Tree Based Models

Roberto Cerminara, Daniele Florio, Lorenzo Piattoli

2025-04-17

Introduzione

In questa analisi ci proponiamo di esplorare i fattori che influenzano il rendimento degli studenti nell'esame finale, con l'obiettivo di individuare le variabili più rilevanti e costruire modelli predittivi capaci di classificare gli studenti in fasce di voto. Il dataset utilizzato contiene variabili sia numeriche (ad es. ore di studio settimanali, percentuale di frequenza, punteggi precedenti, numero di sessioni di tutoraggio) sia categoriali (ad es. livello di coinvolgimento genitoriale, accesso a risorse educative, motivazione, tipo di scuola, reddito familiare). Dopo un'importazione preliminare e la corretta codifica dei fattori, è stata condotta un'analisi descrittiva per comprendere la distribuzione dei punteggi d'esame e le correlazioni con le principali variabili numeriche.

Per gestire il problema della previsione in termini di classi di voto piuttosto che di punteggio continuo, la variabile Exam_Score è stata trasformata in variabile categoriale: sono state create due versioni del dataset, una con sei classi e una con quattro classi, bilanciate attorno al punteggio medio e con fasce più ampie per le code della distribuzione. Lo scopo di questa divisione del dataset è quello di analizzare i limiti dei modelli in situazioni di classi fortemente sbilanciate. in virtù di queste considerazioni, sono stati testati diversi modelli: Random Forest (con e senza pesi di classe), tecniche di data augmentation (SMOTE), il singolo albero decisionale CART e algoritmi di Boosting. In ciascun caso, l'analisi si è concentrata su accuratezza complessiva, errori Out-of-Bag (OOB), metriche di classe (Balanced Accuracy, Specificity) e importanza delle variabili, al fine di valutare performance e robustezza dei modelli in presenza di sbilanciamento tra le classi.

Librerie

```
library(ggplot2)
library(ggcorrplot)
library(scales)
library(randomForest)
library(caret)
library(rpart)
library(dplyr)
library(gbm)
library(smotefamily)
```

Import del dataset e analisi preliminare

Attraverso il seguente codice è stato effettuato l'import del dataset scelto. Inoltre utilizzando la lista delle variabili categoriali è stato possibile convertirle in factor attraverso la funzione lapply. Vediamo in oltre un estratto del dataset in basso.

```
ds <- read.csv("StudentPerformanceFactors.csv")
ds = data.frame(ds)
```

```r
# Lista di variabili categoriali
categorical_vars <- c(
  "Parental_Involvement", "Access_to_Resources", "Extracurricular_Activities",
  "Motivation_Level", "Internet_Access", "Family_Income", "Teacher_Quality",
  "School_Type", "Peer_Influence", "Learning_Disabilities",
  "Parental_Education_Level", "Distance_from_Home", "Gender"
)

ds[categorical_vars] <- lapply(ds[categorical_vars], factor)

head(ds)
```

```
##   Hours_Studied Attendance Parental_Involvement Access_to_Resources
## 1            23         84                  Low                High
## 2            19         64                  Low              Medium
## 3            24         98               Medium              Medium
## 4            29         89                  Low              Medium
## 5            19         92               Medium              Medium
## 6            19         88               Medium              Medium
##   Extracurricular_Activities Sleep_Hours Previous_Scores Motivation_Level
## 1                         No           7              73               Low
## 2                         No           8              59               Low
## 3                        Yes           7              91            Medium
## 4                        Yes           8              98            Medium
## 5                        Yes           6              65            Medium
## 6                        Yes           8              89            Medium
##   Internet_Access Tutoring_Sessions Family_Income Teacher_Quality School_Type
## 1             Yes                 0           Low          Medium      Public
## 2             Yes                 2        Medium          Medium      Public
## 3             Yes                 2        Medium          Medium      Public
## 4             Yes                 1        Medium          Medium      Public
## 5             Yes                 3        Medium            High      Public
## 6             Yes                 3        Medium          Medium      Public
##   Peer_Influence Physical_Activity Learning_Disabilities
## 1       Positive                 3                    No
## 2       Negative                 4                    No
## 3        Neutral                 4                    No
## 4       Negative                 4                    No
## 5        Neutral                 4                    No
## 6       Positive                 3                    No
##   Parental_Education_Level Distance_from_Home Gender Exam_Score
## 1              High School              Near   Male         67
## 2                  College          Moderate Female         61
## 3              Postgraduate              Near   Male         74
## 4               High School          Moderate   Male         71
## 5                  College              Near Female         70
## 6              Postgraduate              Near   Male         71
```

Descrizione delle variabili Prima di iniziare la trattazione è utile fare un breve riassunto su quelle che sono le variabili presenti nel dataset e del loro significato. * **Hours_Studied** Numero di ore spese studiando a settimana. * **Attendance** Percentuale di lezioni frequentate. * **Parental_Involvement** Livello di coinvolgimento genitoriale nella formazione dello studente (Low, Medium, High). * **Access_to_Resources** Disponibilità di risorse educative(Low, Medium, High). * **Extracurricular_Activities** Partecipazione ad attività extracurriculari (Yes, No). * **Sleep_Hours** Numero medio di ore di sonno a notte. * **Previous_Scores**

Punteggio degli esami precedenti. * **Motivation_Level** Livello di motivazione dello studente (Low, Medium, High). * **Internet_Access** Disponibilità di accesso ad Internet (Yes, No). * **Tutoring_Sessions** Numero di sessioni di tutoraggio frequentata al mese. * **Family_Income** Livello di reddito familiare (Low, Medium, High). * **Teacher_Quality** Qualità dell'insegnamento (Low, Medium, High). * **School_Type** Tipo di scuola frequentata (Public, Private). * **Peer_Influence** Influenza dei pari sulla performance accademica (Positive, Neutral, Negative). * **Physical_Activity** Numero medio di ore di attività fisica a settimana. * **Learning_Disabilities** Presenza di difficoltà di apprendimento (Yes, No). * **Parental_Education_Level** Livello più alto di educazione dei genitori (High School, College, Postgraduate). * **Distance_from_Home** Distanza da casa a scuola (Near, Moderate, Far). * **Gender** Genere dello studente (Male, Female). * **Exam_Score** Punteggio dell' esame finale.

Il dataset presenta sia variabili numeriche che categoriali, con valori ben distribuiti. Le ore di studio, la frequenza e le ore di sonno mostrano medie intorno a 20, 80 e 7 rispettivamente. La maggior parte degli studenti ha accesso a Internet e partecipa ad attività extracurricolari. Le categorie Parental Involvement, Motivation Level, e Family Income sono abbastanza bilanciate, mentre alcune categorie come Learning Disabilities e Gender mostrano distribuzioni sbilanciate. Il punteggio Exam_Score ha una media di circa 67, con valori compresi tra 55 e 101.

```
summary(ds)
```

```
##  Hours_Studied     Attendance     Parental_Involvement Access_to_Resources
##  Min.   : 1.00   Min.   : 60.00   High  :1908          High  :1975
##  1st Qu.:16.00   1st Qu.: 70.00   Low   :1337          Low   :1313
##  Median :20.00   Median : 80.00   Medium:3362          Medium:3319
##  Mean   :19.98   Mean   : 79.98
##  3rd Qu.:24.00   3rd Qu.: 90.00
##  Max.   :44.00   Max.   :100.00
##  Extracurricular_Activities  Sleep_Hours     Previous_Scores  Motivation_Level
##  No :2669                  Min.   : 4.000   Min.   : 50.00   High  :1319
##  Yes:3938                  1st Qu.: 6.000   1st Qu.: 63.00   Low   :1937
##                            Median : 7.000   Median : 75.00   Medium:3351
##                            Mean   : 7.029   Mean   : 75.07
##                            3rd Qu.: 8.000   3rd Qu.: 88.00
##                            Max.   :10.000   Max.   :100.00
##  Internet_Access Tutoring_Sessions Family_Income Teacher_Quality  School_Type
##  No : 499        Min.   :0.000     High  :1269         :  78     Private:2009
##  Yes:6108        1st Qu.:1.000     Low   :2672   High  :1947     Public :4598
##                  Median :1.000     Medium:2666   Low   : 657
##                  Mean   :1.494                   Medium:3925
##                  3rd Qu.:2.000
##                  Max.   :8.000
##   Peer_Influence Physical_Activity Learning_Disabilities
##  Negative:1377   Min.   :0.000     No :5912
##  Neutral :2592   1st Qu.:2.000     Yes: 695
##  Positive:2638   Median :3.000
##                  Mean   :2.968
##                  3rd Qu.:4.000
##                  Max.   :6.000
##  Parental_Education_Level Distance_from_Home    Gender       Exam_Score
##              :  90                :  67      Female:2793   Min.   : 55.00
##  College     :1989        Far     : 658      Male  :3814   1st Qu.: 65.00
##  High School :3223        Moderate:1998                    Median : 67.00
##  Postgraduate:1305        Near    :3884                    Mean   : 67.24
##                                                            3rd Qu.: 69.00
##                                                            Max.   :101.00
```

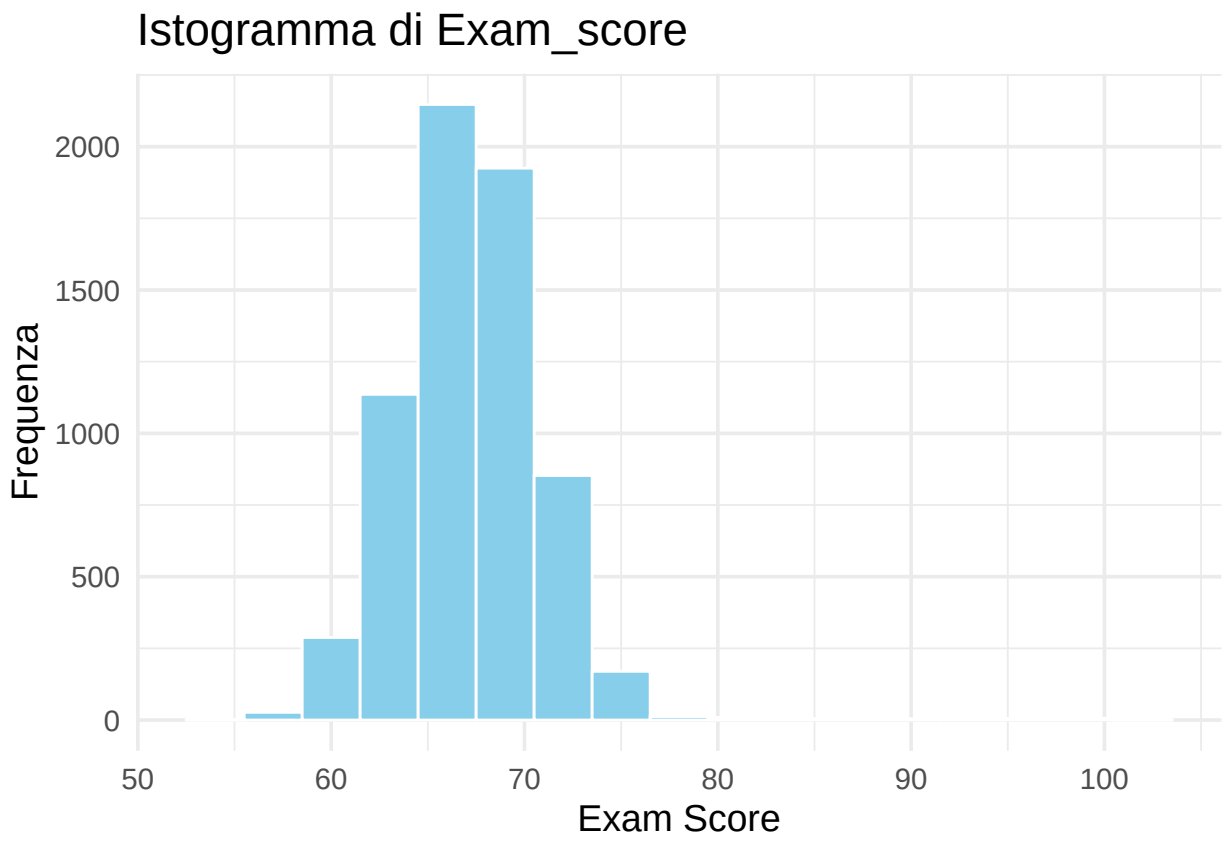
Vediamo il risultato della funzione str sul dataset, utile per comprendere i livelli delle variabili categoriche.

```
str(ds)
```

```
## 'data.frame':    6607 obs. of  20 variables:
##  $ Hours_Studied             : int  23 19 24 29 19 19 29 25 17 23 ...
##  $ Attendance                : int  84 64 98 89 92 88 84 78 94 98 ...
##  $ Parental_Involvement      : Factor w/ 3 levels "High","Low","Medium": 2 2 3 2 3 3 3 2 3 3 ...
##  $ Access_to_Resources       : Factor w/ 3 levels "High","Low","Medium": 1 3 3 3 3 3 2 1 1 3 ...
##  $ Extracurricular_Activities: Factor w/ 2 levels "No","Yes": 1 1 2 2 2 2 2 2 1 2 ...
##  $ Sleep_Hours               : int  7 8 7 8 6 8 7 6 6 8 ...
##  $ Previous_Scores           : int  73 59 91 98 65 89 68 50 80 71 ...
##  $ Motivation_Level          : Factor w/ 3 levels "High","Low","Medium": 2 2 3 3 3 3 2 3 1 3 ...
##  $ Internet_Access           : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Tutoring_Sessions         : int  0 2 2 1 3 3 1 1 0 0 ...
##  $ Family_Income             : Factor w/ 3 levels "High","Low","Medium": 2 3 3 3 3 3 2 1 3 1 ...
##  $ Teacher_Quality           : Factor w/ 4 levels "","High","Low",..: 4 4 4 4 2 4 4 2 3 2 ...
##  $ School_Type               : Factor w/ 2 levels "Private","Public": 2 2 2 2 2 2 1 2 1 2 ...
##  $ Peer_Influence            : Factor w/ 3 levels "Negative","Neutral",..: 3 1 2 1 2 3 2 1 2 3 ...
##  $ Physical_Activity         : int  3 4 4 4 4 3 2 2 1 5 ...
##  $ Learning_Disabilities     : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Parental_Education_Level  : Factor w/ 4 levels "","College","High School",..: 3 2 4 3 2 4 3 3 2 3
##  $ Distance_from_Home        : Factor w/ 4 levels "","Far","Moderate",..: 4 3 4 3 4 4 3 2 4 3 ...
##  $ Gender                    : Factor w/ 2 levels "Female","Male": 2 1 2 2 1 2 2 2 2 2 ...
##  $ Exam_Score                : int  67 61 74 71 70 71 67 66 69 72 ...
```

