

Università degli studi di Milano Bicocca - Corso di High Dimensional Data Analysis - Il Bagging

Lorenzo Bruni (886721), Remo Marconzini (883256)

Abstract

L'obiettivo di questo progetto è quello di approfondire la procedura di *bagging*. A tal fine, è stato dapprima condotto un approfondimento metodologico, per poi passare ad un'implementazione in R, sia per il task di classificazione che per quello di regressione. Per implementare il *bagging* è stato utilizzato il pacchetto *caret*. L'implementazione in R è stata così strutturata: una preliminare fase di analisi esplorativa del dataset, una seconda fase di selezione delle features sulla base di alcune caratteristiche di esse e successivamente testing e valutazione dei modelli. I modelli scelti per l'analisi sono gli alberi di decisione come weak learner, e le stime ottenute da essi sono state poi confrontate con quelle ottenute tramite procedura di *bagging*, al fine di valutarne i benefici.

Introduzione

Il bootstrap aggregation, conosciuto anche come *bagging*, è un meta-algoritmo di apprendimento automatico progettato per migliorare la stabilità e l'accuratezza di algoritmi di statistical e machine learning, utilizzati nei task di classificazione o regressione. Tale procedura viene utilizzata per ridurre la varianza e per evitare il fenomeno di overfitting. Il *bagging* si basa sul concetto di bootstrap, una procedura di ricampionamento che crea b nuovi bootstrap samples, i quali vengono costruiti mediante campionamento con reinserimento dal training set, e attraverso questi è possibile valutare l'accuratezza della stima di un parametro o di una previsione. Il *bagging* sfrutta proprio tale procedura per produrre stime più affidabili e robuste.

Ensemble methods

L'ensemble learning dà credito all'idea della "saggezza delle folle", che suggerisce che il processo decisionale di un gruppo più ampio di persone è in genere migliore di quello di un singolo esperto. Allo stesso modo, l'ensemble learning si riferisce a un gruppo (o ensemble) di weaks learner, che lavorano collettivamente per ottenere una migliore previsione finale. Un singolo weak learner può non avere buone prestazioni individuali, a causa dell'elevata varianza o dell'elevato bias. Tuttavia, quando i weak learners vengono aggregati possono formare uno strong learner, in quanto la loro combinazione aiuta a ridurre il bias o la varianza, portando quindi a prestazioni migliori del modello e producendo previsioni più robuste e affidabili.

I metodi di ensemble sono spesso usati con gli alberi decisionali, i quali risultano semplici da implementare e interpretare, tuttavia risultano essere spesso inclini all'overfitting (ovvero a un ridotto bias ma con una grande variabilità) quando essi non vengono "potati", oppure possono soffrire di underfitting (ridotta varianza ma con alto bias) quando questi sono troppo poco profondi. I metodi di ensemble quindi, tra cui il bagging, grazie alla costruzione di molteplici alberi consentono una migliore generalizzazione del modello, a spese di una perdita dell'interpretabilità dello stesso e ad un aumento del costo computazionale.

Bootstrap

Il *bagging* si basa sul metodo statistico del *bootstrapping*, il quale risulta essere uno strumento ampiamente applicabile ed estremamente potente, utilizzato per quantificare l'incertezza associata a un determinato stimatore o modello di statistical learning.

Il *bootstrap* funziona nel seguente modo. Ipotizziamo la presenza di un campione X di grandezza N . Possiamo creare un nuovo campione, dal campione originale, prelevando N elementi da quest'ultimo in modo casuale e uniforme, con reinserimento. Selezioniamo quindi casualmente elementi dal campione di partenza di grandezza N e lo facciamo B volte. Ciascun elemento del campione ha la medesima probabilità di essere selezionato, pari a $1/N$. Un aspetto sicuramente da notare è che il campionamento viene fatto con reinserimento, di conseguenza la stessa osservazione può comparire più di una volta nel campione estratto. Alla fine di questa procedura otteniamo così B *bootstrap samples* X_1, \dots, X_B . Su ciascun bootstrap sample verrà addestrato il nostro modello e poi esaminato quanto esso riesca a prevedere l'originale training set. In questo modo è possibile ottenere una stima dell'errore di previsione. Considerando $\hat{f}^{*b}(x_i)$ il valore previsto per l'istanza x_i dal nostro modello per il b -esimo bootstrap sample, la stima dell'errore risultante sarà:

$$\hat{Err}_{boot} = \frac{1}{B} \frac{1}{N} \sum_{b=1}^B \sum_{i=1}^N L(y_i, \hat{f}^{*b}(x_i))$$

Tuttavia, è facile vedere come \hat{Err}_{boot} non fornisce una buona stima in generale. Il motivo è che i bootstrap samples, che agiscono come il training sample, e il training sample originale che agisce come il test set, hanno osservazioni in comune. Questa sovrapposizione può rendere le previsioni irrealisticamente buone ed è la ragione per cui la *cross-validation* utilizza esplicitamente dati non sovrapposti per il training e test set.

Un possibile miglioramento è dato dal calcolare \hat{Err}_{boot} solo per quelle osservazioni che non sono incluse nel bootstrap dataset. Un'altra soluzione che offre un possibile miglioramento è data invece dal "Leave-one-out bootstrap", in cui la previsione $\hat{f}^{*b}(x_i)$ è basata solo sul set di dati che non contiene x_i e dove C^{-i} è il set di indici del bootstrap sample che non contiene l'osservazione i :

$$\hat{Err}_{boot} = \frac{1}{n} \sum_{b=1}^B \frac{1}{|C^{-i}|} \sum_{b \in C^{-i}} (y_i - \hat{f}^{*b}(x_i))^2$$

In generale, il principale vantaggio dell'uso della tecnica bootstrap è che permette di stimare la distribuzione campionaria di una certa statistica senza assumere alcuna ipotesi sulla distribuzione dei dati sottostanti. Ciò significa che il bootstrapping è particolarmente utile quando non si conosce la distribuzione dei dati o quando i dati sono non normali o non omogenei. Inoltre, il bootstrapping consente di stimare l'errore standard di una certa statistica in modo robusto e di costruire intervalli di confidenza per i parametri stimati, il che fornisce una valutazione più accurata della precisione delle stime.

Bagging

I bootstrap aggregating (*bagging*) prediction models sono quindi metodi generali per addestrare molteplici versioni di un modello di previsione per poi combinarli (*ensembling*) in un'unica previsione aggregata. Come detto in precedenza, il *bagging* è progettato per migliorare la stabilità e accuratezza degli algoritmi di regressione e classificazione, difatti mediante la media il *bagging* riesce a ridurre la varianza e minimizzare l'overfitting. Nel caso di un task di regressione, il *mean squared error* atteso di uno stimatore può essere decomposto in termini di: distorsione, varianza e rumore. La parte di distorsione, misura l'ammontare medio delle previsioni di uno stimatore

rispetto alle previsioni dello stimatore migliore possibile per il problema. La parte di varianza, misura la variabilità delle previsioni di uno stimatore quando viene addestrato tra diverse istanze del solito problema. Infine, il rumore caratterizza la parte irriducibile dell'errore che è dovuta alla variabilità insita nei dati.

Il *bagging* è un algoritmo piuttosto semplice nel quale b bootstrap samples vengono creati dal training set, e un algoritmo di classificazione o regressione (solitamente chiamato base learner o weak learner) viene applicato ad ogni bootstrap sample e, nel caso della regressione, le nuove previsioni vengono generate mediando le previsioni dei vari base learners addestrati in maniera indipendente. Quando trattiamo invece il caso di classificazione, le previsioni dei base learners sono combinate considerando la classe più votata o mediando le probabilità delle varie classi stimate. Questo processo è rappresentato, nel caso di regressione, dalla seguente equazione, dove: X rappresenta il record per il quale si vuole generare una previsione, \hat{f}_{bag} è la previsione del bagged model, mentre $\hat{f}_1(X), \hat{f}_2(X) \dots \hat{f}_B(X)$ sono le previsioni dei base learners:

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(X)$$

Nel caso di task di classificazione, il modello base produce un classificatore $\hat{G}(x)$ per una variabile target, con un numero di classi uguale a K , in cui $\hat{G}(x) = argmax_k \hat{f}(x)$, dove $\hat{f}(x)$ è una funzione vettore con un valore intero e $K - 1$ zeri. Nelle stime di bagging invece $\hat{f}_{bag}(x)$ è un vettore con K valori, e $p_1(x), \dots, p_K(x)$, dove $p_k(x)$ è la proporzione di alberi che hanno predetto la classe k per l'istanza x . Il classificatore di bagging prodotto sarà quindi il seguente:

$$\hat{G}_{bag}(x) = argmax_k \hat{f}_{bag}(x)$$

dove verrà selezionata la classe con più "voti" fra i vari alberi costruiti.

Inoltre, spesso nel caso di classificazione è comune utilizzare le stime di probabilità di classe per l'istanza x , piuttosto che la classe prevista. In questo modo tramite il processo di bagging si andrà a mediare queste probabilità di classe ottenute per i vari modelli, piuttosto che selezionare la classe con più voti. Data $\hat{p}_k^b(x)$ probabilità stimata della classe k per l'istanza x dal modello b , allora la probabilità di classe stimata dal modello di bagging è data da:

$$\hat{p}_k(x) = \frac{1}{B} \sum_{b=1}^B \hat{p}_k^b(x)$$

Dopodiché, il classificatore finale del modello di bagging semplicemente sceglierà la classe con la più alta probabilità:

$$\hat{G}_{bag}(x) = argmax_k \hat{p}_k(x)$$

In questo modo non otteniamo solo miglioramenti nelle stime di probabilità di classe (non più dovute alla proporzioni di alberi che prevedono una specifica classe), ma il modello tenderà a produrre anche classificatori con più bassa varianza.

Per via del processo di aggregazione, il bagging riduce la varianza di un base learner; tuttavia, il bagging non sempre riesce a migliorare le prestazioni del base learner. Infatti, esso funziona particolarmente bene per base learners instabili e con elevata varianza, nei quali gli output dei modelli riscontrano grandi cambiamenti in risposta

a piccoli cambiamenti nel training set. Fra questi algoritmi sono sicuramente da evidenziare ad esempio gli alberi decisionali. Per quanto riguarda invece algoritmi più stabili, o con un bias elevato, il bagging offre pochi miglioramenti sulle previsioni, dal momento che c'è poca variabilità.

Come detto il bagging viene quindi solitamente applicato agli alberi decisionali. Ciascun albero associato ad ogni bootstrap sample solitamente coinvolgerà differenti features, rispetto all'originale, e potrebbe avere un numero diverso di nodi terminali.

Inoltre con il processo di bagging l'uso di molteplici alberi non porta al fenomeno di overfitting, ma è importante sottolineare che, dato l'elevato numero di alberi costruiti, il costo computazionale potrebbe essere elevato. Infatti, stiamo essenzialmente moltiplicando il lavoro di crescita di un singolo albero per B . E' importante sottolineare che, un altro svantaggio del bagging è dato dalla perdita di interpretabilità, essendo che quando si esegue il bagging di un gran numero di alberi, non è più possibile rappresentare la procedura risultante utilizzando un singolo albero, e non è più chiaro quali variabili siano più importanti per la tale procedura. Pertanto, il bagging migliora l'accuratezza della previsione a discapito dell'interpretabilità. Sebbene l'insieme degli alberi bagged sia molto più difficile da interpretare rispetto a un singolo albero, è possibile ottenere un riepilogo complessivo dell'importanza di ciascuna variabile utilizzando l'RSS (residual sum of square) per gli alberi di regressione, o l'indice di Gini per gli alberi di classificazione.

Un ulteriore beneficio del *bagging* si basa sull'uso del ricampionamento con reinserimento. E' emerso che, esiste un modo molto semplice per stimare l'errore di test di un modello a cui è stata applicata la procedura di bagging, senza la necessità di eseguire una *cross-validation*. Si può dimostrare che, in media, ogni albero "bagged" fa uso di circa due terzi delle osservazioni. Il restante terzo delle osservazioni non utilizzate per il traininig è denominato osservazioni *out-of-bag* (OOB). Possiamo, quindi, prevedere la risposta per l' i -esima osservazione utilizzando ciascuno degli alberi in cui tale osservazione era OOB. In questo modo si otterranno circa $B/3$ previsioni per l' i -esima osservazione, le quali verranno mediate (per task di regressione), o verrà presa la classe maggiormente prevista (per task di classificazione), ottentendo così un'unica previsione OOB. In questo modo si può ottenere una previsione OOB per ciascuna delle N osservazioni, da cui si può calcolare l'MSE complessivo dell'OOB (per un problema di regressione) o il classification error rate (per un problema di classificazione). L'errore OOB risultante, risulta essere una valida stima dell'errore di test per il modello bagged, poiché la risposta per ogni osservazione è predetta utilizzando solo gli alberi in cui quella osservazione non era presente. E' possibile dimostrare che, per valori di B sufficientemente grandi, l'errore OOB è equivalente ad una *leave-one-out cross validation*. L'approccio OOB, per la stima dell'errore di test è particolarmente conveniente quando viene effettuato un bagging su un dataset di grandi dimensioni, ed è comune usare la stima dell'errore tramite OOB come proxy per la performance previsiva del modello.

