Stat 218 - Analytics Project V

Inigo Benavides and Rommel Bartolome

May 20, 2019

Abstract

In this project, we utilized two machine learning techniques involving a dataset of subscribers of a cellular network where we will determine if the particular customer churns or not. First, a Support Vector Machine (SVM) was implemented and was found that the all kernels have similar accuracy but we will choose a radial one, as it has a slightly higher specificity. For SVM, we were able to accurately predict 74% of the customers that churned. Second, an Artificial Neural Network (ANN) was also utilized. Ultimately, it gave us a similar accuracy of 74%, even with varying values.

Introduction

For our last analytics project, we will be utilizing two machine learning techniques called Support Vector Machines (SVM) and Artificial Neural Networks (ANN). We will implement these techniques to cellular customer data. The dataset given is a collection of 2000 subscribers of a cellular network. We will predict whether a subscriber will churn or not. It should be noted that the Churn variable signifies whether the customer had left the company two months after observation.

Data Loading and Cleaning

We load all the libraries we will be using in this project. In addition, similar to our previous projects, we will also clean our data and set our seed for reproducibility.

First, we will generate new predictors and use a "one-hot encoding" for the categorical variables, i.e. generate new features/predictors that are binary for each category. For example, in encoding NonUSTravel, we will make new variables called NonUSTravel_No and NonUSTravel_Yes. Also, we will standardize the numerical variables.

```
#knitr::opts_chunk$set(cache = TRUE)
library(tidyverse)
library(e1071)
library(caret)

binary_encoder <- function(x) {
  case_when(
    x == "Yes" ~ 1,
    TRUE ~ 0
  )
}

seed <- 1
df <- read_csv("data_BaBe.csv")

## create features for category variables
set.seed(seed)
train_size <- 1500
train_index <- sample(seq_len(nrow(df)), size=train_size)

df_process <- df %>%
  mutate(Churn=binary_encoder(Churn),
         MaritalStatus=binary_encoder(MaritalStatus),
         OwnsComputer=binary_encoder(OwnsComputer),
         NonUSTravel=binary_encoder(NonUSTravel),
         RespondsToMailOffers=binary_encoder(RespondsToMailOffers),
         ChildrenInHH=binary_encoder(ChildrenInHH))

# One-hot encode occupation
occupation_dummies <- dummyVars(" ~ .", data=df_process)
df_process <- occupation_dummies %>%
  predict(newdata=df_process) %>%
  as.data.frame
```

Now, we will divide the dataset into training and test datasets, with the training dataset as the 1500 randomly selected observations (out of 2000) and the remaining 500 data points as the test dataset:

```
# Separate into train and test
df_train <- df_process[train_index, ]
df_test <- df_process[-train_index, ]

# Separate X and y for train and test sets
X_train <- df_train %>% select(-Churn)
y_train <- df_train$Churn
X_test <- df_test %>% select(-Churn)
y_test <- df_test$Churn

df_train %>% head %>% as_tibble()
```

```
## # A tibble: 6 x 34
##   Churn MonthlyMinutes OverageMinutes RoamingCalls PercChangeMinut~
##   <dbl>          <dbl>          <dbl>        <dbl>            <dbl>
## 1     1            662             83            1              695
## 2     1           2287            117            1             1499
## 3     0             99              1           38              813
## 4     0           2609            716            1             1418
## 5     0           2194             83          152              919
## 6     1           2405              1          159              189
## # ... with 29 more variables: DroppedCalls <dbl>, BlockedCalls <dbl>,
## #   UnansweredCalls <dbl>, CustomerCareCalls <dbl>, ReceivedCalls <dbl>,
## #   OutboundCalls <dbl>, InboundCalls <dbl>, PeakCallsInOut <dbl>,
## #   OffPeakCallsInOut <dbl>, MonthsInService <dbl>, UniqueSubs <dbl>,
## #   ActiveSubs <dbl>, ChildrenInHH <dbl>, RespondsToMailOffers <dbl>,
## #   NonUSTravel <dbl>, OwnsComputer <dbl>, HasCreditCardNo <dbl>,
## #   HasCreditCardYes <dbl>, RetentionCalls <dbl>,
## #   RetentionOffersAccepted <dbl>, OccupationClerical <dbl>,
## #   OccupationCrafts <dbl>, OccupationHomemaker <dbl>,
## #   OccupationOther <dbl>, OccupationProfessional <dbl>,
## #   OccupationRetired <dbl>, OccupationSelf <dbl>,
## #   OccupationStudent <dbl>, MaritalStatus <dbl>
```

