

Generazione di modelli per predire l'interesse dei clienti per l'acquisto di una polizza

Francesco Gozzoli

29/10/2021

INTRODUZIONE

Il dataset contiene informazioni sull'interesse di clienti di una compagnia di assicurazioni per l'acquisto di polizze auto. Inizialmente le colonne che compongono il dataset sono 12, di cui una, l'ultima è quella che identifica la classe di appartenenza delle istanze. Le features sono informazioni di base delle persone prese in esame, quali età, sesso, ecc, oltrechè informazioni sul veicolo e sulla situazione assicurativa. Le colonne inizialmente si trovano nella situazione mostrata dalla funzione `summary`.

```
insurance <- read.csv("../insurance.csv")
summary(insurance)
```

```
##      id      Gender      Age      Driving_License
## Min.   :    1  Length:381109  Min.   :20.00  Min.   :0.0000
## 1st Qu.: 95278  Class :character  1st Qu.:25.00  1st Qu.:1.0000
## Median :190555  Mode  :character  Median :36.00  Median :1.0000
## Mean   :190555          Mean  :38.82  Mean   :0.9979
## 3rd Qu.:285832          3rd Qu.:49.00  3rd Qu.:1.0000
## Max.   :381109          Max.   :85.00  Max.   :1.0000
## Region_Code  Previously_Insured  Vehicle_Age  Vehicle_Damage
## Min.   : 0.00  Min.   :0.0000  Length:381109  Length:381109
## 1st Qu.:15.00  1st Qu.:0.0000  Class :character  Class :character
## Median :28.00  Median :0.0000  Mode  :character  Mode  :character
## Mean   :26.39  Mean   :0.4582
## 3rd Qu.:35.00  3rd Qu.:1.0000
## Max.   :52.00  Max.   :1.0000
## Annual_Premium  Policy_Sales_Channel  Vintage  Response
## Min.   : 2630  Min.   : 1      Min.   : 10.0  Min.   :0.0000
## 1st Qu.: 24405  1st Qu.: 29      1st Qu.: 82.0  1st Qu.:0.0000
## Median : 31669  Median :133      Median :154.0  Median :0.0000
## Mean   : 30564  Mean   :112      Mean   :154.3  Mean   :0.1226
## 3rd Qu.: 39400  3rd Qu.:152      3rd Qu.:227.0  3rd Qu.:0.0000
## Max.   :540165  Max.   :163      Max.   :299.0  Max.   :1.0000
```

PRE-PROCESSING

Per poter essere utilizzate efficientemente, alcune colonne hanno bisogno di essere manipolate. In particolare:

- La colonna `Gender`, viene trasformata nella colonna `Male`, che ammette valori binari 0 e 1 dove 1 rappresenta il sesso maschile e 0 quello femminile.

```
insurance[which(insurance$Gender == "Male"),]$Gender = 1
insurance[which(insurance$Gender == "Female"),]$Gender = 0
colnames(insurance)[2] = "Male"
insurance$Male <- as.factor(insurance$Male)
```

- La colonna `Vehicle_Age` viene convertita da categorica ordinata a numerica. I valori utilizzati sono:
 - -1 per le auto immatricolate da meno di un anno
 - 0 per le auto immatricolate tra gli 1 e i 2 anni precedenti
 - 1 per le auto immatricolate da più di 2 anni

```
insurance[which(insurance$Vehicle_Age == "< 1 Year"),]$Vehicle_Age <- -1
insurance[which(insurance$Vehicle_Age == "1-2 Year"),]$Vehicle_Age <- 0
insurance[which(insurance$Vehicle_Age == "> 2 Years"),]$Vehicle_Age <- 1
insurance$Vehicle_Age <- as.numeric(insurance$Vehicle_Age)
```

- La colonna `Vehicle_Damage` viene convertita da categorica nominale a factor. I valori utilizzati sono:
 - 0 per le auto che non sono mai state coinvolte in incidenti
 - 1 per le auto che sono state coinvolte in incidenti

```
insurance[which(insurance$Vehicle_Damage == "Yes"),]$Vehicle_Damage <- 1
insurance[which(insurance$Vehicle_Damage == "No"),]$Vehicle_Damage <- 0
insurance$Vehicle_Damage <- as.factor(insurance$Vehicle_Damage)
```

Le colonne categoriche e binarie restanti vengono convertite in factor

```
insurance$Driving_License <- as.factor(insurance$Driving_License)
insurance$Region_Code <- as.factor(insurance$Region_Code)
insurance$Previously_Insured <- as.factor(insurance$Previously_Insured)
insurance$Policy_Sales_Channel <- as.factor(insurance$Policy_Sales_Channel)
insurance$Response <- as.factor(insurance$Response)
```

STATISTICHE DESCRITTIVE

Attraverso la funzione `ggplot` si generano i grafici che mostrano la distribuzione di frequenza delle variabili numeriche `Age`, `Vintage`, `Annual_Premium`

```
library(ggplot2)
age_plot <- ggplot(insurance, aes(x=Age, fill=Response))+
  geom_histogram(alpha=0.4,position="identity")+
  xlab("Età")+
  scale_y_continuous("Numero di osservazioni") +
  guides(fill=guide_legend(title=NULL)) +
  scale_fill_discrete(labels=c("Negativi","Positivi")) +
  theme(legend.position = c(1,1),legend.justification=c(1,1))

#Frequenze Vintage
vintage_plot <- ggplot(insurance, aes( x=Vintage, fill=Response))+
  geom_histogram(alpha=0.4,position="identity")+
  xlab("Vintage")+
```

