BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Machine Learning 2
Project Report

Ilia Ozhmegov
Olena Horyn
Yusif Ifraimov

Berlin 2021

Contents

Introduction 3

Data set 4S

Feature Selection 7

Exploratory Data Analysis 9

Models Selection / Description 14

Testing Performance 34

Conclusion 35

Resources 36

Introduction

Customer churn is one of the biggest challenges any business can possibly be faced with. Losing customers means losing revenue and sales income. This is why it is very important for business to be able to predict which clients might be at risk of leaving the company.
According to Forbes, there are at least three main fiscal reasons why businesses would like to retain their customers:

- Even a 5% reduction in churn rate can increase profits anywhere from 25% to 95%.
- It costs five to seven times more to acquire new customers.
- The probability of selling new products or services to existing customers is 60% to 70%, versus the 5% to 20% chance of selling to a new customer.

In our research we try to tackle this paramount business problem by trying to predict customer churn. We do so by applying and comparing two Machine Learning Methods: Random Forest and Support Vector Machine (SVM). Also, we compare models trained with reduced number of variables after the feature selection step to show that correlation value could be used as a feature selector as well as Boruta approach.

Data set

We use Telecom Churn Dataset in our research. This is a data of a Telecom company Orange of their USA customers and it is publicly available via Kaggle.com.
The Orange Telecom's Churn Dataset, which consists of cleaned customer activity data (features), along with a churn label specifying whether a customer has canceled the subscription or not.

Two datasets are made available: the “churn-bigml-80” and “churn-bigml-20”. The two sets are from the same batch, but have been split by an 80/20 ratio. Furthermore, in our research, we split the “churn-bigml-80” data set by another 75/25 ratio for training and validation purposes. Hence we have 60% for training, 20% for validation and 20% of data for testing.

The "churn-bigml-80" data set contains 2666 observations, each representing a customer, and 20 variables (features). The "churn-bigml-20" data set contains 667 observations (customers) and 20 variables (features). The "Churn" column is the target variable that we would like to predict. It has two levels: “True” for confirmed churn (customer has left the company) and “False” for retained customer (stayed with company).

The data sets have the following attributes or features:

State: *factor*
Account length: *integer*
Area code: *integer*
International plan: *factor*
Voicemail plan: *factor*
Number vmail messages: *integer*
Total day minutes: *double*
Total day calls: *integer*
Total day charge: *double*
Total eve minutes: *double*
Total eve calls: *integer*
Total eve charge: *double*
Total night minutes: *double*
Total night calls: *integer*
Total night charge: *double*
Total intl minutes: *double*
Total intl calls: *integer*
Total intl charge: *double*
Customer service calls: *integer*
Churn: *factor*

As part of our data preparation process, we convert our variable names to all lower case names (for readability) and variable types:

```
> df %>% str(max.level=1)
'data.frame':   2666 obs. of  20 variables:
 $ state                 : Factor w/ 51 levels "AK","AL","AR",..: 17 36 32 36 37 
 ...
 $ account.length        : int  128 107 137 84 75 118 121 147 141 74 ...
 $ area.code             : int  415 415 415 408 415 510 510 415 415 415 ...
 $ international.plan    : Factor w/ 2 levels "No","Yes": 1 1 1 2 2 2 1 2 2 1 ...
 $ voice.mail.plan       : Factor w/ 2 levels "No","Yes": 2 2 1 1 1 1 2 1 2 1 ...
 $ number.vmail.messages : int  25 26 0 0 0 0 24 0 37 0 ...
 $ total.day.minutes     : num  265 162 243 299 167 ...
 $ total.day.calls       : int  110 123 114 71 113 98 88 79 84 127 ...
 $ total.day.charge      : num  45.1 27.5 41.4 50.9 28.3 ...
 $ total.eve.minutes     : num  197.4 195.5 121.2 61.9 148.3 ...
 $ total.eve.calls       : int  99 103 110 88 122 101 108 94 111 148 ...
 $ total.eve.charge      : num  16.78 16.62 10.3 5.26 12.61 ...
 $ total.night.minutes   : num  245 254 163 197 187 ...
 $ total.night.calls     : int  91 103 104 89 121 118 118 96 97 94 ...
 $ total.night.charge    : num  11.01 11.45 7.32 8.86 8.41 ...
 $ total.intl.minutes    : num  10 13.7 12.2 6.6 10.1 6.3 7.5 7.1 11.2 9.1 ...
 $ total.intl.calls      : int  3 3 5 7 3 6 7 6 5 5 ...
 $ total.intl.charge     : num  2.7 3.7 3.29 1.78 2.73 1.7 2.03 1.92 3.02 2.46 ..
 $ customer.service.calls: int  1 1 0 2 3 0 3 0 0 0 ...
 $ churn                 : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1 1 1
```

We also need to make sure that the proportion of entries for each class in the “churn” variable is the same for training/validation data frame and for testing one.

```
>     df      %>% quantity_stats()
# A tibble: 2 x 3
  churn     n n.prop
  <fct> <int>  <dbl>
1 False  2278  0.854
2 True    388  0.146
>     test_df %>% quantity_stats()
# A tibble: 2 x 3
  churn     n n.prop
  <fct> <int>  <dbl>
1 False   572  0.858
2 True     95  0.142
```

There are 2278 “False” churns (customers who did not leave/stayed with the company) and 388 “True” churns (where customers actually left the company) in the train data frame. This is about 85% and 15% respectively. Test data frame contains 572 (86%) of “False” churns and 95 (14%) of “True” churns. So the representation of entries for each class is proportional in each data frame.

Also, this indicates that we are dealing with an unbalanced data (low number of True's and high number of False's). We have to keep this in mind during the modelling process and choosing appropriate metrics before interpreting the results.

The data comes clean and should not contain any missing values. As a part of the analysis, we check our data for any missing values anyway to make sure of it. We confirm that there are none.

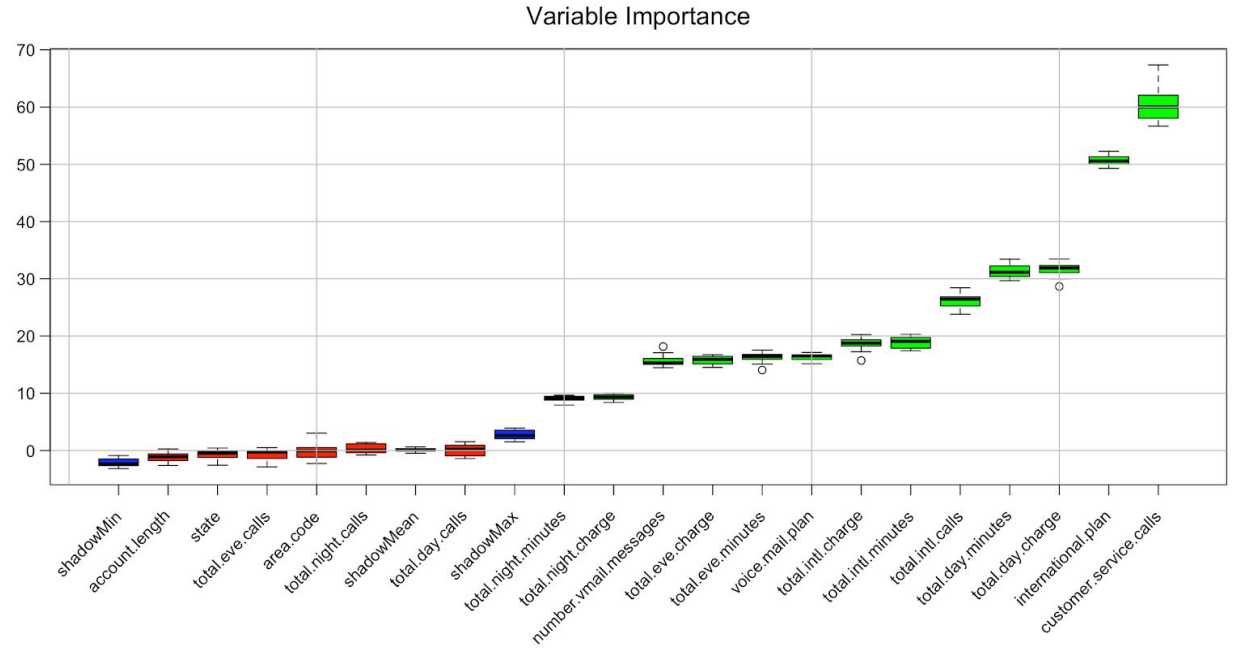
```
> df %>% summarise_all(~sum(is.na(.))) # missingness per feature: no missingness
  state account.length area.code international.plan voice.mail.plan
1     0              0         0                  0               0
  number.vmail.messages total.day.minutes total.day.calls total.day.charge
1                     0                 0               0                 0
  total.eve.minutes total.eve.calls total.eve.charge total.night.minutes
1                 0               0                0                   0
  total.night.calls total.night.charge total.intl.minutes total.intl.calls
1                 0                   0                  0                 0
  total.intl.charge customer.service.calls churn
1                 0                     0     0
```

Feature Selection

Feature selection is the process of choosing variables that are useful in predicting the target variable Y. Since we have 20 predictor variables, not all 20 are equally useful (have the same effect) in predicting outcome variable “churn”.

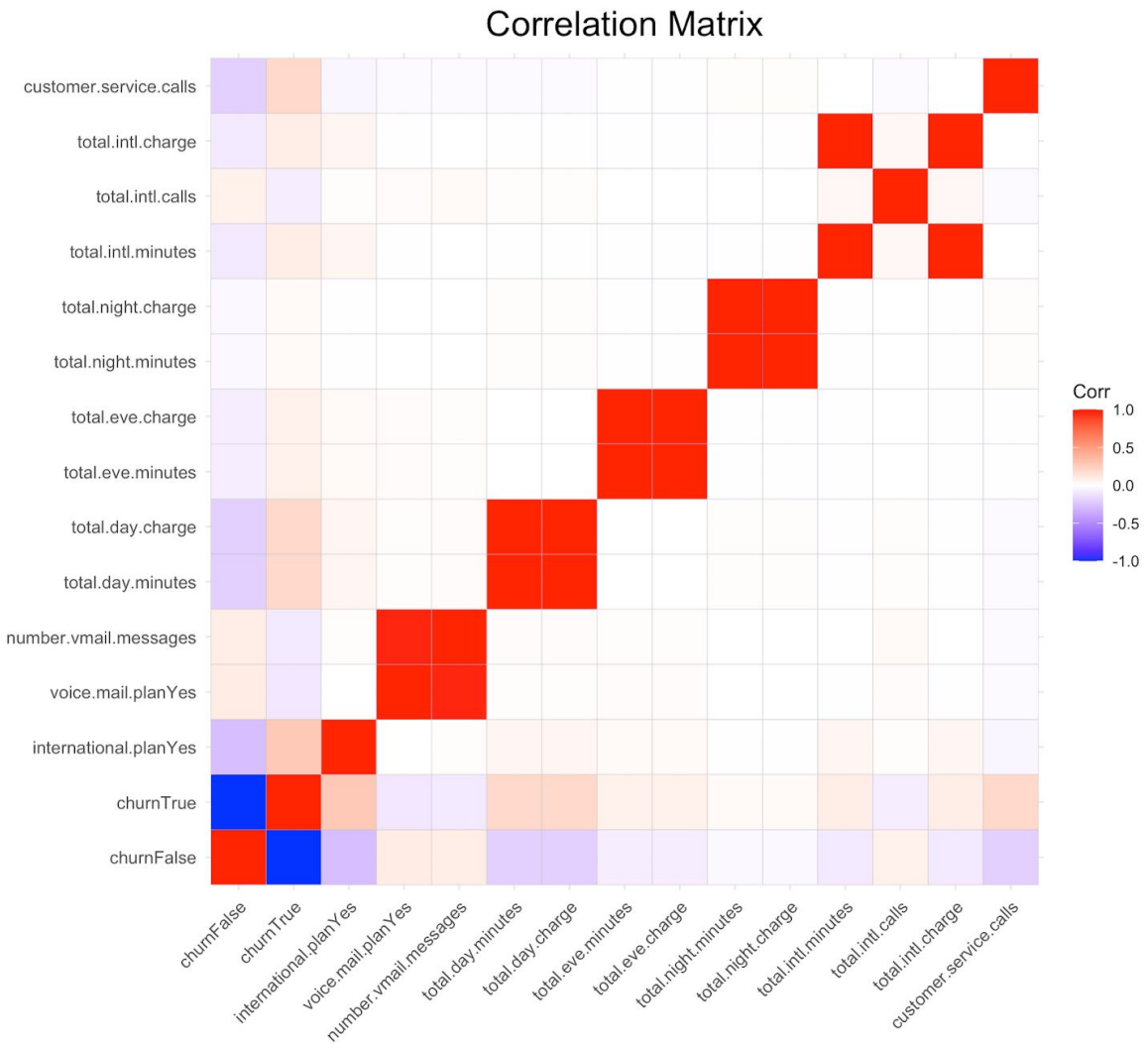
We use the Boruta algorithm and Correlation step function for this. Boruta is a feature ranking and selection algorithm based on a random forest algorithm.
The advantage with Boruta is that it clearly decides if a variable is important or not and helps to select variables that are statistically significant. Besides, it allows to adjust the strictness of the algorithm by adjusting the p values that defaults to 0.01 and the maxRuns.