Exploratory Data Analysis

Before building our model, we first do some Exploratory Data Analysis (EDA) to our original dataset to better understand it. First, we wish to determine the class distribution of the target. We note that there are

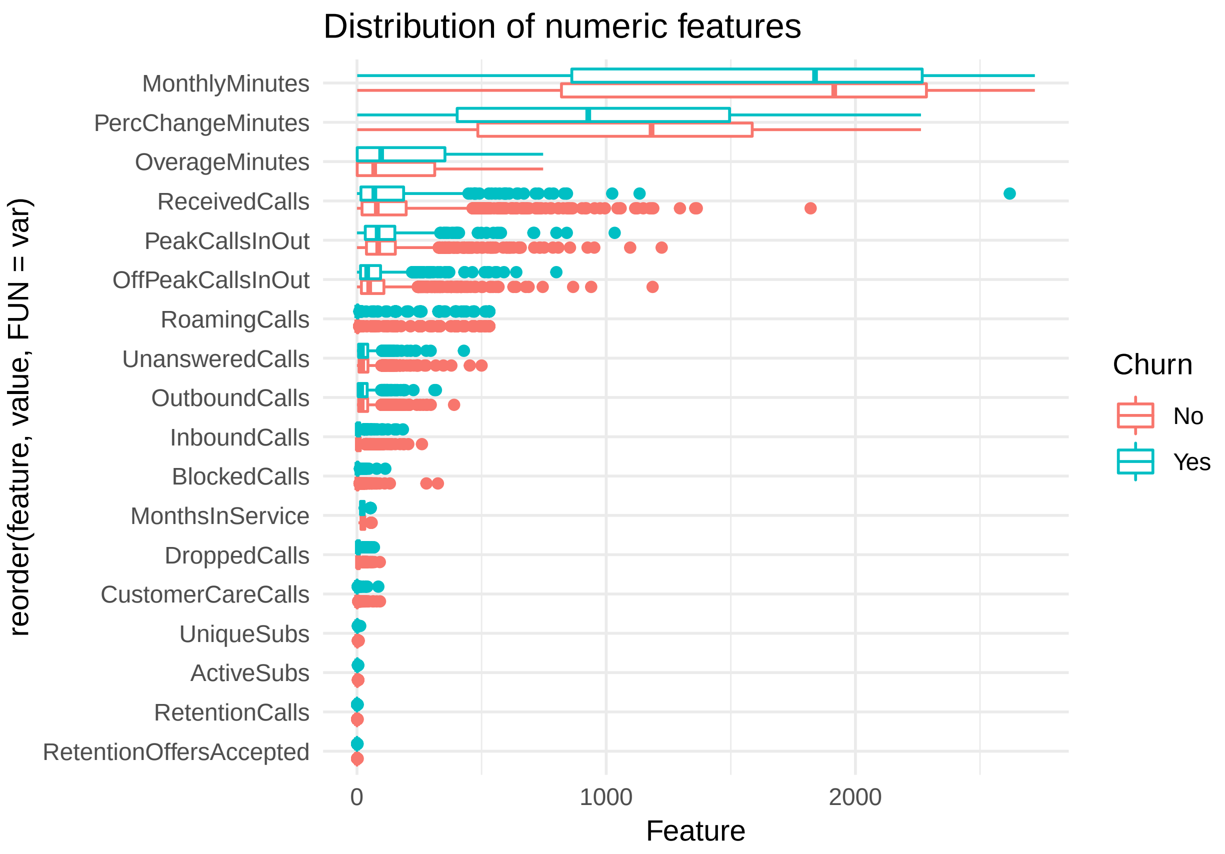
505 “Yes” observations and 1,495 “No” observations. To respect the class distribution in model building, we stratify split according to this distribution.

```
df %>% group_by(Churn) %>% summarise(n=n())
```

```
## # A tibble: 2 x 2
##   Churn     n
##   <chr> <int>
## 1 No     1495
## 2 Yes     505
```