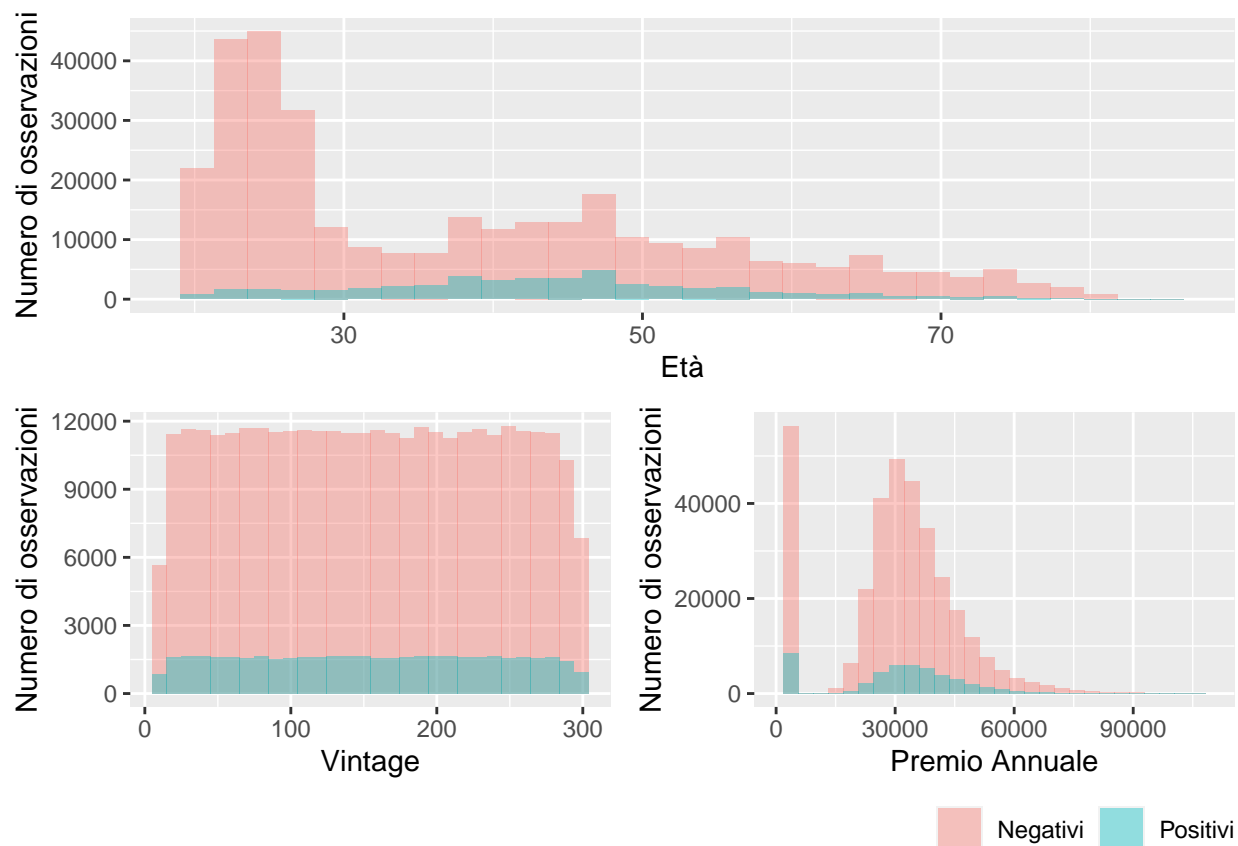
```

scale_y_continuous("Numero di osservazioni")

options(scipen=999)
#Frequenze annual_premium
premium_plot <- ggplot( insurance, aes( x=Annual_Premium, fill=Response))+
  geom_histogram(alpha=0.4,position="identity")+
  xlab("Premio Annuale")+
  scale_y_continuous("Numero di osservazioni") +
  scale_x_continuous(limits = c(0, 110000))

library(ggpubr)
ggarrange(age_plot, ggarrange(vintage_plot, premium_plot, ncol = 2, legend = "none"),
  nrow = 2, common.legend = T, legend = "bottom")