Random Forest

L'algoritmo Random Forest fornisce un miglioramento rispetto agli alberi costruiti tramite procedura bagging, attraverso una piccola modifica che rende gli alberi decorrelati. Come nel bagging, costruiamo una serie di alberi decisionali su dei campioni di training bostrappati. Ma quando vengono costruiti questi alberi, ogni volta che viene preso in considerazione uno split in un albero un campione casuale di m variabili, estratto dalle set p di variabili completo, viene scelto come candidato dello split. Lo split può utilizzare solo uno di queste m variabili, e ad ogni split viene estratto un nuovo campione m (solitamente la dimensione di m è pari a $\sqrt(p)$). Nella costruzione della foresta casuale (random forest), a ogni divisione dell'albero, l'algoritmo non può considerare la stragrande maggioranza delle variabili disponibili.

Qual è la logica alla base di questa scelta?

Supponiamo che nell'insieme di dati ci sia una variabile molto forte, insieme a una serie di altre variabili meno forti. Nella procedura di bagging, la maggior parte o tutti gli alberi utilizzeranno questa variabile forte nel primo split. Tutti gli alberi saranno quindi abbastanza simili tra loro e le previsioni saranno altamente correlate. Abbiamo quindi che, la media di quantità molto correlate non genera una varianza così grande, come la media di molte quantità non correlate. In particolare, questo significa che il bagging non porterà a una riduzione sostanziale della varianza rispetto a un singolo albero. L'algoritmo Random Forest supera questo problema obbligando ogni split a considerare solo un sottoinsieme di variabili. Pertanto, in media $(p - m)/p$ degli split non prenderanno nemmeno in considerazione la variabile forte e quindi le altre variabili avranno più possibilità di essere scelte. Possiamo pensare a questo processo come a una decorrelazione degli alberi, che rende la media degli alberi risultanti meno variabile e quindi più stabile.

Quindi, se il bagging stabilizza gli alberi decisionali e migliora l'accuratezza andando a ridurre la varianza, l'algoritmo Random Forest migliora ulteriormente le prestazioni degli alberi decisionali decorrelando i singoli alberi nell'ensemble di bagging, inoltre le misure di importanza delle variabili delle foreste casuali possono aiutare a determinare quali variabili contribuiscono maggiormente al modello. La sostanziale differenza tra le foreste casuali e la procedura di bagging, sta proprio nell'utilizzo di un sottoinsieme di variabili m per lo split degli alberi. Si può notare, che quando $m = p$ allora il bagging è un caso speciale di Random Forest.

Boosting (accenno)

Il boosting è un altro approccio per migliorare le previsioni risultanti da un modello di statistical o machine learning. Come il bagging, il boosting è un approccio generale che può essere applicato a molti metodi, come gli alberi di regressione o classificazione.

La tecnica di boosting è un metodo di apprendimento supervisionato che mira a migliorare la capacità predittiva di un modello combinando una serie di modelli di base, chiamati anche weak learner addestrati su sottoinsiemi del dataset di training e pesati in modo tale da concentrarsi sui campioni che risultano essere difficili da classificare.

I processi di boosting si può descrivere nei seguenti passaggi:

1. Si addestra weak learning su tutto il dataset di training.
2. Si calcola l'errore del weak learner e si aggiungono dei pesi ai campioni in modo tale che i campioni classificati in modo errato abbiano un peso maggiore rispetto agli altri.
3. Si addestra un nuovo weak leaner su un sottoinsieme del dataset di addestramento, selezionando i campioni con i pesi più alti.
4. Si calcola l'errore del nuovo weak leaner e si aggiornano i pesi dei campioni in modo tale che i campioni mal classificati continuino ad avere un peso maggiore.
5. Si ripete il processo di addestramento aggiungendo nuovi weak learner fino a quando la performance sul dataset di training non migliora, o fino a quando non si raggiunge un numero fissato di weak learner.
6. Si combina l'output dei weak learner in un unico modello finale, utilizzando una combinazione pesata o una votazione.

Tale procedura presenta diversi vantaggi, tra i quali un miglioramento delle performance del modello, minor rischio di overfitting (se viene utilizzata una tecnica di regolarizzazione) e presenta una certa adattabilità a dataset complessi o che presentano del rumore. C'è da tenere in considerazione però che tale procedura ha una complessità computazionale notevole, poiché richiede l'addestramento di molti weak learner in sequenza, oltre che essere difficile da interpretare a causa della sua complessità.

Implementazione in R

Librerie

Faremo uso dei seguenti pacchetti per effettuare l'analisi sul dataset scelto.

```
library(dplyr)
```

```
##  
## Caricamento pacchetto: 'dplyr'
```

```
## I seguenti oggetti sono mascherati da 'package:stats':  
##  
##     filter, lag
```

```
## I seguenti oggetti sono mascherati da 'package:base':  
##  
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)  
library(plotly)
```

```
## Warning: il pacchetto 'plotly' è stato creato con R versione 4.2.1
```

```
##  
## Caricamento pacchetto: 'plotly'
```

```
## Il seguente oggetto è mascherato da 'package:ggplot2':  
##  
##     last_plot
```

```
## Il seguente oggetto è mascherato da 'package:stats':  
##  
##     filter
```

```
## Il seguente oggetto è mascherato da 'package:graphics':  
##  
##     layout
```

```
library(gmodels)
```

```
## Warning: il pacchetto 'gmodels' è stato creato con R versione 4.2.3
```

```
library(rpart)
library(caret)
```

```
## Warning: il pacchetto 'caret' è stato creato con R versione 4.2.1
```

```
## Caricamento del pacchetto richiesto: lattice
```

```
library(rpart.plot)
```

```
## Warning: il pacchetto 'rpart.plot' è stato creato con R versione 4.2.3
```

```
library(vip)
```

```
## Warning: il pacchetto 'vip' è stato creato con R versione 4.2.3
```

```
##
## Caricamento pacchetto: 'vip'
```

```
## Il seguente oggetto è mascherato da 'package:utils':
```

```
##     vi
```

```
library(pdp)
```

```
## Warning: il pacchetto 'pdp' è stato creato con R versione 4.2.3
```

```
library(tibble)
library(forcats)
```

```
## Warning: il pacchetto 'forcats' è stato creato con R versione 4.2.1
```

```
library(doParallel)
```

```
## Warning: il pacchetto 'doParallel' è stato creato con R versione 4.2.1
```

```
## Caricamento del pacchetto richiesto: foreach
```

```
## Warning: il pacchetto 'foreach' è stato creato con R versione 4.2.1
```

```
## Caricamento del pacchetto richiesto: iterators
```

```
## Warning: il pacchetto 'iterators' è stato creato con R versione 4.2.1
```

```
## Caricamento del pacchetto richiesto: parallel
```

```
library(foreach)  
library(randomForest)
```

```
## Warning: il pacchetto 'randomForest' è stato creato con R versione 4.2.1
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##  
## Caricamento pacchetto: 'randomForest'
```

```
## Il seguente oggetto è mascherato da 'package:ggplot2':
```

```
##  
##     margin
```

```
## Il seguente oggetto è mascherato da 'package:dplyr':
```

```
##  
##     combine
```

```
data<- read.csv("churn.csv")  
attach(data)  
data<-na.omit(data)
```

Analisi esplorativa

In questa sezione, è stata effettuata un'analisi esplorativa per indagare meglio la struttura delle variabili del dataset a disposizione su cui basare il modello e per supportare le decisioni intraprese nella parte di feature selection.

```
## [1] 7032    21
```

```

## 'data.frame': 7032 obs. of 21 variables:
## $ customerID : chr "7590-VHVEG" "5575-GNVDE" "3668-QPYBK" "7795-CFOCW" ...
## $ gender      : chr "Female" "Male" "Male" "Male" ...
## $ SeniorCitizen : int 0 0 0 0 0 0 0 0 0 ...
## $ Partner     : chr "Yes" "No" "No" "No" ...
## $ Dependents  : chr "No" "No" "No" "No" ...
## $ tenure      : int 1 34 2 45 2 8 22 10 28 62 ...
## $ PhoneService : chr "No" "Yes" "Yes" "No" ...
## $ MultipleLines : chr "No phone service" "No" "No" "No phone service" ...
## $ InternetService : chr "DSL" "DSL" "DSL" "DSL" ...
## $ OnlineSecurity : chr "No" "Yes" "Yes" "Yes" ...
## $ OnlineBackup   : chr "Yes" "No" "Yes" "No" ...
## $ DeviceProtection: chr "No" "Yes" "No" "Yes" ...
## $ TechSupport    : chr "No" "No" "No" "Yes" ...
## $ StreamingTV     : chr "No" "No" "No" "No" ...
## $ StreamingMovies  : chr "No" "No" "No" "No" ...
## $ Contract       : chr "Month-to-month" "One year" "Month-to-month" "One year" ...
## $ PaperlessBilling: chr "Yes" "No" "Yes" "No" ...
## $ PaymentMethod   : chr "Electronic check" "Mailed check" "Mailed check" "Bank transfer (automatic)" ...
## $ MonthlyCharges  : num 29.9 57 53.9 42.3 70.7 ...
## $ TotalCharges    : num 29.9 1889.5 108.2 1840.8 151.7 ...
## $ Churn          : chr "No" "No" "Yes" "No" ...
## - attr(*, "na.action")= 'omit' Named int [1:11] 489 754 937 1083 1341 3332 3827 4381 5219 66
71 ...
## ... attr(*, "names")= chr [1:11] "489" "754" "937" "1083" ...

```

Il dataset è composto da **7032** righe e **21** colonne. Fra queste abbiamo: *CustomerID*, *gender*, *SeniorCitizen* (variabile dicotomica che indica se il cliente è un cittadino anziano o no), *Partner*, *Dependents* (se il cliente ha persone a carico o meno), *tenure* (numero di mesi che il consumatore è rimasto cliente dell'azienda), una serie di variabili legate ai servizi del cliente (*PhoneService*, *MultipleLines*, *InternetService*, *OnlineSecurity*, *OnlineBackup*, *DeviceProtection*, *TechSupport*, *StreamingTV*, *StreamingMovies*), *Contract* (variabile che indica i termini del contratto del cliente), *PaperlessBilling* (se il cliente ha o meno una fatturazione senza carta), *PaymentMethod*, *MonthlyCharges* (l'importo addebitato mensilmente al cliente), *TotalCharges* (l'importo totale addebitato al cliente), *Churn*.

```
summary(data$tenure)
```

```

##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 1.00   9.00  29.00  32.42  55.00  72.00

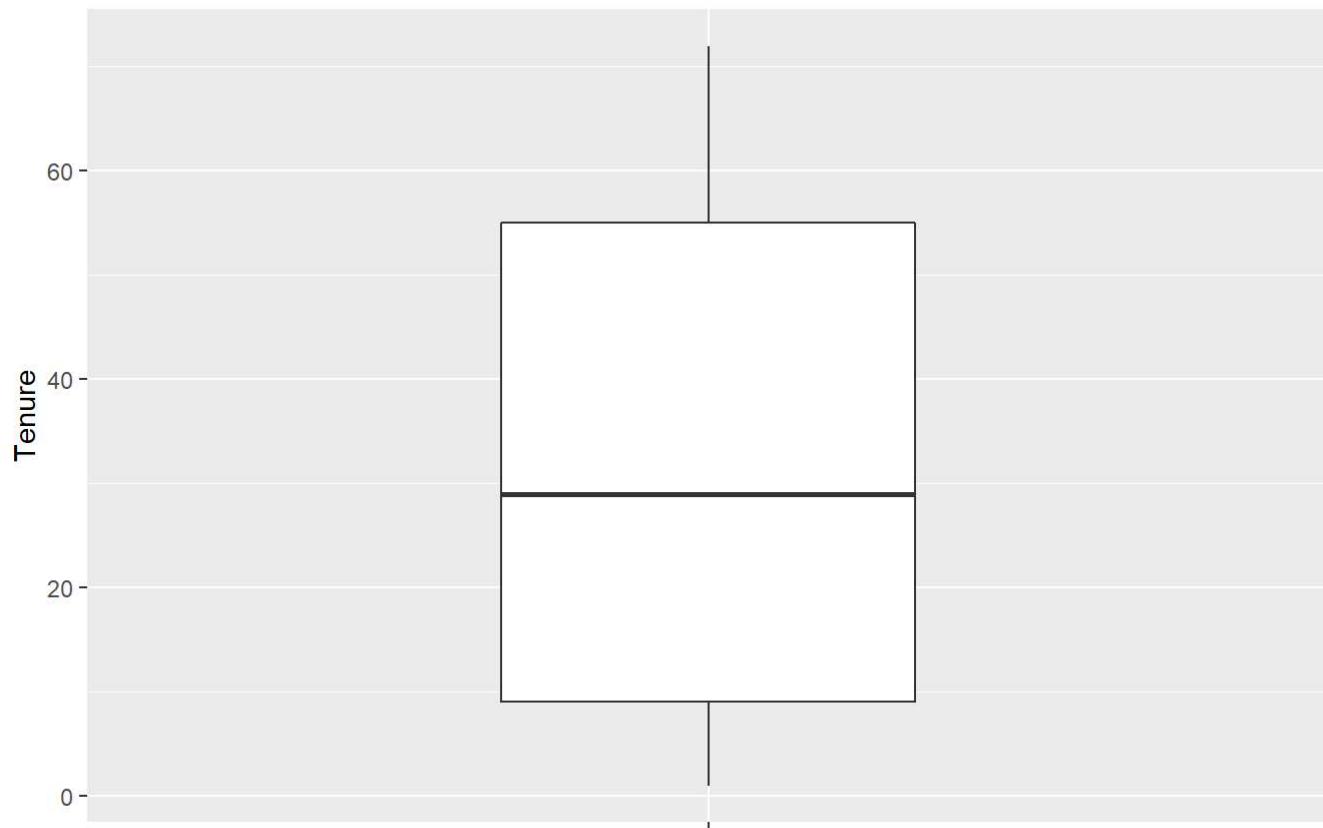
```

```
sd(data$tenure)
```

```
## [1] 24.54526
```