Here we can see than we have 505 customers that churned while 1,495 did not. Next, we inspect the numerical features and run horizontal box plots to see the distribution against the target.

```
# Boxplot of numerics against target distribution
numerics <- unlist(lapply(df, is.numeric))
numerics[1] <- TRUE # Set Churn column to be included
df[, numerics] %>% gather(key="feature", value="value", -Churn) %>%
  ggplot(aes(x=reorder(feature, value, FUN=var), y=value, color=Churn)) + geom_boxplot() +
  theme_minimal() +
  coord_flip() +
  labs(title="Distribution of numeric features") +
  ylab("Feature")
```



Based on the above analysis, it appears that there may be large discriminative power in `MonthlyMinutes` and `PercChangeMinutes`. The percentage change in minutes of use appears especially discriminative, and we can consider this intuitive because lower usage in the service would be indicative of lower future usage and eventually churn.

Next, we can plot the correlation matrix of these numeric covariates:

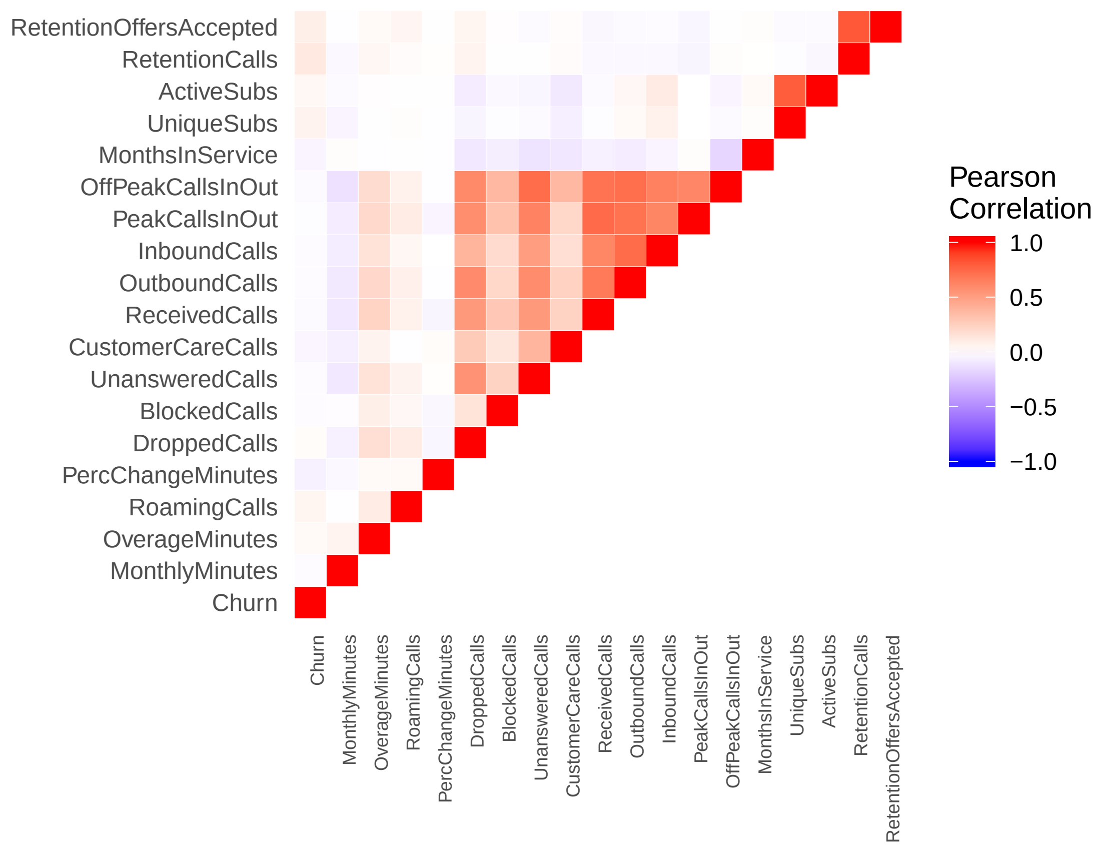
```
# Correlation matrix
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
# Get lower triangle of the correlation matrix
get_lower_tri<-function(cormat){
  cormat[upper.tri(cormat)] <- NA
  return(cormat)
}
# Get upper triangle of the correlation matrix
get_upper_tri <- function(cormat){
  cormat[lower.tri(cormat)]<- NA
  return(cormat)
}

df_numeric <- df[,numerics]
df_numeric %>%
  mutate(Churn=case_when(Churn=="Yes" ~ 1, TRUE ~ 0)) %>%
  cor %>%
  get_upper_tri() %>%
  melt(na.rm=TRUE) %>% ggplot(aes(x=Var1, y=Var2, fill=value)) +
  geom_tile(color="white") +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
   midpoint = 0, limit = c(-1,1), space = "Lab",
   name="Pearson\nCorrelation") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 1,
    size = 7, hjust = 1),
    axis.title.x = element_blank(),
  axis.title.y = element_blank(),
  panel.grid.major = element_blank(),
  panel.border = element_blank(),
  panel.background = element_blank(),
  axis.ticks = element_blank()) +
  coord_fixed()
```