After running it, the algorithm confirms 13 attributes as important and 6 attributes as unimportant. Amongst an unimportant features are: account.length, area.code, state, total.day.calls, total.eve.calls and total.night.calls. We can drop them and use only the remaining 13 confirmed features. Below is the visualization of the results.



The columns in green are confirmed and the ones in red are not. There are a couple of blue bars representing “ShadowMax” and “ShadowMin”. They are not actual features, but are used by the boruta algorithm to decide if a variable is important or not.
Features "international.plan" and "customer.service.calls" have the highest importance.

After dropping an unimportant feature, we can draw a correlation matrix.



From the Correlation Matrix plot, we can see that all the *.minutes and *.charge variables have a high correlation with respect to each other, so they can be removed to.

Therefore we now have only 9 variables left which we save in a new data frame “narrow_df”. Meanwhile the data frame that has 14 features is saved as “wide_df”. It will be done to show that a method of excluding highly correlated features can be used as a feature selection act as well as Boruta algorithm.

Exploratory Data Analysis

Problem Description.

We have a binomial Classification task, since our goal is to predict one of the two possible Churn classes: class “True” for confirmed/positive churn and class “False” otherwise, for every retained customer.

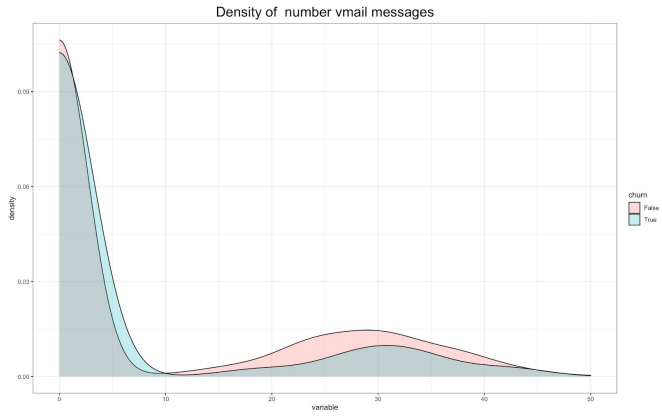
EDA.

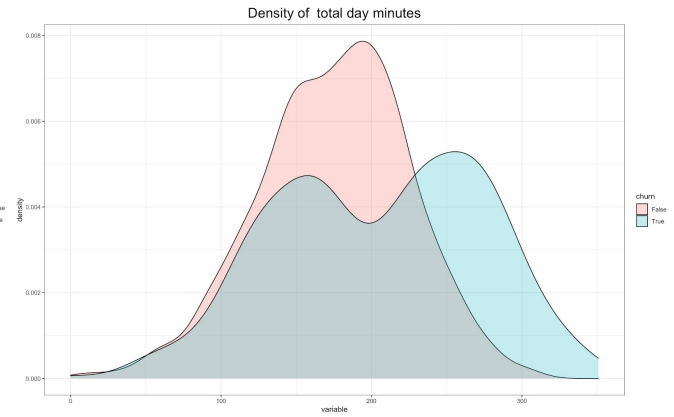
We first look at the summary of our data frame to investigate it:

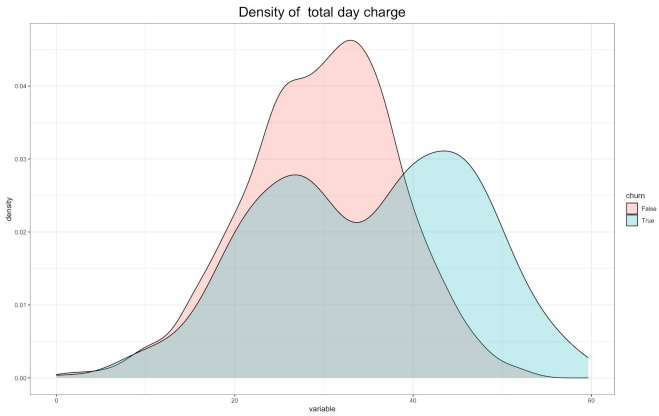
```
> summary(df)
     state      account.length    area.code     international.plan voice.mail.plan
 WV     :  88   Min.   :  1.0   Min.   :408.0   No :2396           No :1933
 MN     :  70   1st Qu.: 73.0   1st Qu.:408.0   Yes: 270           Yes: 733
 NY     :  68   Median :100.0   Median :415.0
 VA     :  67   Mean   :100.6   Mean   :437.4
 AL     :  66   3rd Qu.:127.0   3rd Qu.:510.0
 OH     :  66   Max.   :243.0   Max.   :510.0
 (Other):2241
 number.vmail.messages total.day.minutes total.day.calls total.day.charge
 Min.   : 0.000        Min.   :  0.0     Min.   :  0.0   Min.   : 0.00
 1st Qu.: 0.000        1st Qu.:143.4     1st Qu.: 87.0   1st Qu.:24.38
 Median : 0.000        Median :179.9     Median :101.0   Median :30.59
 Mean   : 8.022        Mean   :179.5     Mean   :100.3   Mean   :30.51
 3rd Qu.:19.000        3rd Qu.:215.9     3rd Qu.:114.0   3rd Qu.:36.70
 Max.   :50.000        Max.   :350.8     Max.   :160.0   Max.   :59.64

 total.eve.minutes total.eve.calls total.eve.charge total.night.minutes
 Min.   :  0.0     Min.   :  0     Min.   : 0.00    Min.   : 43.7
 1st Qu.:165.3     1st Qu.: 87     1st Qu.:14.05    1st Qu.:166.9
 Median :200.9     Median :100     Median :17.08    Median :201.2
 Mean   :200.4     Mean   :100     Mean   :17.03    Mean   :201.2
 3rd Qu.:235.1     3rd Qu.:114     3rd Qu.:19.98    3rd Qu.:236.5
 Max.   :363.7     Max.   :170     Max.   :30.91    Max.   :395.0

 total.night.calls total.night.charge total.intl.minutes total.intl.calls
 Min.   : 33.0     Min.   : 1.970     Min.   : 0.00      Min.   : 0.000
 1st Qu.: 87.0     1st Qu.: 7.513     1st Qu.: 8.50      1st Qu.: 3.000
 Median :100.0     Median : 9.050     Median :10.20      Median : 4.000
 Mean   :100.1     Mean   : 9.053     Mean   :10.24      Mean   : 4.467
 3rd Qu.:113.0     3rd Qu.:10.640     3rd Qu.:12.10      3rd Qu.: 6.000
 Max.   :166.0     Max.   :17.770     Max.   :20.00      Max.   :20.000
```

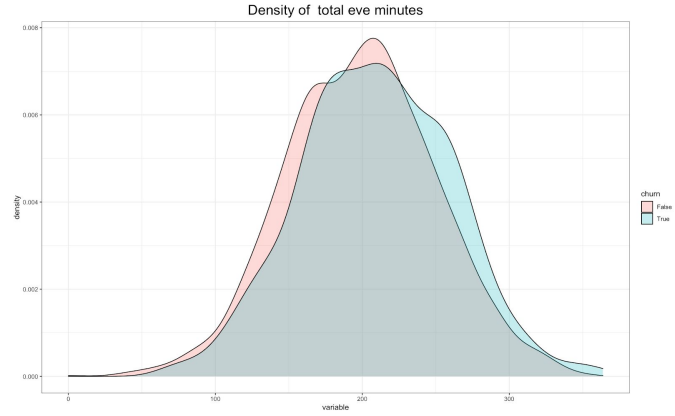
```
total.intl.charge customer.service.calls   churn
Min.   :0.000     Min.   :0.000          False:2278
1st Qu.:2.300     1st Qu.:1.000          True : 388
Median :2.750     Median :1.000
Mean   :2.764     Mean   :1.563
3rd Qu.:3.270     3rd Qu.:2.000
Max.   :5.400     Max.   :9.000
```

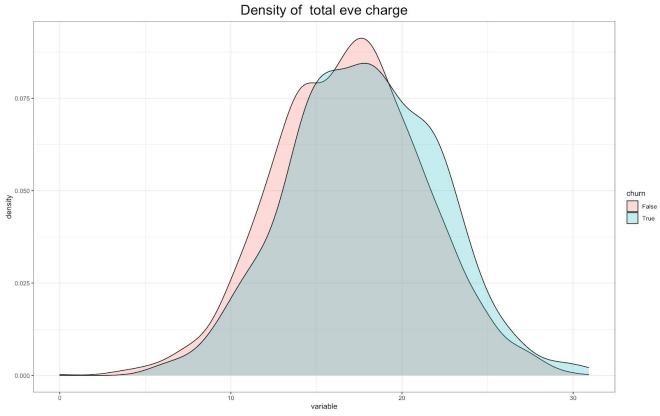
To find out if there is a skewness in the data we will draw density plots target variable against every predictor variable.