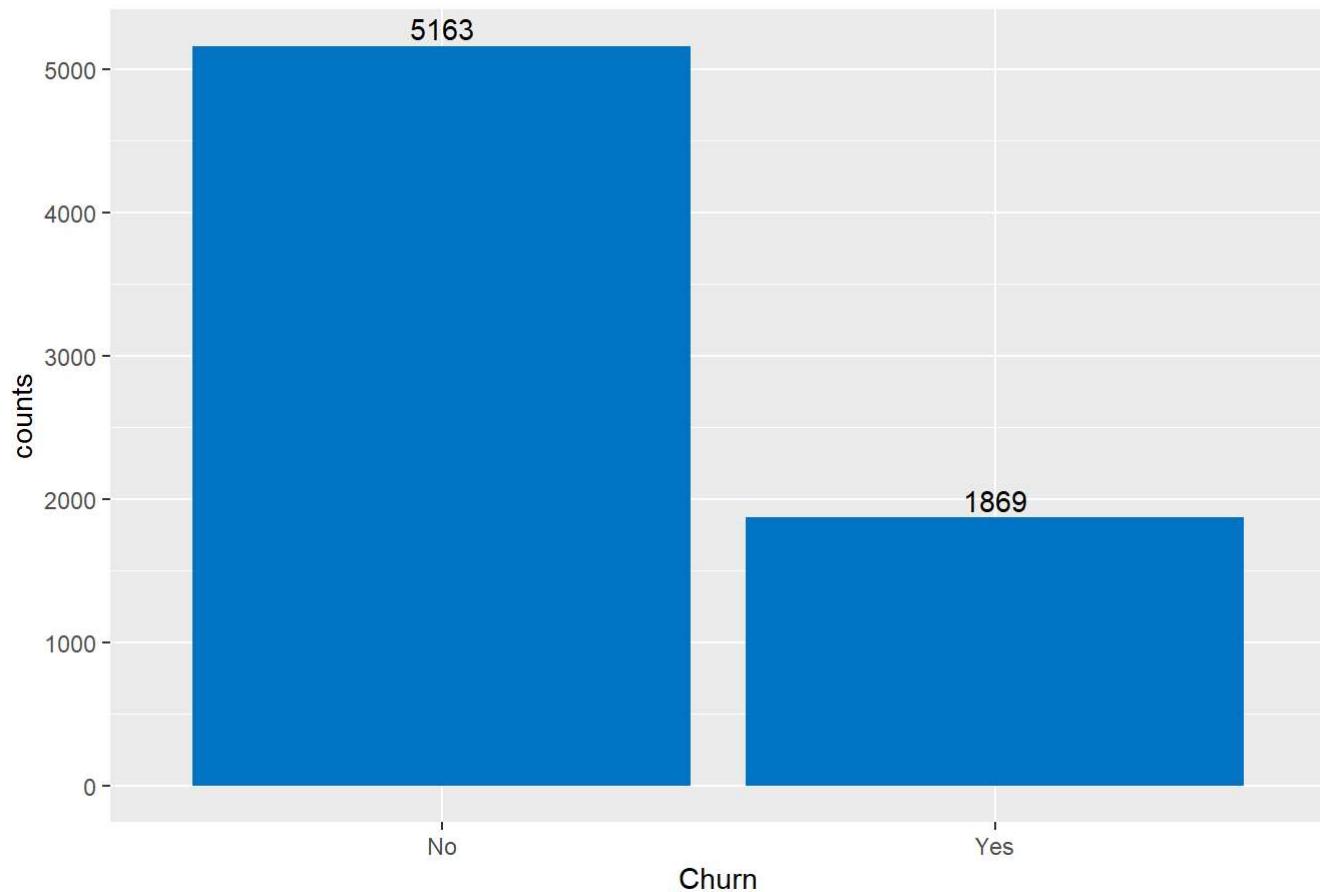
Particolarmente interessante per la nostra analisi di regressione è lo studio della variabile target *tenure*. Come possiamo vedere la deviazione standard di questa variabile è pari a **24.5**, con una media di **32.4**.

Tenure distribution



L'analisi del boxplot relativo alla variabile target, mostra come tale variabile presenti un'elevata variabilità.

Churn distribution

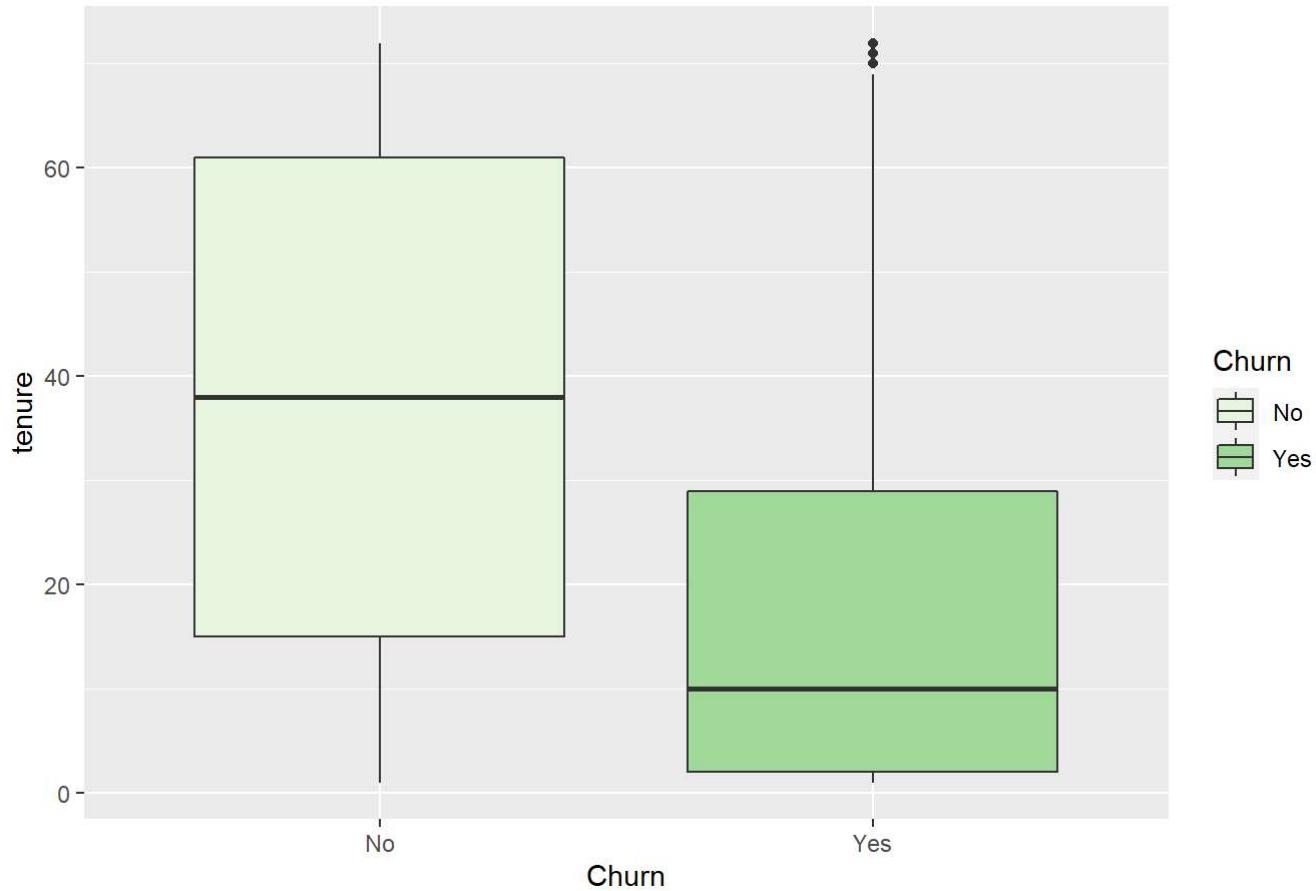


Inoltre, per quanto riguarda la variabile di *churn*, la quale risulterà essere la variabile target per il task di classificazione, si nota come essa sia particolarmente sbilanciata, con un numero di clienti che hanno effettuato il churn nettamente inferiore rispetto a coloro che non lo hanno fatto.

Analizziamo ora, il comportamento della variabile *churn* condizionata ai valori della variabile *tenure*:

```
## Warning in pal_name(palette, type): Unknown palette Green
```

Churn vs tenure

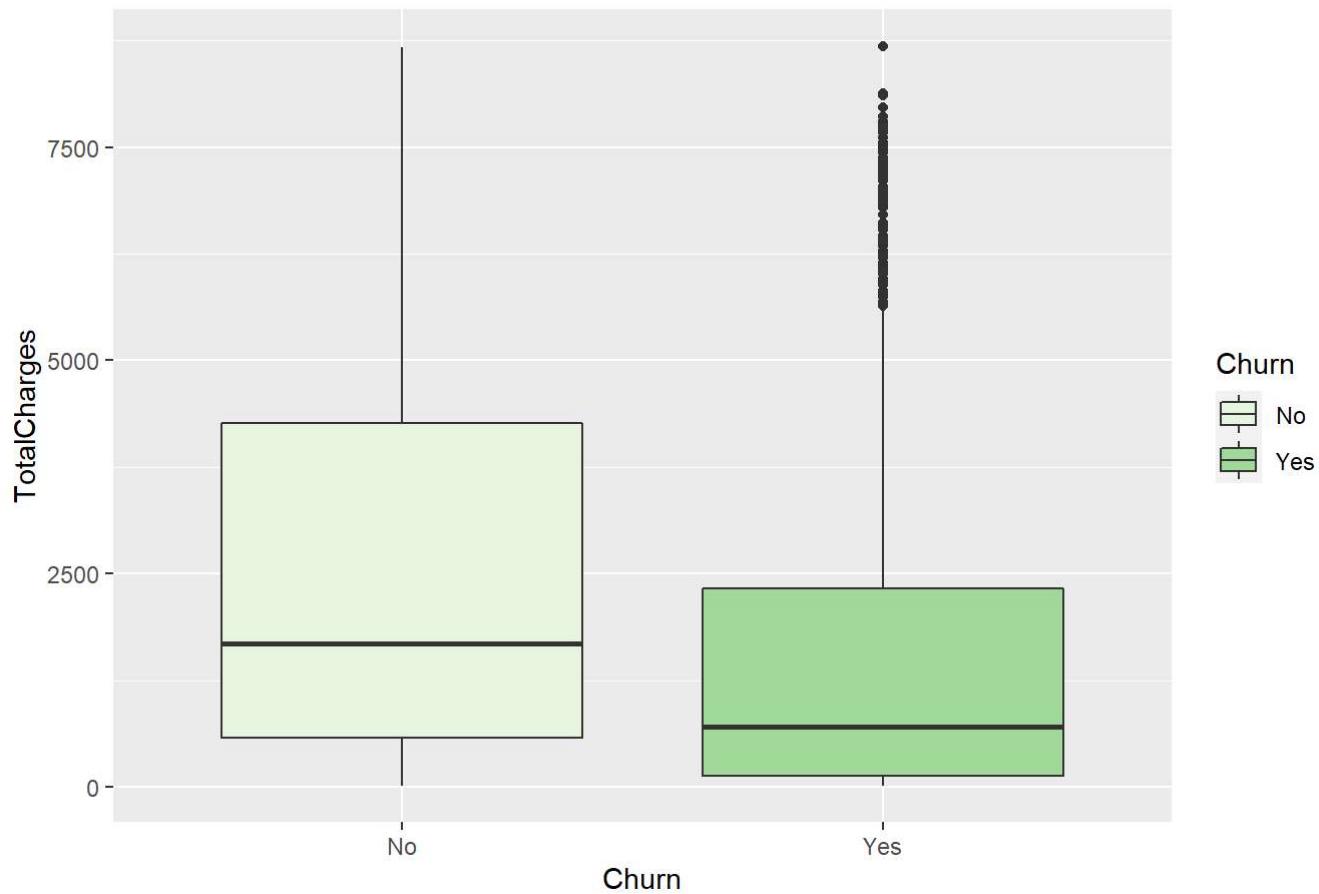


Dal precedente grafico possiamo notare come la variabile *tenure* sia una variabile potenzialmente utile nello spiegare il comportamento della variabile *churn*, indicando che clienti con un numero di mesi trascorsi con l'azienda maggiore presentano una minore probabilità di abbandono.