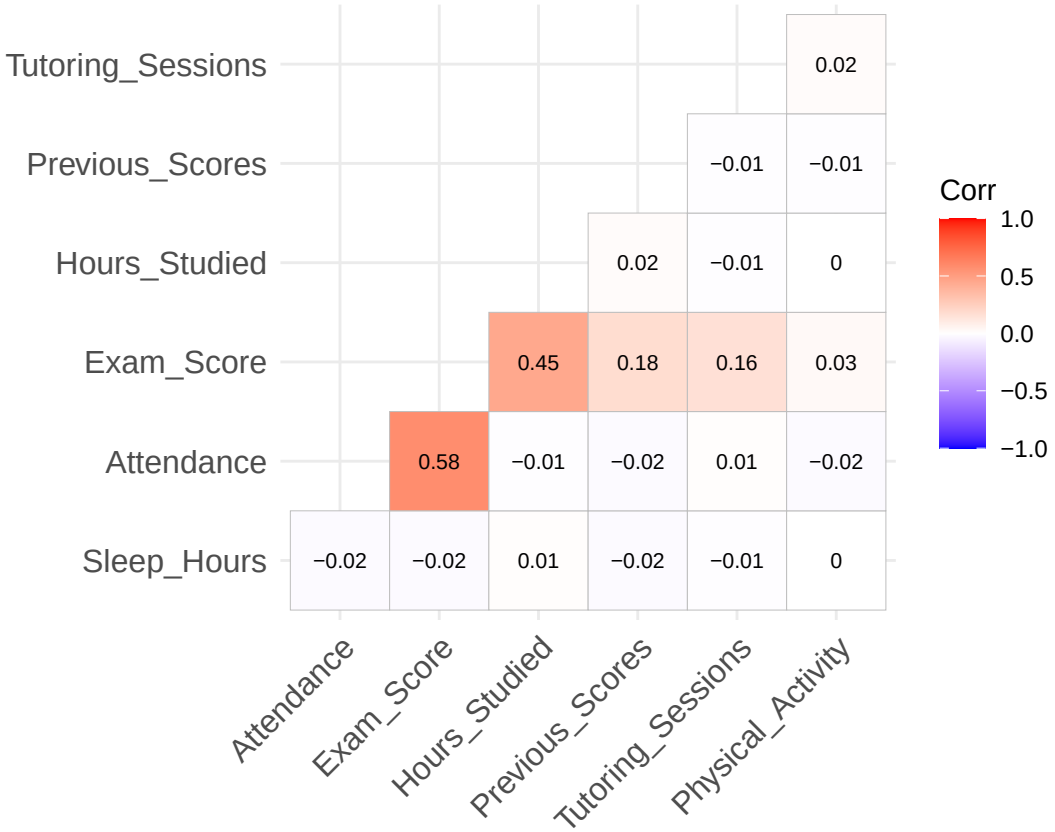
La variabile di interesse in questa analisi è *Exam_Score*. Per comprenderne meglio la sua natura, osserviamo la sua distribuzione. Per fa ciò è stato realizzato un istogramma che ci mostra un andamento quasi normale della variabile in questione, con la maggior parte dei punteggi compresa tra 63 e 75, e una presenza limitata di valori estremi.

```
ggplot(ds, aes(x = Exam_Score)) +
  geom_histogram(
    binwidth = 3,
    fill     = "skyblue",
    color    = "white"
  ) +
  labs(
    x     = "Exam Score",
    y     = "Frequenza",
    title = "Istogramma di Exam_score"
  ) +
  theme_minimal(base_size = 14)
```



Analizzando la matrice di correlazione calcolata sul dataset, si osserva che la variabile Exam_Score mostra una forte correlazione con Attendance (0.58) e Hours_Studied (0.45). La stessa variabile presenta inoltre una correlazione, seppur più debole, anche con Previous_Scores e Tutoring_Sessions. Non emergono invece correlazioni rilevanti tra le altre variabili numeriche del dataset.

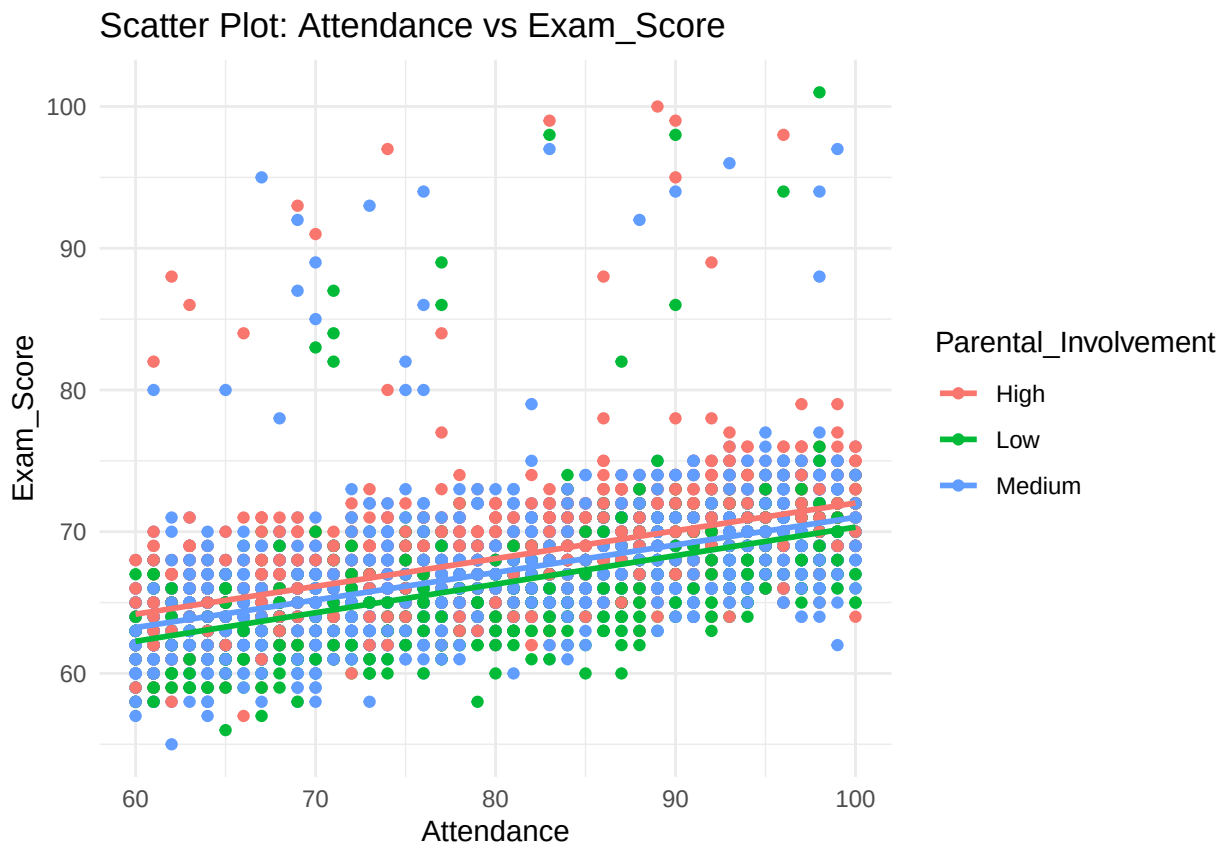
```
matrix_corrplot = round(cor(select_if(ds, is.numeric), method="pearson"),4)
ggcorrplot(matrix_corrplot, hc.order=T, type="lower", lab=T, lab_size = 2.7)
```



Alla luce dei risultati ottenuti dalla matrice di correlazione, è stato deciso di realizzare alcuni scatter plot per analizzare le variabili più significative. In questo grafico, è evidente come la variabile Exam_Score mostri una relazione lineare con la variabile Attendance. Inoltre, si osserva come la relazione tra le due variabili sembri essere influenzata dalla variabile categoriale Parental_Involvement, evidenziando una suddivisione dei bias presenti.

```
ggplot(ds, aes(x = Attendance, y = Exam_Score, color=Parental_Involvement)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(
    x = "Attendance",
    y = "Exam_Score",
    title = "Scatter Plot: Attendance vs Exam_Score"
  ) +
  theme_minimal()
```

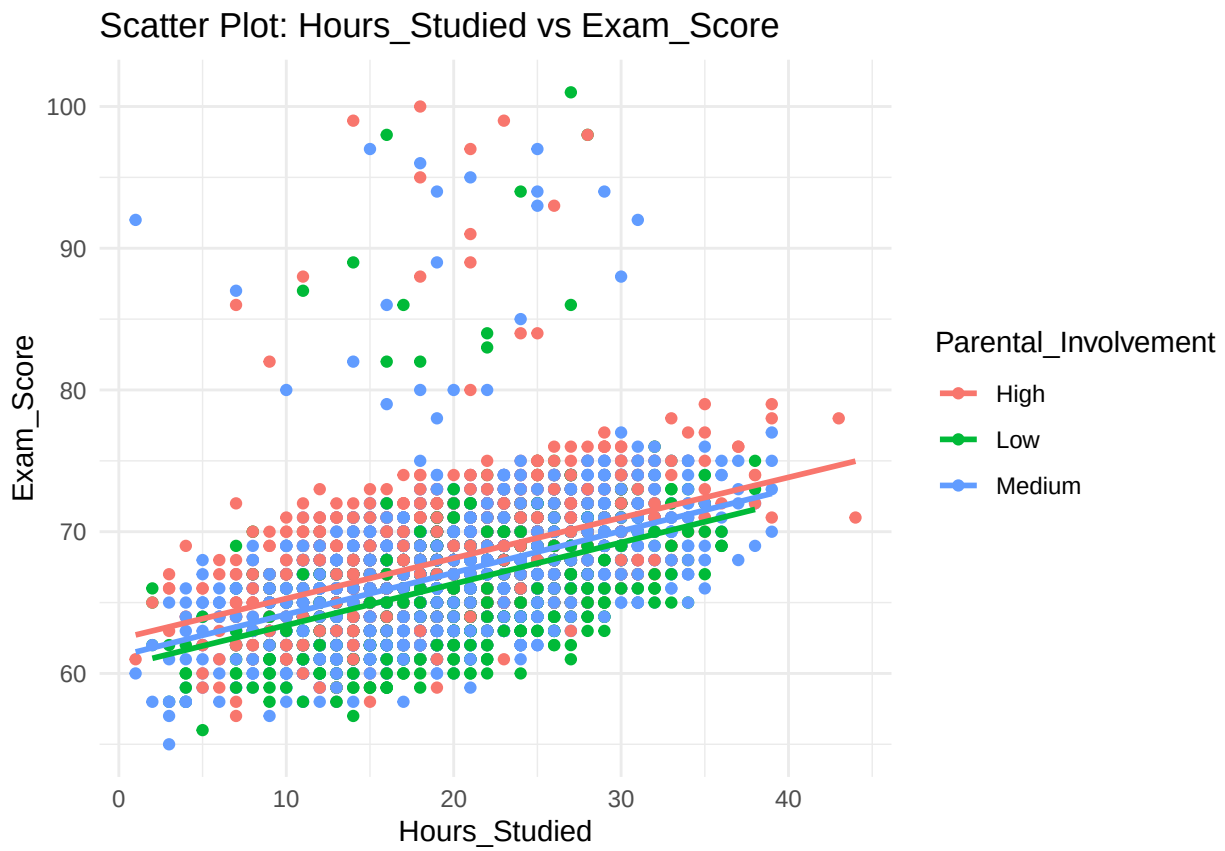
```
## `geom_smooth()` using formula = 'y ~ x'
```



Analogamente a prima, è stata effettuata la analisi precedente ma questa volta sono state considerate le ore di studio settimanali (Hours_Studied). Anche in questo caso si nota una forte dipendenza lineare fra le due variabili con una distinzione in base Parental_Involvement.

```
ggplot(ds, aes(x = Hours_Studied, y = Exam_Score, color=Parental_Involvement)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(
    x = "Hours_Studied",
    y = "Exam_Score",
    title = "Scatter Plot: Hours_Studied vs Exam_Score"
  ) +
  theme_minimal()
```

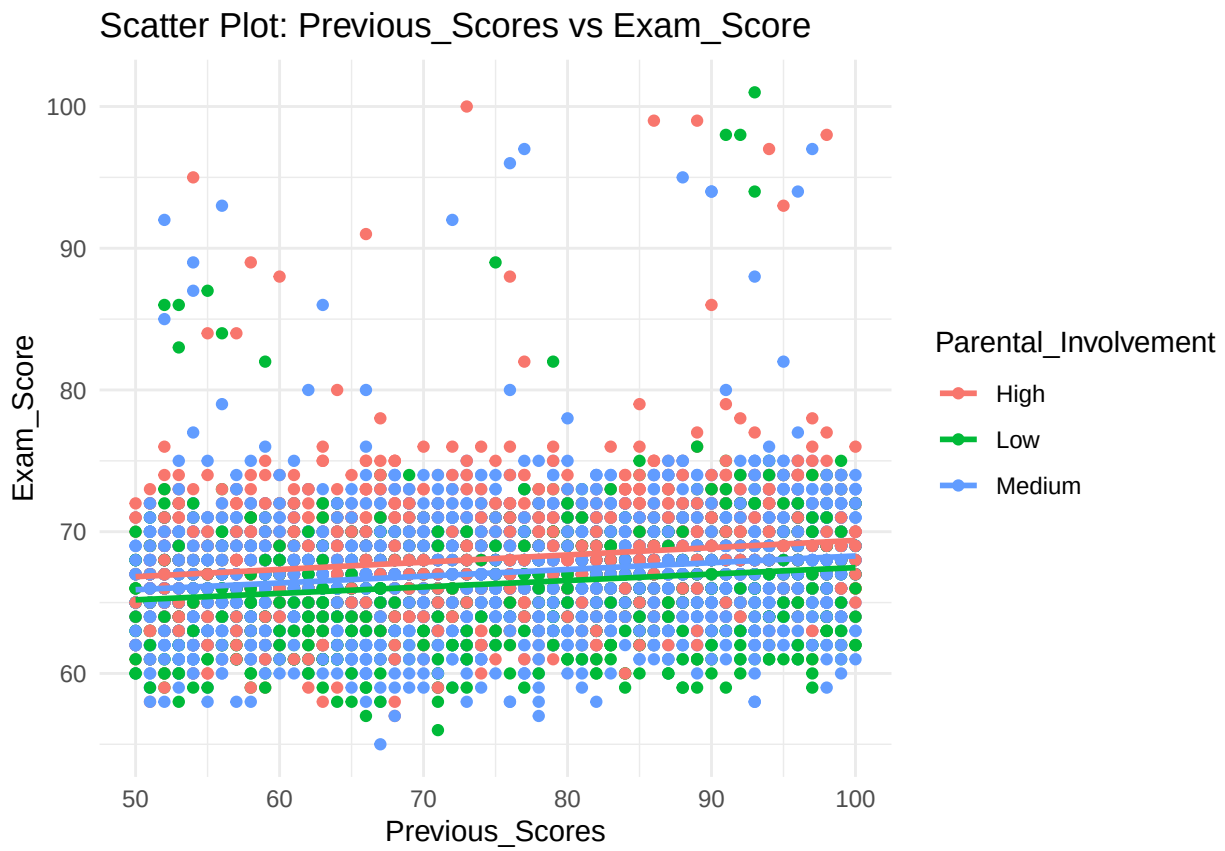
```
## `geom_smooth()` using formula = 'y ~ x'
```