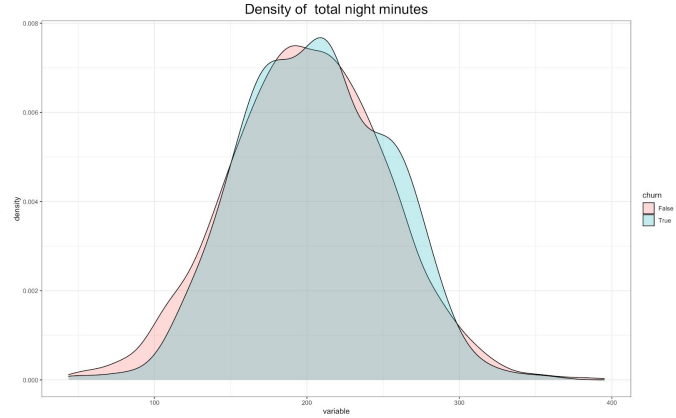


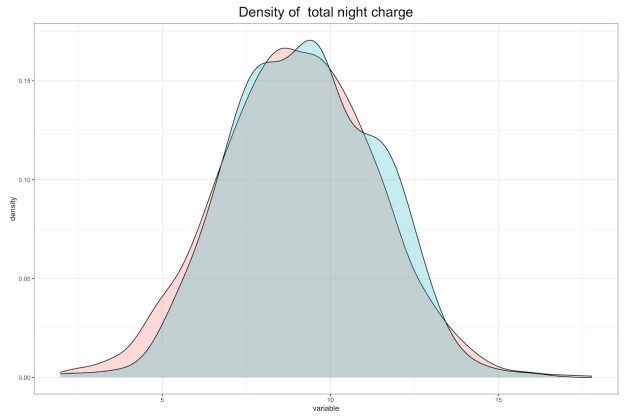


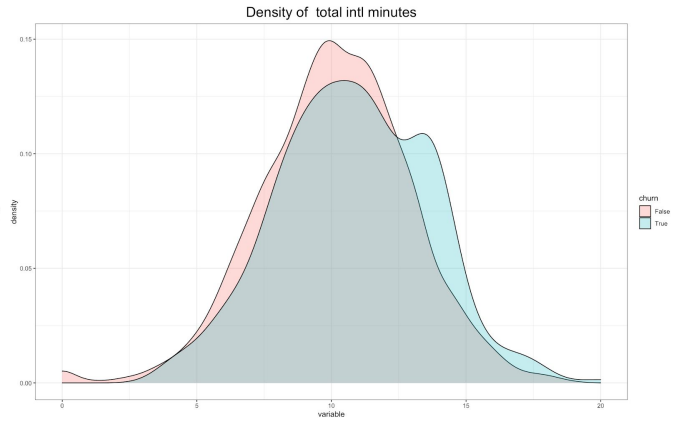


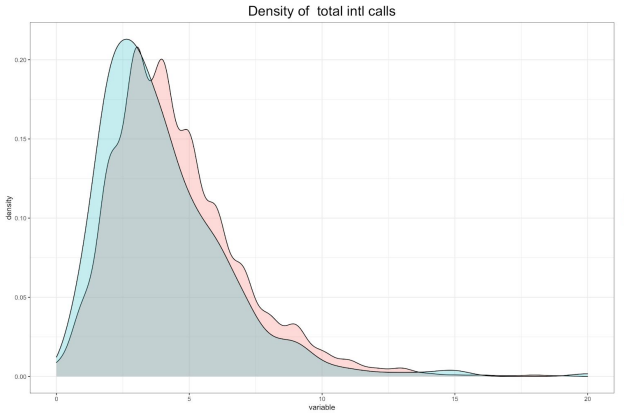


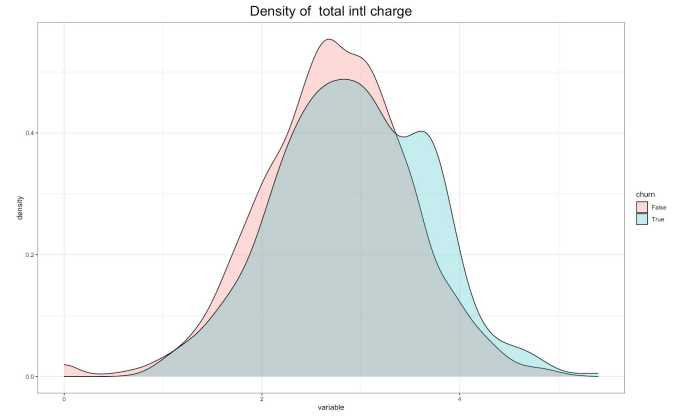


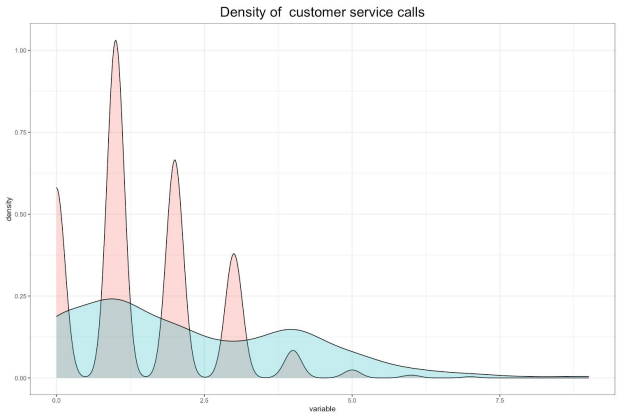


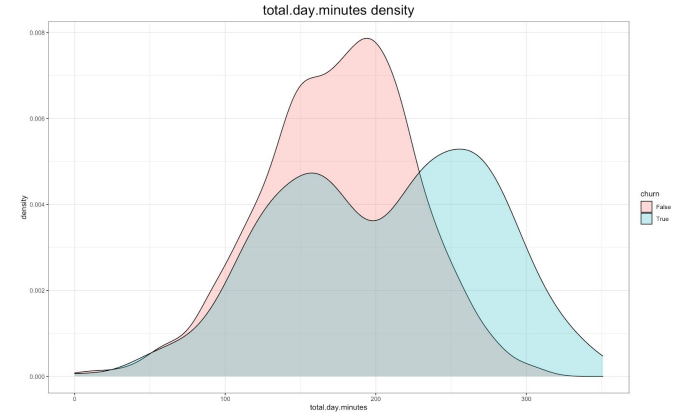

Density of total night charge
density
variable
churn
False
True

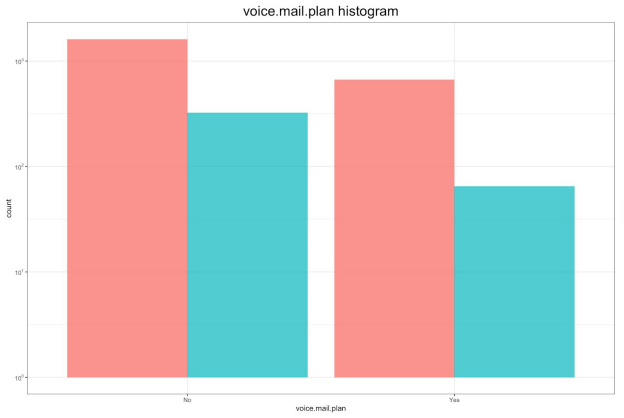

Density of total intl minutes
density
variable
churn
False
True


Density of total intl calls
density
variable
churn
False
True


Density of total intl charge
density
variable
churn
False
True


Density of customer service calls
density
variable
churn
False
True


total.day.minutes density
density
total.day.minutes
churn
False
True


voice.mail.plan histogram
count
voice.mail.plan
No
Yes
churn
False
True

We can observe that almost all our variables are equally distributed. Some variables are normally distributed, such as “total.eve.minutes”, “total.eve.charge”, “total.night.minutes”, “total.night.charge”, “total.intl.minutes”, “total.intl.charge”.

Few variables are highly skewed (positive skewness): “number.vmail.messages” and “customer.service.calls” (especially for “False” churn). Although it should not be an issue for our Random Forests model, since it is robust to outliers, but for our SVM model it might be, since this algorithm typically is less robust to outliers. This means that we might get not very accurate results.

The histogram for discrete variables indicates almost equal distribution (slightly more for plan “no”).

During our initial Data Analysis, we looked into the “states” variable. We were curious to find out what states were with the highest number of customers that left and the ones with the lowest churn rate.

state	n	n_churn	churn.prop
TX	55	16	0.290909090909091
NJ	50	14	0.28
AR	47	11	0.234042553191489
MD	60	14	0.233333333333333
MS	48	11	0.229166666666667
ME	49	11	0.224489795918367
SC	49	11	0.224489795918367
MI	58	13	0.224137931034483
PA	36	8	0.222222222222222
NV	61	13	0.213114754098361
NH	43	9	0.209302325581395
CA	24	5	0.208333333333333
WA	48	10	0.208333333333333
KS	52	10	0.192307692307692
MT	53	10	0.188679245283019
CT	59	11	0.186440677966102
MN	70	13	0.185714285714286
NY	68	12	0.176470588235294
GA	49	8	0.163265306122449
NC	56	9	0.160714285714286
DE	51	8	0.156862745098039
MA	52	8	0.153846153846154
OH	66	10	0.151515151515152
KY	43	6	0.13953488372093
OK	52	7	0.134615384615385
UT	60	8	0.133333333333333
FL	54	7	0.12962962962963
SD	49	6	0.122448979591837
TN	41	5	0.121951219512195
WY	66	8	0.121212121212121

CO	59	7	0.11864406779661
OR	62	7	0.112903225806452
DC	45	5	0.111111111111111
IN	54	6	0.111111111111111
AL	66	7	0.106060606060606
VT	57	6	0.105263157894737
MO	51	5	0.0980392156862745
ND	44	4	0.0909090909090909
NM	44	4	0.0909090909090909
ID	56	5	0.0892857142857143
IL	45	4	0.0888888888888889
NE	45	4	0.0888888888888889
LA	35	3	0.0857142857142857
WV	88	7	0.0795454545454545
IA	38	3	0.0789473684210526
AK	43	3	0.0697674418604651
AZ	45	3	0.0666666666666667
WI	61	4	0.0655737704918033
RI	48	3	0.0625
VA	67	4	0.0597014925373134
HI	44	2	0.0454545454545455

Since during the Feature Selection process, variable “state” was confirmed as unimportant, it was dropped from the model. And we did not include this in our final R script file.

Models Selection / Description

1. Random Forest.

Model overview.

Random Forest is a machine learning method that is based on generating a large number of individual decision trees, each constructed using a different subset of the training set. These subsets are selected by sampling at random and with replacement from the original data set, and this approach is called Bootstrapping.

Random Forests can be used for Regression and Classification problems. Since we are dealing with a Classification Problem, we will use Random Forests Classifier.

Another reason for why we have selected this model is its interpretability. And this is also why random forest models are being widely used and often preferred over other models such as neural networks.

The way the algorithm works when building individual decision trees, is that each time a split in a tree is considered, a random sample (with replacement) of *m* variables is chosen from the full set of all available *p* variable predictors. Only one of these *m* predictors is used for a split. A new sample of *m* variables is chosen at each split. The general criterion for calculating the best number of *m* is approximately equal to the square root of the total number of *p* variables:

$m \approx \sqrt{p}$ or $m = \left[\sqrt{p}\right]$ (rounding down the square root)

In our case, the wide data frame has 14 variables, so our *m* = 3, whereas the narrow data frame has 9 variables therefore m = 3 as well.

Using a small value of *m* in building random forests is an advantage when dealing with a large number of correlated variables. Since the predictions from the regular bagged trees are usually highly correlated, random forests overcome this problem by having each split to consider only a subset of the variables. This creates a low correlation between models/trees. Uncorrelated trees can produce ensemble predictions that are more accurate than any of the individual tree predictions.

The Class prediction decision by using Random Forests classifier can be demonstrated on figure 1.

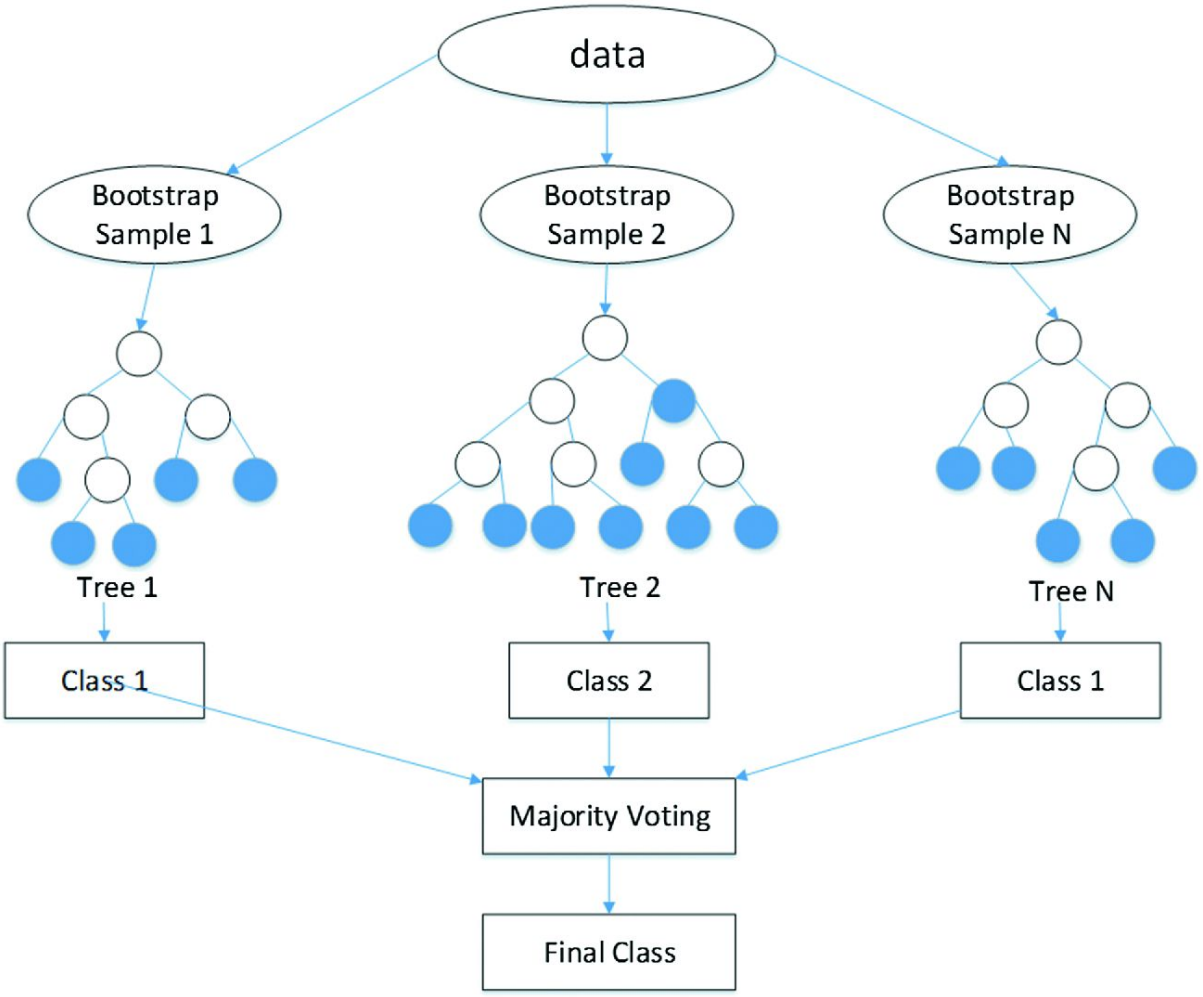


Figure 1. Random Forests Classifier.