Mentre, analizzando il comportamento della variabile *churn* condizionata alla variabile *TotalCharges*:

```
## Warning in pal_name(palette, type): Unknown palette Green
```

Churn vs TotalCharges

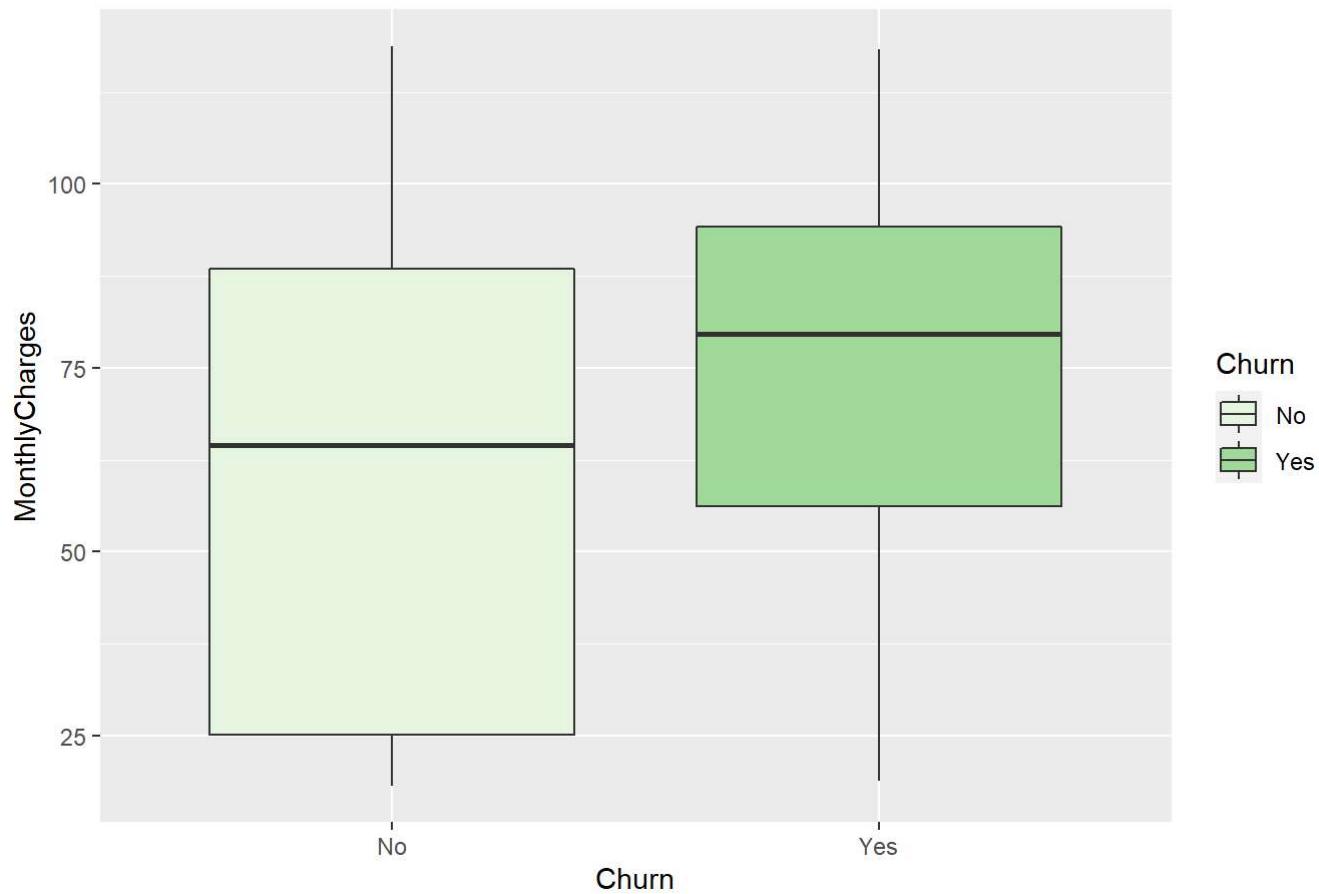


Notiamo come la variabile *TotalCharges* rappresenta anch'essa un fattore potenzialmente determinante nello spiegare il comportamento della variabile di *Churn*. Maggiore l'ammontare totale speso, minore la probabilità di abbandono.

Infine, mostriamo il comportamento della variabile *churn* condizionata ai valori della variabile *MonthlyCharges*:

```
## Warning in pal_name(palette, type): Unknown palette Green
```

Churn vs MonthlyCharges



Analizziamo ora, la correlazione tra le variabili *TotalCharges* e *MonthlyCharges*:

```
cor(data$TotalCharges, data$MonthlyCharges)
```

```
## [1] 0.6510648
```

Data la dipendenza abbastanza marcata tra le variabili *TotalCharges* e *MonthlyCharges*, decidiamo di escludere una delle due variabili e teniamo per l'analisi la variabile *MonthlyCharges*.

Ragionamento simile per le variabili *PhoneService* e *MultipleLines* nel quale la seconda è inevitabilmente legata ai valori assunti dalla prima. Decidiamo così di includere nell'analisi solo la variabile *PhoneService*.

Lo stesso ragionamento è stato effettuato per quanto riguarda il legame tra la variabile *InternetService* e le variabili *OnlineSecurity*, *OnlineBackup*, *DeviceProtection*, *TechSupport*, *StreamingTV*, *StreamingMovies*. Anche in questo caso decidiamo di tenere in considerazione esclusivamente la variabile *InternetService* poiché i valori assunti dalle altre variabili risultano influenzati dai valori di quest'ultima.

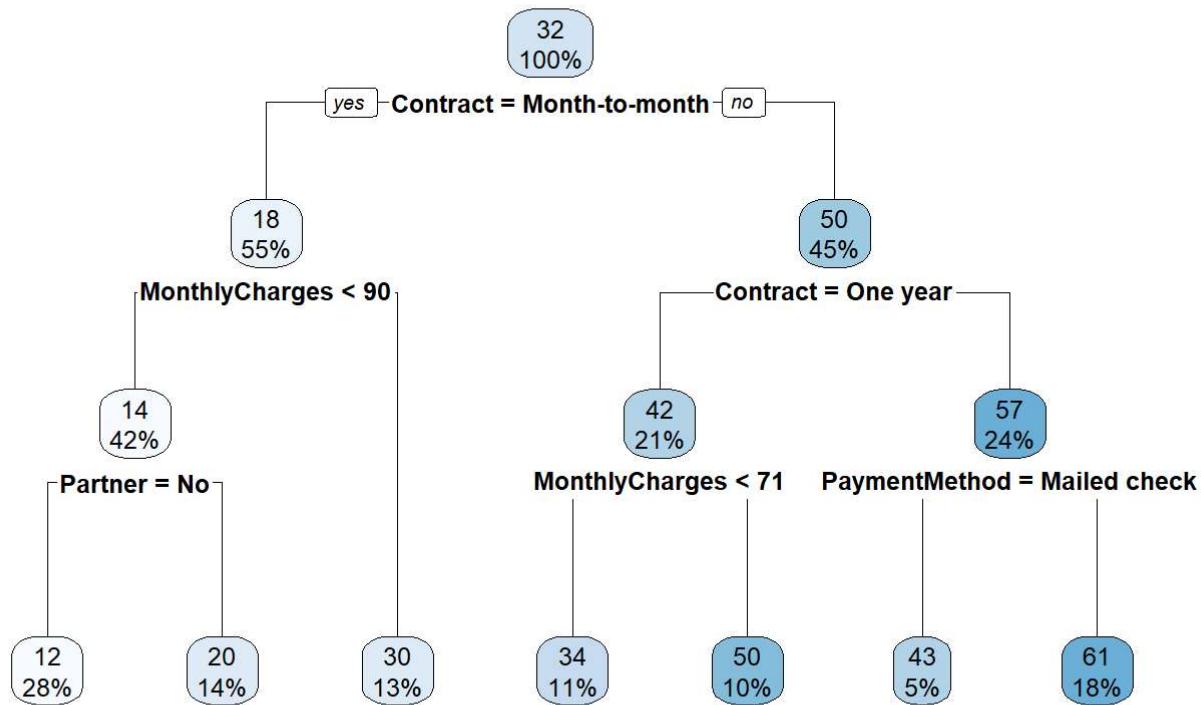
Analisi di Regressione: Regression Tree

L'analisi di regressione viene effettuata cercando di prevedere i valori assunti dalla variabile *tenure* mediante l'utilizzo delle altre variabili rimaste come regressori.

Albero di regressione

Come detto in precedenza il modello che decidiamo di utilizzare come weak-learner è l'albero decisionale. Per la costruzione dell'albero di regressione usiamo la funzione `rpart()` contenuta nel pacchetto rpart, dopodiché per la visualizzazione dell'albero utilizziamo la funzione `rpart.plot()`. Il processo di addestramento e visualizzazione del modello è molto simile sia per la parte di regressione che per quella di classificazione.

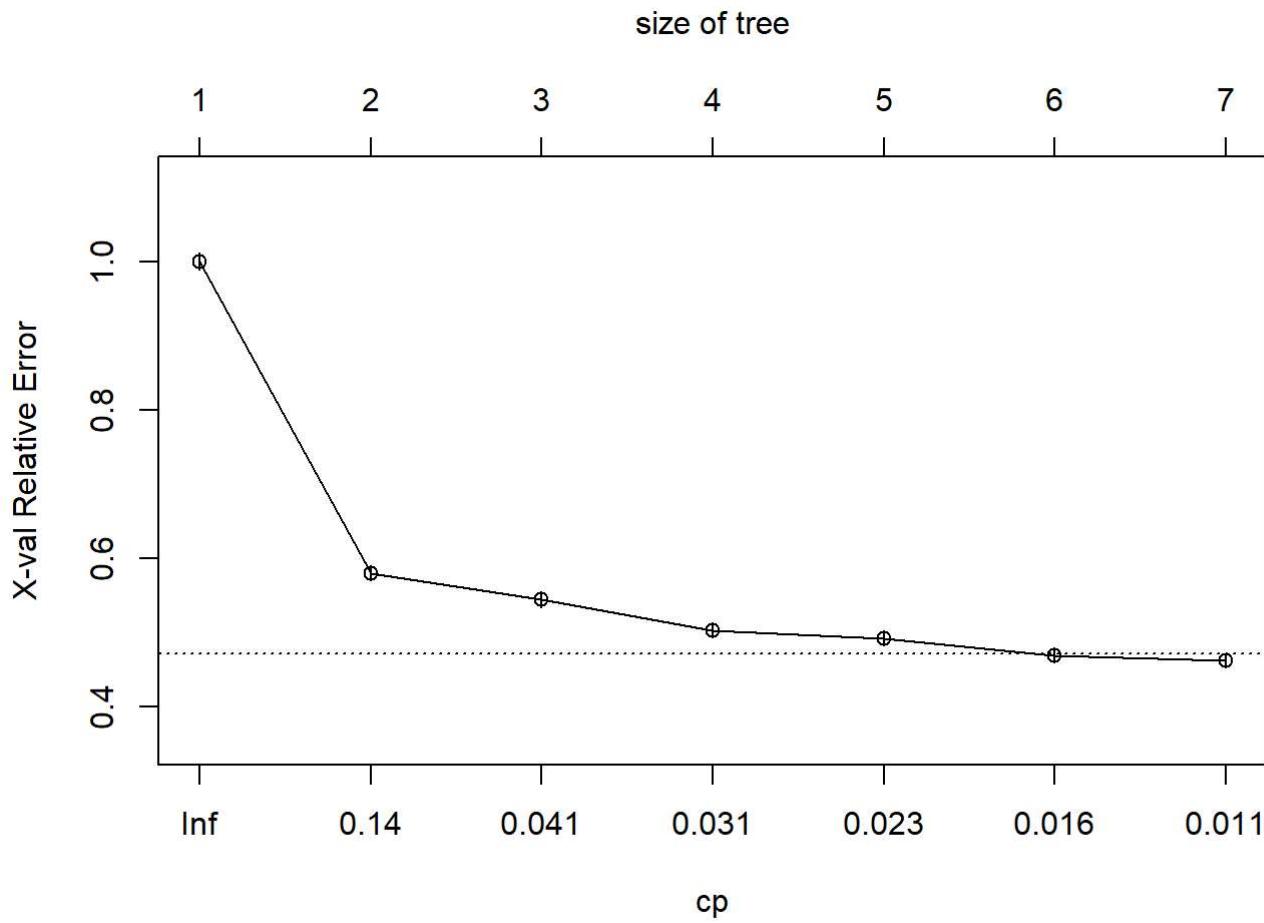
```
fit <- rpart(
  formula = tenure ~ .,
  data     = traindf,
  method   = "anova"
)
```



La variabile che decreta il primo split (cioè la variaile che restituisce la più grande riduzione nel SSE) è `Contract`.

Visualizzando il modello ad albero con `rpart.plot()`, possiamo notare che l'albero contiene 6 nodi interni e 7 risultanti nodi terminali.

Di default `rpart()` automaticamente applica un range di valori di cost complexity (`cp values`) per potare l'albero. Per confrontare l'errore associato a ciascun `cp value`, `rpart()` performa un 10-fold CV.



