socialist) for the social-democratic pillar (see appendix for cypher code). After creating the labels, we used the following cypher code to extract all nodes for the 5 biggest political parties which are associated with one of the three pillars:

```
CALL apoc.export.graphml.query(
'MATCH (p:party)-[m:member_of]-(n:politician)-[r:mandate_at]-(b:organisation)
WHERE (r.year=2017) AND (b.type ="CHR" or b.type="LIB" or b.type="SOC")
AND (p.name = "OPEN VLD" or p.name= "GROEN" or p.name= "N-VA" or
p.name = "CD&V" or p.name = "SP.A")
RETURN *',
"c:/users/hotze/desktop/subset99.graphml",
)
```

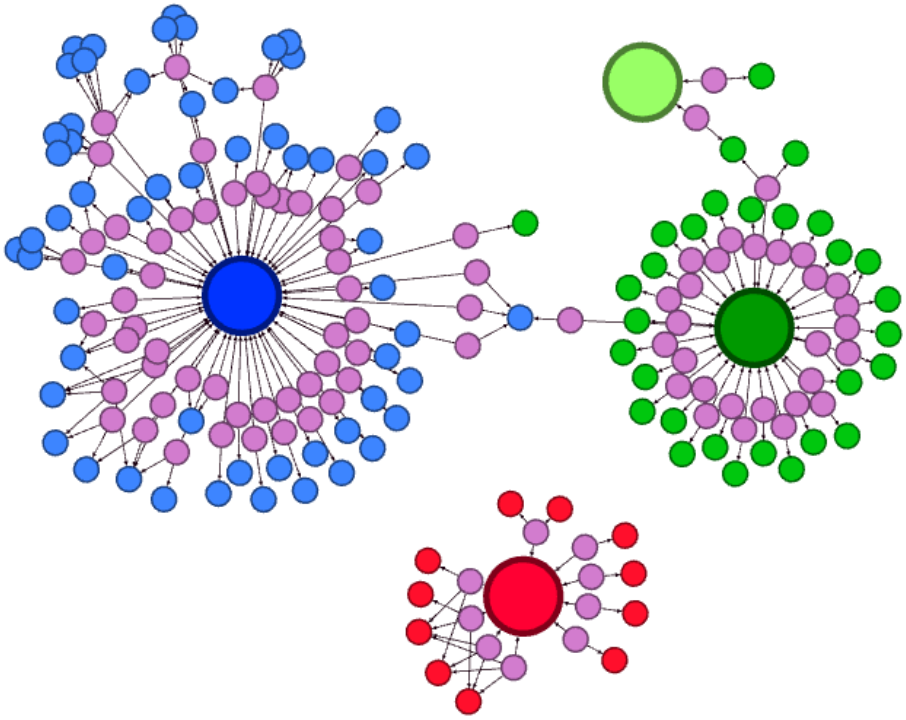


Figure 38

This query created 197 nodes, 228 relationships and 980 properties. The next step is to run the python "cleaning" script which fixes the GraphML file. We do this by running the following code in Jupyter QtConsole: **%run fixme.py subset99.graphml**, which gave the cleaned graph (subset99.graphml_fixed.graphml). Now the graph of political mandates obtained using Neo4j is further visualized using Gephi. In order to do this, we import the graph obtained by Neo4j into Gephi. Different algorithms were tried out to obtain a decent looking graph, the ForceAtlas algorithm combined with the Expansion algorithm produced the best result. Gephi has some additional option for making graphs more representative. We added colours and changed the size of the nodes to distinguish between politicians, organisations and political parties.

Interpretation graph

The big coloured nodes in the graph (figure 38) represent all political parties (red for the sp.a, blue for Open Vld, darkgreen for CD&V and lightgreen for Groen). The purple nodes are the politicians. The small coloured nodes are organisations (blue for liberal organisations, green for christian/catholic organisations and red for social-democratic organisations).