Interpretation of results.

During the Feature selection process we created two data frames. A “narrow_df” - a data frame with 8 most important predictor variables (9 variables in total, including an outcome variable). And “wide_df” - a data frame with 13 most important predictor variables (14 variables in total, including an outcome variable). In this section, we investigate and compare the performance and results of the Random Forests model on both data frames. We also use 10-fold cross-validation.

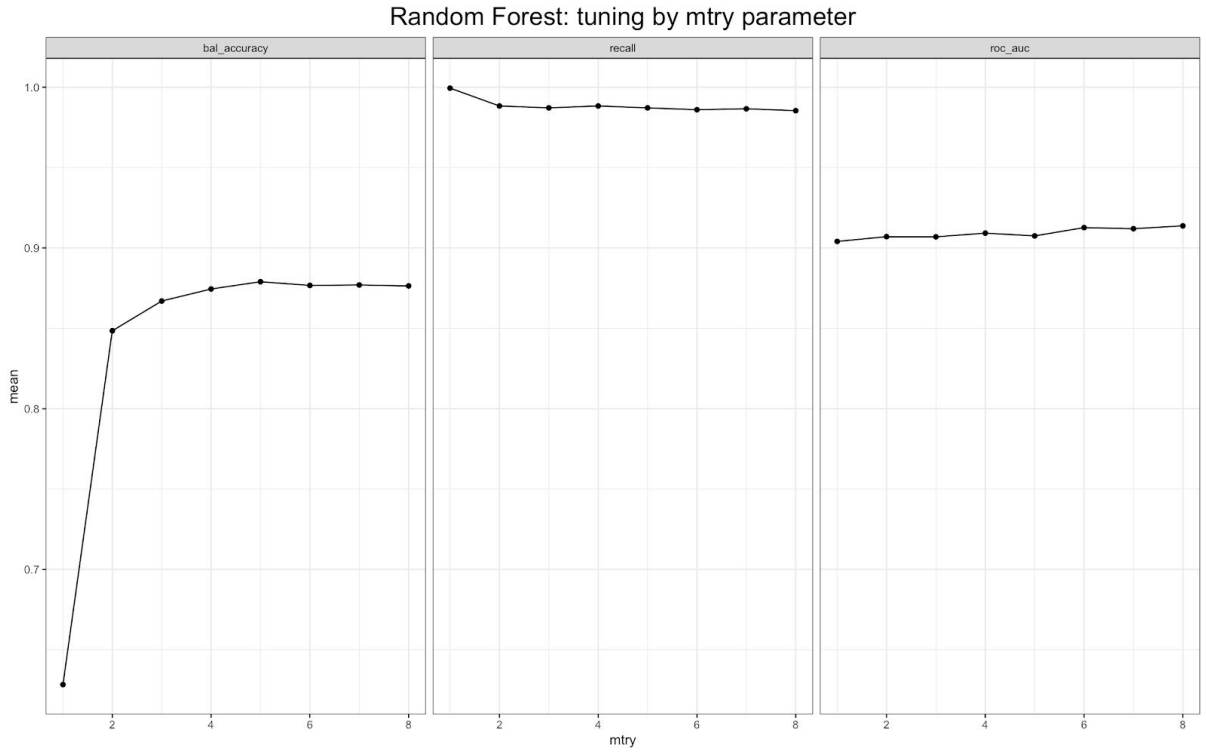
We start with defining the recipe which is part of the ‘recipes’ package. A recipe is a description of what steps should be applied to a data set in order to get it ready for data analysis or model tuning. The idea of the recipes package is to define a blueprint that can be used to sequentially define the encodings and preprocessing of the data.

We use the “rand_forest()” function for our Random Forests model. It has three main arguments:

- *mtry:* The number of predictors that will be randomly sampled at each split when creating the tree models. Below we proof why the best value for mtry is 3 ($\sqrt{9}$).
- *trees:* The number of trees contained in the ensemble. We will tune from 125 to 500 trees.
- *min_n:* The minimum number of data points in a node that are required for the node to be split further. It will be tuned from 2 to 8.

We also use the ranger version of Random Forest with the set_engine() function and set mode as “classification”.

We tune the mtry parameter and investigate if the mtry = $\sqrt{9}$ = 3 is the best number.



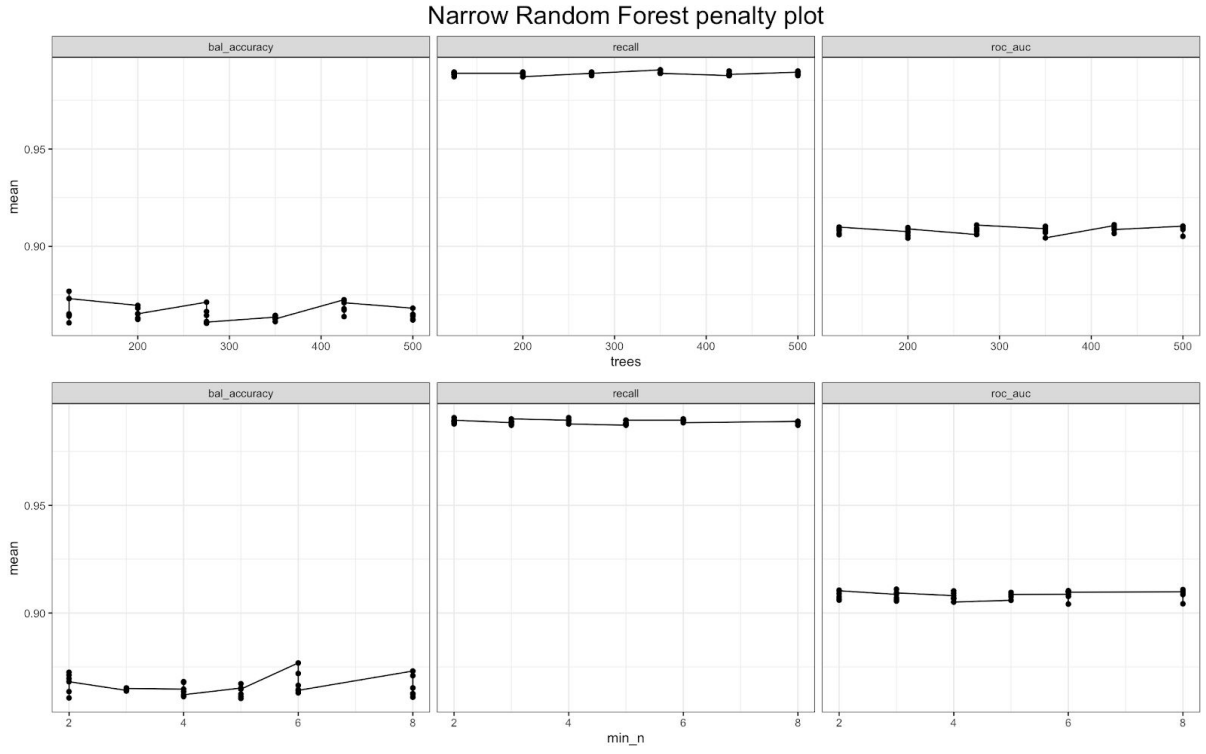
The “bal_accuracy” parameter clearly demonstrates that mtry more than 3 does not bring a significant improvement. The same is true for our Wide model, since mtry = $\sqrt{14}$ would be 3 after rounding down. So we use mtry = 3 for both our models (Narrow and Wide).

After we train our model, we can look at the summary.

```
[[1]]
══ Workflow [trained] ══════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: rand_forest()

── Preprocessor ────────────────────────────────────────────────────────────────
0 Recipe Steps

── Model ───────────────────────────────────────────────────────────────────────
Ranger result

Call:
 ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~m,      x), num.trees = ~42
5L, min.node.size = min_rows(~3L, x),      importance = ~"impurity", num.threads = 1, verbos
e = FALSE,      seed = sample.int(10^5, 1), probability = TRUE)

Type:                             Probability estimation
Number of trees:                  425
Sample size:                      2000
Number of independent variables:  8
Mtry:                             3
Target node size:                 3
Variable importance mode:         impurity
Splitrule:                        gini
OOB prediction error (Brier s.):  0.0438291
```

Our model fitted 425 trees out of 2000. Number of mtry is 3, number of explanatory variables are considered for generating trees - 8. Trees where the nodes separate the different categories well, will have a low Gini index. And Out of Bag (OOB) prediction error is 0.043.

The plot of the results for “narrow_df” for validation predictions is below.



From the above penalty plot, we can observe that the balanced accuracy is below 85% and recall is relatively high, which suggests a good prediction of the majority class. We then choose our hyperparameters by the area under the ROC curve.

Below is our classification matrix (or confusion matrix) for the Narrow model:

```
          Truth
Prediction False True
     False   559   24
     True      7   76
```

Using this classification matrix we can calculate the accuracy of our model (76+559 / 666 = 0.953) and recall or sensitivity (76 / 24+76 = 0.76).

We can check our calculations and other metrics.

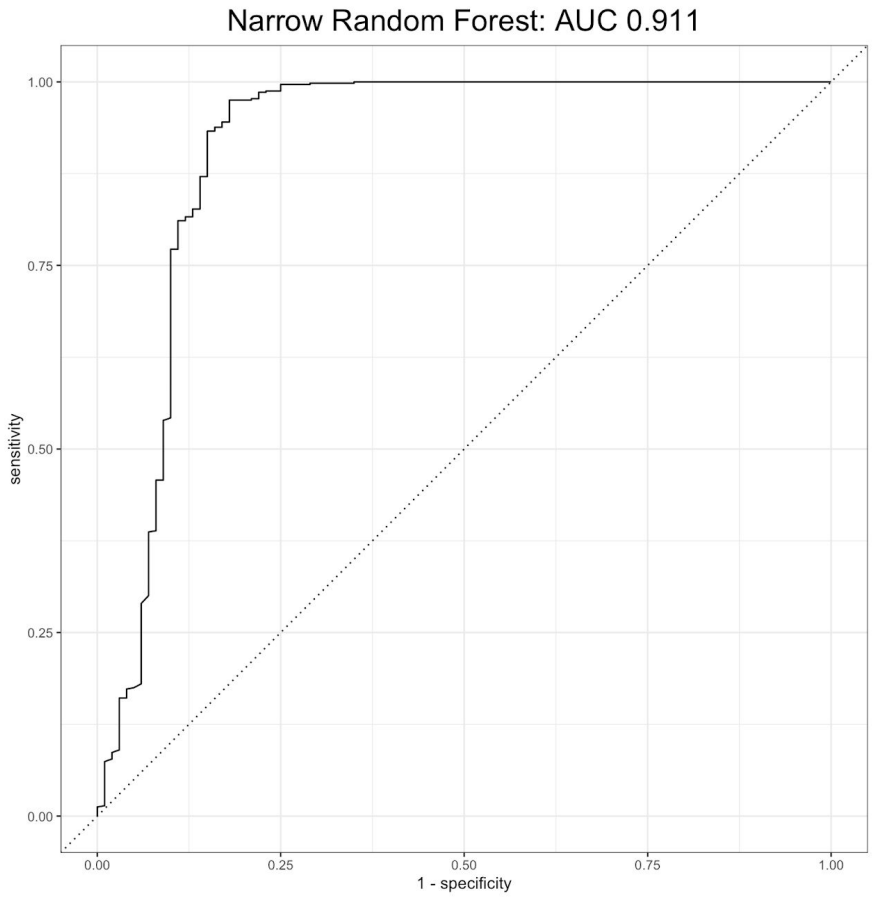
```
# A tibble: 4 x 3
  .metric      .estimator .estimate
  <chr>        <chr>          <dbl>
1 recall       binary         0.76
2 accuracy     binary         0.953
3 bal_accuracy binary         0.874
4 kap          binary         0.804
```

Balanced accuracy of 0.874 is the average of sensitivity and specificity.
The Kappa statistic here (or value) is a metric that compares an Observed Accuracy with an Expected Accuracy (random chance). In our case kappa = 0.804, and is considered to be excellent for kappa > 0.75.