Il grafico di pruning complexity parameter (*cp*) illustra il relativo cross validation error (y-axis) per vari *cp* values (lower x-axis). Piccoli valori di *cp* portano ad alberi più grandi (upper x-axis). Dal precedente grafico si può notare come un valore ottimale di *cp* è **0.031**, offrendo un buon bilanciamento tra complessità del modello e relativo errore.

```
pruned <- prune(fit, cp=0.031)
preds <- predict(pruned, testdf)
rmse <- RMSE(
  pred = preds,
  obs = testdf$tenure
)
rmse
```

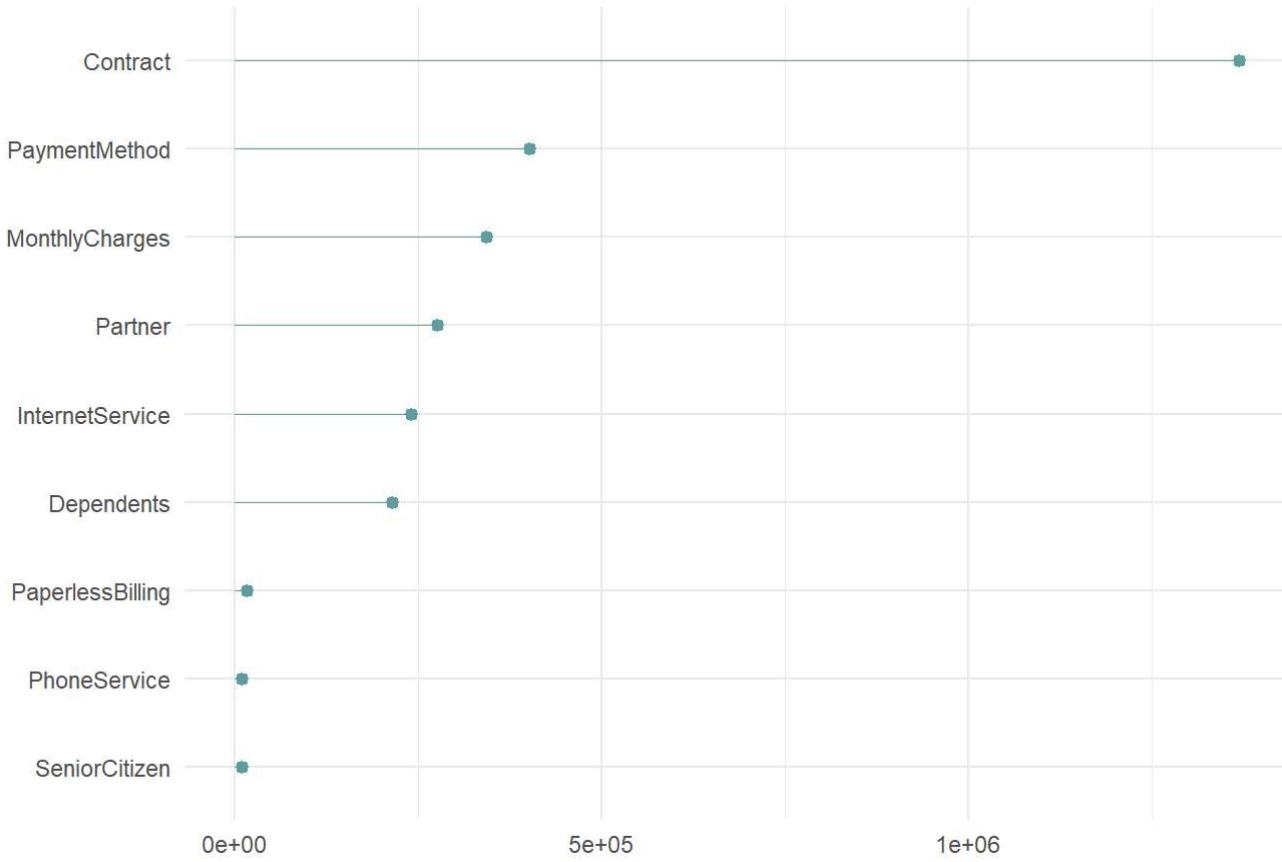
```
## [1] 17.50233
```

Il root mean square error ottenuto valutando il modello finale sul test set è pari a **17.5**. Considerando che lo scarto quadratico medio per la variabile *tenure* come visto in precedenza è **24.5**, il risultato ottenuto dal nostro modello può essere considerato soddisfacente.

Per misurare l'importanza che le varie features assumono nello spiegare il comportamento della variabile target, viene considerata la riduzione nella funzione di perdita (ovvero, SSE) attribuita ad ogni variabile ad ogni split. In alcuni casi, una variable potrebbe essere usata molte volte in un albero; di conseguenza, la riduzione totale nella

funzione di perdita causata da una variabile nei vari split sono sommate e usate per la feature importance. Gli alberi decisionali eseguono automaticamente feature selection dal momento che le features non informative non vengono usate dal modello.

Variable Importance with Simple Regression



Dal precedente grafico, che illustra l'importanza che le varie features hanno nello spiegare il comportamento della variabile target, sono sicuramente da evidenziare le variabili *Contract*, *PaymentMethod* e *MonthlyCharges*.

In conclusione, gli alberi decisionali hanno diversi vantaggi:

- Richiedono poco pre-processing, questo non vuol dire che il feature engineering non migliori le prestazioni del modello, ma piuttosto che non ci sono particolari requisiti di pre-processing.
- Solitamente gli outliers non distorgono tanto i risultati.
- Gli alberi decisionali possono facilmente gestire categorical features senza pre-processing.

Tuttavia, essi spesso non riescono a raggiungere ottimi risultati in termini di performance.

Bagging

La performance ottimale nei bagged models viene spesso trovata unendo dai 50 ai 500 alberi. Dataset con pochi predittori richiedono spesso meno alberi; mentre i set di dati con molto rumore o più predittori forti potrebbero aver bisogno di più alberi.

Per questa analisi decidiamo di utilizzare 100 alberi non potati (non potando gli alberi stiamo mantenendo una bassa distorsione ma un'elevata varianza ed è in questo caso che è possibile ottenere un effetto maggiore dal bagging). Una cosa da notare è che tipicamente, più alberi vengono utilizzati e migliore saranno le prestazioni del

modello di bagging, dal momento che aggiungendo più alberi mediamo tra più modelli decisionali ad elevata varianza. Tipicamente, l'errore di previsione si appiattisce e si stabilizza una volta raggiunto un numero adeguato di alberi.

```
set.seed(1234)
bag_model <- train(
  tenure ~.,
  data = traindf,
  nbagg = 100,
  trControl = trainControl(method = "cv", number = 10),
  method = "treebag",
  control = rpart.control(minsplit = 2, cp = 0)
)
bag_model
```

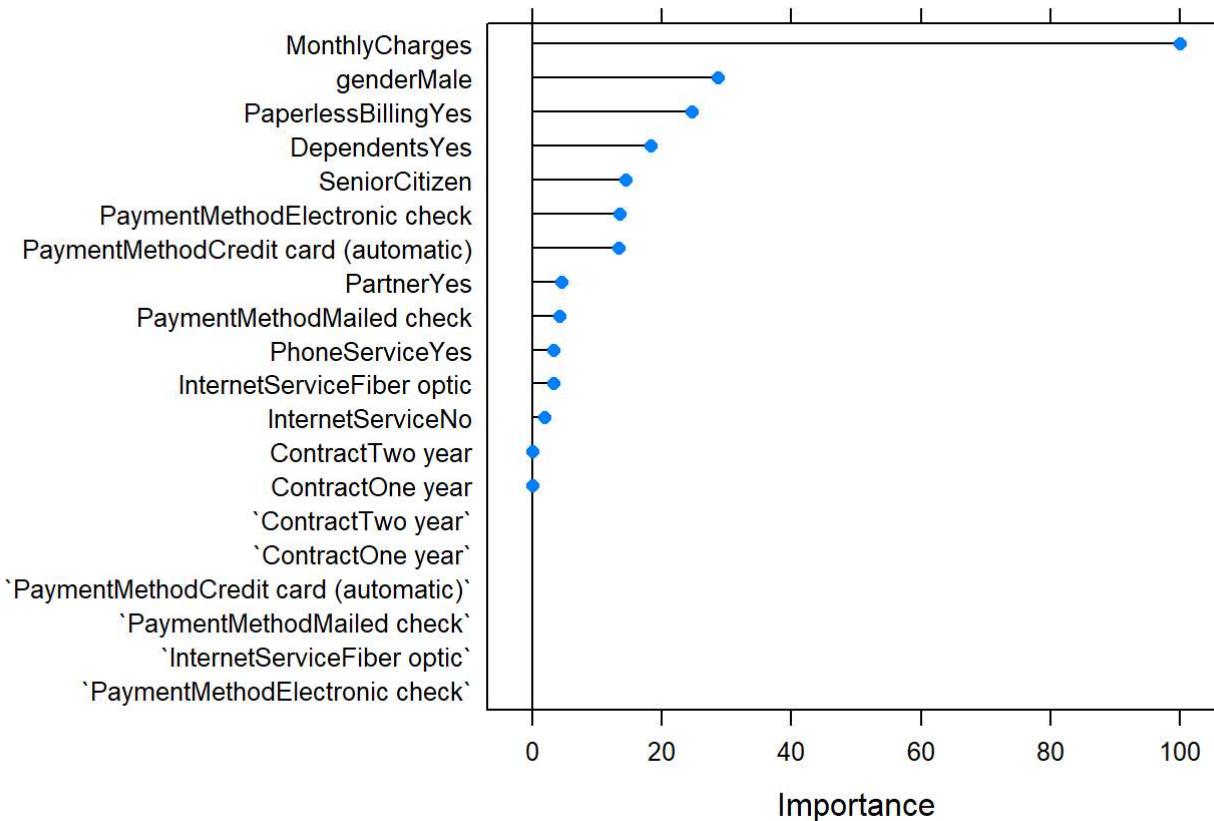
```
## Bagged CART
##
## 4922 samples
##   10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4430, 4429, 4429, 4430, 4430, 4429, ...
## Resampling results:
##
##    RMSE      Rsquared     MAE
##    16.46922  0.5564838  12.68355
```

```
## [1] 16.58013
```

Il mean squared error ottenuto dal nostro modello risulta pari a **16.58**.

Sfortunatamente, per via del processo di bagging, modelli come gli alberi decisionali che sono percepiti come interpretabili e visualizzabili adesso non lo sono più. Tuttavia, possiamo ancora fare inferenza su come le varie features influenzano il nostro modello. Per i bagged decision trees, questo processo è simile a quello degli alberi decisionali. Per ciascun albero, si calcola la somma delle funzioni di perdita fra tutti gli split. Dopodiché per ciascuna feature si aggrega questa misura per tutti gli alberi. Le features con la più grande riduzione nel SSE (per la regressione) sono considerate importanti.

Variable Importance with Bagging



In questo caso vediamo come la variabile più importante per il nostro modello è *MonthlyCharges* e non più *Contract* come nel precedente modello.

```
##          Model      RMSE
## 1      Bagging 16.58013
## 2 Single Tree 17.50233
```

Confrontando il root mean squared error del modello di bagging con quello ottenuto dal decision tree, vediamo come siamo riusciti ad ottenere una buona riduzione dell'errore commesso dal precedente modello.

Classificazione: Classification Tree

Passiamo ora al task di classificazione. In questo caso vogliamo prevedere la variabile *Churn*, la quale assume valori “Yes”, se il cliente ha abbandonato la compagnia, “No” nel caso in cui risulti ancora presente.

```
data <- read.csv("churn.csv")
attach(data)
```

```
## I seguenti oggetti sono mascherati da data (pos = 3):
##
##   Churn, Contract, customerID, Dependents, DeviceProtection, gender,
##   InternetService, MonthlyCharges, MultipleLines, OnlineBackup,
##   OnlineSecurity, PaperlessBilling, Partner, PaymentMethod,
##   PhoneService, SeniorCitizen, StreamingMovies, StreamingTV,
##   TechSupport, tenure, TotalCharges
```

```
data<-na.omit(data)
```

Un albero di classificazione è molto simile ad un albero di regressione, ma viene utilizzato per prevedere una risposta qualitativa anziché quantitativa.

Ricordiamo che per un albero di regressione, la risposta prevista per un'osservazione è data dalla risposta media delle osservazioni di training che appartengono allo stesso nodo terminale.

Nell'interpretare i risultati di un albero di classificazione, spesso si è interessati non solo alla previsione della classe corrispondente a una particolare regione del nodo terminale, ma anche alle proporzioni delle classi tra le osservazioni di training che rientrano in quella regione.

Come nel caso della regressione, per costituire un albero di classificazione si utilizza la suddivisione binaria ricorsiva. Tuttavia, nell'impostazione della classificazione, la *Residual Sum of Squares* non può essere utilizzata come criterio per effettuare le suddivisioni binarie. Si può invece utilizzare uno dei 3 metodi seguenti: Classification Error Rate, Indice di Gini, Cross-Entropy.

Quando si costruisce un albero di classificazione, per valutare la qualità di una particolare suddivisione si utilizzano in genere l'indice di Gini o la cross-entropy, poiché sono più sensibili alla "purezza" dei nodi rispetto al classification error rate.