```

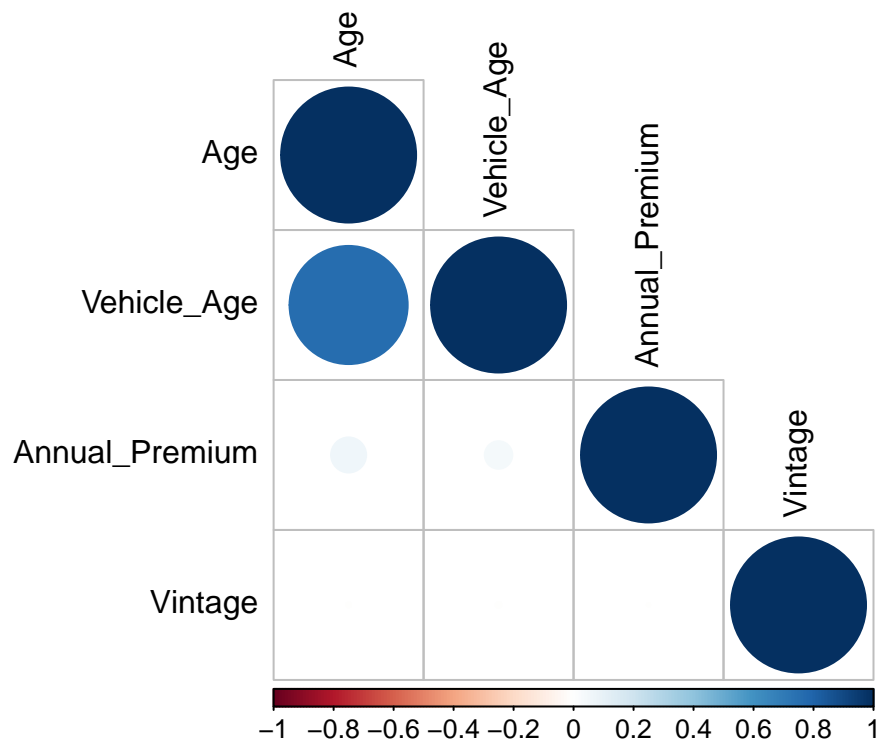


Nonostante il numero non elevato di features presenti nel dataset possiamo cercare la correlazione presente tra esse. Dalla correlation matrix si può notare che le features **Vehicle_Age** e **Age** sono piuttosto correlate tra loro. Nonostante questo ho deciso di mantenerle entrambe poichè il vantaggio dal punto di vista computazionale è minimo, evitando così perdita di informazioni

```

library(corrplot)
correlationMatrix <- stats::cor(insurance[c(3, 7, 9, 11)])
corrplot(correlationMatrix, method="circle", type="lower", tl.col="black")

```

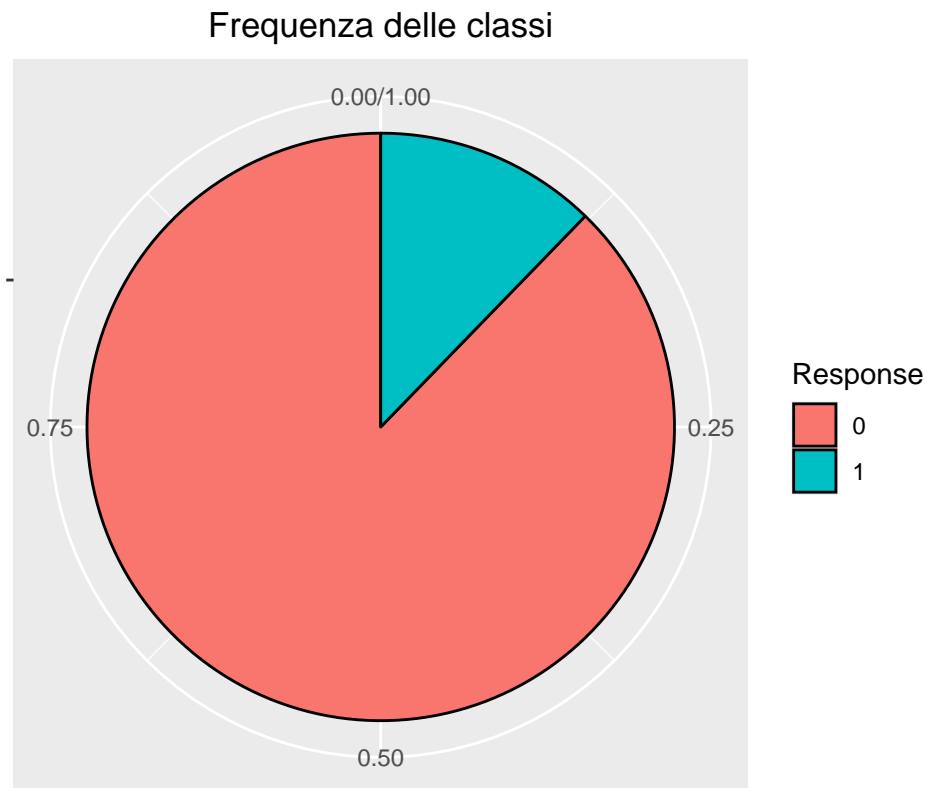


Uno degli aspetti più critici del dataset in analisi è lo sbilanciamento tra le classi, come si evince dal seguente piechart. Infatti le percentuali evidenziano uno sbilanciamento di 88% a favore della classe dei negativi contro il 12% della classe dei positivi

```
freq <- as.data.frame(table(insurance$Response))
colnames(freq)[1] = "Response"
freq$perc <- prop.table(freq$Freq)
print(freq)
```

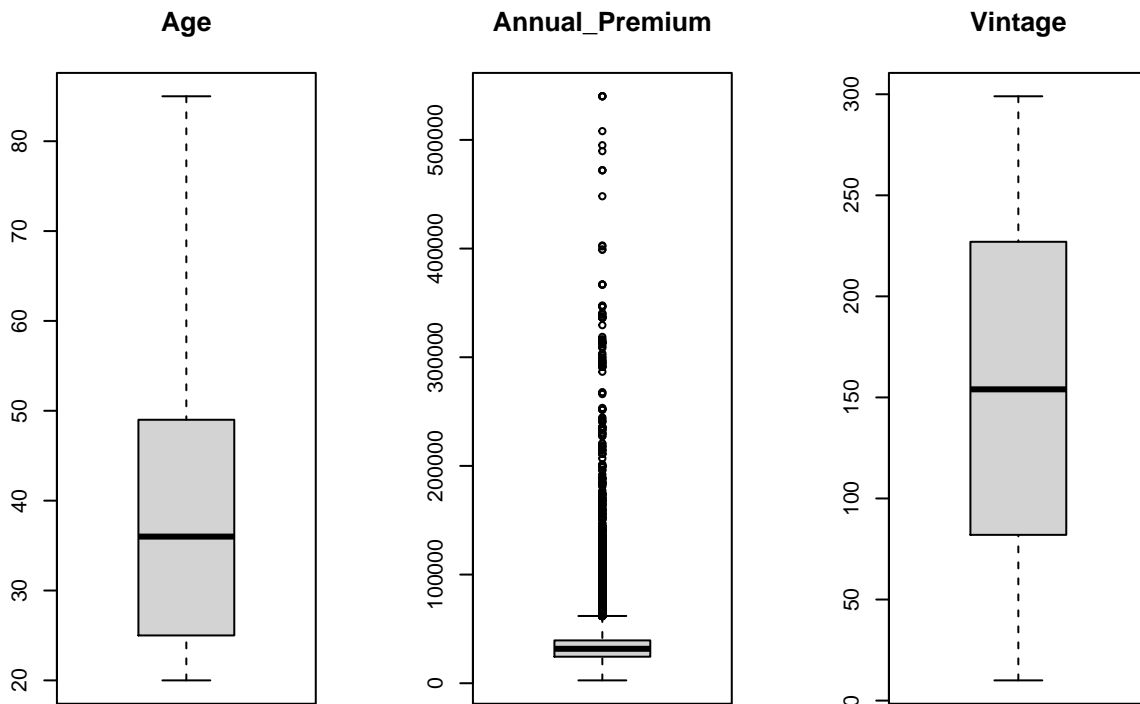
```
##  Response  Freq    perc
## 1         0 334399 0.8774366
## 2         1  46710 0.1225634
```

```
ggplot(freq, aes(x = "", y = perc, fill = Response)) +
  geom_col(color = "black") +
  coord_polar(theta = "y") +
  xlab("") +
  ylab("") +
  ggtitle("Frequenza delle classi") +
  theme(plot.title = element_text(hjust = 0.5))
```