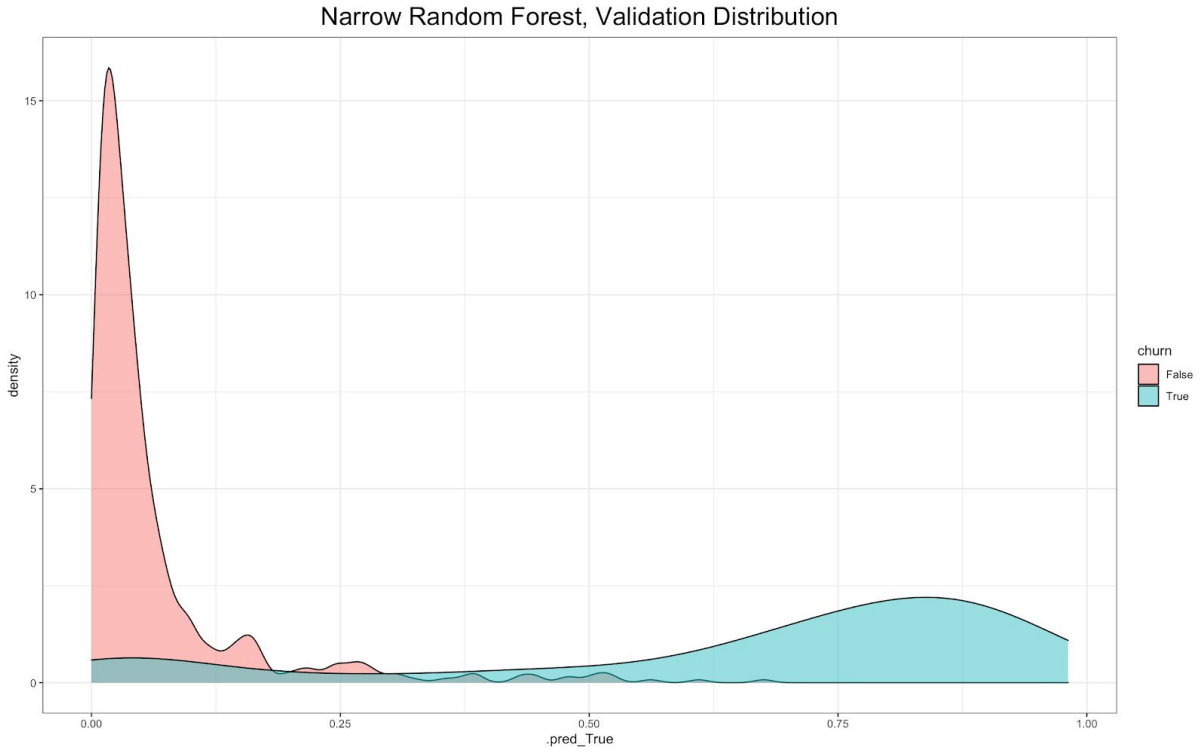
From the confusion matrix we can assume that our Random Forests classifier for our Narrow model is quite good with balanced accuracy of 87.4%, however it is not that good in predicting “True” or positive churn cases with recall rate being 0.76.

Let us also explore the ROC curve (below).

The ROC curve indicates good classification results with the curve being very close to the top left corner. And Area Under the Curve (AUC) is 0.911 which is also quite good.



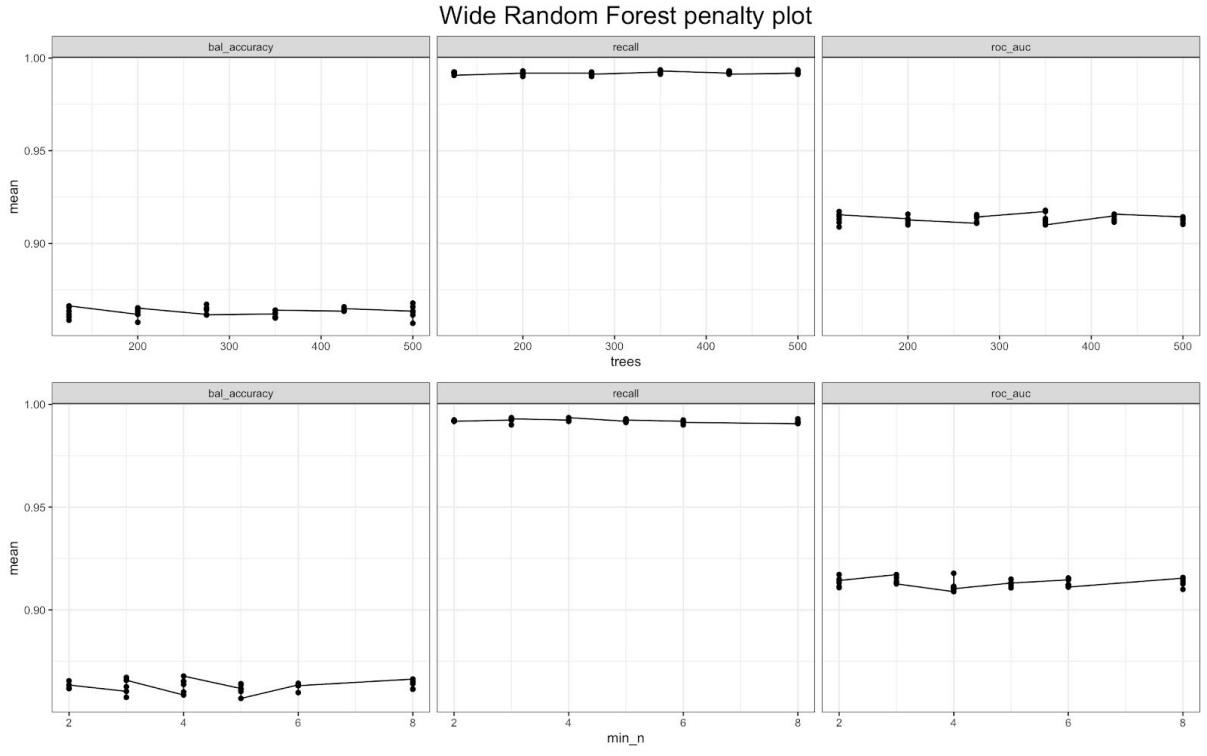
We can also look at the Validation distribution of churn predictions for our narrow model. This plot demonstrates the ability of our model to discriminate well between the two classes.



Next, let's investigate our Wide model with 13 predictor variables and compare its performance to the Narrow model with 8 predictor variables. Model fitted 350 trees, with mrty = 3 and OOB error 0.046 (slightly higher than with narrow model).

```
[[1]]
══ Workflow [trained] ══════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: rand_forest()

── Preprocessor ────────────────────────────────────────────────────────────────
0 Recipe Steps

── Model ───────────────────────────────────────────────────────────────────────
Ranger result

Call:
 ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~m,      x), num.trees = ~350
L, min.node.size = min_rows(~4L, x),      importance = ~"impurity", num.threads = 1, verbose =
 FALSE,      seed = sample.int(10^5, 1), probability = TRUE)

Type:                             Probability estimation
Number of trees:                  350
Sample size:                      2000
Number of independent variables:  13
Mtry:                             3
Target node size:                 4
Variable importance mode:         impurity
Splitrule:                        gini
OOB prediction error (Brier s.):  0.0461716
```

This is the Random Forests penalty plot for the Wide model.



This penalty plot looks very similar to the one of the Narrow model. So it is difficult to evaluate it visually. Instead, let's investigate the classification matrix for the Wide model and its evaluation metrics.

	Truth	
Prediction	False	True
False	560	20
True	9	77

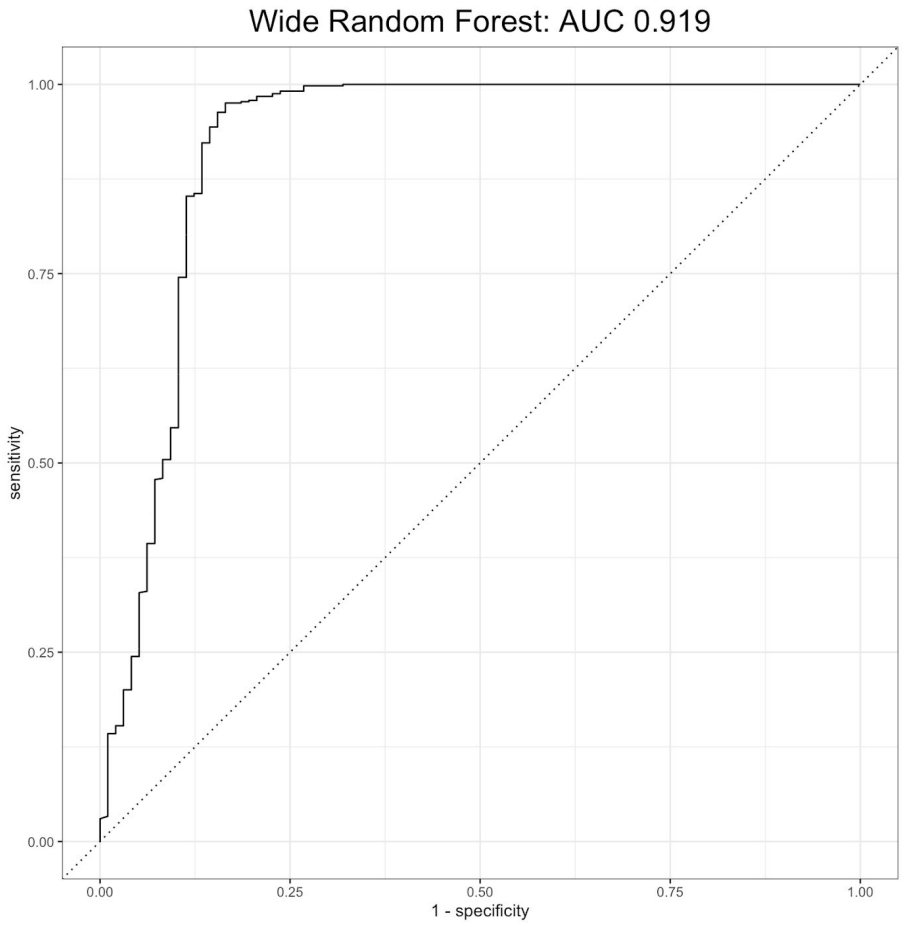
We can instantly see that this Random Forests model has predicted one more True Positive instance (77), than with the previous model. And also has identified one more True Negative (560). This makes a total of misclassified observations 29 (9+20), two more when compared with Narrow model 31 (24+7). A very slight improvement.
But let us look at what our metrics are.

```
# A tibble: 4 x 3
  .metric      .estimator .estimate
  <chr>        <chr>          <dbl>
1 recall       binary         0.794
2 accuracy     binary         0.956
3 bal_accuracy binary         0.889
4 kap          binary         0.816
```

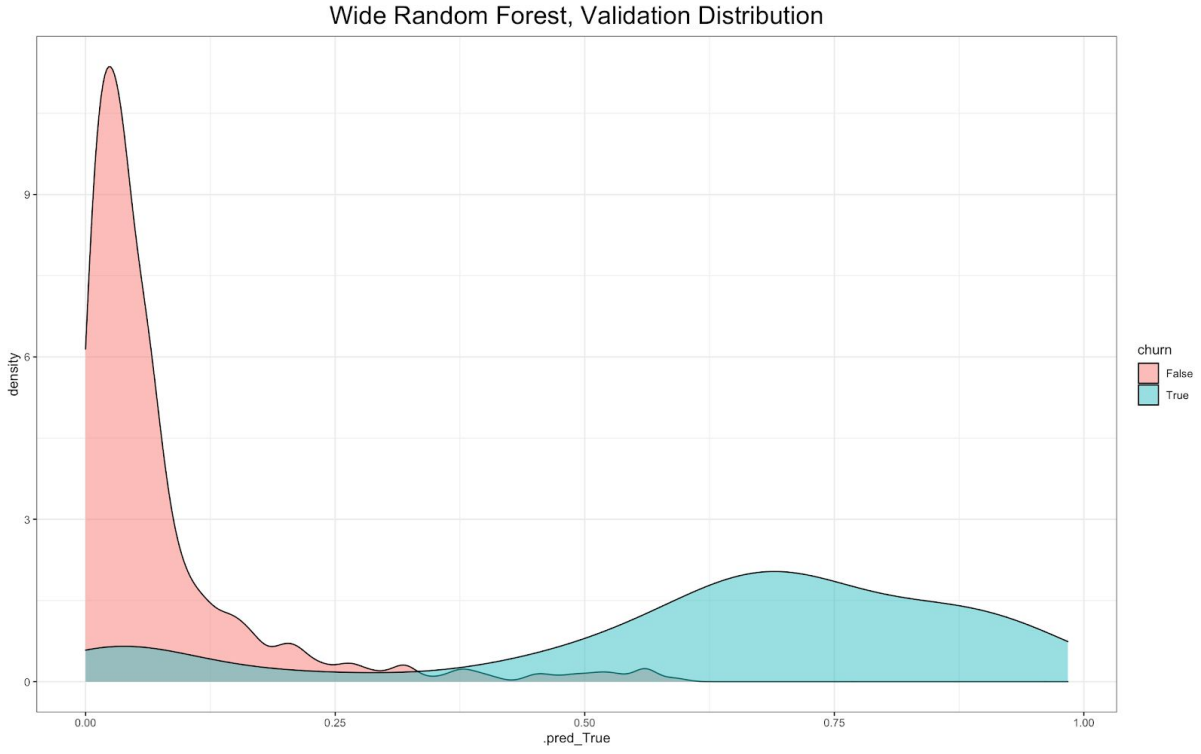
The recall rate for the Wide model is 0.794 (77 / 77+20) and is little higher than for the Narrow model (0.76). We can consider this as an improvement, as our goal is to identify as many positive or "Ture" churns.