Di particolare interesse è anche la variabile Previous_Scores, che intuitivamente potrebbe sembrare quella più adatta per predire il valore di Exam_Score. Tuttavia, questa analisi mette in luce una notevole variabilità fra i dati, suggerendo che altre variabili potrebbero giocare un ruolo altrettanto rilevante nella previsione dei punteggi dell'esame finale.

```
ggplot(ds, aes(x = Previous_Scores, y = Exam_Score,
               color=Parental_Involvement)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(
    x = "Previous_Scores",
    y = "Exam_Score",
    title = "Scatter Plot: Previous_Scores vs Exam_Score"
  ) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

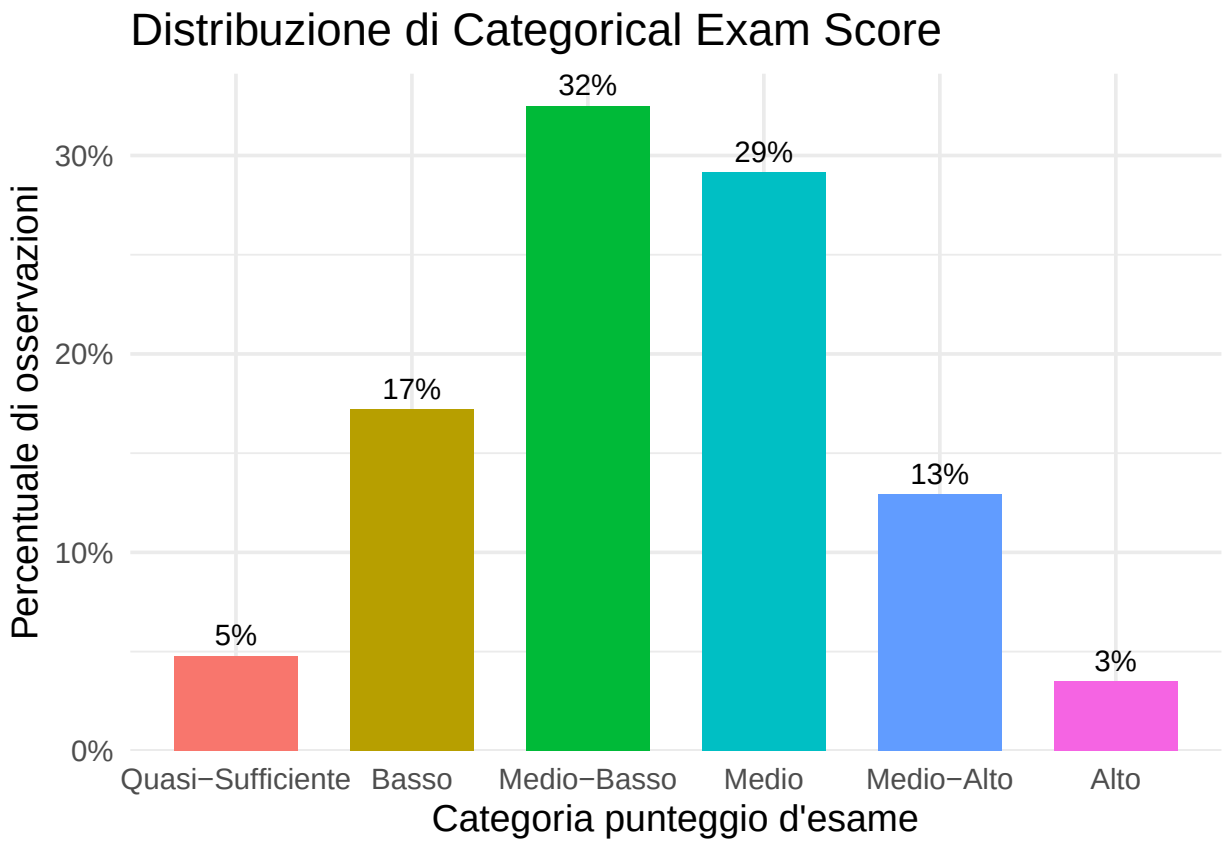


Poiché l'obiettivo di questa analisi è costruire un modello capace sia di individuare le variabili più rilevanti, sia di prevedere il rendimento di uno studente nell'esame finale, si è scelto di trasformare la variabile Exam_Score da numerica a categoriale. Infatti, non è tanto importante stimare il punteggio esatto che uno studente potrebbe ottenere, quanto piuttosto individuare la fascia di voto in cui è più probabile che si collochi. Questo approccio è utile anche per ipotizzare eventuali interventi didattici mirati, con l'intento di migliorare il percorso formativo degli studenti. Per raggiungere questo scopo, il codice seguente effettua tale trasformazione. Tuttavia, come mostrato nell'istogramma precedente, i punteggi sono concentrati in un intervallo molto ristretto. Per questo motivo sono stati creati due dataset con classificazioni differenti: - Il primo suddivide i voti in 6 classi, fornendo una stima più precisa del rendimento ma includendo due classi scarsamente rappresentate. - Il secondo utilizza 4 classi, semplificando il lavoro del modello a scapito però della precisione nella previsione del voto finale.

In entrambi i casi, le fasce sono state definite sfruttando la simmetria della distribuzione attorno al valore medio (67), con l'obiettivo di bilanciare le classi. Inoltre, gli intervalli non sono equidistanti, in quanto le fasce più estreme hanno ampiezze maggiori per compensare la minore densità dei dati in quelle zone.

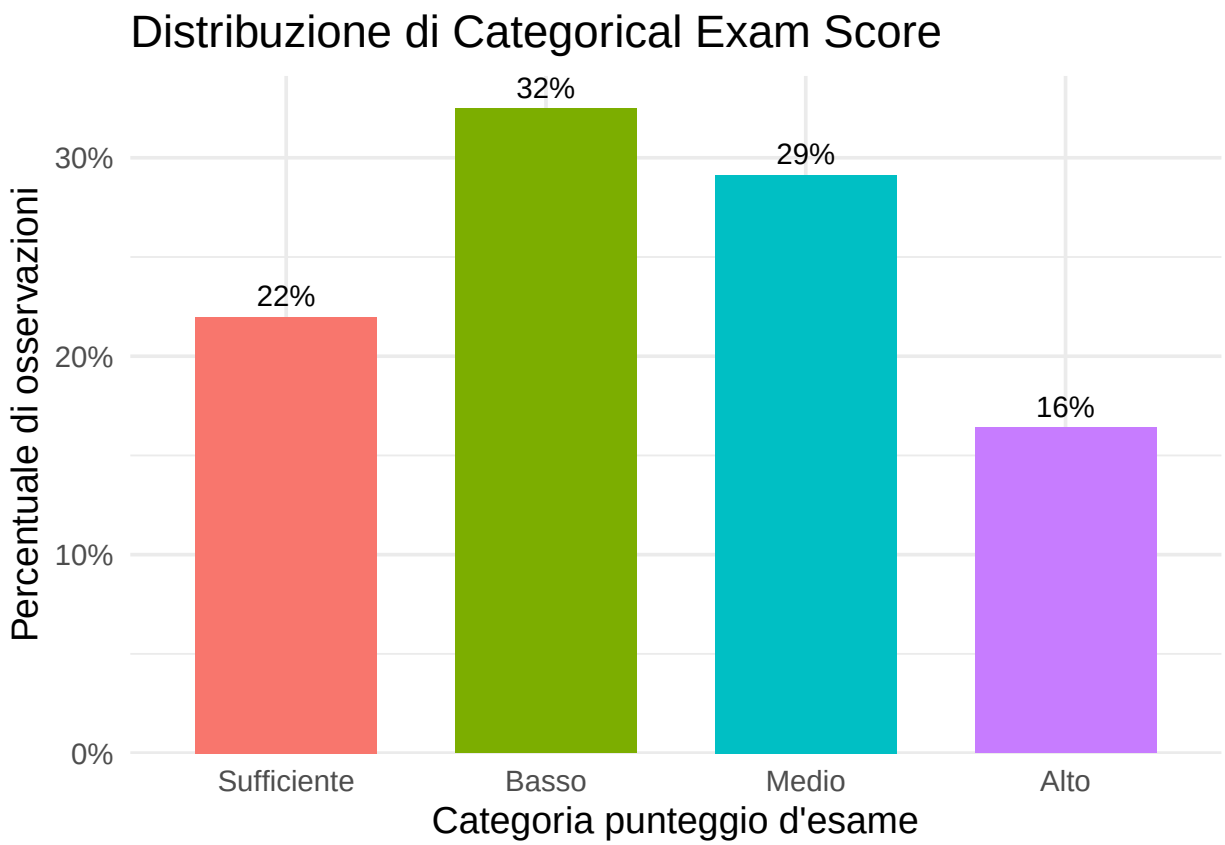
```
ds_2 = ds

ds_2$Categorical_Exam_Score <- cut(
  ds$Exam_Score,
  breaks = c(54, 64, 67, 70, 102),
  labels = c("Sufficiente", "Basso", "Medio", "Alto"),
  include.lowest = FALSE,
  right = TRUE
)
```

```r
ds$Categorical_Exam_Score <- cut(
  ds$Exam_Score,
  breaks = c(54, 61, 64, 67, 70, 73, 102),
  labels = c("Quasi-Sufficiente", "Basso", "Medio-Basso", "Medio",
             "Medio-Alto", "Alto"),
  include.lowest = FALSE,
  right = TRUE
)

ggplot(ds, aes(x = Categorical_Exam_Score,
               fill = Categorical_Exam_Score)) +
  # barre con proporzione
  geom_bar(
    aes(y = after_stat(count) / sum(after_stat(count))),
    stat = "count",
    width = 0.7,
    show.legend = FALSE
  ) +
  # percentuali sopra le barre
  geom_text(
    aes(
      label = percent(after_stat(count) / sum(after_stat(count)), accuracy = 1),
      y     = after_stat(count) / sum(after_stat(count))
    ),
    stat = "count",
    vjust = -0.5
  ) +
  # scala y in percentuale e un po' di spazio in alto
  scale_y_continuous(
    labels = percent_format(accuracy = 1),
    expand = expansion(mult = c(0, 0.05))
  ) +
  labs(
    x     = "Categoria punteggio d'esame",
    y     = "Percentuale di osservazioni",
    title = "Distribuzione di Categorical Exam Score"
  ) +
  theme_minimal(base_size = 14)
```




```r
ggplot(ds_2, aes(x = Categorical_Exam_Score,
                 fill = Categorical_Exam_Score)) +
  # barre con proporzione
  geom_bar(
    aes(y = after_stat(count) / sum(after_stat(count))),
    stat = "count",
    width = 0.7,
    show.legend = FALSE
  ) +
  # percentuali sopra le barre
  geom_text(
    aes(
      label = percent(after_stat(count) / sum(after_stat(count)), accuracy = 1),
      y     = after_stat(count) / sum(after_stat(count))
    ),
    stat = "count",
    vjust = -0.5
  ) +
  # scala y in percentuale e un po' di spazio in alto
  scale_y_continuous(
    labels = percent_format(accuracy = 1),
    expand = expansion(mult = c(0, 0.05))
  ) +
  labs(
    x     = "Categoria punteggio d'esame",
    y     = "Percentuale di osservazioni",
```

```
    title = "Distribuzione di Categorical Exam Score"
  ) +
  theme_minimal(base_size = 14)
```



Divisione del dataset in train e test

In questa sezione si è effettuata la suddivisione dei due dataset in train e test. Il seguente codice divide il datsaet con le classi di Categorical_Exam_Score più numerose.

```
set.seed(123)
ds$Exam_Score= NULL

trainIndex <- createDataPartition(ds$Categorical_Exam_Score,
                                  p    = 0.5,
                                  list = FALSE)

train <- ds[ trainIndex, ]
test  <- ds[-trainIndex, ]
```

Mentre il seguente codice divide il datsaet con le classi di Categorical_Exam_Score meno numerose.

```
set.seed(123)
ds_2$Exam_Score= NULL

trainIndex_2 <- createDataPartition(ds_2$Categorical_Exam_Score,
                                  p    = 0.5,
                                  list = FALSE)
```

```
train_2 <- ds_2[ trainIndex, ]
test_2  <- ds_2[-trainIndex, ]
```

Analisi

Random forest

Le Random Forest si basano sulla costruzione di numerosi alberi decisionali generati tramite campionamento bootstrap. A differenza degli alberi classici, introducono il parametro mtry, che seleziona un sottoinsieme casuale delle p variabili presenti nel dataset ad ogni split. Questo sottoinsieme viene utilizzato per individuare la variabile su cui effettuare la divisione, contribuendo a ridurre la correlazione tra gli alberi e migliorare la capacità di generalizzazione del modello. Solitamente nei problemi di classificazione si tende a costruire un numero di alberi pari a 10*p e un numero di variabili $mtry = sqrt(p)$. Procediamo quindi a fittate il modello usando questi parametri.