Per rappresentare graficamente la distribuzione dei valori delle 3 features numeriche `Age`, `Annual_Premium`, `Vintage` e controllare la presenza di outliers si stampano i relativi boxplot

```
par(mfrow=c(1,3))
boxplot(insurance$Age, main="Age")
boxplot(insurance$Annual_Premium, main="Annual_Premium")
boxplot(insurance$Vintage, main="Vintage")
```



SEPARAZIONE STRATIFICATA

Per addestrare e validare i modelli che verranno realizzati si divide il dataset originale nei due dataset di training e testing nella classica proporzione percentuale di 70-30. Il criterio di separazione scelto è quello stratificato perchè consente di mantenere la proporzione tra le istanze della classe di maggioranza e quella di minoranza all'interno dei due dataset. E' stata fatta questa scelta perchè il dataset originale è molto sbilanciato, ed in questo modo evitiamo di creare dei dataset ancora più sbilanciati

```
library(splitstackshape)
set.seed(1)
train <- as.data.frame(stratified(insurance, c('Response'), 0.7))
test <- insurance[which(!(insurance$id %in% train$id)),]
insurance <- insurance[, -1]
train <- train[, -1]
test <- test[, -1]
```

Nonostante lo sbilanciamento si prova a generare un primo modello, in particolare viene generato un albero di decisione. Per fare ciò ci si può avvalere della funzione `rpart`, che genera un albero implementando l'algoritmo CART ed utilizzando quindi l'indice di Gini come criterio per lo splitting ¹

¹<https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf>

ALBERO DI DECISIONE SU DATASET SBILANCIATO

```
library(rpart)
set.seed(1)
tree <- rpart(Response~., train, method = "class", control = rpart.control(cp = 0.00001))
prediction_unbalanced <- predict(tree, test, type = "class")
library(caret)
confusionMatrix(as.factor(prediction_unbalanced), as.factor(test$Response))
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction      0      1
##              0 94540 11131
##              1  5780  2882
##
##              Accuracy : 0.8521
##              95% CI : (0.85, 0.8541)
##      No Information Rate : 0.8774
##      P-Value [Acc > NIR] : 1
##
##              Kappa : 0.1771
##
##  Mcnemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.9424
##              Specificity : 0.2057
##              Pos Pred Value : 0.8947
##              Neg Pred Value : 0.3327
##              Prevalence : 0.8774
##              Detection Rate : 0.8269
##      Detection Prevalence : 0.9242
##              Balanced Accuracy : 0.5740
##
##              'Positive' Class : 0
##
```

Come si può notare dalla ConfusionMatrix la classificazione non è per niente soddisfacente poichè nonostante riesca a classificare le istanze della classe di maggioranza fatica a classificare correttamente le istanze della classe di minoranza. Questo è un problema perchè oltre ad essere un punto debole del modello, l'obiettivo della classificazione è quello di massimizzare la corretta identificazione di potenziali clienti interessati piuttosto che di quelli non interessati. In generale quindi si preferisce avere un maggior tasso di falsi positivi piuttosto che di falsi negativi.

RIBILANCIAMENTO

Alla luce delle prestazioni mostrate da questo primo modello è bene procedere ad un ribilanciamento delle classi. Per fare ciò mi sono avvalso di due tecniche: * Tecnica di sottocampionamento casuale che consiste nel rimuovere casualmente istanze della classe di maggioranza fino ad avere un bilanciamento di circa 50-50 * Sovracampionamento effettuato attraverso SMOTE, che genera istanze sintetiche della classe di minoranza sfruttando un algoritmo di k-nearest neighbors (K = 5 nel mio caso)

```
library(unbalanced)
set.seed(1)
rem <- ubUnder(train[,1:11], train$Response, perc=14, method = "percUnder")
casual.balance.train <- data.frame(rem$X, rem$Y)
casual.balance.train$rem.Y <- NULL
names(casual.balance.train)[11] <- "Response"
table(casual.balance.train$Response)
```

```
##
##      0      1
## 32771 32697
```

Per quanto riguarda l'applicazione della tecnica di ribilanciamento SMOTE (synthetic minority oversampling technique) si è scelto un approccio non esageratamente drastico cercando quindi di mantenere un leggero sbilanciamento. Per evitare di creare un dataset enorme effettuando quindi soltanto un oversampling della classe di minoranza, ho optato per una scelta ibrida abbinando all'oversampling una rimozione casuale (undersampling) delle istanze della classe di maggioranza, questo approccio ibrido è infatti consigliato dalla documentazione ufficiale di SMOTE ². A questo scopo i parametri scelti per l'esecuzione di `ubBalance` sono:

- * `percOver = 100` questo significa che l'algoritmo crea istanze sintetiche della classe di minoranza in numero pari a $\frac{percOver}{100} \times \text{num.istanze classe minoranza}$ delle istanze di quest'ultima

- `percUnder = 280` questo significa che l'algoritmo effettua un undersampling della classe di maggioranza mantenendone un numero pari a $\frac{280}{100} \times \text{num.istanze sintetiche}$

```
set.seed(1)
SMOTE <- ubBalance(train[, -c(11)], train[, 11], type="ubSMOTE", positive = 1,
                  percOver = 100, percUnder = 280, verbose = T)
```

```
## Proportion of positives after ubSMOTE : 41.67 % of 156945 observations
```

```
SMOTE_train <- SMOTE$X
SMOTE_train$Response <- SMOTE$Y
table(SMOTE_train$Response)
```

```
##
##      0      1
## 91551 65394
```

MODELLI DI CLASSIFICAZIONE

ALBERI DI DECISIONE

Per prima cosa si valutano le prestazioni dell'albero di decisione applicato ai dataset bilanciati, prima il dataset bilanciato con la tecnica di sottocampionamento e dopo il dataset bilanciato con lo SMOTE

```
set.seed(1)
tree <- rpart(Response ~ ., casual.balance.train, method = "class",
              control = rpart.control(cp = 0.00001))
prediction_casual_balance <- predict(tree, test, type = "class")
confusionMatrix(as.factor(prediction_casual_balance), as.factor(test$Response))
```

²<https://arxiv.org/pdf/1106.1813.pdf>


```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 70913 2746
##           1 29407 11267
##
##           Accuracy : 0.7188
##           95% CI : (0.7162, 0.7214)
##       No Information Rate : 0.8774
##       P-Value [Acc > NIR] : 1
##
##           Kappa : 0.281
##
##  McNemar's Test P-Value : <2e-16
##
##           Sensitivity : 0.7069
##           Specificity : 0.8040
##       Pos Pred Value : 0.9627
##       Neg Pred Value : 0.2770
##           Prevalence : 0.8774
##       Detection Rate : 0.6202
##       Detection Prevalence : 0.6442
##       Balanced Accuracy : 0.7555
##
##       'Positive' Class : 0
##
```

```
set.seed(1)
tree <- rpart(Response~., SMOTE_train, method = "class", control = rpart.control(cp = 0.0001))
prediction_smote <- predict(tree, test, type = "class")
confusionMatrix(as.factor(prediction_smote),as.factor(test$Response))
```

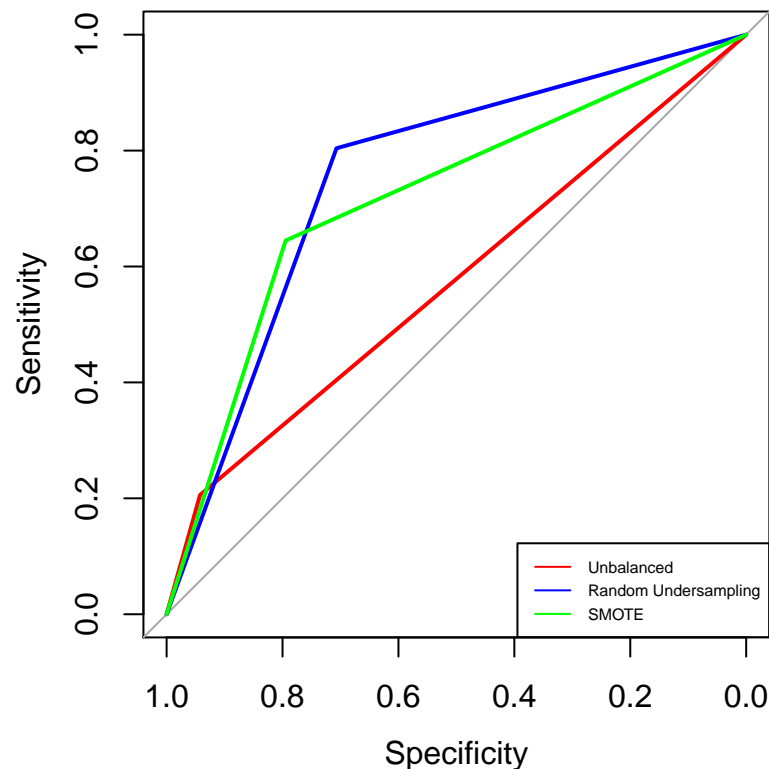
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 79721 4978
##           1 20599 9035
##
##           Accuracy : 0.7763
##           95% CI : (0.7739, 0.7787)
##       No Information Rate : 0.8774
##       P-Value [Acc > NIR] : 1
##
##           Kappa : 0.297
##
##  McNemar's Test P-Value : <2e-16
##
##           Sensitivity : 0.7947
##           Specificity : 0.6448
##       Pos Pred Value : 0.9412
##       Neg Pred Value : 0.3049
##           Prevalence : 0.8774
```

```
##          Detection Rate : 0.6973
##    Detection Prevalence : 0.7408
##          Balanced Accuracy : 0.7197
##
##          'Positive' Class : 0
##
```

Come si può notare i risultati sono migliorati molto rispetto al primo modello con dataset sbilanciato, in particolare si nota come la tecnica di undersampling fornisca prestazioni peggiori rispetto a SMOTE per quanto riguarda la sensitivity ma migliori per quanto riguarda la specificity, quest'ultima è preferibile in questo caso come già accennato prima

QUI TOCCA SPIEGARE COSA È LA CURVA ROC