Dapprima vengono escluse le variabili in base a quanto detto in fase di esplorazione del dataset, e poi vengono trasformate le restanti variabili qualitative in *factor*:

```
data <- subset(data,select=-c(customerID, MultipleLines, OnlineSecurity, OnlineBackup, DeviceProtection, TotalCharges, TechSupport, StreamingTV, StreamingMovies))

str(data)
```

```
## 'data.frame': 7032 obs. of 12 variables:
## $ gender : chr "Female" "Male" "Male" "Male" ...
## $ SeniorCitizen : int 0 0 0 0 0 0 0 0 ...
## $ Partner : chr "Yes" "No" "No" "No" ...
## $ Dependents : chr "No" "No" "No" "No" ...
## $ tenure : int 1 34 2 45 2 8 22 10 28 62 ...
## $ PhoneService : chr "No" "Yes" "Yes" "No" ...
## $ InternetService : chr "DSL" "DSL" "DSL" "DSL" ...
## $ Contract : chr "Month-to-month" "One year" "Month-to-month" "One year" ...
## $ PaperlessBilling: chr "Yes" "No" "Yes" "No" ...
## $ PaymentMethod : chr "Electronic check" "Mailed check" "Mailed check" "Bank transfer (automatic)" ...
## $ MonthlyCharges : num 29.9 57 53.9 42.3 70.7 ...
## $ Churn : chr "No" "No" "Yes" "No" ...
```

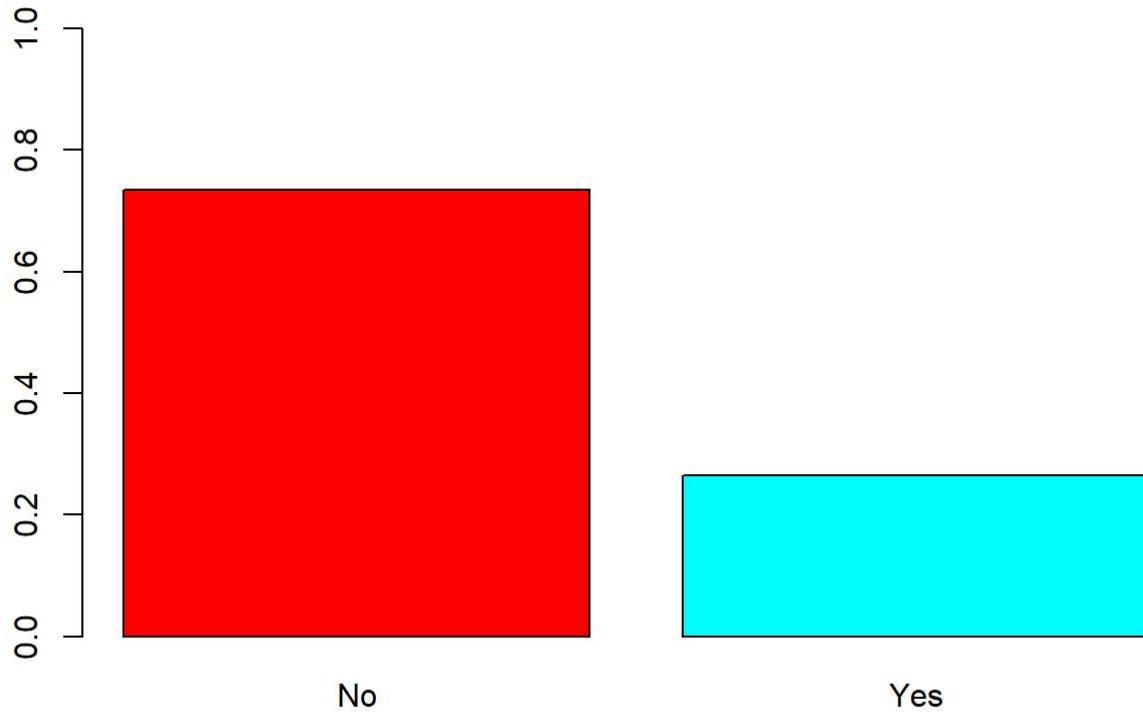
```
categorical_var = c("gender", "SeniorCitizen", "Partner", "Dependents", "PhoneService", "InternetService", "Contract", "PaperlessBilling", "PaymentMethod", "Churn")
data[,categorical_var] <- lapply(data[,categorical_var], as.factor)
str(data)
```

```
## 'data.frame': 7032 obs. of 12 variables:
## $ gender : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 1 2 1 1 2 ...
## $ SeniorCitizen : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
## $ Partner : Factor w/ 2 levels "No","Yes": 2 1 1 1 1 1 1 1 2 1 ...
## $ Dependents : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 2 1 1 2 ...
## $ tenure : int 1 34 2 45 2 8 22 10 28 62 ...
## $ PhoneService : Factor w/ 2 levels "No","Yes": 1 2 2 1 2 2 2 1 2 2 ...
## $ InternetService : Factor w/ 3 levels "DSL","Fiber optic",...: 1 1 1 1 2 2 2 1 2 1 ...
## $ Contract : Factor w/ 3 levels "Month-to-month",...: 1 2 1 2 1 1 1 1 1 2 ...
## $ PaperlessBilling: Factor w/ 2 levels "No","Yes": 2 1 2 1 2 2 2 1 2 1 ...
## $ PaymentMethod : Factor w/ 4 levels "Bank transfer (automatic)",...: 3 4 4 1 3 3 2 4 3 1 ...
...
## $ MonthlyCharges : num 29.9 57 53.9 42.3 70.7 ...
## $ Churn : Factor w/ 2 levels "No","Yes": 1 1 2 1 2 2 1 1 2 1 ...
```

Come già detto in precedenza, la variabile target presenta uno sbilanciamento marcato verso la classe "No", che rappresenta circa il 70% del totale delle osservazioni, come è possibile vedere in figura:

```
barplot(prop.table(table(Churn)),
       col = rainbow(2),
       ylim = c(0, 1),
       main = "Class Distribution")
```

Class Distribution



La presenza di classi sbilanciate, può causare problemi nella fase di modellizzazione e di valutazione delle prestazioni del modello.

In particolare, gli ostacoli associati al problema delle classi sbilanciate includono:

- *Prestazioni del modello*: se la classe di minoranza è sotto-rappresentata, il modello potrebbe essere meno preciso e meno affidabile nella classificazione di questa classe rispetto alla classe di maggioranza, poiché il modello tende ad assegnare la classe maggioritaria alla maggior parte delle osservazioni.
- *Bias di valutazione*: se si utilizzano metriche di valutazione del modello che non tengono conto dello sbilanciamento delle classi (ad esempio l'accuratezza), si potrebbe ottenere una valutazione del modello che non rispecchia la sua vera capacità di classificazione.
- *Overfitting*: se la classe di minoranza è poco rappresentata, il modello potrebbe soffrire di overfitting, ovvero potrebbe adattarsi eccessivamente ai dati di training, ma non generalizzare adeguatamente sui dati di test.

Per gestire il problema delle classi sbilanciate, ci sono diverse tecniche che possono essere utilizzate, tra cui:

- *Oversampling* della classe di minoranza: consiste nell'aumentare il numero di campioni della classe di minoranza, ad esempio mediante tecniche di replicazione o sintesi di campioni.
- *Undersampling* della classe di maggioranza: consiste nel ridurre il numero di campioni della classe di maggioranza, ad esempio mediante tecniche di campionamento casuale.
- Utilizzo di algoritmi di classificazione *cost sensitive*: L'idea alla base di questa tecnica è quella di associare all'algoritmo di classificazione una matrice di costo, la quale può essere utilizzata per assegnare un costo maggiore alla classificazione errata della classe di minoranza, rispetto alla classe di maggioranza. In questo modo, il modello sarà incentivato a dare più importanza alla classificazione corretta della classe minoritaria,

in quanto l'errore in questa classe avrà un peso maggiore rispetto all'errore nella classe di maggioranza.

Tuttavia, è importante notare che l'utilizzo di una matrice di costo richiede la conoscenza del costo

associato a ciascuna combinazione di classificazione, il che può essere difficile da definire in alcuni casi.

Nel nostro caso, non potendo ipotizzare una adeguata matrice di costo e non essendo disposti a escludere troppe osservazioni tramite l'utilizzo di una tecnica di *undersampling*, si decide di procedere tramite un *oversampling* della classe minotaria. In particolare, in questo contesto verrà utilizzata la libreria *ROSE()*.

Dividiamo dapprima il dataset in training e test set, mantenendo così la vera distribuzione dei dati per il test set. Dopodichè, verrà applicata la tecnica di oversampling sul dataset di training.

```
set.seed(1234)
split <- sample(nrow(data), floor(0.7*nrow(data)))
traindf <- data[split,]
testdf <- data[-split,]
```

Applichiamo l'algoritmo ROSE, Random Over-Sampling Examples, il quale genera sinteticamente nuovi campioni della classe minoritaria. Questa tecnica di sintesi di campioni è basata sull'algoritmo SMOTE (Synthetic Minority Over-sampling Technique), ma con alcune modifiche per migliorarne l'efficacia.

```
library(ROSE)
```

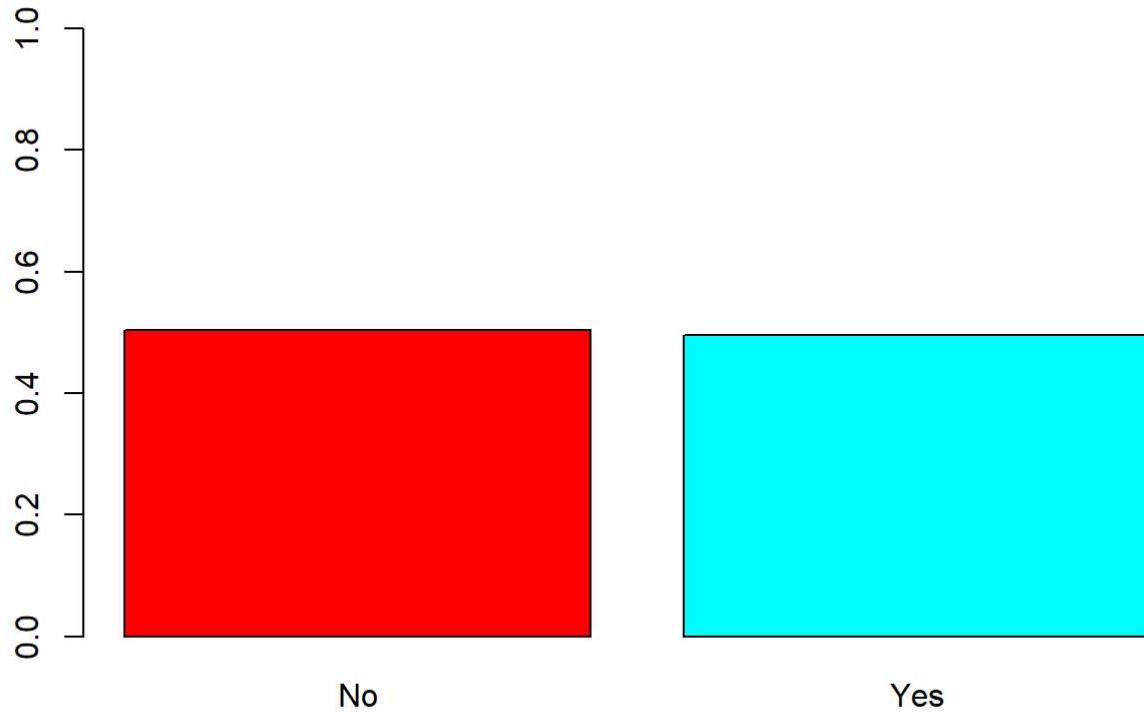
```
## Warning: il pacchetto 'ROSE' è stato creato con R versione 4.2.3
```

```
## Loaded ROSE 0.0-4
```

```
traindf_ROSE <- ROSE(Churn ~ ., data = traindf, seed = 123)$data
```

```
barplot(prop.table(table(traindf_ROSE$Churn)),
       col = rainbow(2),
       ylim = c(0, 1),
       main = "Class Distribution")
```

Class Distribution



Dal grafico sovrastante, possiamo notare come il problema delle classi sbilanciate sia risolto. Passiamo ora all' stima dell'albero di classificazione.

Albero di Classificazione

```
fit <- rpart(  
  formula = Churn ~ .,  
  data    = traindf_ROSE,  
  method = "class"  
)
```

```
summary(fit)
```