Next, we inspect the categoricals and plot the distribution against the target variable:

```
# Inspect categoricals
df %>% select_if(negate(is.numeric)) %>% summarise_each(funs(n_distinct)) %>% as_tibble()
```

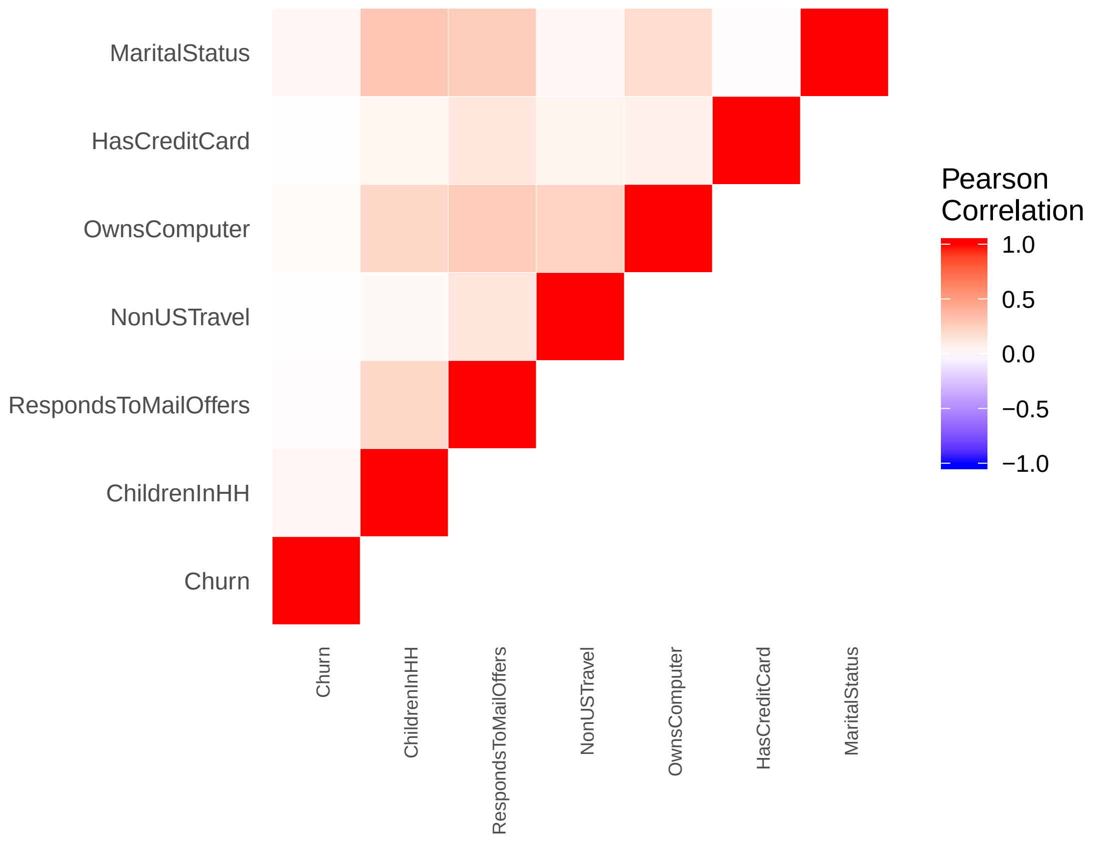
```
## `summarise_each()` is deprecated.
## Use `summarise_all()`, `summarise_at()` or `summarise_if()` instead.
## To map `funs` over all variables, use `summarise_all()`
```

```
## # A tibble: 1 x 8
##   Churn ChildrenInHH RespondsToMailO~ NonUSTravel OwnsComputer
##   <int>        <int>            <int>       <int>        <int>
## 1     2            2                2           2            2
## # ... with 3 more variables: HasCreditCard <int>, Occupation <int>,
## #   MaritalStatus <int>
```