```
library(pROC)
par(mfrow=c(1,1), pty = "s")
plot.roc(as.numeric(test$Response), as.numeric(prediction_unbalanced), col="red")
lines.roc(as.numeric(test$Response), as.numeric(prediction_casual_balance), col="blue")
lines.roc(test$Response, as.numeric(prediction_smote), col="green")
legend("bottomright", legend = c("Unbalanced", "Random Undersampling", "SMOTE"), lty=1, col=c("red", "blue", "green"))
```



NAIVE BAYES

Adesso si utilizza il metodo di classificazione Naive-Bayes, come per gli alberi di decisione si valuteranno le prestazioni prima sul dataset originale e poi per entrambi i dataset bilanciati, prima quello bilanciato con Undersampling e dopo quello bilanciato con SMOTE

```
#Naive-Bayes su dataset sbilanciato
library(e1071)
bayes_bil <- naiveBayes(Response ~ ., data = casual.balance.train)
b_prediction_unbalanced <- predict(bayes_bil, newdata = test)
confusionMatrix(as.factor(b_prediction_unbalanced),as.factor(test$Response))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##           0 69269 1557
##           1 31051 12456
##
##           Accuracy : 0.7148
##           95% CI : (0.7122, 0.7174)
##       No Information Rate : 0.8774
##       P-Value [Acc > NIR] : 1
##
##           Kappa : 0.3041
##
##  McNemar's Test P-Value : <2e-16
##
##           Sensitivity : 0.6905
##           Specificity : 0.8889
##           Pos Pred Value : 0.9780
##           Neg Pred Value : 0.2863
##           Prevalence : 0.8774
##           Detection Rate : 0.6059
##       Detection Prevalence : 0.6195
##       Balanced Accuracy : 0.7897
##
##       'Positive' Class : 0
##
```

```
#Naive-Bayes su dataset bilanciato con Undersampling casuale
bayes <- naiveBayes(Response ~ ., data = train)
b_prediction_casual_balance <- predict(bayes, newdata = test)
confusionMatrix(as.factor(b_prediction_casual_balance),as.factor(test$Response))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##           0 73853 3097
##           1 26467 10916
##
##           Accuracy : 0.7414
##           95% CI : (0.7389, 0.744)
##       No Information Rate : 0.8774
##       P-Value [Acc > NIR] : 1
##
##           Kappa : 0.3
##
```

```
## McNemar's Test P-Value : <2e-16
##
##      Sensitivity : 0.7362
##      Specificity : 0.7790
##      Pos Pred Value : 0.9598
##      Neg Pred Value : 0.2920
##      Prevalence : 0.8774
##      Detection Rate : 0.6459
##      Detection Prevalence : 0.6730
##      Balanced Accuracy : 0.7576
##
##      'Positive' Class : 0
##
```

#Naive-Bayes su dataset bilanciato con SMOTE

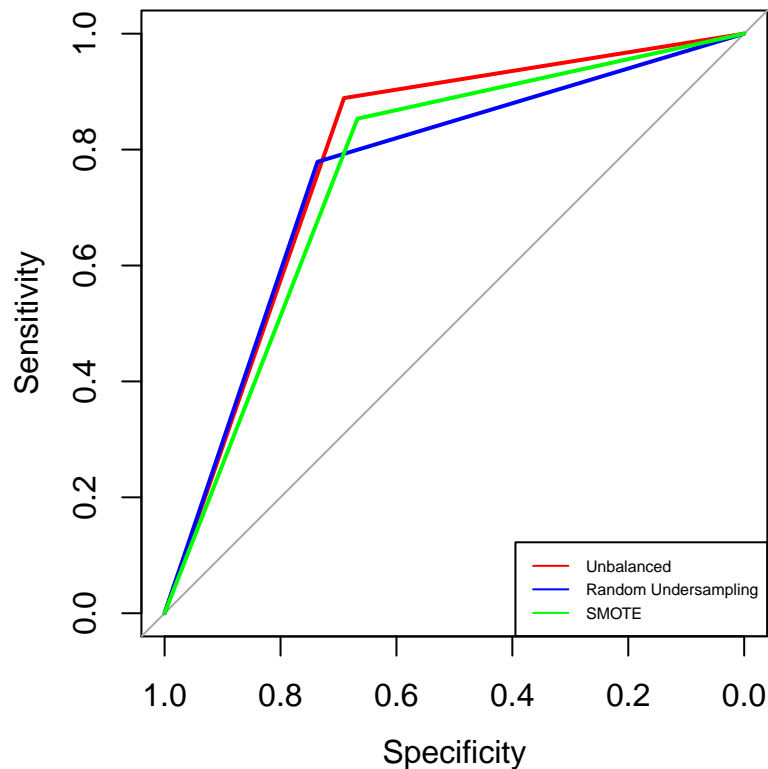
```
bayes_bil <- naiveBayes(Response ~ ., data = SMOTE_train)
b_prediction_smote <- predict(bayes_bil, newdata = test)
confusionMatrix(as.factor(b_prediction_smote),as.factor(test$Response))
```