As we can see from the graph, remnants of pillarisation still play a major role related to political mandates even today. Almost all mandates for christian/catholic organisations were performed by politicians from CD&V, only three mandates were performed by politicians from other parties (one for Open Vld and two for Groen). The same is true for the liberal organisations, only one mandate was performed by a politician from another party (CD&V). For the socialist-democratic organisations, the situation is even

more extreme: none of the mandates were performed by a politician other than the politicians from sp.a. Noteworthy point is the fact that the biggest party in Flanders - N-VA has zero(!) mandates at organisations which were associated with pillarisation. This graph clearly shows us the remnants of pillarisation still play a major role in political mandates today.

Network metrics

In this paragraph we will analyse some network metrics. As we can see from the table 13 below, Open Vld have the most party members in this network with degree=54 whereas CD&V and s.pa with 29 party members and 9 party members also have a reasonable amount of party members. N-VA is not present in this graph and Groen only has 2 party members.

Something notable related to the network metrics are the centrality measures for s.pa. Since s.pa is not connected to other parties in this network, we could say that s.pa have their own network. S.pa scores highest on closeness centrality (0.645) but on the contrary they score very low on betweeness centrality (0.008). This is because when calculating closeness centrality Gephi recognizes two separate networks, for betweeness centrality all the nodes are viewed as one large network.

Party	Degree	Betweeness	Closeness	Eigenvector centr.
Open Vld	54	0.672	0.336	1.000
CD&V	29	0.451	0.243	0.133
s.pa	9	0.008	0.645	0.016
GROEN	2	0.018	0.126	0.010

Table 13: Network metrics

A fun fact is that the *Katholieke Universiteit Leuven* is an exception regarding pillarisation, it is one of the three organisations from the christian pillar which has a mandate that is not performed by a politician from CD&V, Joost Venken from Groen has a mandate at KU Leuven. Notewhorthy is the fact that KU Leuven scores most low on all network metrics compared to all the other nodes.

Pillarisation part 2

Process in Neo4j/Gephi

We will dive deeper into the concept of pillarisation. We follow the same process as in the previous chapter, but now we will focus on key words which are only to a certain amount related to a pillar.

For the catholic pillar the following key words are now used: *Kerk* (church), *Kerkbestuur* (church administration), *Landbouw* (agriculture) and *Boeren* (farmers). The catholic pillar had a lot of members on the countryside whereas the social-democratic and the liberal pillar had relatively more members in cities. For the key word *Kerk* the problem exists that a lot of Belgium villages do have the word *Kerk* in their name (for example Middelkerke or Kerkhove). In order to get relevant results, we added a space after the word *Kerk* in our query.

For the liberal pillar we use the following keywords: *Openbare* (public) and *Vrije* (free). Now we face the problem that a lot of results consists of organisations related to *Vrije tijd* (leisure time) and *Publieke werken* (public works). We solve this problem by excluding the keywords *Vrije tijd* and *Publieke werken* in our query.

For the social-democratic pillar we will not use any specific keywords. However we do expect that politicians from sp.a are relatively more often active at organizations related to the liberal pillar since there was a certain overlap between these pillars during the pillarisation. For example, public schools are an element of both the liberal and social-democratic pillar. We also expect that there is far more variety for these mandates and to see more mandates for politicians from N-VA and Groen. However, we still expect to see some signs of pillarisation, especially for the catholic pillar.

Regarding our work in Neo4j and Gephi exactly the same steps as described in the previous chapter were performed (see the appendix for the queries). Our graph consists of 157 nodes, 169 relationships and 736 properties.

Interpretation graph

The graph (figure 39) has the same structure as the previous graph. Again the purple nodes represent politicians and the small coloured nodes are organisations, blue for the liberal pillar and green for the catholic pillar. The big coloured nodes are political parties: blue for Open Vld, red for sp.a, yellow for N-VA, lightgreen for Groen and darkgreen for CD&V.