```
mtry=sqrt(ncol(train))
mtry
```

```
## [1] 4.472136
```

```
B=200
rf_1 <- randomForest(Categorical_Exam_Score ~ . , data = train, ntree=B,
                     mtry= mtry, method="class")
rf_1
```

```
##
## Call:
##  randomForest(formula = Categorical_Exam_Score ~ ., data = train,      ntree = B, mtry = mtry, metho
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 34.92%
## Confusion matrix:
##                    Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
## Quasi-Sufficiente                60    93           5     0          0    0
## Basso                             3   309         256     0          0    0
## Medio-Basso                       0    61         840   173          0    0
## Medio                             0     0         213   720         30    0
## Medio-Alto                        0     0           3   209        209    6
## Alto                              0     5           8    12         77   13
##                    class.error
## Quasi-Sufficiente    0.6202532
## Basso                0.4559859
## Medio-Basso          0.2178771
## Medio                0.2523364
## Medio-Alto           0.5105386
## Alto                 0.8869565
```

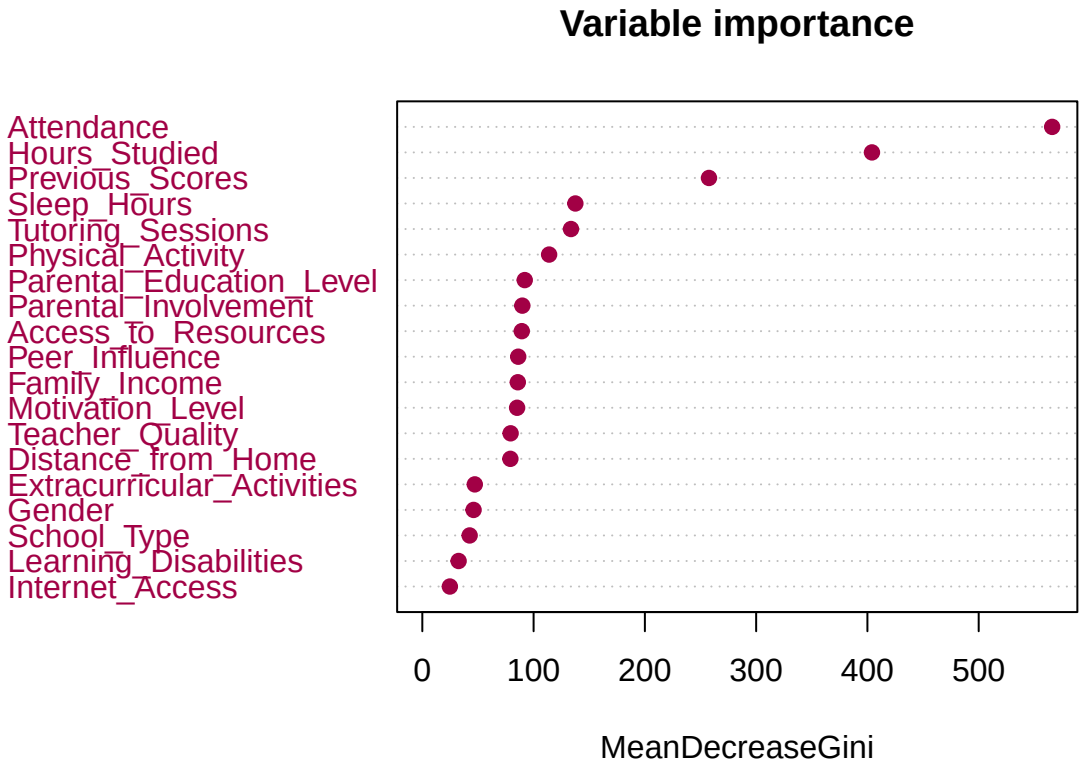
Dall'output della matrice di confusione si osserva come, nelle classi più sbilanciate, l'algoritmo — con le impostazioni adottate e considerando quattro variabili di split che cambiano da simulazione a simulazione — evidenzi una scarsa capacità di apprendere correttamente le caratteristiche delle classi sottorappresentate. Questo comportamento è tipico nei contesti in cui il dataset presenta un forte sbilanciamento tra le categorie, e può compromettere l'efficacia predittiva del modello per le classi meno frequenti. Per approfondire l'analisi, rappresentiamo l'andamento dell'errore OOB (Out-of-Bag) in funzione del numero di alberi. Tale errore

misura la capacità dell'algoritmo di predire correttamente le osservazioni che non sono state utilizzate durante l'addestramento di ciascun albero, fornendo una stima realistica dell'errore di generalizzazione. Il grafico mostra sia l'errore OOB complessivo, sia quello calcolato separatamente per ciascuna classe, offrendo una panoramica dettagliata delle difficoltà che il modello incontra nelle diverse categorie. Infine, analizziamo la variable importance, un indicatore che misura il contributo di ciascuna variabile nella riduzione dell'impurità all'interno degli alberi. Questo strumento permette di individuare quali variabili risultano più informative nel processo di classificazione.

```
  library(RColorBrewer)

n_classi <- ncol(rf_1$err.rate)
n_classi
```

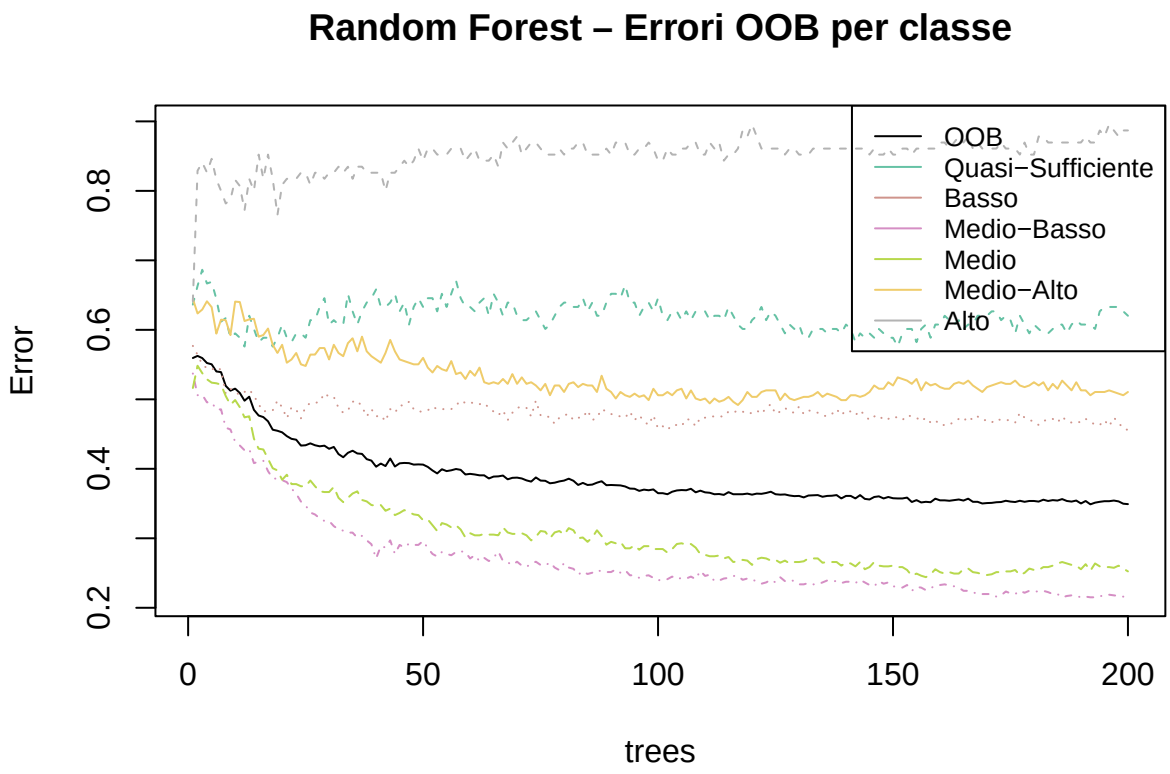
```
## [1] 7
```

```
varImpPlot(rf_1, main="Variable importance", pch = 19, color="#A20045")
```




```
# Colori: OOB in nero, classi con una palette armoniosa
colori <- c("black", colorRampPalette(brewer.pal(8, "Set2"))(n_classi - 1))

# Plot errori OOB
plot(rf_1,
     col = colori,
     main = "Random Forest - Errori OOB per classe")

# Aggiunta legenda
legend("topright",
       legend = colnames(rf_1$err.rate),
```

```
    col = colori,
        lty = 1,
        cex = 0.8)
```



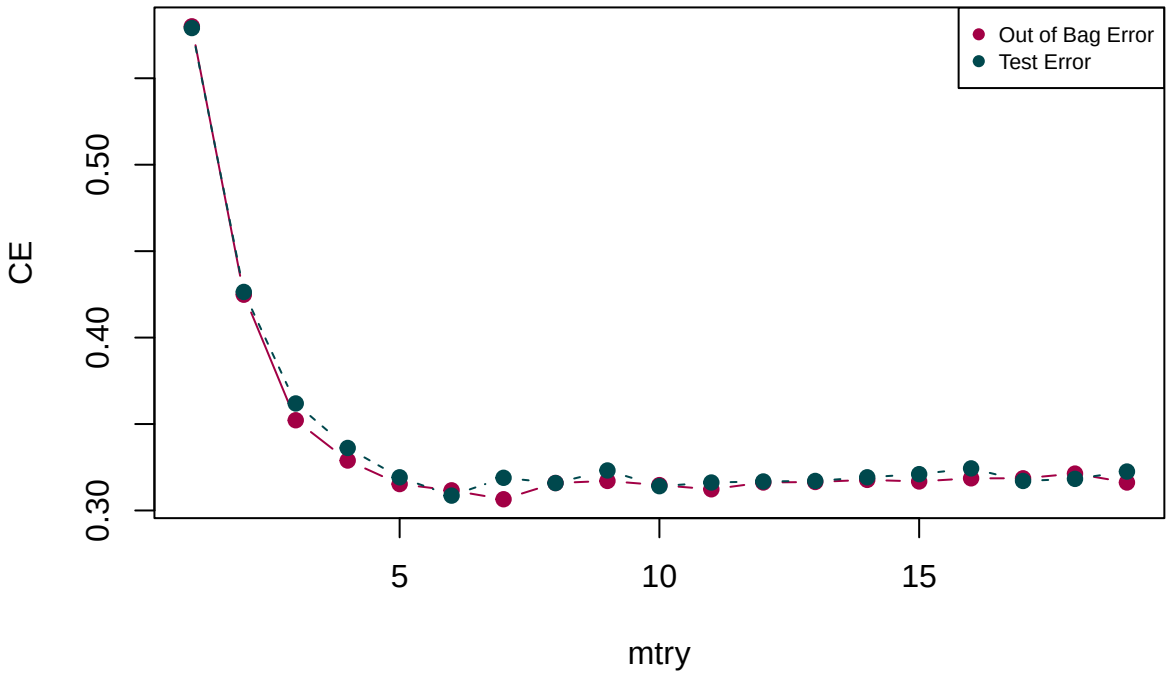
Un aspetto interessante emerso dall'analisi è che l'errore OOB complessivo, così come quello associato a diverse classi, tende a diminuire progressivamente all'aumentare del numero di alberi costruiti sui campioni bootstrap. Questo comportamento conferma l'efficacia dell'aggregazione nel ridurre la varianza del modello e migliorare la sua stabilità. Tuttavia, si nota che per le classi quasi sufficiente e alto, l'errore OOB rimane pressoché costante, suggerendo che il modello fatica a distinguere correttamente queste categorie.

Inoltre, viene rappresentata anche la variable importance: essa misura il contributo di ciascuna variabile nella riduzione dell'impurità (indice di Gini) durante la costruzione degli split. Le variabili più rilevanti risultano essere Hours_Studied, che indica il numero di ore dedicate allo studio settimanale, Attendance, che rappresenta la percentuale di lezioni frequentate, e Previous_Scores, ovvero il punteggio ottenuto negli esami precedenti. Questi fattori sembrano avere un peso decisivo nella classificazione degli studenti. Anche le ore di sonno forniscono un contributo, sebbene meno marcato.

A questo punto dell'analisi, vogliamo studiare come cambia il comportamento del modello al variare del parametro mtry, che indica quante variabili vengono considerate a ogni split di ciascun albero. Per farlo, confronteremo l'errore OOB e l'errore sul test set per diversi valori di mtry. Per rendere le stime più stabili, decidiamo inoltre di aumentare il numero di alberi a 1000.

```
set.seed(112)

B <- 1000
oob.err <- c()
test.err <- c()
p <- NCOL(ds) - 1
```

```
for(mtry in 1:p){
  rf <- randomForest(Categorical_Exam_Score ~ . , data = train , mtry=mtry,
                     ntree=B)
  oob.err[mtry] <- rf$err.rate[B, "OOB"]
  pred=predict(rf, newdata = test)
  test.err[mtry] <- mean(pred != test$Categorical_Exam_Score)
}

matplot(1:mtry , cbind(oob.err,test.err), pch=19 ,
        col=c("#A20045","#00484D"),type="b",ylab="CE",xlab="mtry")
legend("topright",legend=c("Out of Bag Error","Test Error"),pch=19,
       col=c("#A20045","#00484D"), cex = 0.7)
```

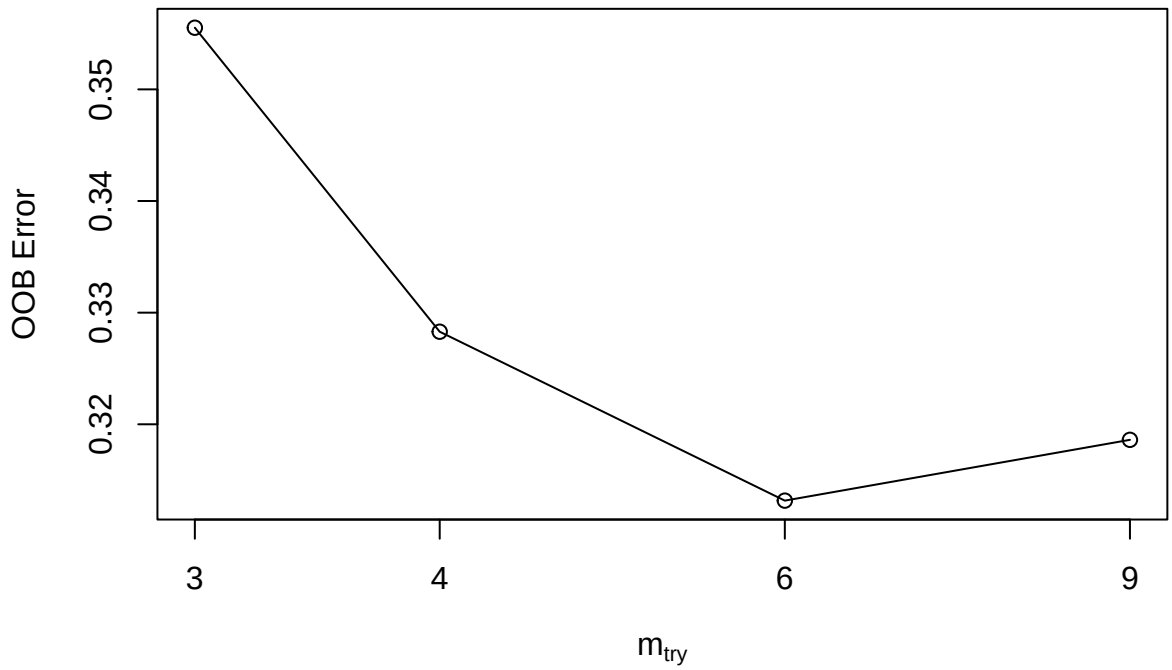


Il grafico riportato sopra fornisce informazioni preziose sull'andamento dell'errore OOB al variare del parametro mtry. Si osserva che, all'aumentare di mtry, l'errore OOB tende a diminuire rispetto all'errore calcolato sul test set. La scelta del parametro mtry è molto importante: un valore troppo elevato di mtry, infatti, può portare gli alberi ad essere troppo simili tra loro, con il rischio di adattarsi eccessivamente ai dati di training. Al contrario, valori troppo bassi di mtry riducono la correlazione tra gli alberi, il che è positivo in termini di generalizzazione, ma può diventare problematico quando nel dataset sono presenti molte variabili irrilevanti. In questi casi, selezionare un numero troppo basso di predittori rischia di far scegliere, durante gli split, variabili non informative, peggiorando la qualità dell'albero. Per affrontare questo trade-off e individuare il valore ottimale di mtry, abbiamo effettuato un tuning del parametro sul training set utilizzando la funzione tuneRF, che consente di testare diversi valori e selezionare quello che minimizza l'errore OOB.