```
## Confusion Matrix and Statistics
##
##      Reference
## Prediction    0    1
##      0 66941  2054
##      1 33379 11959
##
##      Accuracy : 0.6901
##      95% CI : (0.6874, 0.6928)
##      No Information Rate : 0.8774
##      P-Value [Acc > NIR] : 1
##
##      Kappa : 0.2654
##
## McNemar's Test P-Value : <2e-16
##
##      Sensitivity : 0.6673
##      Specificity : 0.8534
##      Pos Pred Value : 0.9702
##      Neg Pred Value : 0.2638
##      Prevalence : 0.8774
##      Detection Rate : 0.5855
##      Detection Prevalence : 0.6035
##      Balanced Accuracy : 0.7603
##
##      'Positive' Class : 0
##
```

A differenza degli alberi di decisione il classificatore Naive-Bayes ha le prestazioni abbastanza allineate per tutti e 3 i dataset utilizzati. In particolare il dataset sbilanciato ha prestazioni abbastanza simili per sensitivity e specificity, mentre i dataset bilanciati hanno una specificity più alta a discapito della sensitivity, il dataset bilanciato con SMOTE però restituisce delle prestazioni simili al dataset bilanciato con Undersampling casuale ma più basse in ogni circostanza. In definitiva se si vogliono prestazioni più allineate tra di loro è preferibile usare il dataset originale, mentre prediligendo come fatto fino ad ora la specificity il dataset bilanciato con undersampling casuale risulta il migliore

ANCHE QUA CITIAMO ROC

```
par(pty = "s")
plot.roc(as.numeric(test$Response), as.numeric(b_prediction_unbalanced), col="red")
lines.roc(as.numeric(test$Response), as.numeric(b_prediction_casual_balance), col="blue")
lines.roc(as.numeric(test$Response), as.numeric(b_prediction_smote), col="green")
legend("bottomright", legend = c("Unbalanced", "Random Undersampling", "SMOTE"), lty=1, col=c("red", "blue", "green"))
```



SVM

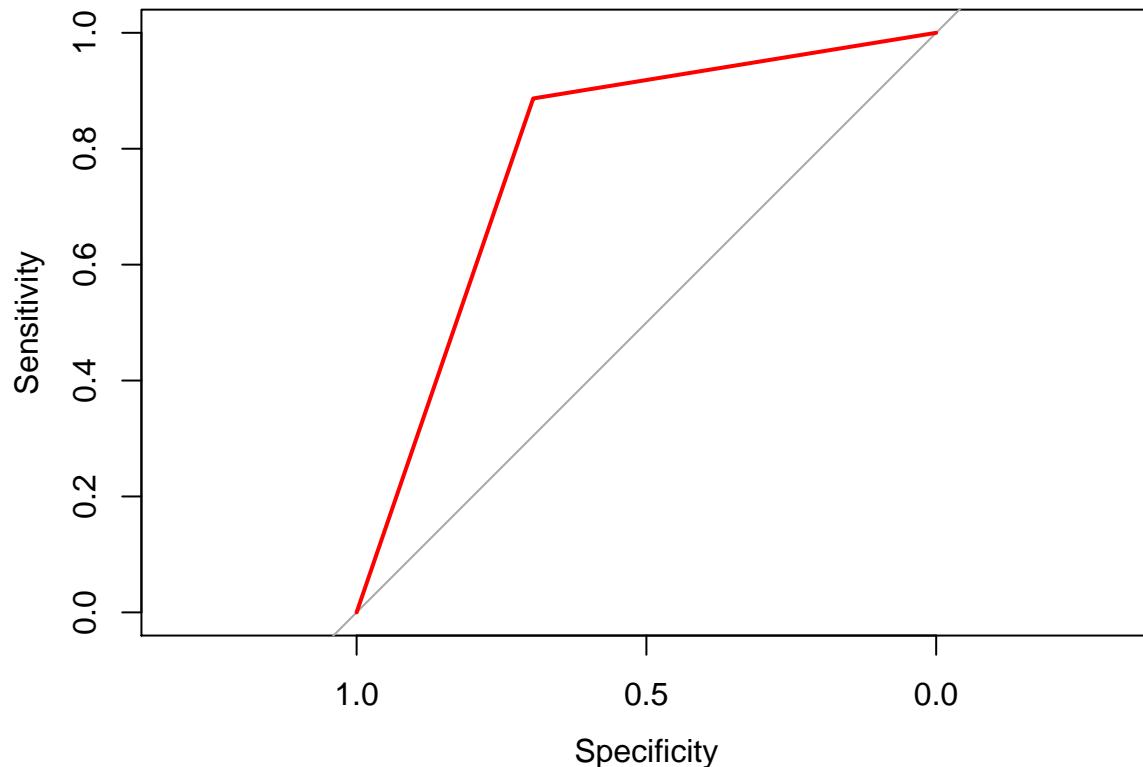
Una soluzione allo sbilanciamento alternativa alla alterazione del dataset è quella di applicare una funzione di peso che permette di dare più importanza alle istanze di una classe rispetto a quelle dell'altra. In questo caso si applica la funzione di peso al metodo di classificazione SVM utilizzato perchè molto efficace nei problemi di classificazione binaria. Utilizzando kernel lineare la SVM separa tramite un iperpiano le istanze in classi. In particolare i pesi (parametro `weights`) utilizzati sono 1 per la classe di maggioranza e 6 per la classe di minoranza, per ottenere un peso simile tra le classi.