For the organisations associated with the catholic pillar most mandates are unsurprisingly performed by politicians from CD&V. Only five mandates are not (four for Open Vld and one for N-VA). For the organisations associated with the liberal pillar we see more diversity. As expected some of the mandates are performed by politicians from sp.a and also Open Vld holds a relative large amount of mandates. Very surprising however is the fact that CD&V(!) possesses by far most of the mandates for liberal organizations.

The non-pillar parties have unsurprisingly the lowest amount of mandates. Groen again only has one mandate whereas N-VA has seven mandates. The Vrije Universiteit Brussel (the blue cirlce in the middle) has the most mandates from different political parties: two from Open Vld, one from N-VA and one from sp.a.

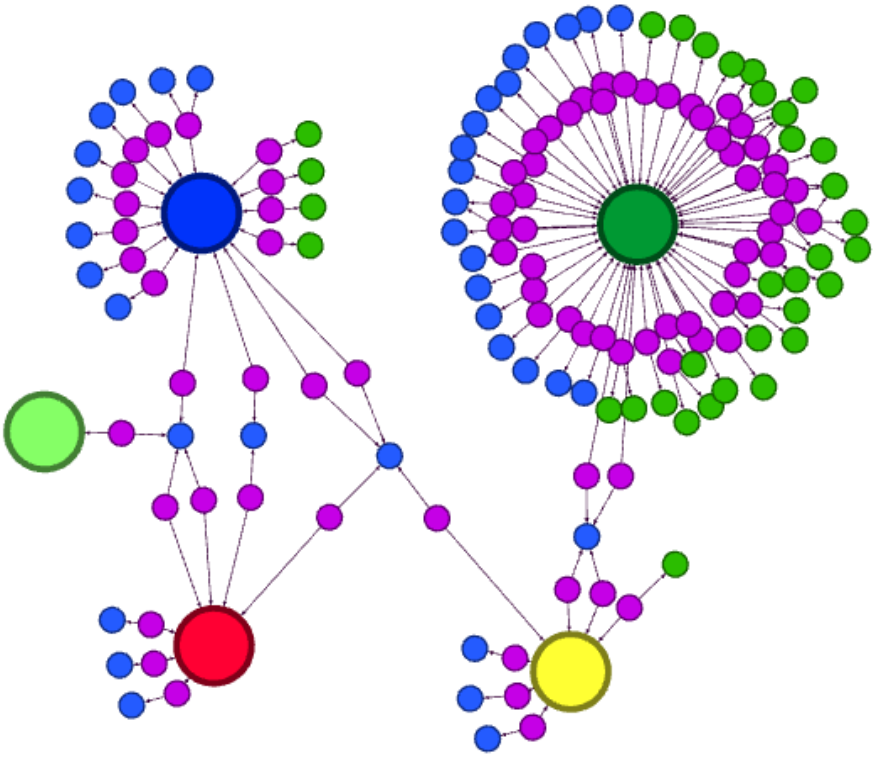


Figure 39

We conclude again that despite the fact that pillarisation is not present anymore in Belgium daily life, it still is a huge factor regarding political mandates for organizations, especially those associated with the catholic pillar.

Network metrics

Contrary to the previous graph, Belgium's largest political party, N-VA is present in this graph. CD&V with 50 politicians has by far the highest closeness and betweeness this graph. However despite having only 7 mandates N-VA scores high on those metrics as well, compared to Open Vld which possesses more than double the amount of mandates.

Party	Degree	Betweeness	Closeness	Eigenvector centr.
CD&V	50	0.843	0.247	1.000
Open Vld	16	0.335	0.142	0.320
s.pa	7	0.122	0.133	0.140
N-VA	7	0.493	0.197	0.140
GROEN	1	0.000	0.094	0.020

Table 14: Network metrics

Notable is the fact that 2 organisations have mandates from 3 different political parties, one of which is the *Vrije Universiteit Brussel*, where politicians from s.pa, N-VA

and Open Vld have a mandate.