```
set.seed(112)

my.mtry <- tuneRF(train[,-20],train$Categorical_Exam_Score, ntreeTry=1000,
```

```
                stepFactor=1.5,improve=0.001, trace=TRUE, plot=TRUE)
```

```
## mtry = 4  OOB error = 32.83%
## Searching left ...
## mtry = 3     OOB error = 35.55%
## -0.08294931 0.001
## Searching right ...
## mtry = 6     OOB error = 31.32%
## 0.04608295 0.001
## mtry = 9     OOB error = 31.86%
## -0.0173913 0.001
```




```
set.seed(112)
best_mtry <- my.mtry[which.min(my.mtry[, 2]),1]
best_mtry
```

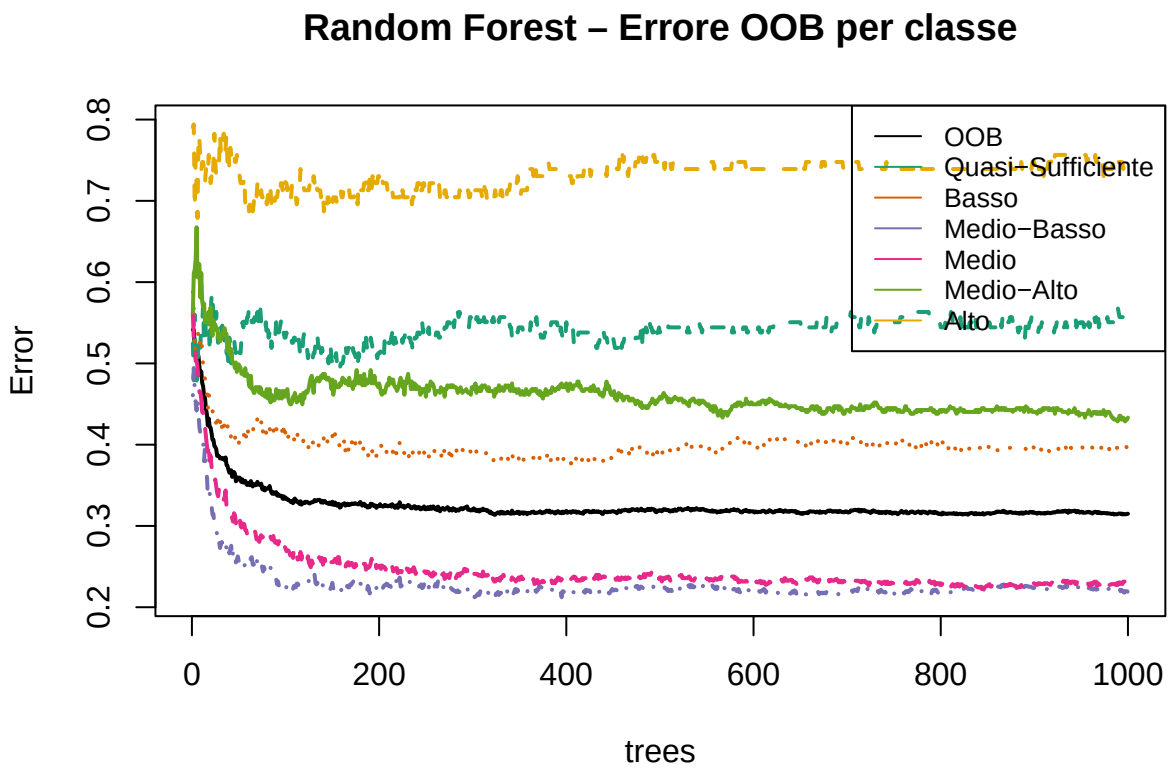
```
## [1] 6
```

```
mtry=6
rf_1_o <- randomForest(Categorical_Exam_Score ~ . , data = train,
                       ntree=1000, mtry= mtry)
rf_1_o
```

```
##
## Call:
##  randomForest(formula = Categorical_Exam_Score ~ ., data = train,      ntree = 1000, mtry = mtry)
##                Type of random forest: classification
##                      Number of trees: 1000
```

```
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 31.5%
## Confusion matrix:
##                    Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
## Quasi-Sufficiente                70    87           1     0          0    0
## Basso                            10   342         216     0          0    0
## Medio-Basso                       0    76         839   159          0    0
## Medio                             0     0         184   741         38    0
## Medio-Alto                        0     0           3   177        242    5
## Alto                              0     4           9     8         64   30
##                    class.error
## Quasi-Sufficiente    0.5569620
## Basso                0.3978873
## Medio-Basso          0.2188082
## Medio                0.2305296
## Medio-Alto           0.4332553
## Alto                 0.7391304
```

```r
n_classi <- ncol(rf_1_o$err.rate)

colori <- c("black", brewer.pal(n_classi - 1, "Dark2"))  # palette per 6 classi

plot(rf_1_o,
     col = colori,
     main = "Random Forest - Errore OOB per classe",
     lwd = 2)

legend("topright",
       legend = colnames(rf_1_o$err.rate),
       col = colori,
       lty = 1,
       cex = 0.8)
```



La situazione non mostra miglioramenti significativi: l'errore OOB complessivo e quello di ciascuna classe, rimane pressoché invariato e l'algoritmo continua a mostrare difficoltà nel classificare correttamente le categorie meno rappresentate, in particolare alto e quasi sufficiente, anche se quest'ultimo ha un OOB error più basso.

Per valutare in modo più completo le prestazioni del modello, procediamo ora a confrontare le predizioni ottenute sui dati di test con i valori osservati, calcolando così l'accuratezza su dati mai visti in fase di addestramento.

```
rf_1p= predict(rf_1_o, newdata = test)
table(rf_1p)
```

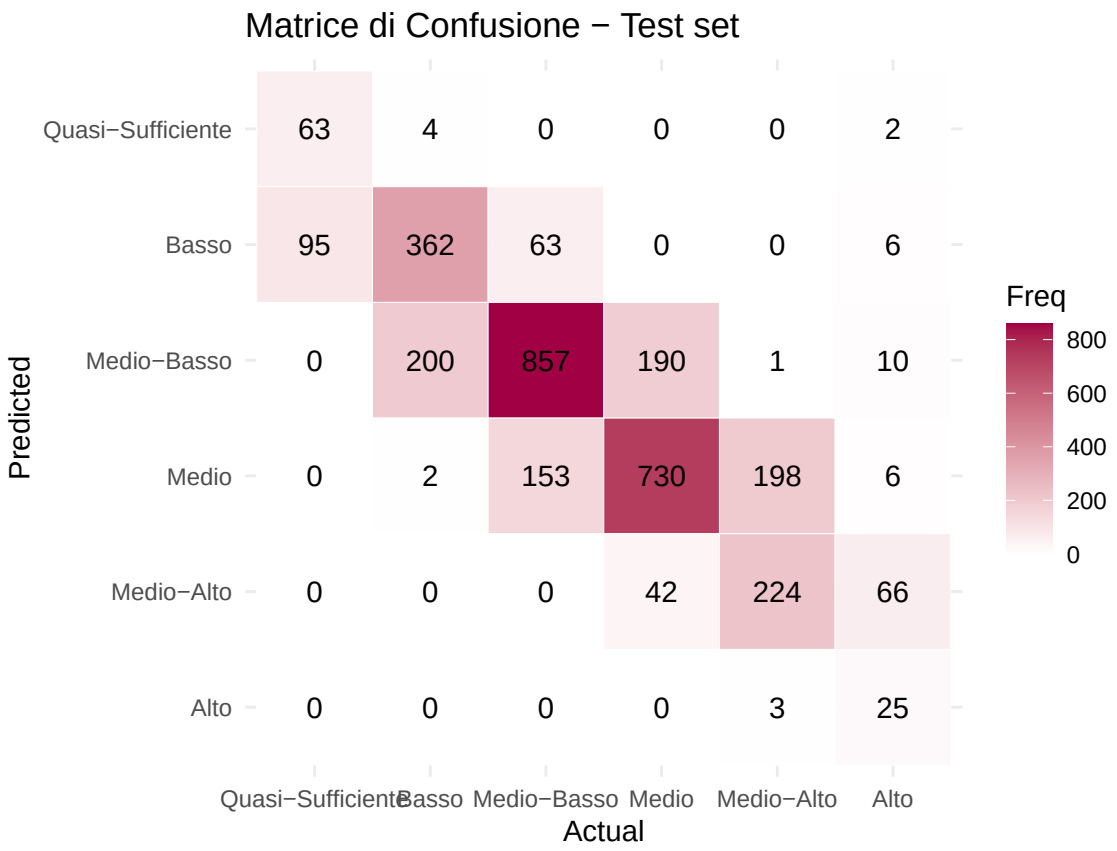
```
## rf_1p
## Quasi-Sufficiente             Basso       Medio-Basso             Medio
##                69               526              1258              1089
##        Medio-Alto              Alto
##               332                28
```

```
actuals <- test$Categorical_Exam_Score
confusion_matrix <- table(Predicted = rf_1p, Actual = actuals)
print(confusion_matrix)
```

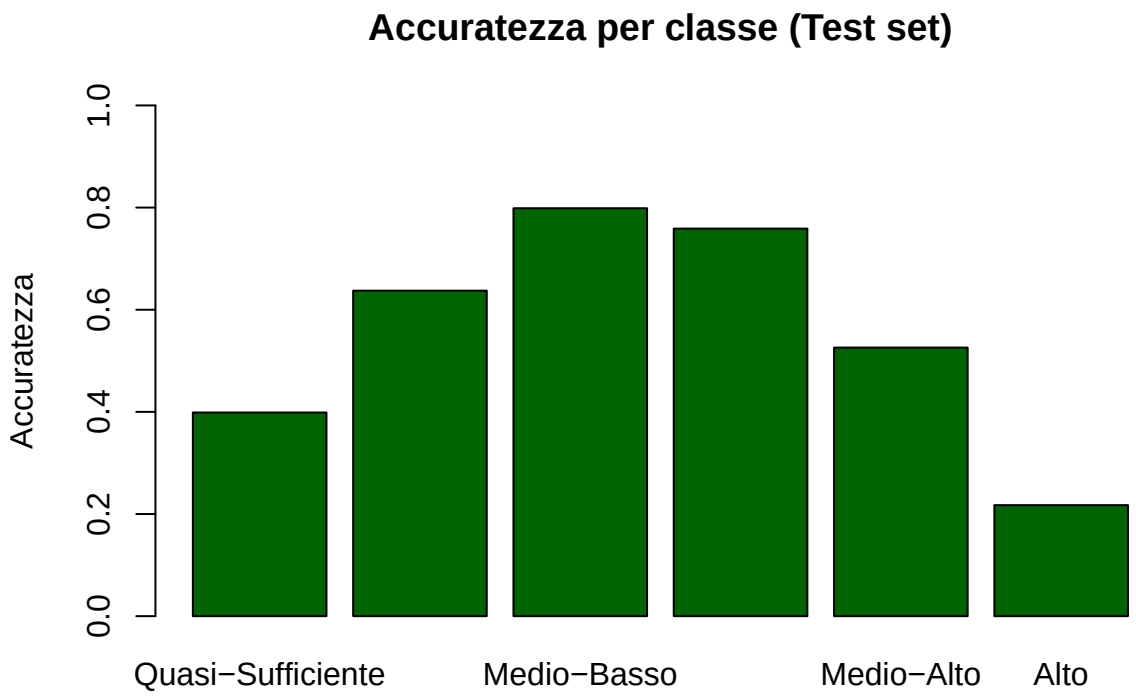
```
##                   Actual
## Predicted          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente                63     4           0     0          0    2
##   Basso                            95   362          63     0          0    6
##   Medio-Basso                       0   200         857   190          1   10
##   Medio                             0     2         153   730        198    6
##   Medio-Alto                        0     0           0    42        224   66
```

```
##   Alto                            0      0           0      0          3    25
```

```r
conf_df <- as.data.frame(confusion_matrix)
ggplot(conf_df, aes(x = Actual, y = Predicted)) +
  geom_tile(aes(fill = Freq), color = "white") +
  geom_text(aes(label = Freq), vjust = 0.5) +
  scale_fill_gradient(low = "white", high = "#A20045") +
  scale_y_discrete(limits = rev) +
  theme_minimal() +
  labs(title = "Matrice di Confusione - Test set") +
  coord_fixed()
```