The Accuracy rate is slightly higher as well - 0.956 (77+560 / 666) from 0.953 with the Narrow model. And the other two matrices: balanced accuracy (0.889) and kappa (0.816) are a little higher too.

These are very small improvements. And let us see if our the AUC of ROC curve has improved too.



The ROC curve looks very similar to the prior one. However now the AUC = 0.919, which is slightly better than for the Narrow model (0.911).



The Density plot above demonstrates how well our model can discriminate (or differentiate) between two classes. It is usually very confident when predicting a class. When the probability for the observation is high for being “True”, at the same time the probability for it being “False” is low. The overlapping part is very small, which means that there are only a few instances when the model is “unsure” and can’t tell apart e.i. is not confident in making a prediction.

Conclusion.

Random Forest has worked really well for both of our Narrow and Wide models. However it did perform slightly better with the Wide model, which has more predictor variables (13). But that is a very slight improvement compared with the Narrow model and we do no’t gain much when adding more predictor variables that were excluded due to a high correlation. The balanced accuracy rate of the Narrow model is 0.874 and of the Wide model - 0.889. And the recall rate for the Narrow and Wide models is 0.76 and 0.794 respectively.

One of our goals was to see if the increase in the number of predictor variables that were excluded due to a high correlation results in the significantly better performance. Also the concern was that if we remove too many predictor variables - it can have an adverse effect and worsen our results. However, we have established that removal of some “unimportant” variables does nott gain worse results when use correlation.

Our research has proved that in certain cases with Random Forest, the model with more predictor variables does not gain significantly better results. This means that during the Feature Selection process it is sufficient to use Correlation Matrix as feature selector besides such well accepted Boruta algorithm. And that even if we add more “unimportant” variables, a model does not gain much performance increase.

This is not a reason to use a Narrow model, but the reason not to use a Wide model as it is a more complex one without an appropriate difference.

2. Support Vector Machine (SVM).

Model overview.

SVM (Support Vector Machine) is a supervised machine learning method. Support Vector Machine can be used for both regression and classification tasks. But, it is widely used in classification cases, usually as a binary classifier. This makes this method very attractive and appropriate for our case where we have K=2 classes.

The SVM algorithm tries to find a separating hyperplane in an p-dimensional space (p — number of features) that are used to classify the data points. The hyperplane is a boundary between two classes. Since there are many possible hyperplanes that could be chosen to separate our classes. Algorithm's objective is to find an optimal hyperplane that has the maximum margin, i.e the maximum distance (M) between data points of both classes or between the boundary and margin edge, i.e. half the margin width. Maximizing the margin distance M provides some reinforcement so that future data points can be classified with more confidence.
Figure 2 illustrates the possible hyperplanes vs the optimal hyperplane.

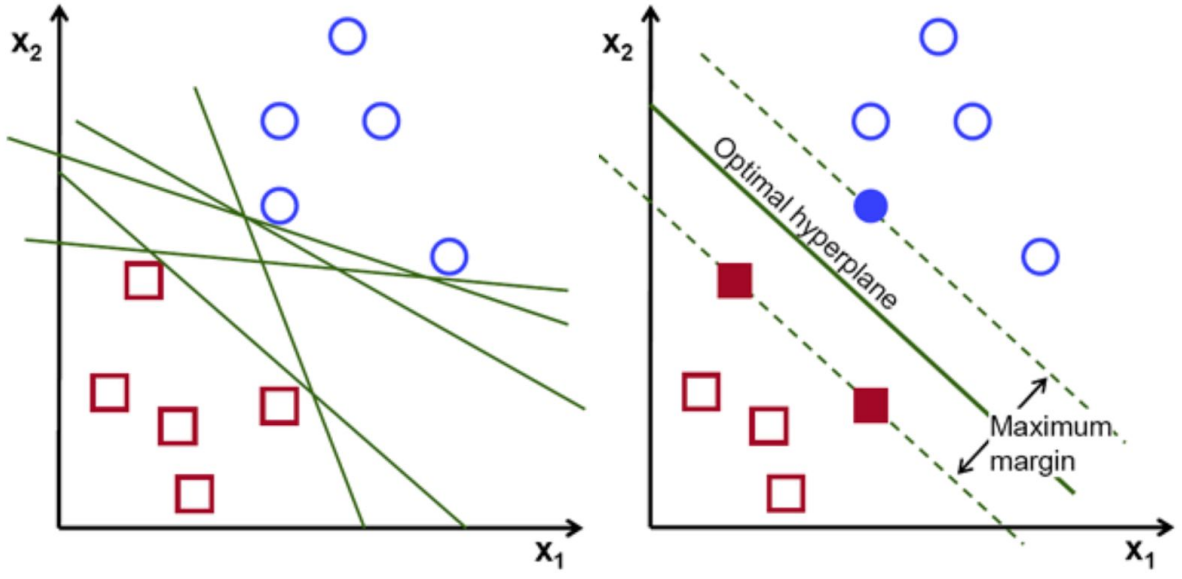


Figure 2. Possible hyperplanes (left) vs optimal hyperplane (right).

The dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane (Figure 3). It becomes difficult to illustrate a hyperplane for higher dimensions when the number of features exceeds 3.

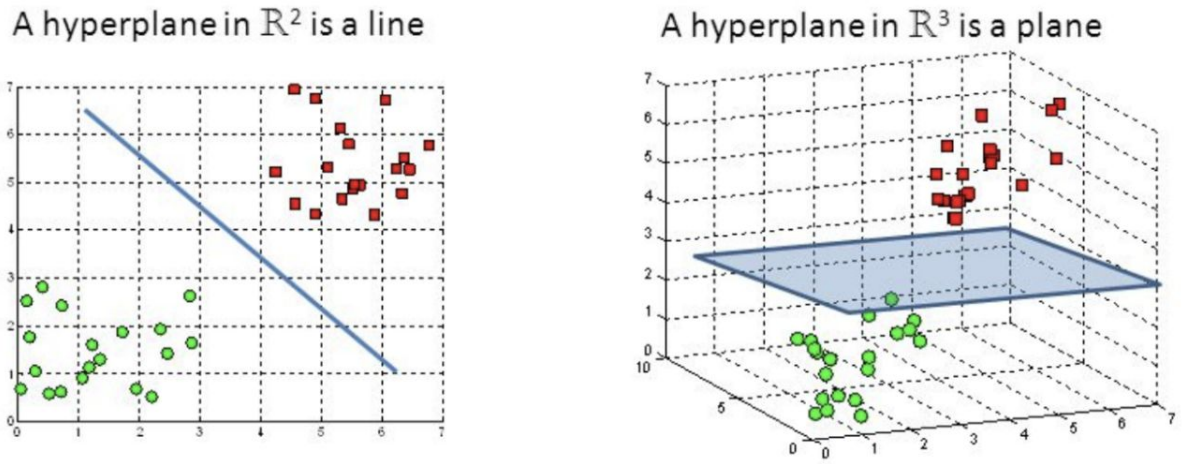


Figure 3. Hyperplanes in 2D and 3D feature space.

If there are training observations that are equally distant from the maximal margin hyperplane and lie along the width of the margin, then these observations are known as support vectors. And since they “support” the hyperplane in the sense that if these points were moved slightly then the hyperplane would move as well.
SVMs address the problem when hyperplane does not perfectly separate the two classes by allowing some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane.

The SVM algorithm tries to find the solution to the optimization problem.

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,M}{\text{maximize}} M$$
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$
$$\epsilon_i \geq 0, \ \sum_{i=1}^{n} \epsilon_i \leq C,$$

Where M is the distance between the boundary and margin edge, C is a positive smoothing parameter, $\epsilon_1, \ldots, \epsilon_n$ are slack variables that allow individual observations to be on the wrong side of the margin or the hyperplane.

Once the best hyperplane has been found, the predictions are obtained based on the sign of the hyperplane function. ie. for a test observation vector **x**t:

$$\widehat{y}_t = \begin{cases} 1 & \text{for } \beta_0 + \beta_1 x_{t1} + \beta_2 x_{t2} \geqslant 0 \\ -1 & \text{for } \beta_0 + \beta_1 x_{t1} + \beta_2 x_{t2} < 0 \end{cases}$$

Slack variable ϵ_i tells us where the ith observation is located, relative to the hyperplane and relative to the margin. If ϵ_i = 0 then the ith observation is on the correct side of the margin. If ϵ_i > 0 then the ith observation is on the wrong side of the margin, and the ith observation has violated the margin. If ϵ_i > 1 then it is on the wrong side of the hyperplane.

We allow some misclassified points but we end up having a more generalized model.

Parameter C (cost) adds a penalty for each misclassified data point. If C is small, the penalty for misclassified points is low, so a decision boundary with a large margin is chosen at the expense of a greater number of misclassifications. If C is large, SVM tries to minimize the number of misclassified examples due to high penalty which results in a decision boundary with a smaller margin. Penalty is not the same for all misclassified examples. It is directly proportional to the distance to the decision boundary.
In practice, C is treated as a tuning parameter that is generally chosen via cross-validation.

There are cases where two classes are not linearly separable. Kernel approach is used for non-linear decision boundary problems. One of the commonly used kernel functions is radial basis function (RBF). Gamma parameter of RBF controls the distance of influence of a single training point. Low values of gamma indicates a large similarity radius which results in more points being grouped together. For high values of gamma, the points need to be very close to each other in order to be considered in the same class.
As the gamma decreases, the regions separating different classes get more generalized. Very large gamma values result in too overfitting.

Overall, SVM method provides a better robustness to individual observations and better classification of most of the training observations.

Interpretation of results.

The overall workflow at this step is the same as described for Random Forests algorithm - we define a recipe, meaning we define a blueprint for our model.

We use *svm_model_poly* and *svm_model_rbf* representing a Polynomial of the 1st Degree and Radial Basis Function - kernel SVM respectively. They have one common argument:

- *cost:* Parameter defines how much an SVM should be allowed to “bend” with the data. It is also simply referred to as the cost of misclassification.