Conclusion

The remnants of pillarisation regarding political mandates in Flanders were explored using Neo4j and Gephi. The 5 largest parties from Flanders were identified and connected to several mandates. Despite the fact that pillarisation is no longer a part of daily life anymore in Flanders it could be concluded that pillarisation still plays a major role when it comes to political mandates.

Appendix - Cypher queries

Cypher queries - Descriptive part

```
MATCH (x:party)--(n:politician)-[r:mandate_at]-(b:organisation)
WHERE r.year = 2017 AND r.compensated = 1
RETURN x.name AS Party, n.name AS politician, COUNT(r) AS mandates
ORDER BY mandates DESC

#we then exported the data as a .CVS and plotted the results in Excel.
```

Cypher queries - pillarisation part 1

```
#Creating labels
#CHR
MATCH (x:party)-[m:member_of]-(n:politician)-[r:mandate_at]-(b:organisation)
WHERE (r.year=2017) AND (b.name contains "Christelijk"  or b.name contains "Christelijk" or b.name contains "Katholiek" or b.name contains "katholiek")
SET b.type="CHR"
#Set 41 properties, completed after 1055 ms.

#SOC
MATCH (x:party)-[m:member_of]-(n:politician)-[r:mandate_at]-(b:organisation)
WHERE (r.year=2017) AND (b.name contains "Socialistisch"  or b.name contains "socialistisch")
SET b.type="SOC"
#Set 19 properties, completed after 573 ms.

#LIB
MATCH (x:party)-[m:member_of]-(n:politician)-[r:mandate_at]-(b:organisation)
WHERE (r.year=2017) AND (b.name contains "Liberale"  or b.name contains "liberale"
or b.name contains "Liberaal" or b.name contains "liberaal")
SET b.type="LIB"
#Set 93 properties, completed after 214 ms.

#Transport graph
CALL apoc.export.graphml.query(
'MATCH (p:party)-[m:member_of]-(n:politician)-[r:mandate_at]-(b:organisation)
WHERE (r.year=2017) AND (b.type ="CHR" or b.type="LIB" or b.type="SOC") AND (p.name  =~ "OPEN VLD" or p.name=~ "GROEN" or p.name=~ "N-VA" or p.name =~"CD&V" or p.name = "SP.A")
RETURN *',
"c:/users/hotze/desktop/subset99.graphml", {}
)
```

Cypher queries - pillarisation part 2

```
#Creating labels
#LIB
MATCH (x:party)-[m:member_of]-(n:politician)-[r:mandate_at]-(b:organisation)
WHERE (r.year=2017) AND (b.name contains "Vrije"  or b.name contains "Openbare")  AND  NOT (b.name contains "Vrije Tijd" or b.name contains "Openbare Werken")
SET b.name2 = "LIB"
#Set 57 properties, completed after 1330 ms.

#CHR
MATCH (x:party)-[m:member_of]-(n:politician)-[r:mandate_at]-(b:organisation)
WHERE (r.year=2017) AND (b.name contains "Kerk " or b.name contains "Boeren" or b.name contains "Landbouw" or b.name contains "Kerkbestuur")
set b.name2 = "CHR"
#Set 43 properties, completed after 286 ms.

#Transport graph
CALL apoc.export.graphml.query(
'MATCH (p:party)-[m:member_of]-(n:politician)-[r:mandate_at]-(b:organisation)
WHERE (r.year=2017) AND (b.name2 ="CHR" or b.name2="LIB") AND
    (p.name  =~ "OPEN VLD" or p.name=~ "GROEN" or p.name=~ "N-VA" or
    p.name =~"CD&V" or p.name = "SP.A")
RETURN *',
"c:/users/hotze/desktop/subset99extra.graphml", {})
```

References

[1] Sotiris Kotsiantis and Dimitris Kanellopoulos. Cascade generalisation for ordinal problems. *International Journal of Artificial Intelligence and Soft Computing*, 2(1/2):46–57, 2010.

[2] Arend Lijphart. *Democracy in plural societies: A comparative exploration*. Yale University Press, 1977.