```r
accuratezza_classe <- diag(prop.table(confusion_matrix, 2))
barplot(accuratezza_classe,
        main = "Accuratezza per classe (Test set)",
        col = "darkgreen",
        ylab = "Accuratezza",
        ylim = c(0, 1))
```



Dall'analisi dei risultati sul test set emerge che il modello riesce a classificare correttamente le classi basso, medio-basso, medio e medio-alto, confermando quanto osservato anche sul training. Tuttavia, come già evidenziato nelle osservazioni precedenti, lo squilibrio tra le classi continua a penalizzare le categorie meno frequenti: quasi sufficiente e alto risultano ancora le più difficili da predire, con una bassa accuratezza anche sul test set.

Weighted Class:

Dopo aver osservato le difficoltà del modello nel classificare correttamente le classi meno rappresentate — e constatato che né l'aumento del numero di alberi né l'ottimizzazione del parametro mtry sono riusciti a risolvere il problema — abbiamo deciso di pesare ciascuna classe, attribuendo un maggior peso alle classi con meno osservazioni. In termini pratici, i pesi sono stati calcolati come l'inverso della frequenza assoluta di ciascuna classe. Tali valori sono stati poi normalizzati dividendo per la somma complessiva dei pesi grezzi e moltiplicati per il numero totale delle classi. Questo approccio assegna un peso maggiore alle classi più rare, con l'obiettivo di aumentare la probabilità che il modello le riconosca correttamente, senza che vengano "schiacciate" dalle classi predominanti. Dal punto di vista teorico, l'introduzione dei pesi ha un impatto diretto sul criterio di splitting adottato negli alberi. Di conseguenza, il modello è incentivato a costruire partizioni che favoriscano anche le classi meno rappresentate.

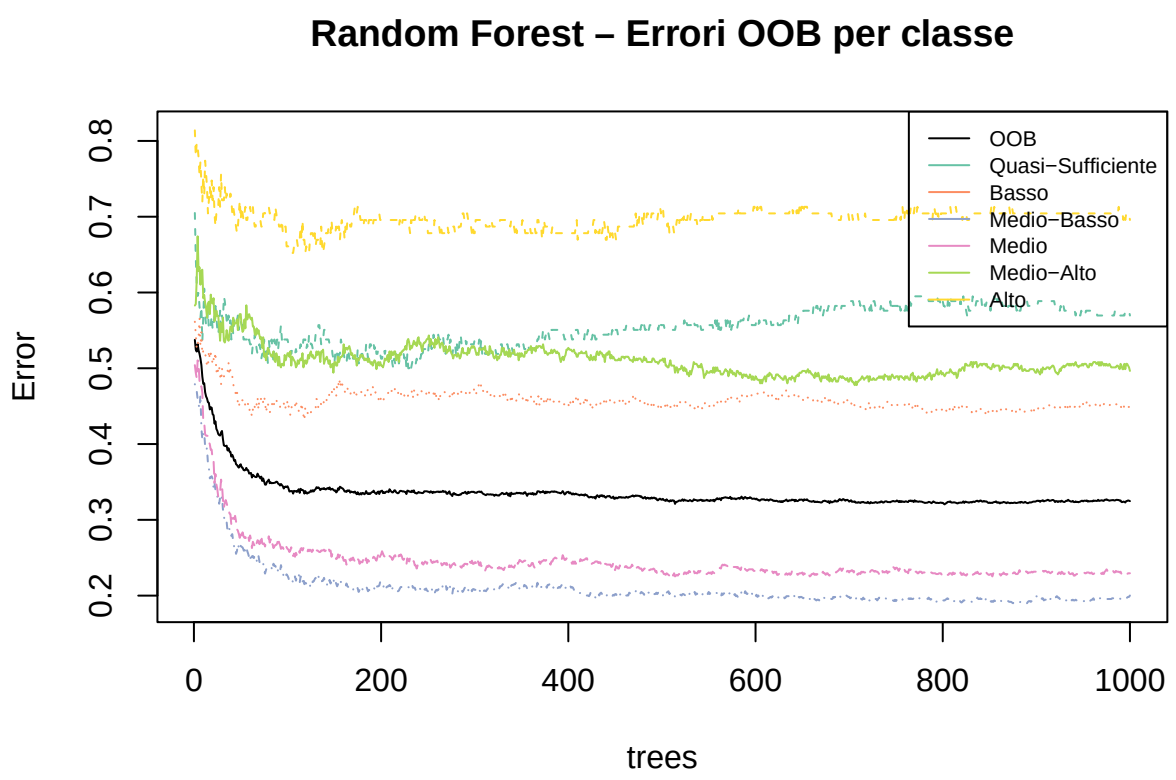
```
#modifico l'influenza di ogni classe sul modello in modo inversamente proporzionale alla sua frequenza
class_weights= 1/table(train$Categorical_Exam_Score)
#costruisco i pesi omega(i)*numero di classi/sum(omega(i))
class_weights_1 <- class_weights / sum(class_weights) * length(class_weights)
```

```
set.seed(112)
rf_weighted <- randomForest(Categorical_Exam_Score~ .,
                            data = train,
```

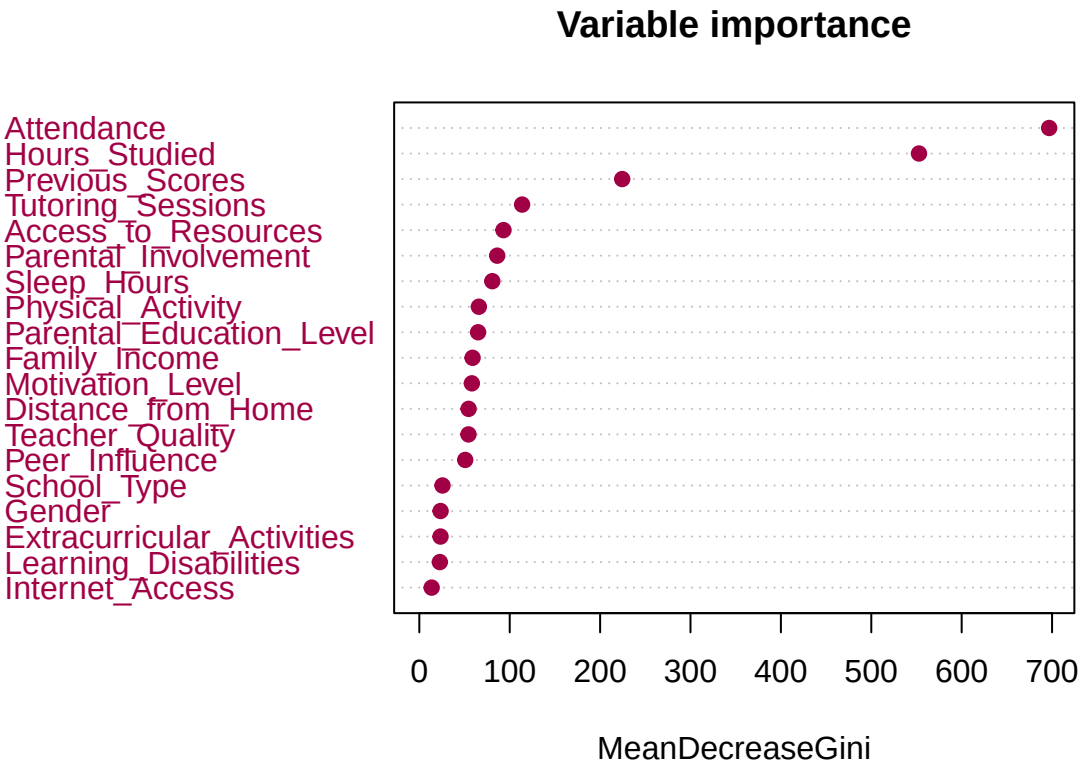
```
                            mtry=6,
                            ntree=1000,
                            importance = FALSE,
                            classwt = class_weights_1)
rf_weighted
```

```
##
## Call:
##  randomForest(formula = Categorical_Exam_Score ~ ., data = train,      mtry = 6, ntree = 1000, impor
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 32.47%
## Confusion matrix:
##                    Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
## Quasi-Sufficiente                68    87           3     0          0    0
## Basso                            10   313         245     0          0    0
## Medio-Basso                       0    52         859   163          0    0
## Medio                             0     0         182   742         39    0
## Medio-Alto                        0     0           2   200        215   10
## Alto                              0     5           8     8         59   35
##                    class.error
## Quasi-Sufficiente    0.5696203
## Basso                0.4489437
## Medio-Basso          0.2001862
## Medio                0.2294912
## Medio-Alto           0.4964871
## Alto                 0.6956522
```

```
n_classi <- ncol(rf_weighted$err.rate)


# Colori: OOB in nero, classi con una palette armoniosa
colori <- c("black", RColorBrewer::brewer.pal(n_classi - 1, "Set2"))

# Plot OOB error per classe
plot(rf_weighted,
     col = colori,
     main = "Random Forest - Errori OOB per classe")

# Legenda
legend("topright",
       legend = colnames(rf_weighted$err.rate),
       col = colori,
       lty = 1,
       cex = 0.7)
```

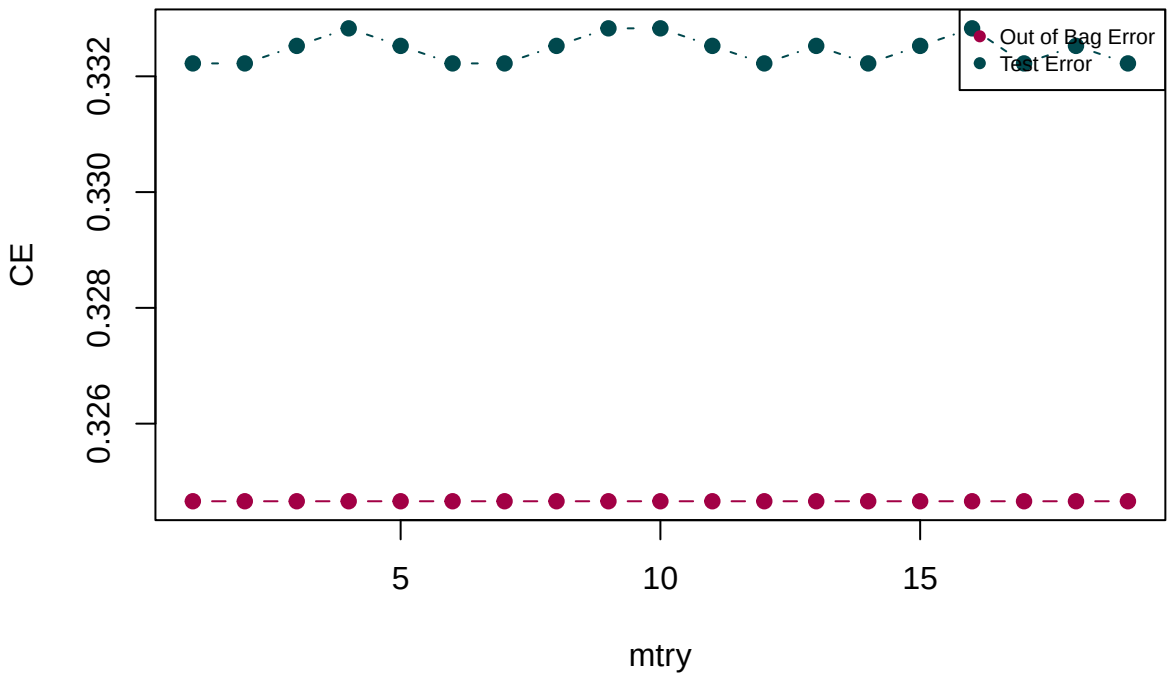



```r
varImpPlot(rf_weighted, main="Variable importance", pch = 19, color="#A20045")
```



Dal grafico è possibile osservare che l'errore associato alle classi più sbilanciate si mantiene pressoché costante anche con l'aumentare del numero di alberi, segnalando che il modello continua ad avere difficoltà nel gestire queste categorie. Riproviamo a fare un tuning del parametro mtry per vedere se il modello ci dà un parametro ottimale diverso che riduca di più l'OOB error e l'errore sul test sample.

```
set.seed(112)

B <- 1000
oob.err_w <- c()
test.err_w <- c()
p <- NCOL(train) - 1
for(mtry in 1:p){
  rf <- randomForest(Categorical_Exam_Score ~ . , data = train ,
                     mtry=mtry, ntree=B)
  oob.err_w[mtry] <- rf_weighted$err.rate[B, "OOB"]
  pred_w=predict(rf_weighted, newdata = test)
  test.err_w[mtry] <- mean(pred_w != test$Categorical_Exam_Score)
}

matplot(1:mtry , cbind(oob.err_w,test.err_w), pch=19 ,
        col=c("#A20045","#00484D"),type="b",ylab="CE",xlab="mtry")
legend("topright",legend=c("Out of Bag Error","Test Error"),pch=19,
       col=c("#A20045","#00484D"), cex = 0.7)
```



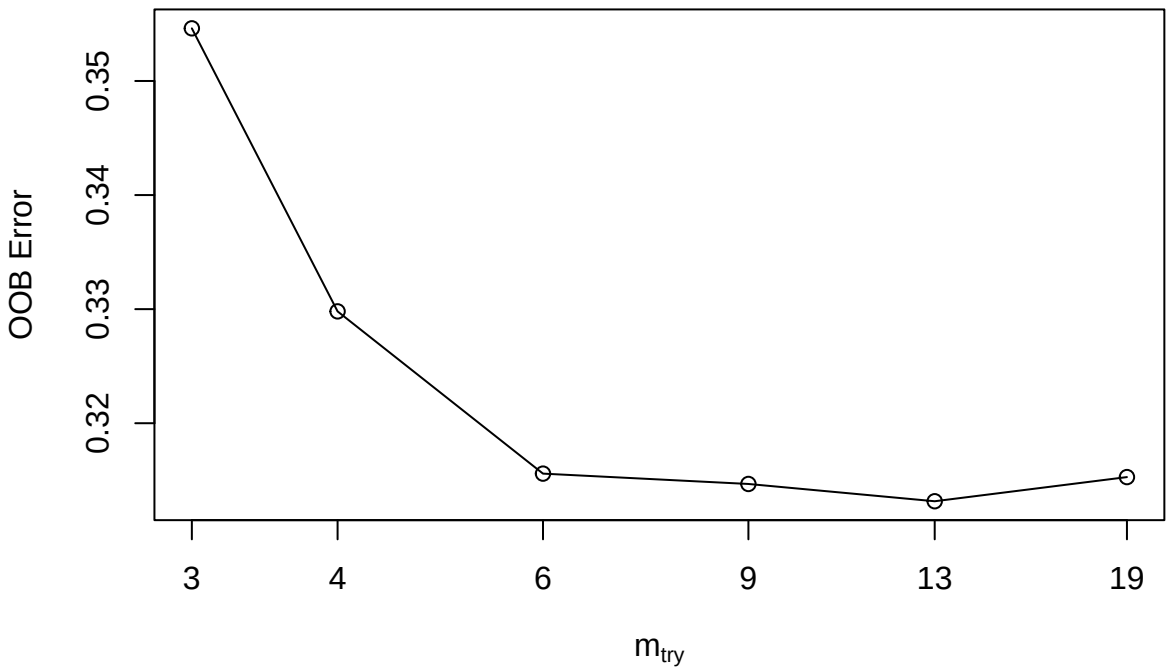
Dal grafico si vede una stabilizzazione dell'OOB error, che si mantiene costante al variare del parametro mtry, dopo aver ripesato le classi; mentre il test error mostra delle piccole oscillazioni.