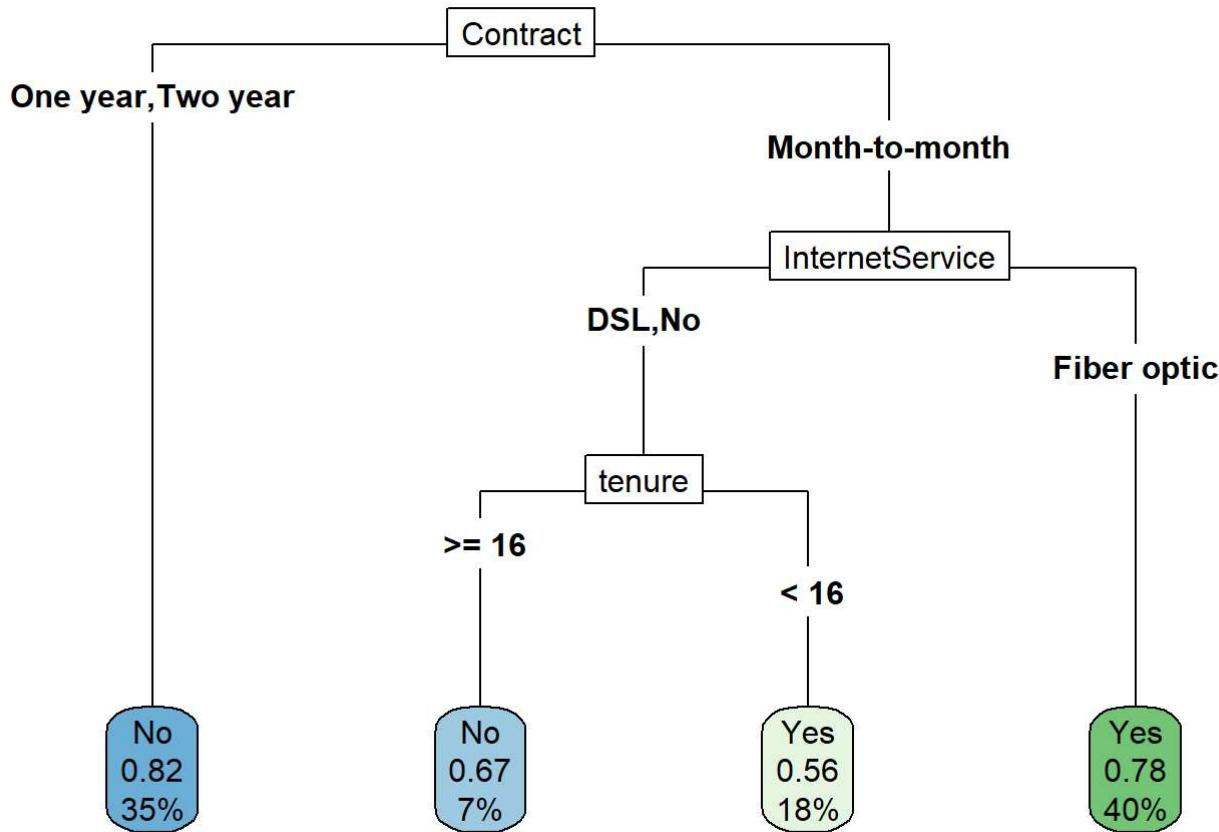
```

## Call:
## rpart(formula = Churn ~ ., data = traindf_ROSE, method = "class")
## n= 4922
##
##          CP nsplit rel error      xerror      xstd
## 1 0.43342892     0 1.0000000 1.0000000 0.01437006
## 2 0.02580909     1 0.5665711 0.5665711 0.01291852
## 3 0.01000000     3 0.5149529 0.5256043 0.01261728
##
## Variable importance
##          Contract        tenure InternetService MonthlyCharges
##             41            22           13              10
## PaymentMethod Dependents PhoneService PaperlessBilling
##             7              4              2                  1
##
## Node number 1: 4922 observations,      complexity param=0.4334289
## predicted class=No  expected loss=0.4959366  P(node) =1
##   class counts: 2481 2441
##   probabilities: 0.504 0.496
## left son=2 (1710 obs) right son=3 (3212 obs)
## Primary splits:
##   Contract      splits as RLL,      improve=526.6005, (0 missing)
##   InternetService splits as LRL,      improve=320.6511, (0 missing)
##   tenure         < 18.11504 to the right, improve=279.1541, (0 missing)
##   PaymentMethod   splits as LLRL,      improve=256.0469, (0 missing)
##   PaperlessBilling splits as LR,      improve=143.2739, (0 missing)
## Surrogate splits:
##   tenure         < 39.10118 to the right, agree=0.817, adj=0.472, (0 split)
##   PaymentMethod   splits as LLRR,      agree=0.690, adj=0.109, (0 split)
##   Dependents      splits as RL,      agree=0.683, adj=0.087, (0 split)
##   InternetService splits as RRL,      agree=0.680, adj=0.078, (0 split)
##   MonthlyCharges < 108.2742 to the right, agree=0.668, adj=0.044, (0 split)
##
## Node number 2: 1710 observations
## predicted class=No  expected loss=0.1789474  P(node) =0.3474197
##   class counts: 1404 306
##   probabilities: 0.821 0.179
##
## Node number 3: 3212 observations,      complexity param=0.02580909
## predicted class=Yes  expected loss=0.3353051  P(node) =0.6525803
##   class counts: 1077 2135
##   probabilities: 0.335 0.665
## left son=6 (1251 obs) right son=7 (1961 obs)
## Primary splits:
##   InternetService splits as LRL,      improve=122.77790, (0 missing)
##   MonthlyCharges    < 63.68517 to the left,  improve= 74.44489, (0 missing)
##   PaymentMethod      splits as LLRL,      improve= 59.32190, (0 missing)
##   PaperlessBilling   splits as LR,      improve= 50.39530, (0 missing)
##   tenure            < 17.7115 to the right, improve= 37.51608, (0 missing)
## Surrogate splits:
##   MonthlyCharges    < 63.41834 to the left,  agree=0.942, adj=0.851, (0 split)
##   PaymentMethod      splits as RRRL,      agree=0.720, adj=0.282, (0 split)

```

```
##      PhoneService      splits as  LR,           agree=0.700, adj=0.229, (0 split)
##      PaperlessBilling splits as  LR,           agree=0.667, adj=0.145, (0 split)
##      tenure            < 3.480934 to the left, agree=0.620, adj=0.024, (0 split)
##
## Node number 6: 1251 observations,    complexity param=0.02580909
##   predicted class=No  expected loss=0.4916067 P(node) =0.254165
##   class counts:  636 615
##   probabilities: 0.508 0.492
##   left son=12 (362 obs) right son=13 (889 obs)
## Primary splits:
##   tenure          < 15.75918 to the right, improve=27.952710, (0 missing)
##   PhoneService    splits as  RL,           improve=21.632350, (0 missing)
##   InternetService splits as  R-L,          improve= 9.673739, (0 missing)
##   MonthlyCharges < 62.08806 to the right, improve= 9.273496, (0 missing)
##   PaymentMethod   splits as  LLRL,         improve= 9.034981, (0 missing)
##
## Node number 7: 1961 observations
##   predicted class=Yes  expected loss=0.2248853 P(node) =0.3984153
##   class counts:  441 1520
##   probabilities: 0.225 0.775
##
## Node number 12: 362 observations
##   predicted class=No  expected loss=0.3259669 P(node) =0.07354734
##   class counts:  244 118
##   probabilities: 0.674 0.326
##
## Node number 13: 889 observations
##   predicted class=Yes  expected loss=0.4409449 P(node) =0.1806176
##   class counts:  392 497
##   probabilities: 0.441 0.559
```

```
rpart.plot(fit, extra=108, type=5)
```



Il summary del modello mostra i risultati dell'adattamento dell'albero di decisione per la variabile di risposta Churn. Il modello ha un parametro di complessità di 0,433 e ha suddiviso il dataset in due nodi principali, uno con 1710 osservazioni e l'altro con 3212 osservazioni. Il nodo 1 ha una probabilità del 50,4% per la classe No e del 49,6% per la classe Yes, mentre il nodo 3 ha una probabilità del 33,5% per la classe No e del 66,5% per la classe Yes.

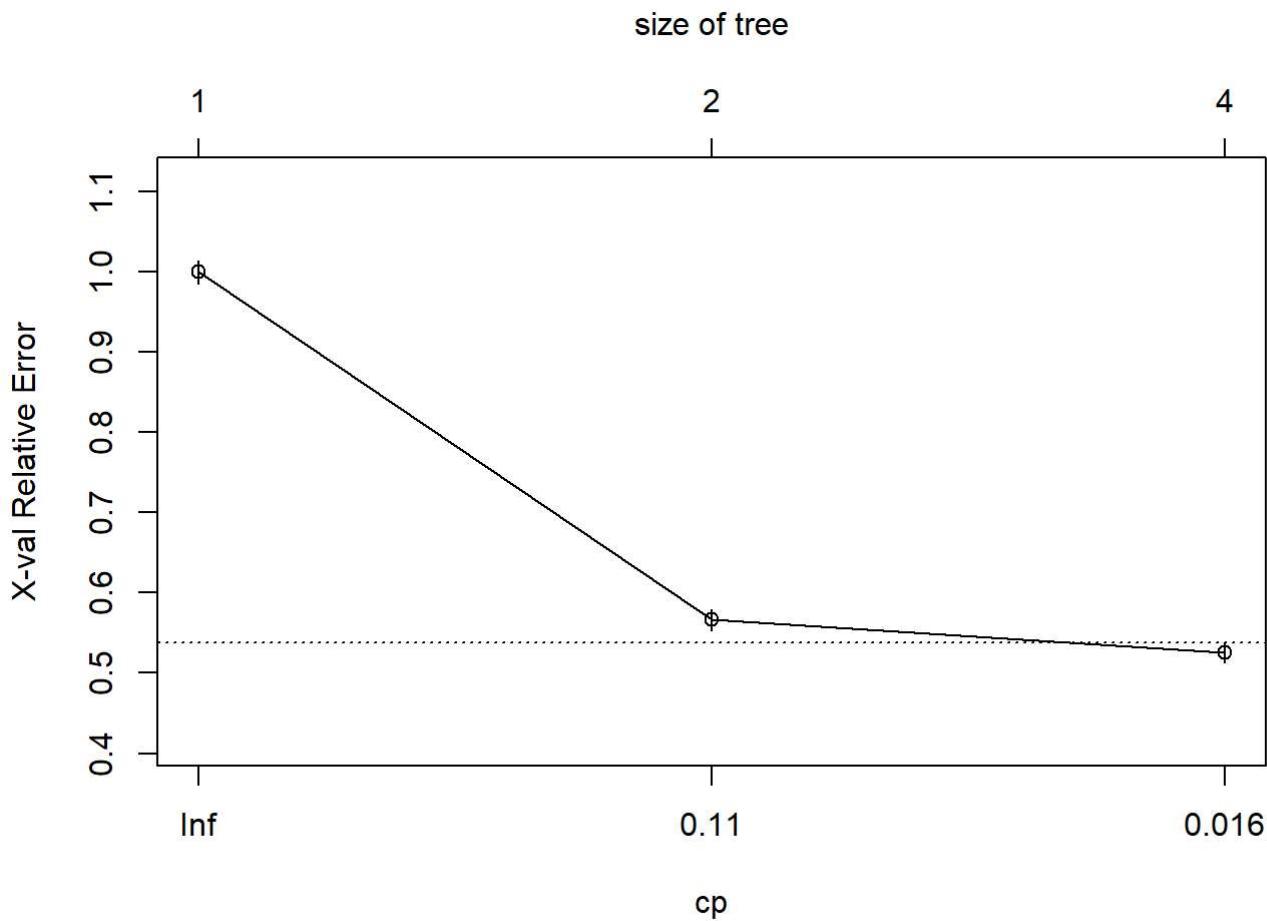
La variabile utilizzata per la suddivisione del primo nodo è Contract, mentre per il nodo 3 è InternetService.

```
printcp(fit)
```

```

## 
## Classification tree:
## rpart(formula = Churn ~ ., data = traindf_ROSE, method = "class")
## 
## Variables actually used in tree construction:
## [1] Contract      InternetService tenure
## 
## Root node error: 2441/4922 = 0.49594
## 
## n= 4922
## 
##          CP nsplit rel error  xerror     xstd
## 1 0.433429     0   1.00000 1.00000 0.014370
## 2 0.025809     1   0.56657 0.56657 0.012919
## 3 0.010000     3   0.51495 0.52560 0.012617
  
```

```
plotcp(fit)
```



Osservando il grafico e analizzando il summary, scegliamo un cp pari a 0.025

```
pruned <- prune(fit, cp=0.025)
```

```
preds_train <- predict(pruned, traindf_ROSE, type = "class")
confusionMatrix(preds_train, traindf_ROSE$Churn, positive = "Yes", mode="prec_recall")
```

```

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No   Yes
##           No 1648  424
##           Yes  833 2017
##
##           Accuracy : 0.7446
##             95% CI : (0.7322, 0.7568)
##   No Information Rate : 0.5041
## P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.4899
##
## Mcnemar's Test P-Value : < 2.2e-16
##
##           Precision : 0.7077
##             Recall : 0.8263
##               F1 : 0.7624
##             Prevalence : 0.4959
##       Detection Rate : 0.4098
## Detection Prevalence : 0.5790
##   Balanced Accuracy : 0.7453
##
## 'Positive' Class : Yes
##

```

Il valore di accuratezza sul training set risulta pari a 0.74 con livelli di Precision e Recall accettabili. Valutiamo il livello di generalizzazione del modello, attraverso l'utilizzo del test set.

```

preds_test <- predict(pruned, testdf, type = "class")
confusionMatrix(preds_test, testdf$Churn, positive = "Yes", mode="prec_recall")

```

```

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No   Yes
##           No 1055    77
##           Yes  517  461
##
##           Accuracy : 0.7185
##             95% CI : (0.6988, 0.7376)
##   No Information Rate : 0.745
##   P-Value [Acc > NIR] : 0.9974
##
##           Kappa : 0.4161
##
## McNemar's Test P-Value : <2e-16
##
##           Precision : 0.4714
##             Recall : 0.8569
##             F1 : 0.6082
##           Prevalence : 0.2550
##   Detection Rate : 0.2185
## Detection Prevalence : 0.4635
##   Balanced Accuracy : 0.7640
##
## 'Positive' Class : Yes
##