```
weights <- c(1,6)
names(weights) <- c(0, 1)
set.seed(1)
support <- svm(formula = Response ~ ., data = train[, 1:11], kernel = "linear", class.weights = weights)
svm_prediction <- predict(support, newdata = test)
confusionMatrix(as.factor(svm_prediction), as.factor(test$Response))
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    0    1
##           0 69748 1587
##           1 30572 12426
##
##           Accuracy : 0.7187
##           95% CI : (0.7161, 0.7213)
##           No Information Rate : 0.8774
##           P-Value [Acc > NIR] : 1
##
##           Kappa : 0.308
##
## Mcnemar's Test P-Value : <2e-16
##
##           Sensitivity : 0.6953
##           Specificity : 0.8867
##           Pos Pred Value : 0.9778
##           Neg Pred Value : 0.2890
##           Prevalence : 0.8774
##           Detection Rate : 0.6100
##           Detection Prevalence : 0.6239
##           Balanced Accuracy : 0.7910
##
##           'Positive' Class : 0
##
```

```
plot.roc(as.numeric(test$Response), as.numeric(svm_prediction), col="red")
```



Le prestazioni del classificatore sono buone in generale e soprattutto hanno ottima specificity, purtroppo però il metodo risulta molto costoso dal punto di vista del tempo, motivo per il quale mi sono limitato ad applicarlo ad uno solo dei dataset a disposizione

XGBOOST

L'algoritmo XGBoost si basa sul metodo del gradient boosting che utilizza le derivate di secondo ordine, cioè il gradiente, per trovare il miglior modello ad albero grazie ad una esaustiva valutazione di tutte le possibili suddivisioni ad ogni passo dell'algoritmo. Viste le prestazioni del metodo SVM abbinata alla funzione dei pesi anche l'algoritmo XGBoost si applica allo stesso modo

```
library(xgboost)
library(Matrix)

xgb_preprocess <- function(df)
{
  df.xgb <- df[,c(1,2,3,5,6,7,8,10,11)]
  df.xgb$Male <- as.numeric(as.character(df.xgb$Male))
  df.xgb$Driving_License <- as.numeric(as.character(df.xgb$Driving_License))
  df.xgb$Vehicle_Damage <- as.numeric(as.character(df.xgb$Vehicle_Damage))
  df.xgb$Previously_Insured <- as.numeric(as.character(df.xgb$Previously_Insured))
  df.xgb$Response <- as.numeric(as.character(df.xgb$Response))
  df.xgb$Age <- as.numeric(df.xgb$Age)
  df.xgb$Vintage <- as.numeric(df.xgb$Vintage)
  df.xgb <- as.matrix(df.xgb)
```

```

    return(df.xgb)
}
train.xgb = xgb_preprocess(train)
weight <- ifelse(train$Response==1, 6,1)
set.seed(1)
bst <- xgboost(data = train.xgb[,-9], label = train.xgb[,9],
               nthread = -1, nrounds = 100, weight = weight, verbose=0)

test.xgb = xgb_preprocess(test)
test.xgb <- test.xgb[,-9]

xgb_prediction <- predict(bst, test.xgb)
xgb_prediction <- as.numeric(xgb_prediction > 0.5)
confusionMatrix(as.factor(xgb_prediction),as.factor(test$Response))

```

```

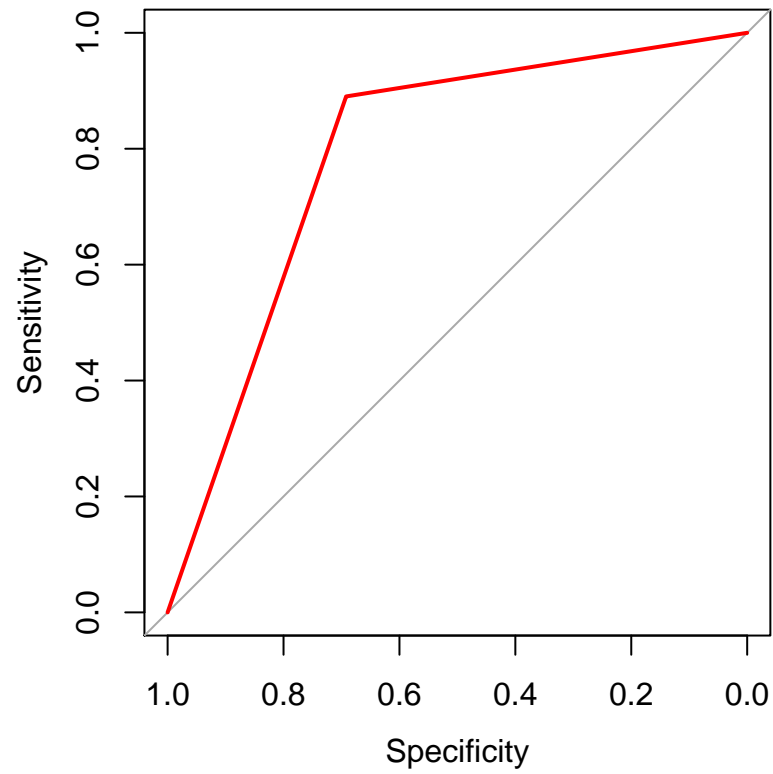
## Confusion Matrix and Statistics
##
##              Reference
## Prediction      0      1
##              0 69417 1538
##              1 30903 12475
##
##              Accuracy : 0.7163
##              95% CI : (0.7136, 0.7189)
##      No Information Rate : 0.8774
##      P-Value [Acc > NIR] : 1
##
##              Kappa : 0.3062
##
##      McNemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.6920
##              Specificity : 0.8902
##              Pos Pred Value : 0.9783
##              Neg Pred Value : 0.2876
##              Prevalence : 0.8774
##              Detection Rate : 0.6071
##      Detection Prevalence : 0.6206
##              Balanced Accuracy : 0.7911
##
##              'Positive' Class : 0
##

```

```

par(pty = "s")
plot.roc(as.numeric(test$Response), as.numeric(xgb_prediction), col="red")

```

Le prestazioni sono allineate a quelle dell'SVM, ma a differenza di quest'ultimo i tempi di esecuzioni sono estremamente rapidi, questa differenza sostanziale rende preferibile l'XGBoost

CONCLUSIONI