```
best_mtry <- my.mtry[which.min(my.mtry[, 2]),1]
best_mtry
```

```
## [1] 6
```

```
my.mtry <- tuneRF(train[,-20],train$Categorical_Exam_Score, ntreeTry=1000,
                  stepFactor=1.5,improve=0.001, trace=TRUE, plot=TRUE)
```

```
## mtry = 4  OOB error = 32.98%
## Searching left ...
## mtry = 3     OOB error = 35.46%
## -0.07522936 0.001
## Searching right ...
## mtry = 6     OOB error = 31.56%
## 0.04311927 0.001
## mtry = 9     OOB error = 31.47%
## 0.002876318 0.001
## mtry = 13    OOB error = 31.32%
## 0.004807692 0.001
## mtry = 19    OOB error = 31.53%
## -0.006763285 0.001
```

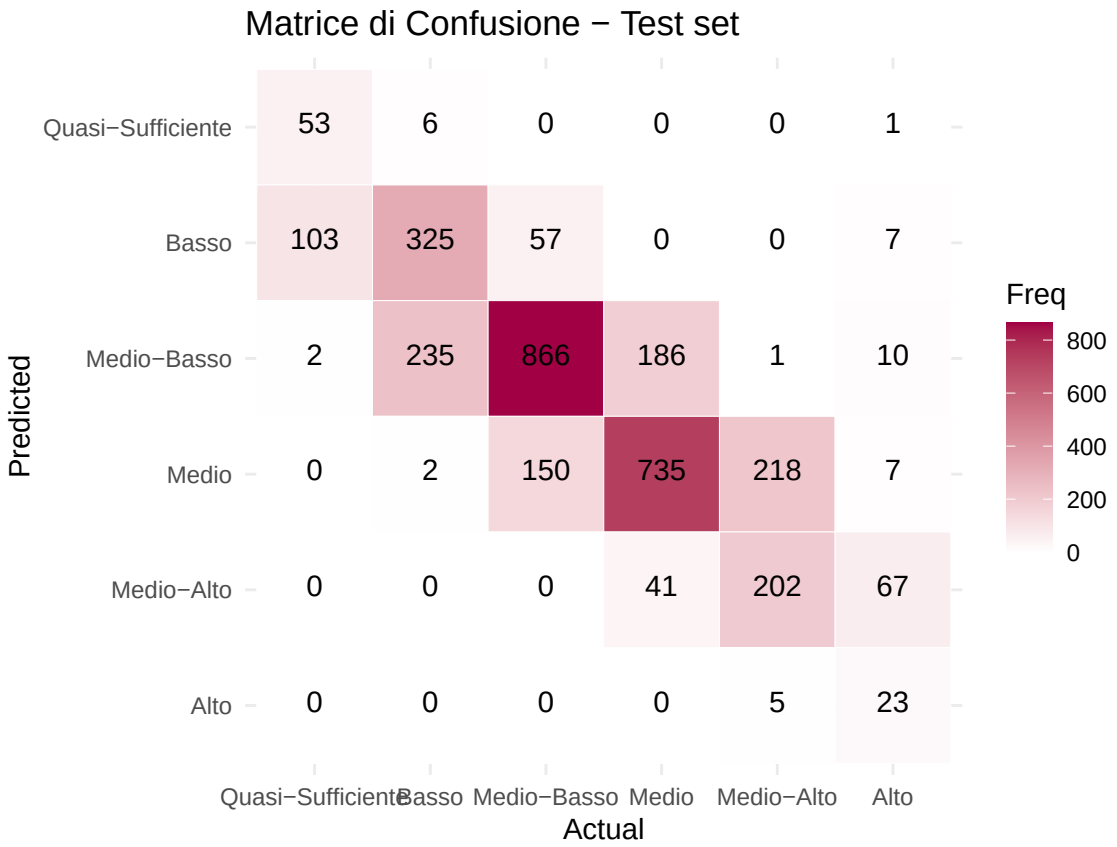


Per comprendere se l'introduzione di una redistribuzione dei pesi abbia effettivamente migliorato la capacità predittiva, vediamo come si comporta il modello con i dati di test.

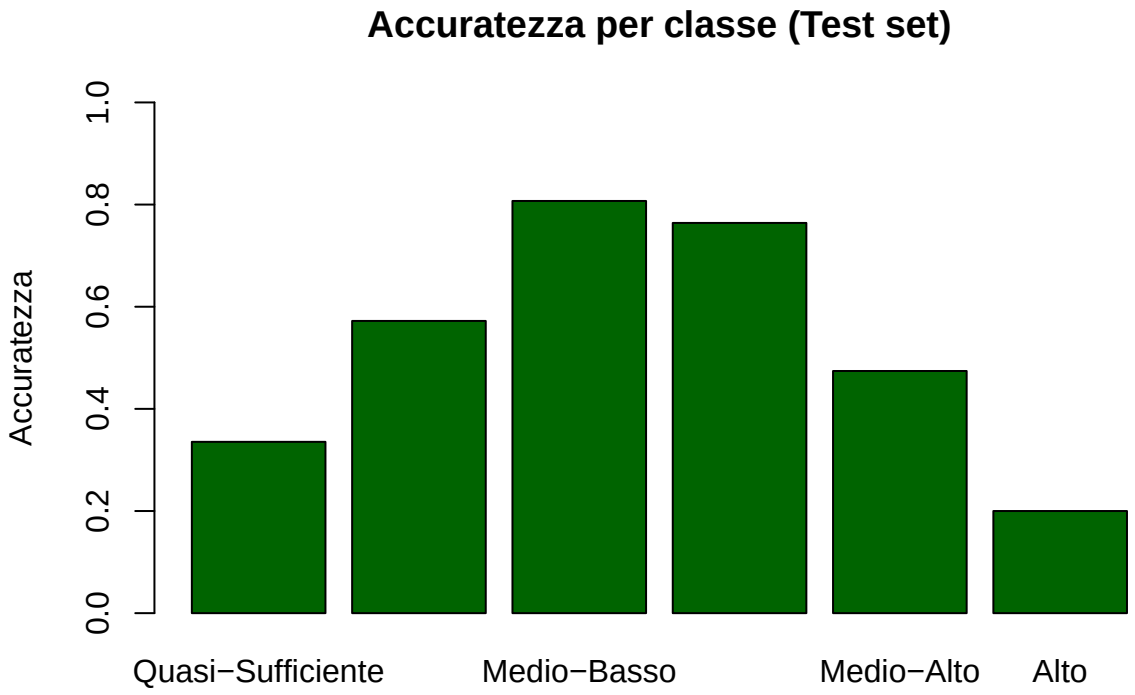
```
actuals <- test$Categorical_Exam_Score
rf_wp= predict(rf_weighted, newdata = test )
confusion_matrix_w <- table(Predicted = rf_wp, Actual = actuals)
print(confusion_matrix_w)
```

```
##                   Actual
## Predicted          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente                53     6           0     0          0    1
##   Basso                           103   325          57     0          0    7
##   Medio-Basso                       2   235         866   186          1   10
##   Medio                             0     2         150   735        218    7
##   Medio-Alto                        0     0           0    41        202   67
##   Alto                              0     0           0     0          5   23
```

```
conf_df2 <- as.data.frame(confusion_matrix_w)
ggplot(conf_df2, aes(x = Actual, y = Predicted)) +
  geom_tile(aes(fill = Freq), color = "white") +
  geom_text(aes(label = Freq), vjust = 0.3) +
  scale_fill_gradient(low = "white", high = "#A20045") +
  scale_y_discrete(limits = rev) +
  theme_minimal() +
  labs(title = "Matrice di Confusione - Test set") +
  coord_fixed() # Opzionale: mantiene le celle quadrate
```




```
accuratezza_classe2<- diag(prop.table(confusion_matrix_w, 2))
barplot(accuratezza_classe2,
        main = "Accuratezza per classe (Test set)",
        col = "darkgreen",
        ylab = "Accuratezza",
        ylim = c(0, 1))
```



Purtroppo, dall'analisi del grafico emerge che non si riscontra un miglioramento significativo nella classificazione delle classi minoritarie, nonostante l'introduzione dei pesi. Questo suggerisce che il semplice ribilanciamento dei pesi, pur utile, non è sufficiente da solo a risolvere le difficoltà legate allo sbilanciamento del dataset.

Data augmentation

In questo paragrafo ci siamo concentrati sul sottoinsieme di addestramento (train), dove sono presenti le categorie con il maggior numero di osservazioni, con l'obiettivo di incrementare le classi meno rappresentate. A tal fine, abbiamo sfruttato la libreria smotefamily, che mette a disposizione la funzione SMOTE. Lo scopo di questa procedura è verificare se, bilanciando le classi tramite algoritmi di over-sampling, è possibile migliorare le prestazioni predittive dei nostri modelli

Il codice applica la funzione SMOTE per generare nuovi esempi sintetici per le due classi meno rappresentate, "rare", indicate nella lista rare_classes. La funzione trasformando il problema in un'etichettatura binaria, selezionando solo le variabili numeriche e applicando l'algoritmo SMOTE. I campioni sintetici generati vengono successivamente aggiunti al dataset originale, con la pulizia delle colonne temporanee e la verifica della nuova distribuzione delle classi per assicurarsi che l'over-sampling abbia riequilibrato il dataseto quanto meno migliorato il problema .

```
table(train$Categorical_Exam_Score)
```

```
##
## Quasi-Sufficiente             Basso       Medio-Basso             Medio
##               158               568              1074               963
##        Medio-Alto              Alto
##               427               115
```

```r
# Quasi-Sufficiente" e "Alto" sono le classi con pochi esempi
rare_classes <- c("Quasi-Sufficiente", "Alto")

apply_smote_to_class <- function(data, class_target, rate = 2) {
  # Crea etichetta binaria
  data$binary_target <- ifelse(data$Categorical_Exam_Score == class_target,
                               1, 0)

  # SMOTE lavora solo su variabili numeriche → isoliamo features numeriche
  x_vars <- data %>% select(where(is.numeric))
  y_bin  <- data$binary_target

  # Applica SMOTE
  smote_out <- SMOTE(x_vars, y_bin, K = 12, dup_size = rate)

  # Recupera solo i sintetici generati (classe = 1)
  synthetic <- smote_out$syn_data %>%
    mutate(Categorical_Exam_Score = class_target)

  # Rimuove colonna target binaria
  synthetic <- synthetic %>% select(-class)

  return(synthetic)
}

# Applichiamo SMOTE a ciascuna classe rara
synthetics <- lapply(rare_classes, function(cl) {
  apply_smote_to_class(train, class_target = cl, rate = 2)
})

# Combiniamo i sintetici
synthetic_data <- bind_rows(synthetics)

# Unisci al train originale
train_augmented <- bind_rows(train, synthetic_data)
train_augmented$binary_target = NULL


train_augmented$Categorical_Exam_Score = as.factor(
  train_augmented$Categorical_Exam_Score
  )

# Controlla la nuova distribuzione
table(train_augmented$Categorical_Exam_Score)
```

```
##
##              Alto             Basso             Medio        Medio-Alto
##               345               568               963               427
##       Medio-Basso Quasi-Sufficiente
##              1074               474
```

Poiché la funzione SMOTE calcola nuovi valori solo per le variabili numeriche, lasciando vuoti i campi relativi alle variabili categoriali, si è deciso di riempire questi campi con il valore più comune della variabile categoriale per ciascuna classe. Questa scelta riduce la variabilità dei dati, ma, a seguito di valutazioni preliminari,

si è constatato che eliminare completamente le variabili categoriali dal dataset comportava un notevole peggioramento delle performance. Pertanto, si è preferito inserire i valori mancanti invece di rimuovere del tutto queste variabili, per mantenere la coerenza e la ricchezza informativa del dataset.