```

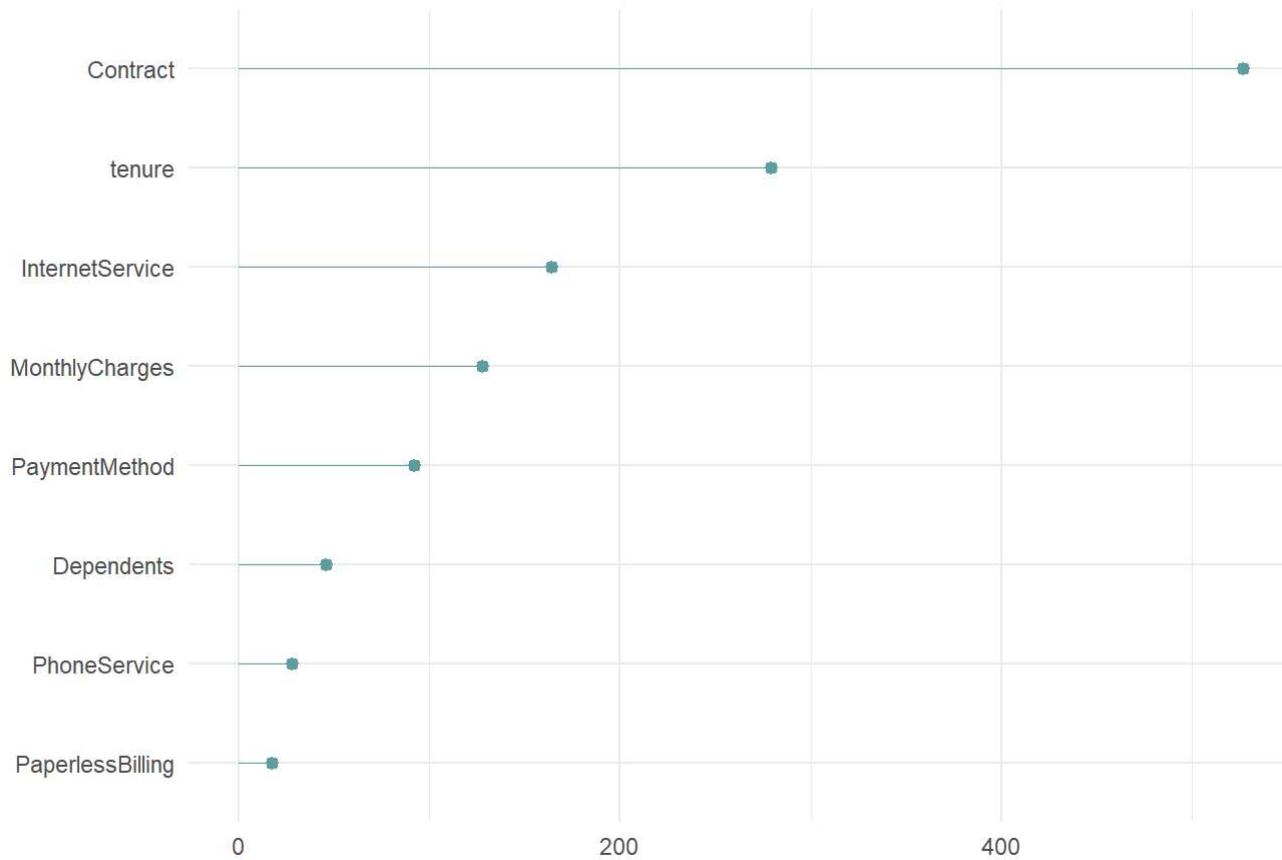
Il modello sembra generalizzare abbastanza bene, dal momento che il valore di accuracy risulta pari a 0.71 in linea con il valore ottenuto sul training set. Notiamo comunque un lieve peggioramento delle performance, in quanto il valore di precision passa da 0.70 a 0.47, il che equivale ad un aumento di falsi positivi.

```

pruned$variable.importance %>%
  data.frame() %>%
  rownames_to_column(var = "Feature") %>%
  rename(Overall = '.') %>%
  ggplot(aes(x = fct_reorder(Feature, Overall), y = Overall)) +
  geom_pointrange(aes(ymin = 0, ymax = Overall), color = "cadetblue", size = .3) +
  theme_minimal() +
  coord_flip() +
  labs(x = "", y = "", title = "Variable Importance")

```

Variable Importance



Il grafico mostra l'importanza delle variabili utilizzate. Maggiore è il valore della metrica *Gini importance*, che valuta la riduzione del criterio di impurità di Gini che si ottiene quando una variabile viene utilizzata per dividere un nodo dell'albero, maggiore è l'importanza della variabile per la classificazione. In questo caso, la variabile più importante è "Contract" seguita da "tenure" e "InternetService".

Bagging

```
cvcontrol <- trainControl(method = "repeatedcv", number = 10,
allowParallel = TRUE)

set.seed(1234)
bag_model <- train(
  Churn ~.,
  data = traindf_ROSE,
  nbagg = 150,
  trControl = cvcontrol,
  method = "treebag",
  control = rpart.control(minsplit = 2, cp = 0)
)
```

```
bag_model
```

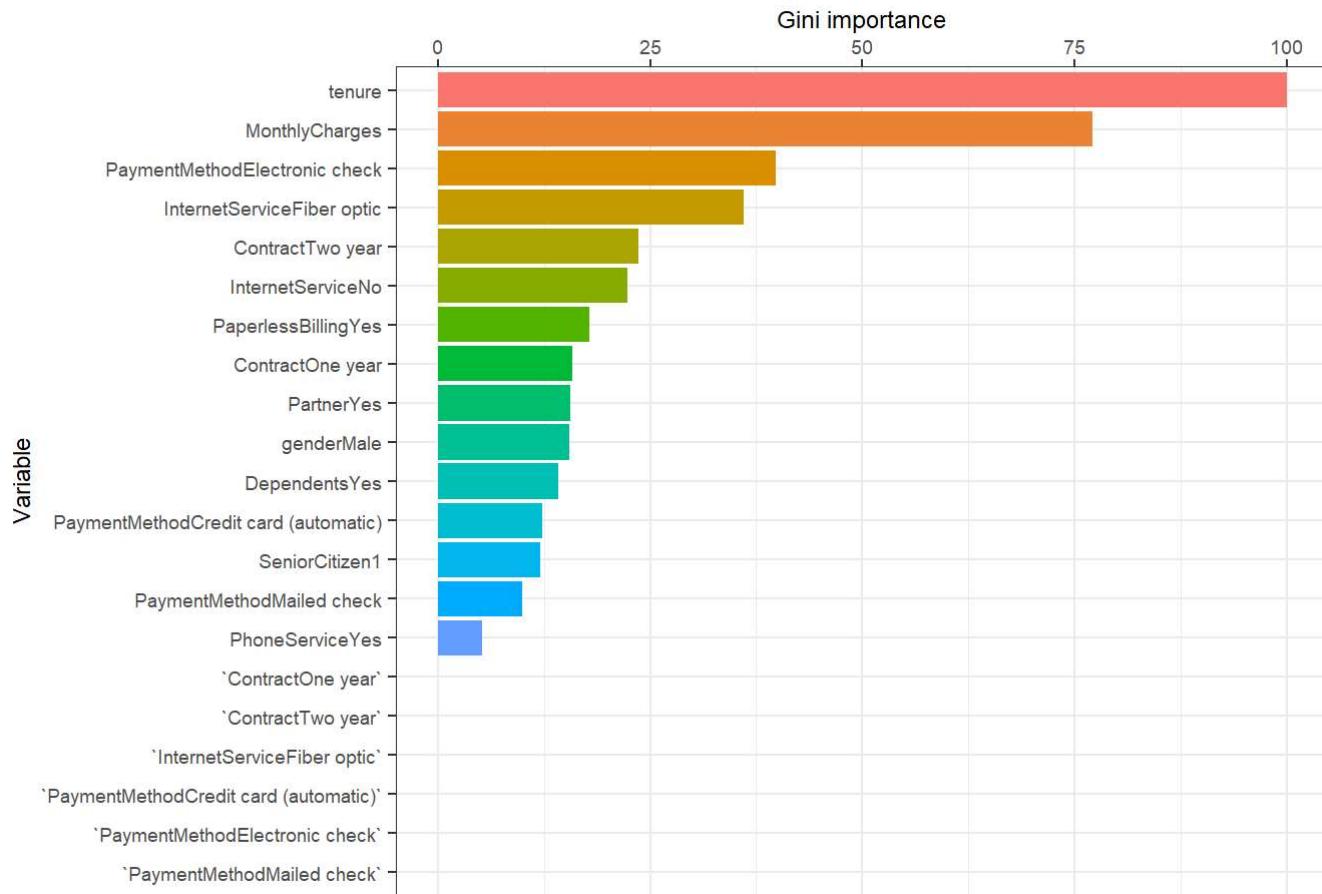
```
## Bagged CART
##
## 4922 samples
##   11 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 4430, 4430, 4429, 4430, 4430, 4429, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.757813  0.5157368
```

```
importance <- varImp(bag_model)$importance

# Creiamo il plot di importanza delle variabili
var_imp <- data.frame(variable = rownames(importance),
                       MeanDecreaseGini = importance$Overall)
var_imp$variable <- reorder(var_imp$variable, -var_imp$MeanDecreaseGini)

# Creiamo il plot di importanza delle variabili
ggplot(var_imp, aes(x = MeanDecreaseGini, y = variable, fill = variable)) +
  geom_col() +
  labs(title = "Variable Importance with Bagging",
       x = "Gini importance",
       y = "Variable") +
  theme_bw() +
  theme(plot.title = element_text(size = 10, face = "bold"),
        axis.text.y = element_text(size = 7),
        axis.text.x = element_text(size = 7),
        axis.title = element_text(size = 9),
        legend.position = "none") +
  scale_x_continuous(position = "top") +
  scale_y_discrete(limits = rev(levels(var_imp$variable)))
```

Variable Importance with Bagging



Il grafico ci mostra l'importanza stimata di ogni variabile, in ordine decrescente. La variabile "tenure" (durata del servizio) è quella considerata più importante, con un punteggio di 100. La seconda variabile più importante è "MonthlyCharges" (canone mensile), con un punteggio di 77.127. Altre variabili che sembrano avere un impatto significativo sulla classificazione sono "PaymentMethodElectronic check" (metodo di pagamento tramite assegno elettronico), "InternetServiceFiber optic" (tipo di servizio internet) e "ContractTwo year" (tipo di contratto a due anni).

```

preds_bag <- bind_cols(
  predict(bag_model, newdata = testdf, type = "prob"),
  Predicted = predict(bag_model, newdata = testdf, type = "raw"),
  Actual = testdf$Churn
)

confusionMatrix(preds_bag$Predicted, preds_bag$Actual, positive = "Yes", mode="prec_recall")

```

```

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No   Yes
##           No 1166  141
##           Yes  406  397
##
##           Accuracy : 0.7408
##             95% CI : (0.7215, 0.7593)
##   No Information Rate : 0.745
##   P-Value [Acc > NIR] : 0.6836
##
##           Kappa : 0.4128
##
## McNemar's Test P-Value : <2e-16
##
##           Precision : 0.4944
##             Recall : 0.7379
##             F1 : 0.5921
##   Prevalence : 0.2550
##   Detection Rate : 0.1882
## Detection Prevalence : 0.3806
##   Balanced Accuracy : 0.7398
##
## 'Positive' Class : Yes
##

```

L'accuratezza ottenuta in fase di validazione è del 0.758, con un indice kappa di 0,5157. La matrice di confusione mostra che il modello ha classificato correttamente il 74,16% dei casi positivi e il 49,5% dei casi negativi. Il livello di accuratezza è in linea con quanto osservato in fase di training, sintomo di una buona generalizzazione del modello. In generale, il modello sembra avere una buona capacità di classificazione.

Potrebbe essere utile esplorare altre tecniche di modellizzazione quali ad esempio Random Forest, oppure tecniche di boosting (ad esempio XGBoost o Adaptive Boosting).

Random Forest

```
fit.rf <- randomForest(Churn ~ ., data = traindf_ROSE, ntree = 500, importance = T)
```

```
preds_test_rf <- predict(fit.rf, testdf, type = "class")
confusionMatrix(preds_test_rf, testdf$Churn, positive = "Yes", mode="prec_recall")
```

```

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No   Yes
##           No 1145  113
##           Yes  427  425
##
##           Accuracy : 0.7441
##                 95% CI : (0.7249, 0.7626)
##   No Information Rate : 0.745
##   P-Value [Acc > NIR] : 0.5513
##
##           Kappa : 0.4349
##
## McNemar's Test P-Value : <2e-16
##
##           Precision : 0.4988
##           Recall    : 0.7900
##           F1        : 0.6115
##           Prevalence : 0.2550
##           Detection Rate : 0.2014
## Detection Prevalence : 0.4038
##           Balanced Accuracy : 0.7592
##
##           'Positive' Class : Yes
##

```

Se confrontiamo i risultati con quelli ottenuti con il precedente albero di classificazione con metodo bagging, notiamo come i modelli hanno un'accuratezza simile pari a circa il 74%, ma il modello random forest ha un valore K leggermente più alto (0,4286 rispetto a 0,4142) che indica una maggiore concordanza tra i risultati del modello e i dati reali. Inoltre, il modello random forest ha una recall significativamente più alta (0,78 rispetto a 0,73), il che significa che è in grado di identificare correttamente una maggiore percentuale di veri positivi. In generale, il modello random forest sembra avere prestazioni leggermente migliori rispetto al modello di bagging su questo dataset.

Conclusioni

Il processo di *bagging* migliora l'accuratezza delle previsioni per modelli ad elevata varianza (e basso bias) a spese dell'interpretabilità e della velocità computazionale. Tuttavia, usando vari algoritmi e strumenti di interpretabilità, possiamo ancora fare inferenza su come il nostro bagged model sfrutta la feature information. Tuttavia, con il processo di bagging degli alberi un problema continua a sussistere. Nonostante il modello esegue i vari step in maniera indipendente, gli alberi nel processo di bagging non sono completamente indipendenti tra di loro, dal momento che tutte le features sono considerate ad ogni split di ogni albero. Di conseguenza, alberi provenienti da diversi bootstrap samples hanno una struttura simile fra loro (specialmente nella parte iniziale dell'albero) a causa di eventuali relazioni forti sottostanti. Questa caratteristica è conosciuta come **tree correlation** e previene il *bagging* dal ridurre ulteriormente la varianza del base learner. Gli algoritmi di Random forest estendono e migliorano i bagged decision trees riducendo questa correlazione e migliorando così la precisione dell'insieme complessivo.