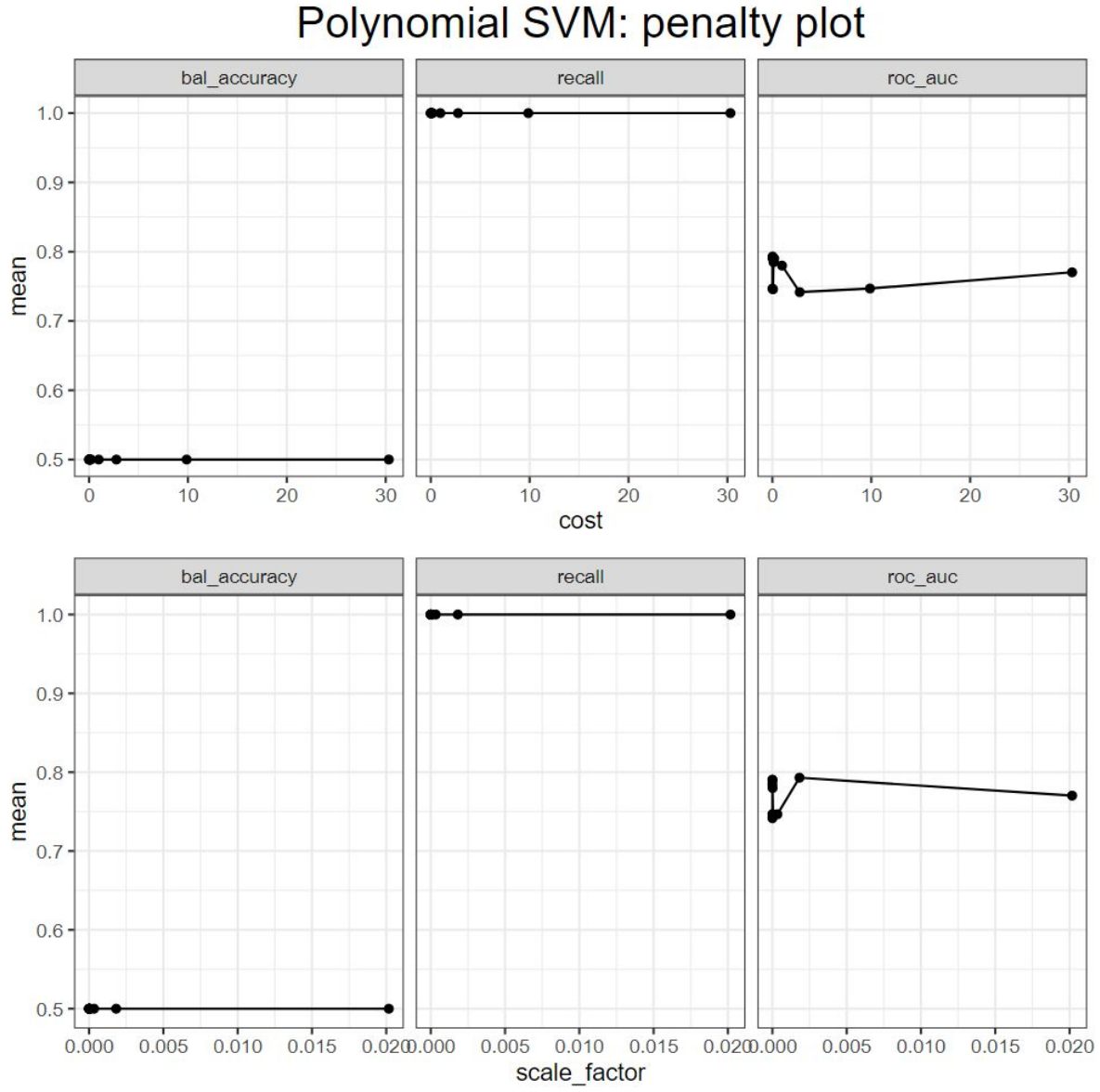
svm_model_poly has the following parameter:

- scale_factor: A scaling factor for the kernel.

svm_model_rbf has the following parameter:

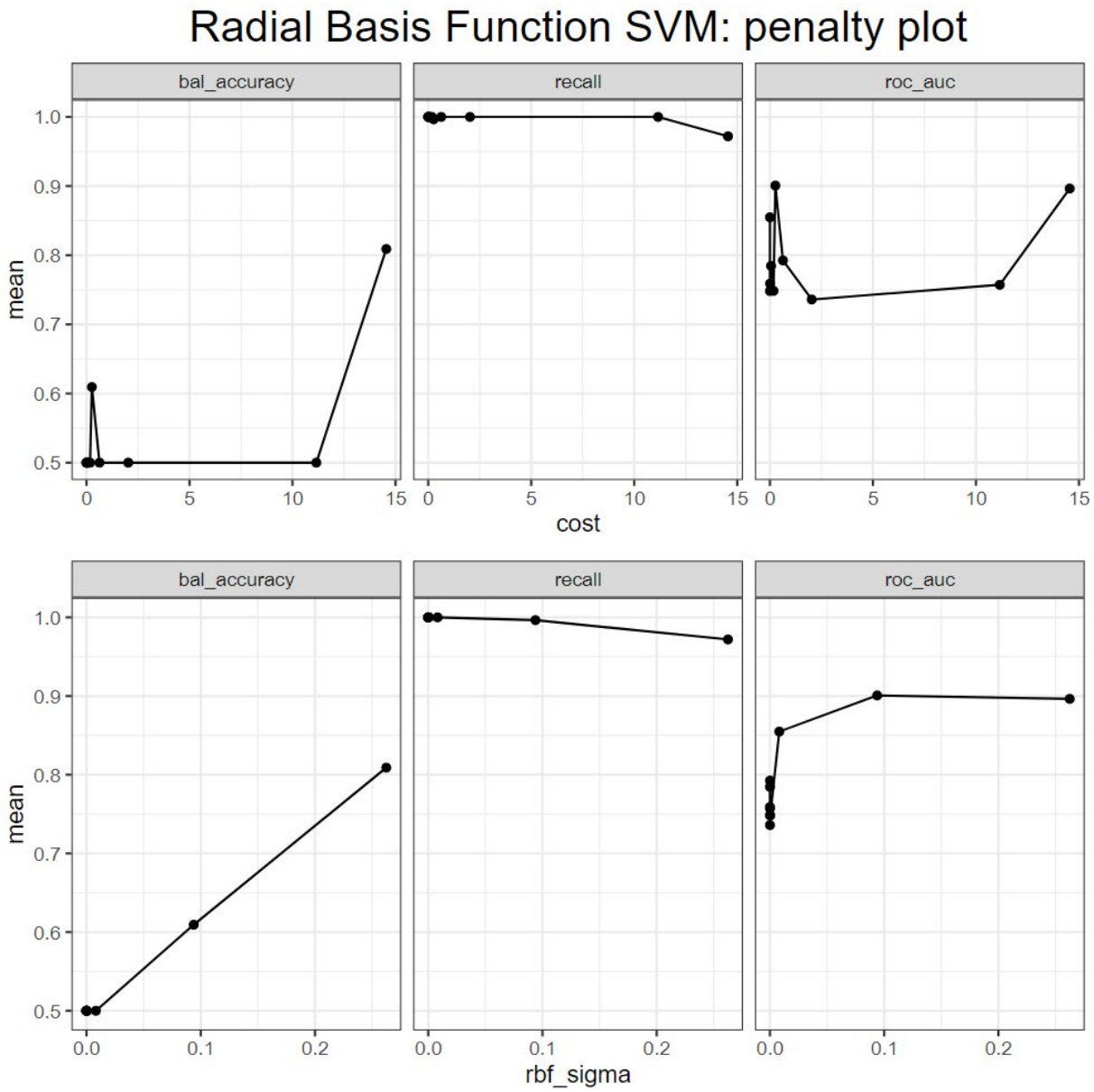
- *rbf_sigma:* determines the reach of a single training instance. If the value of the parameter is low, then every training instance will have a far reach. Conversely, high values mean that training instances will have a close reach.

Starting with *svm_model_poly* we tune the two parameters *cost* and *scale_factor* in order to find out whether that has any significant influence on our accuracy:



Both “bal_accuracy” and “recall” demonstrate that changing both parameters does not bring any noticeable change at all. Looking at “roc_auc” we can notice a “mirror effect” - for *cost* we can see an upward slide while for scale_factor a downward slide followed by increasing and decreasing incline respectively. Ultimately, tuning these two parameters does not provide any significant improvement.

Next we will have a look at *svm_model_rbf* Penalty Plot:

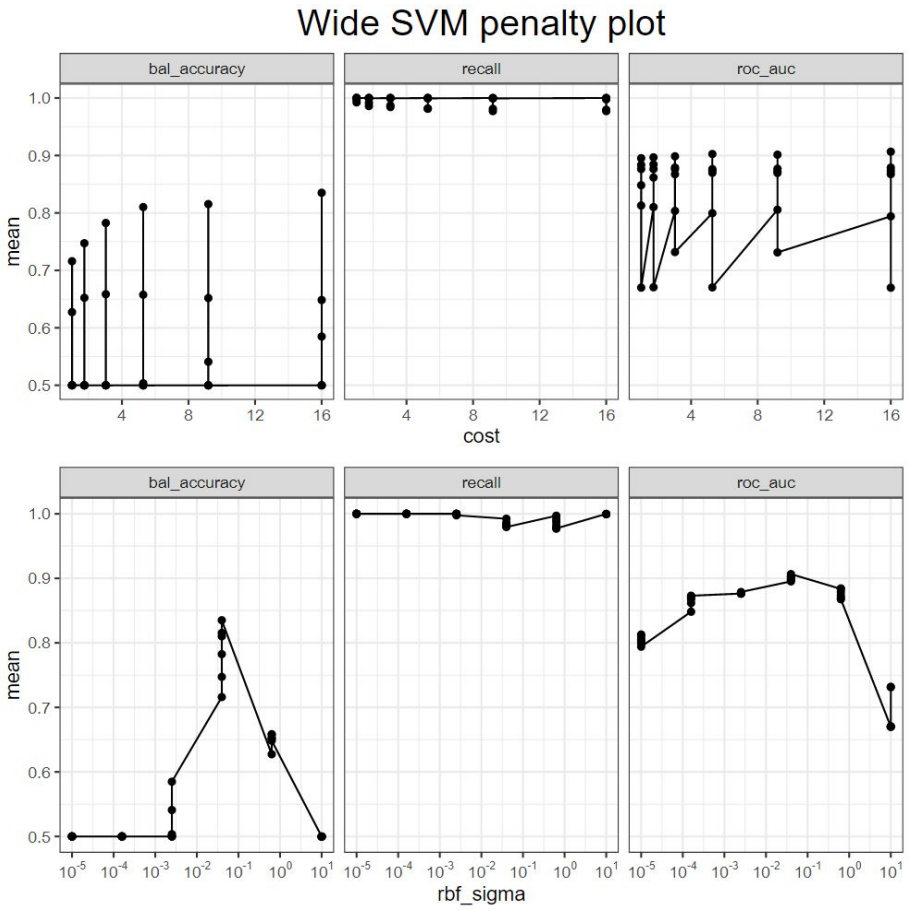


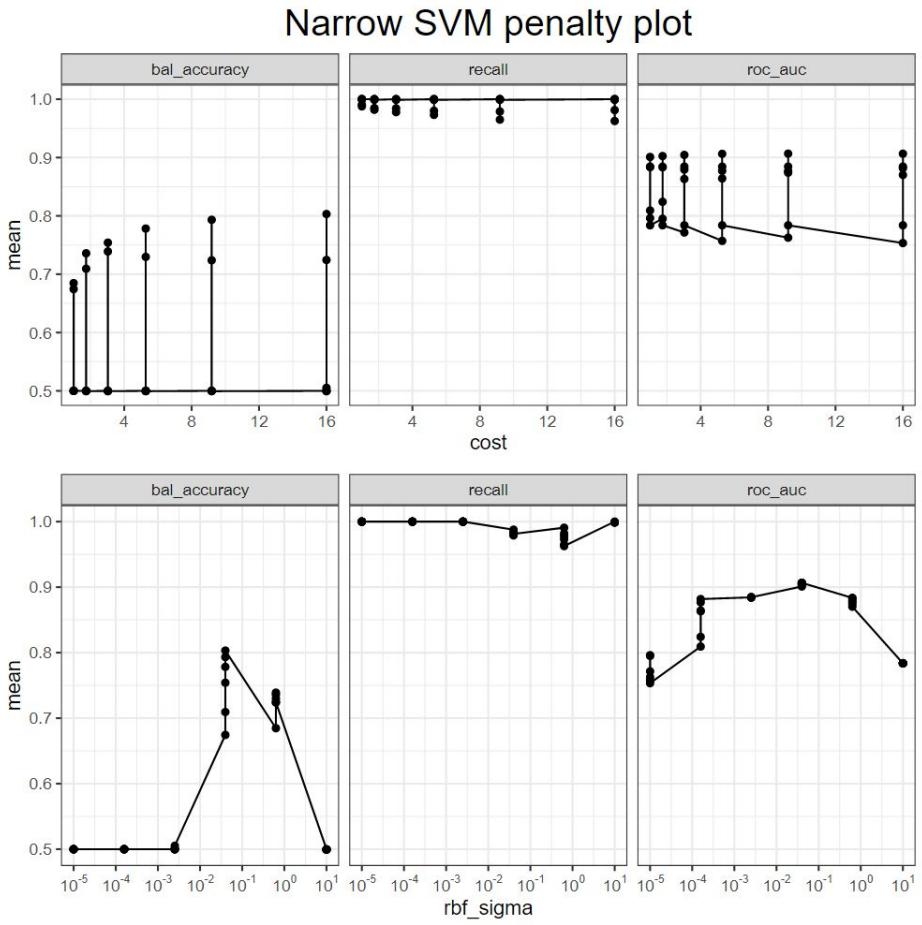
For this SVM the results are noticeably different. For the cost parameter the “bal_accuracy” shows that there is a significant improvement in increasing it. The fact that recall decreases with the increase of balanced accuracy may suggest that the model switches towards better prediction of negative class. For *rbf_sigma* we see the same picture but the increase in “bal_accuracy” with the increase of the parameter starts immediately. Overall, in this case the “roc_auc” metric clearly demonstrates that increasing this parameter up to 0.1 may bring significant change.