We find that all categoricals except `Occupation` are binary. We can encode these as 1 or 0 instead and also plot the correlation matrix.

```
#
df %>%
  select_if(negate(is.numeric)) %>%
  select(-Occupation) %>%
  mutate_all(binary_encoder) %>%
  cor %>%
  get_upper_tri() %>%
  melt(na.rm=TRUE) %>% ggplot(aes(x=Var1, y=Var2, fill=value)) +
  geom_tile(color="white") +
```

```r
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
   midpoint = 0, limit = c(-1,1), space = "Lab",
   name="Pearson\nCorrelation") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 1,
    size = 7, hjust = 1),
    axis.title.x = element_blank(),
  axis.title.y = element_blank(),
  panel.grid.major = element_blank(),
  panel.border = element_blank(),
  panel.background = element_blank(),
  axis.ticks = element_blank()) +
  coord_fixed()
```



There do not seem to be any strong categorical correlations with `Churn`.

Support Vector Machines

For the SVM part, we will try different kernels and check what kernel will be the best in classifying if a customer will churn. Our goal is to be able to have an error rate of $\leq 10\%$.

First, we will fit an SVM with a radial kernel, and tuning it by finding the best cost and gamma features:

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
set.seed(seed)
df_train[,1] <- as.factor(df_train[,1])
df_test[,1] <- as.factor(df_test[,1])
svm_radial <- tune(svm, Churn ~., data = df_train, kernel = "radial",
                   ranges = list(cost = c(0.01, 1, 10),
                                 gamma = c(0.01, 1, 10)))
svm_radial$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Churn ~ ., data = df_train,
##     ranges = list(cost = c(0.01, 1, 10), gamma = c(0.01, 1, 10)),
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  1490
```

```
confusionMatrix(predict(svm_radial$best.model, df_test),
                df_test$Churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 369 129
##          1   1   1
##
##                Accuracy : 0.74
##                  95% CI : (0.6992, 0.7779)
##     No Information Rate : 0.74
##     P-Value [Acc > NIR] : 0.5236
##
##                   Kappa : 0.0073
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.997297
##             Specificity : 0.007692
##          Pos Pred Value : 0.740964
##          Neg Pred Value : 0.500000
##              Prevalence : 0.740000
##          Detection Rate : 0.738000
##    Detection Prevalence : 0.996000
##       Balanced Accuracy : 0.502495
##
##        'Positive' Class : 0
##
```

Surprisingly, our model classified almost all of it as “Not Churn” with only one classified to be “Churn”. We check if this will also be the case for a linear kernel:

```
set.seed(100)
svm_linear <- tune(svm, Churn~., data = df_train, kernel="linear", ranges = list(cost = c(0.01,1,10),
                                                                                 gamma = c(0.01,1,10))
svm_linear$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Churn ~ ., data = df_train,
##     ranges = list(cost = c(0.01, 1, 10), gamma = c(0.01, 1, 10)),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.01
##
## Number of Support Vectors:  791
```

```
confusionMatrix(predict(svm_linear$best.model, df_test), df_test$Churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 370 130
##          1   0   0
##
##                Accuracy : 0.74
##                  95% CI : (0.6992, 0.7779)
##     No Information Rate : 0.74
##     P-Value [Acc > NIR] : 0.5236
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.00
##             Specificity : 0.00
##          Pos Pred Value : 0.74
##          Neg Pred Value :  NaN
##              Prevalence : 0.74
##          Detection Rate : 0.74
##    Detection Prevalence : 1.00
##       Balanced Accuracy : 0.50
##
##        'Positive' Class : 0
```

```
##
```

Again and worse, it classified all of it as not churning. We try for a polynomial kernel:

```
set.seed(seed)
svm_poly <- tune(svm, Churn~., data = df_train, kernel="polynomial",ranges = list(cost = c(0.01,1,10),
                                                                                   degree = c(2,3,4)))
svm_poly$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Churn ~ ., data = df_train,
##     ranges = list(cost = c(0.01, 1, 10), degree = c(2, 3, 4)),
##     kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##       gamma:  0.03030303
##      coef.0:  0
##
## Number of Support Vectors:  838
```

```
confusionMatrix(predict(svm_poly$best.model, df_test), df_test$Churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 370 130
##          1   0   0
##
##                Accuracy : 0.74
##                  95% CI : (0.6992, 0.7779)
##     No Information Rate : 0.74
##     P-Value [Acc > NIR] : 0.5236
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.00
##             Specificity : 0.00
##          Pos Pred Value : 0.74
##          Neg Pred Value :  NaN
##              Prevalence : 0.74
##          Detection Rate : 0.74
##    Detection Prevalence : 1.00
##       Balanced Accuracy : 0.50
##
##        'Positive' Class : 0
##
```

```
#df_train[,1] <- as.numeric(df_train[,1])
#df_test[,1] <- as.numeric(df_test[,1])
```

Again, the predictions are all NOT churn. Overall, the accuracy is at 74% but for this exercise, we will choose a radial one as it has a slightly higher specificity.

Artificial Neural Networks

Now, we will fit an Artificial Neural Network. We try to create one with a 3 hidden layers.

```
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'

## The following object is masked from 'package:dplyr':
##
##     compute
```

```
fit_nn <- neuralnet(Churn~., data=df_train, hidden=3)
plot(fit_nn)
tune_nn <- caret::train(Churn~.,
                        data=df_train,
                        method='mlpWeightDecayML',
                        tuneGrid=expand.grid(layer1 = c(2:5),
                                             layer2 = c(1:3),
                                             layer3 = c(0:1),
                                             decay=0.1),
                        verbose=TRUE)

tune_nn
```

```
## Multi-Layer Perceptron, multiple layers
##
## 1500 samples
##   33 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1500, 1500, 1500, 1500, 1500, 1500, ...
## Resampling results across tuning parameters:
##
##   layer1  layer2  layer3  Accuracy   Kappa
##   2       1       0       0.7490245  0
##   2       1       1       0.7490245  0
##   2       2       0       0.7490245  0
##   2       2       1       0.7490245  0
##   2       3       0       0.7490245  0
##   2       3       1       0.7273060  0
##   3       1       0       0.7287608  0
##   3       1       1       0.7287704  0
##   3       2       0       0.7289517  0
##   3       2       1       0.7490245  0
##   3       3       0       0.7287704  0
##   3       3       1       0.7490245  0
##   4       1       0       0.7095625  0
##   4       1       1       0.7300130  0
##   4       2       0       0.7304505  0
##   4       2       1       0.7280790  0
##   4       3       0       0.7287608  0
##   4       3       1       0.7490245  0
##   5       1       0       0.7490245  0
##   5       1       1       0.7490245  0
##   5       2       0       0.7294194  0
##   5       2       1       0.7301167  0
##   5       3       0       0.7490245  0
##   5       3       1       0.7490245  0
##
## Tuning parameter 'decay' was held constant at a value of 0.1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were layer1 = 2, layer2 = 1, layer3
##  = 0 and decay = 0.1.
```

Here we see that for layer 1 we used 2 nodes, for layer 2 we used 1 node and none for layer 3. We check the performance of this model:

```
confusionMatrix(predict(tune_nn, df_test), df_test$Churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 370 130
##          1   0   0
##
##                Accuracy : 0.74
##                  95% CI : (0.6992, 0.7779)
##     No Information Rate : 0.74
##     P-Value [Acc > NIR] : 0.5236
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.00
##             Specificity : 0.00
##          Pos Pred Value : 0.74
##          Neg Pred Value :  NaN
##              Prevalence : 0.74
##          Detection Rate : 0.74
##    Detection Prevalence : 1.00
##       Balanced Accuracy : 0.50
##
##        'Positive' Class : 0
##
```

Here, we see the same performance as above, where most of the predictions were NOT Churn. We check a similar 2-layer ANN, with varying decay value:

```
fit_nn_2 <- neuralnet(Churn~., data=df_train, hidden=2)
plot(fit_nn_2)
tune_nn_2 <- caret::train(Churn~.,
                        data=df_train,
                        method='mlpWeightDecayML',
                        tuneGrid=expand.grid(layer1 = c(2:5),
                                             layer2 = c(0:2),
                                             layer3 = c(0),
                                             decay=c(0.001, 0.1, 1)),
                        verbose=TRUE)

tune_nn_2
```

```
## Multi-Layer Perceptron, multiple layers
##
## 1500 samples
##   33 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1500, 1500, 1500, 1500, 1500, 1500, ...
## Resampling results across tuning parameters:
##
##   layer1  layer2  decay  Accuracy   Kappa
##   2       0       0.001  0.7502642   0.0000000000
##   2       0       0.100  0.7265449   0.0000000000
##   2       0       1.000  0.6305529   0.0000000000
##   2       1       0.001  0.7502642   0.0000000000
##   2       1       0.100  0.7502642   0.0000000000
##   2       1       1.000  0.5916884   0.0000000000
##   2       2       0.001  0.7502642   0.0000000000
##   2       2       0.100  0.7302642   0.0000000000
##   2       2       1.000  0.5712736   0.0000000000
##   3       0       0.001  0.7502642   0.0000000000
##   3       0       0.100  0.7502642   0.0000000000
##   3       0       1.000  0.5508068   0.0000000000
##   3       1       0.001  0.7502642   0.0000000000
##   3       1       0.100  0.7502642   0.0000000000
##   3       1       1.000  0.6691799   0.0000000000
##   3       2       0.001  0.7502642   0.0000000000
##   3       2       0.100  0.7502642   0.0000000000
##   3       2       1.000  0.6938574   0.0000000000
##   4       0       0.001  0.7502642   0.0000000000
##   4       0       0.100  0.7501900  -0.0001479985
##   4       0       1.000  0.6538796   0.0000000000
##   4       1       0.001  0.7502642   0.0000000000
##   4       1       0.100  0.7502642   0.0000000000
##   4       1       1.000  0.5685305   0.0000000000
##   4       2       0.001  0.7502642   0.0000000000
##   4       2       0.100  0.7304824   0.0000000000
##   4       2       1.000  0.6139261   0.0000000000
##   5       0       0.001  0.7502642   0.0000000000
##   5       0       0.100  0.7502642   0.0000000000
##   5       0       1.000  0.6286242   0.0000000000
##   5       1       0.001  0.7502642   0.0000000000
##   5       1       0.100  0.7502642   0.0000000000
##   5       1       1.000  0.6252793   0.0000000000
##   5       2       0.001  0.7502642   0.0000000000
##   5       2       0.100  0.7502642   0.0000000000
##   5       2       1.000  0.6708572   0.0000000000
##
## Tuning parameter 'layer3' was held constant at a value of 0
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were layer1 = 2, layer2 = 0, layer3
##  = 0 and decay = 0.001.
```

Now we have layer 1 with two nodes, the decay being 0.001. We now check its performance:

```
confusionMatrix(predict(tune_nn_2, df_test), df_test$Churn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 370 130
##          1   0   0
##
##                Accuracy : 0.74
##                  95% CI : (0.6992, 0.7779)
##     No Information Rate : 0.74
##     P-Value [Acc > NIR] : 0.5236
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.00
##             Specificity : 0.00
##          Pos Pred Value : 0.74
##          Neg Pred Value :  NaN
##              Prevalence : 0.74
##          Detection Rate : 0.74
##    Detection Prevalence : 1.00
##       Balanced Accuracy : 0.50
##
##        'Positive' Class : 0
##
```

Unfortunately, this is the case again. We also tried different models to the dataset, but unfortunately, this has also been the case.

What we can infer is that the features that we were given were not enough to identify if a customer will Churn or Not Churn, with an error rate of $\leq 10\%$. All the models classified most of the data as "not churn".

Conclusions

For the SVM technique, we have the following accuracy metrics:

Model	**Accuracy**

Model	**Accuracy**
Radial Kernel	74%
Linear Kernel	74%
Polynomial Kernel	74%

For the ANN technique, we have the following accuracy metrics:

Model	**Accuracy**
ANN - 3 Layers	74%
ANN - 2 Layers	74%

Unfortunately, we were not able to lower the error rate to $\leq 10\%$, but only to 26%. This may mean that the features in the data were not able to capture what a customer that “Churn”