Again predicting positive “True” churn is the priority - both models provide almost perfect results in this regard, but recall being a highly biased metric is not quite a good choice to make decisions. The recall as a metric is considered to be quite “blind” as it focuses on True Positive Rate, and overlooks everything else (False Positive Rates, Accuracy etc). On the other side roc_auc aggregates most of the important performance features and provides an unbiased representation of the model’s performance. Considering this , we haven’t seen any

notable reason to refrain from using “roc_auc” as baseline for model selection at this step and it was decided to move forward with *svm_model_rbf* as it has a better performance (roc_auc = 0.9) than *svm_model_poly* (roc_auc = 0.8)

After training our models we can have a look at penalty plot for both Wide and Narrow - SVM:





As it was the case with random forests, penalty plots both for Narrow and Wide models look the same, meaning no comparison can be made based solely on the visual representations. Let's move forward with numeric representations of the metrics for comparison:

As in our task we have a particular interest in correctly predicting positive class we start by comparing the confusion matrices of two SVM models.

Confusion matrix for Narrow SVM

Prediction	Truth: False	Truth: True
False	559	35
True	7	65

Confusion matrix for Wide SVM

Prediction	Truth: False	Truth: True
False	558	40
True	11	57

Prior to calculating the metrics, we can instantly see that the Narrow model has detected more of both classes (559 for "False" or no churn and 65 for "False" or confirmed churn).

The metrics results for each model.

Metrics for Narrow SVM

Metrics for Wide SVM

	.metric	.estimator	.estimate
	<chr>	<chr>	<dbl>
1	recall	binary	0.65
2	accuracy	binary	0.937
3	bal_accuracy	binary	0.819
4	kap	binary	0.721

	.metric	.estimator	.estimate
	<chr>	<chr>	<dbl>
1	recall	binary	0.588
2	accuracy	binary	0.923
3	bal_accuracy	binary	0.784
4	kap	binary	0.649

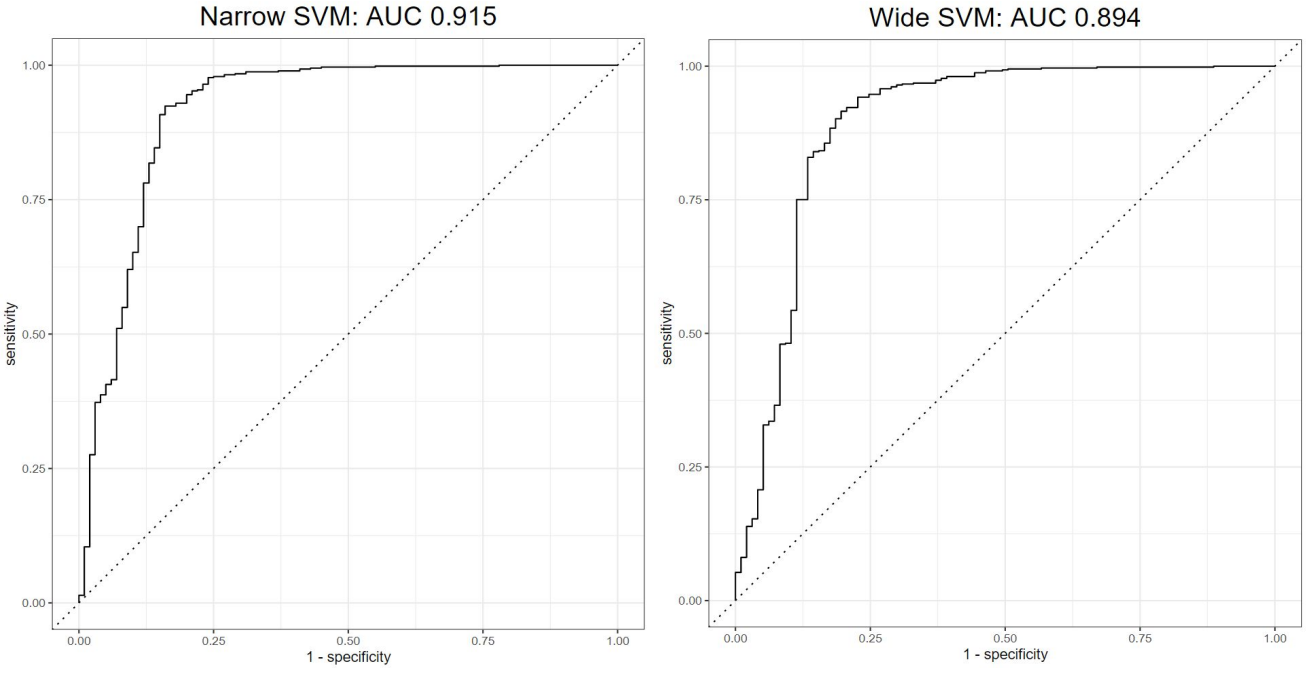
From the summary above we can see that Narrow SVM outperforms Wide one in all 4 criterias, meaning that for SVM using less variables as an input gives better results.

The difference between the two (highlighted in yellow) is not significantly large:

A table: 4 x 4

	metric	wide_svm	narrow_svm	diff
	<chr>	<dbl>	<dbl>	<dbl>
1	recall	0.588	0.65	0.0624
2	accuracy	0.923	0.937	0.0135
3	bal_accuracy	0.784	0.819	0.0347
4	kap	0.649	0.721	0.0720

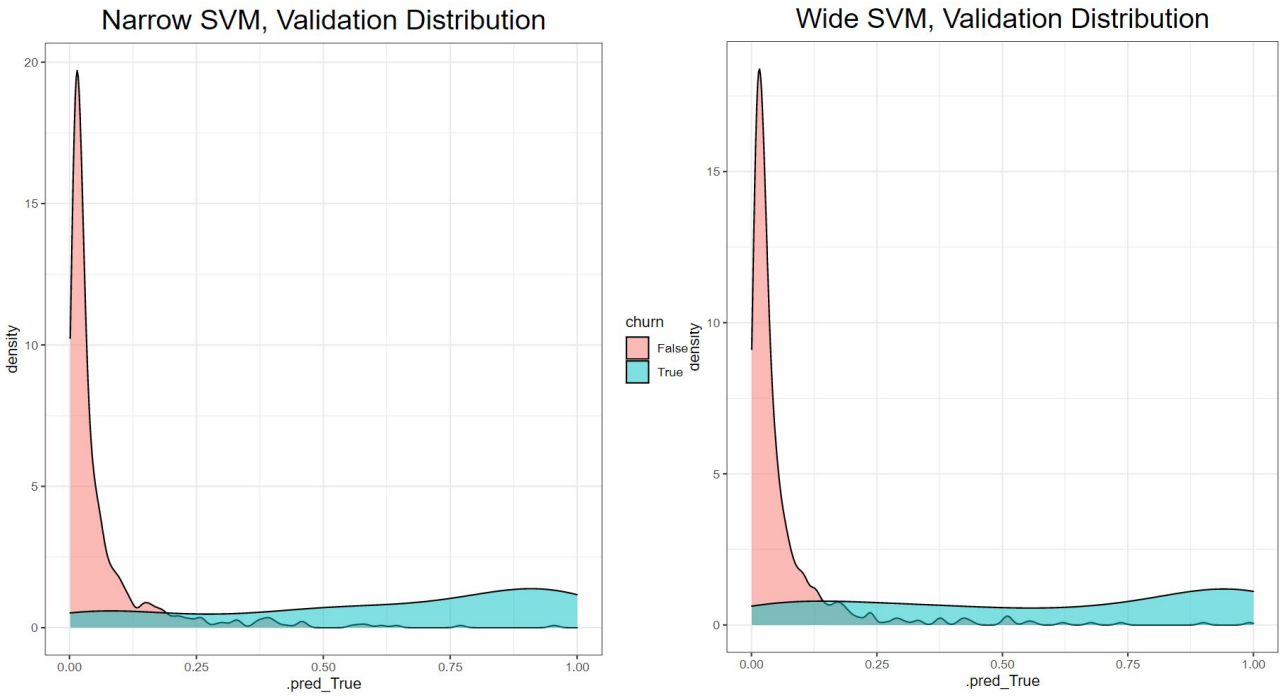
Now let's compare the ROC curves of Narrow and Wide models below:



While the ROC curves of both models are quite close to the left top corner, which as mentioned before is an indication of good model performance. While the AUC values of both

models are quite good, the AUC of Narrow model (0.915) slightly outperforms the AUC value of Wide model (0.894) by 0.021 (0.915 - 0.894).

Finally we will compare the Density plots of both models which should help us evaluate our models differentiate between two classes:



At the first glance two plots look very similar to one another, and that can be well explained by the slight performance difference between the two models we had observed beforehand. Most of the time our models are quite confident at predicting the classes. The probability for the observation is high for being “True”, at the same time the probability for it being “False” is low for both Narrow and Wide models. There is an overlap on the left side, which means that there are instances when the model is “unsure” and cannot distinguish between positive and negative classes. Additionally for positive or churn equals “True” the distribution is quite uniform.

Conclusion.

Support Vector Machines as Random Forests (RF) before, worked really well for both our Wide and Narrow models, with the exception of SVM models having a lower recall rate in general than RF. Compared to Wide SVM, Narrow SVM model, which has less predictor variables (8), worked slightly better in all regards. Usually we tend to use a “narrower” dataset when using a “wider” provides small or no gain, in case with SVM the choice is quite clear.

As mentioned before our goal was to find out whether more predictor variables provide significantly better results, and removing variables from the dataset can significantly decrease the performance of our models. Again, it was established in context of this research that this is not a valid case.

Our research has proven that in case with SVM, the model with more predictor parameters does not outperform the model with less predictor parameters, actually quite opposite. It has been proven again, in context of this research, that using the Correlation Matrix as methodology for feature selection is sufficient. We demonstrated again that adding more “unimportant” variables will not provide gain in performance and in case of SVM wll even result in reducing it (probably due to noise introduced by those “unimportant” variables).

Thus we keep the Narrow SVM model for further testing.

Testing Performance

As a result of comparing the performances of Random Forests and SVM models, we have two “finalists”: a Narrow SVM model and a Narrow Random Forests model.

	metric	narrow_svm	narrow_rf	diff
	<chr>	<dbl>	<dbl>	<dbl>
1	recall	0.65	0.78	0.13
2	accuracy	0.937	0.958	0.0210
3	bal_accuracy	0.819	0.885	0.0659
4	kap	0.721	0.824	0.103

Since Random Forests outperforms the SVM model by all metrics, we select the Narrow Random Forests model as our final model for testing. But it is important to mention that we tried only a linear SVM model and a SVM model that uses a Radial Basis Function, so it could be worth considering a polynomial SVM model of a higher degree in the future research.

As was mentioned, we are testing our final model on the “churn-bigml-20” data set, which we have converted to the same format as it was done outside of the recipe in the Random Forest function. And the model was retrained regarding the entire training dataset as hyperparameters were found.

The classification matrix and performance metrics for our test data frame.

Prediction	Truth False	Truth True
False	567	25
True	5	70

	.metric	.estimate
	<chr>	<dbl>
1	recall	0.737
2	accuracy	0.955
3	bal_accuracy	0.864
4	kap	0.798

Our testing model, as expected, has performed quite well by reaching the balanced accuracy rate of 0.864 and recall rate of 0.737. As mentioned earlier, it is important that our model identifies as many “True” churns accurately, so having a good recall rate is very important.

Conclusion

Customer churn is a great problem for businesses, since losing customers means income and profit loses. In our research we used and compared two machine learning algorithms to predict the customer churn.

As a result of our research, we have built two principally different models: Random Forest and SVM with good prediction results.

We have also established that correlation value can be used as a feature selection approach. We have researched the performance of two model variations: the Narrow model with fewer predictor variables (8) and Wide model with larger number of predictor variables (13). The better performance of the SVM model with smaller number of predictor variables was expected and did not come as a surprise.

In this regard we consider the model choice as acceptable to show that the smaller number of predictor variables do not make the results worse, but actually improve them. The comparison of the metrics results of two models confirms this.

Overall we are very pleased with the results of our research and consider the final Random Forest model satisfactory.

Resources

1. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani - An Introduction to Statistical Learning with Applications in R Springer - Science+Business Media New York 2013 (Corrected at 8th printing 2017).
2. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
3. https://towardsdatascience.com/hyperparameter-tuning-for-support-vector-machines-c-and-gamma-parameters-6a5097416167
4. https://recipes.tidymodels.org/
5. https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/#:~:text=mtry%3A%20Number%20of%20variables%20randomly,Number%20of%20trees%20to%20grow.
6. https://www.machinelearningplus.com/machine-learning/feature-selection/