```
impute_categorical_na_by_class_mode <- function(data, class_col, rare_classes) {
  # Identifica colonne categoriali (escluse quelle già numeriche o il target)
  categorical_cols <- data %>% select(where(~is.factor(.) || is.character(.))) %>%
    select(-all_of(class_col)) %>% colnames()

  for (cat_col in categorical_cols) {
    for (rare_class in rare_classes) {
      # Subset dei dati per la classe rara
      subset_class <- data %>%
        filter(!!sym(class_col) == rare_class)

      # Calcola la moda ignorando gli NA
      mode_val <- subset_class %>%
        filter(!is.na(!!sym(cat_col))) %>%
        count(!!sym(cat_col), sort = TRUE) %>%
        slice(1) %>%
        pull(!!sym(cat_col))

      # Sostituisci NA con la moda solo per la classe rara corrente
      data <- data %>%
        mutate(!!sym(cat_col) := ifelse(
          is.na(!!sym(cat_col)) & (!!sym(class_col) == rare_class),
          mode_val,
          !!sym(cat_col)
        ))
    }
  }
  return(data)
}


train_augmented <- impute_categorical_na_by_class_mode(
  data = train_augmented,
  class_col = "Categorical_Exam_Score",
  rare_classes = rare_classes
)
```

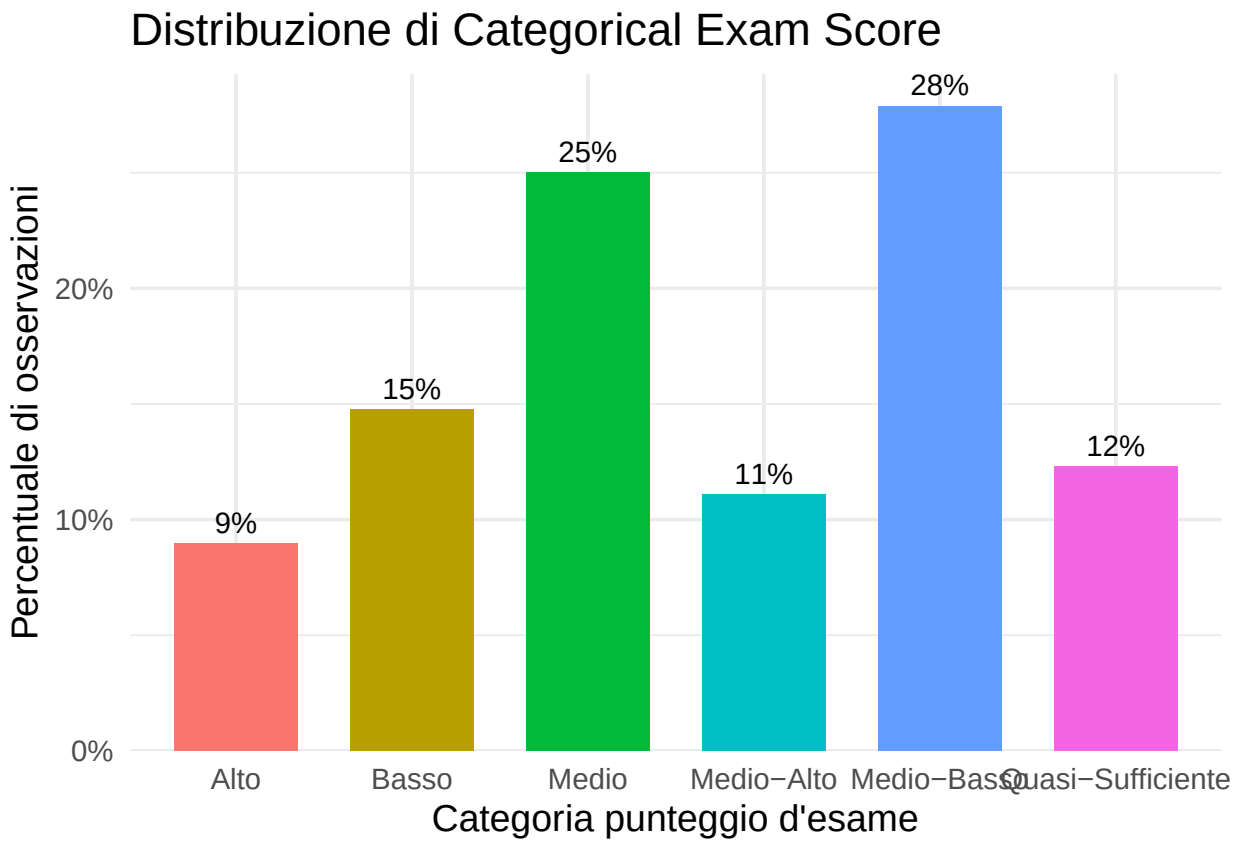
Per motivi di efficienza computazionale, le variabili categoriali presenti nel dataset aumentato nel codice precedente sono state codificate come indici corrispondenti ai livelli delle variabili categoriali stesse. Per rendere nuovamente il dataset facilmente interpretabile, il codice seguente converte questi indici nei rispettivi valori testuali, rendendo il dataset nuovamente leggibile e comprensibile.

```
levels_list <- lapply(train[categorical_vars], function(x) {
  # se sono factor mantieni i livelli, altrimenti estrai i valori unici
  if (is.factor(x)) levels(x) else unique(as.character(x))
})
names(levels_list) <- categorical_vars

# Decodifica in train_augmented gli indici numerici usando levels_list
for (var in categorical_vars) {
  train_augmented[[var]] <- factor(
```

```
    train_augmented[[var]],
    levels = seq_along(levels_list[[var]]),
    labels = levels_list[[var]]
  )
}
```

Con il seguente codice è possibile visualizzare la frequenza percentuale delle variabili nel dataset aumentato. Si osserva come l'algoritmo abbia incrementato la rappresentatività delle classi meno numerose, bilanciando meglio la distribuzione delle categorie.

```
ggplot(train_augmented, aes(x = Categorical_Exam_Score,
                fill = Categorical_Exam_Score)) +
  # barre con proporzione
  geom_bar(
    aes(y = after_stat(count) / sum(after_stat(count))),
    stat = "count",
    width = 0.7,
    show.legend = FALSE
  ) +
  # percentuali sopra le barre
  geom_text(
    aes(
      label = percent(after_stat(count) / sum(after_stat(count)), accuracy = 1),
      y     = after_stat(count) / sum(after_stat(count))
    ),
    stat = "count",
    vjust = -0.5
  ) +
  # scala y in percentuale e un po' di spazio in alto
  scale_y_continuous(
    labels = percent_format(accuracy = 1),
    expand = expansion(mult = c(0, 0.05))
  ) +
  labs(
    x     = "Categoria punteggio d'esame",
    y     = "Percentuale di osservazioni",
    title = "Distribuzione di Categorical Exam Score"
  ) +
  theme_minimal(base_size = 14)
```

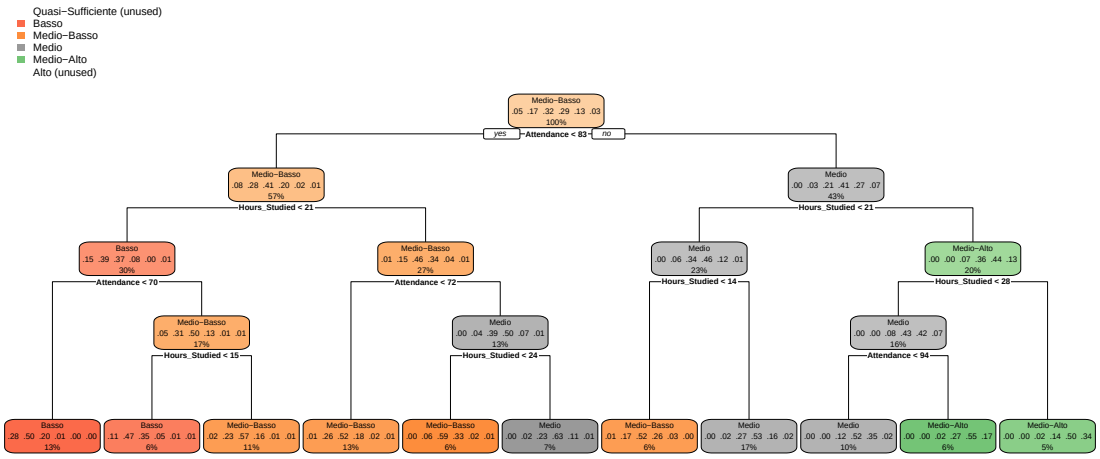


CART:

Dopo aver valutato modelli complessi come la Random Forest, anche con l'introduzione di pesi per il bilanciamento delle classi, si è deciso di analizzare un modello più semplice ma interpretabile: il CART (Classification and Regression Tree). Il modello CART si basa sulla costruzione di un singolo albero decisionale, ed è particolarmente utile in fase di interpretazione grazie alla sua struttura visiva. A differenza della Random Forest, che è un ensemble di molti alberi, il CART consente di comprendere in modo diretto quali variabili guidano le decisioni del modello e come vengono effettuati gli split. In questo contesto, il CART viene introdotto non come alternativa in termini di accuratezza, ma come strumento complementare.

```
mod0 <- rpart(Categorical_Exam_Score ~ ., data = train, method = "class")

# Visualizzazione dell'albero
rpart.plot::rpart.plot(mod0,main = "CART")
```

CART




```
# Predizione sul training set
pred_mod0 <- predict(mod0, type = "class", newdata = test)
summary(pred_mod0)
```

```
## Quasi-Sufficiente             Basso       Medio-Basso             Medio
##                 0               636              1170              1183
##        Medio-Alto              Alto
##               313                 0
```

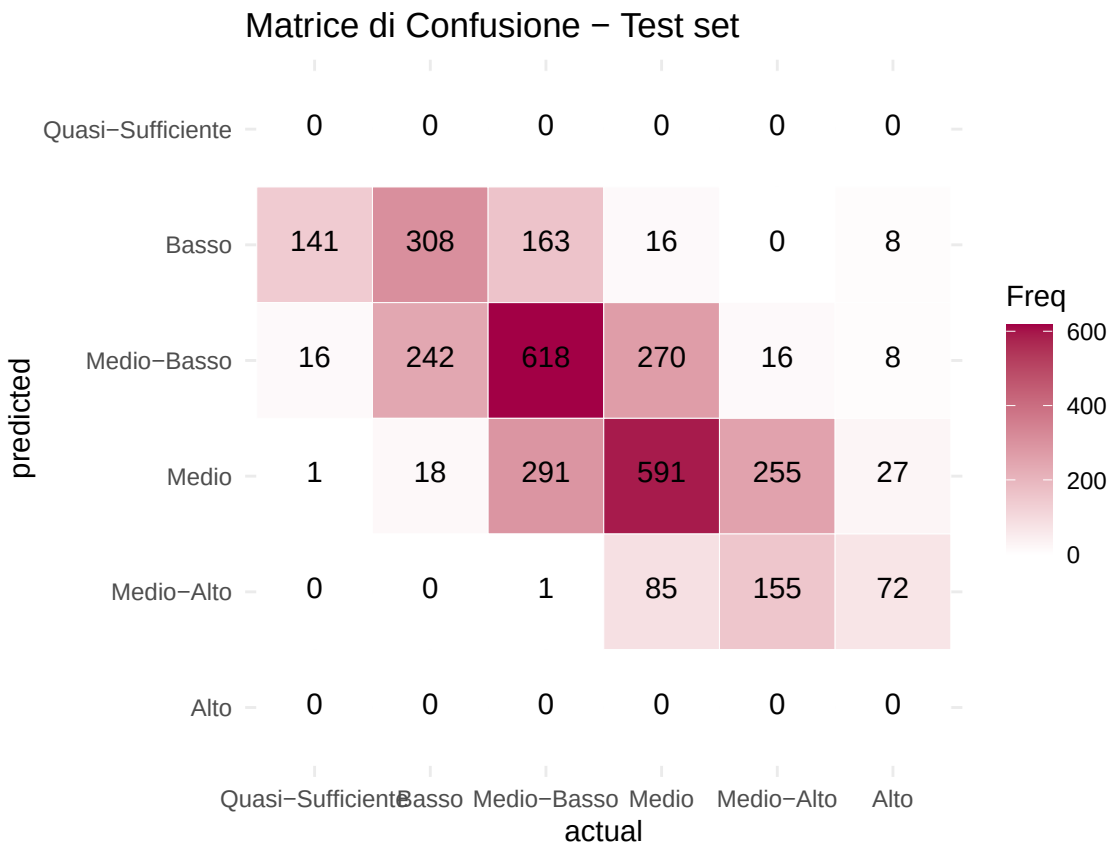
```
# Confusion matrix
conf_base <- table(predicted = pred_mod0, actual = test$Categorical_Exam_Score)
# Valutazione
caret::confusionMatrix(conf_base)
```

```
## Confusion Matrix and Statistics
##
##                  actual
## predicted          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente                 0     0           0     0          0    0
##   Basso                           141   308         163    16          0    8
##   Medio-Basso                      16   242         618   270         16    8
##   Medio                             1    18         291   591        255   27
##   Medio-Alto                        0     0           1    85        155   72
##   Alto                              0     0           0     0          0    0
##
## Overall Statistics
##
```

```
##                Accuracy : 0.5064
##                  95% CI : (0.4892, 0.5236)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3285
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                           0.00000      0.54225             0.5760
## Specificity                           1.00000      0.88003             0.7524
## Pos Pred Value                            NaN      0.48428             0.5282
## Neg Pred Value                        0.95215      0.90248             0.7866
## Prevalence                            0.04785      0.17202             0.3250
## Detection Rate                        0.00000      0.09328             0.1872
## Detection Prevalence                  0.00000      0.19261             0.3543
## Balanced Accuracy                     0.50000      0.71114             0.6642
##                      Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                0.6143           0.36385     0.00000
## Specificity                0.7470           0.94506     1.00000
## Pos Pred Value             0.4996           0.49521         NaN
## Neg Pred Value             0.8249           0.90933     0.96517
## Prevalence                 0.2913           0.12901     0.03483
## Detection Rate             0.1790           0.04694     0.00000
## Detection Prevalence       0.3583           0.09479     0.00000
## Balanced Accuracy          0.6807           0.65446     0.50000
```

La matrice di confusione ottenuta dal modello CART mostra evidenti difficoltà nel riconoscere correttamente le classi meno rappresentate. In particolare, la classe "alto" non viene mai predetta, mentre "quasi sufficiente" e "basso" sono frequentemente confuse. Il modello tende a collassare verso le classi più numerose, come "medio" e "medio-basso", segno che la struttura ad albero fatica a modellare le sottili differenze tra categorie. Questi limiti confermano la sensibilità del CART allo sbilanciamento delle classi e motivano l'uso di modelli più robusti, come la Random Forest, potenzialmente integrati con pesi di classe.

```r
conf_df3 <- as.data.frame(conf_base) %>%
  mutate(is_correct = predicted == actual)
ggplot(conf_df3, aes(x = actual, y = predicted)) +
  geom_tile(aes(fill = Freq), color = "white") +
  geom_text(aes(label = Freq), vjust = 0.3) +
  scale_fill_gradient(low = "white", high = "#A20045") +
  scale_y_discrete(limits = rev) +
  theme_minimal() +
  labs(title = "Matrice di Confusione - Test set") +
  coord_fixed() # Opzionale: mantiene le celle quadrate
```



Pre-processing output:

Tra le tecniche di data-augmentation, abbiamo provato anche a raggruppare le classi in modo da ridimensionare la variabilità fra di esse. L'idea è quella di fare delle comparazioni in termini di performance, in quanto, con classi meno separate ci si aspetterebbe una migliore capacità predittiva del modello a causa di un miglior bilanciamento fra le classi. Questo dovrebbe aiutare il modello stesso a classificare e a riconoscere più facilmente le unità statistiche. Per avere una comparazione corretta dei risultati rieseguiamo le procedure discusse fin ora utilizzando il dataset 2 precedentemente generato.

```
# Fit del modello
mtry <- 6
rf_2 <- randomForest(Categorical_Exam_Score ~ .,
                     data = train_2,
                     ntree = 1000,
                     mtry = mtry)
rf_2
```

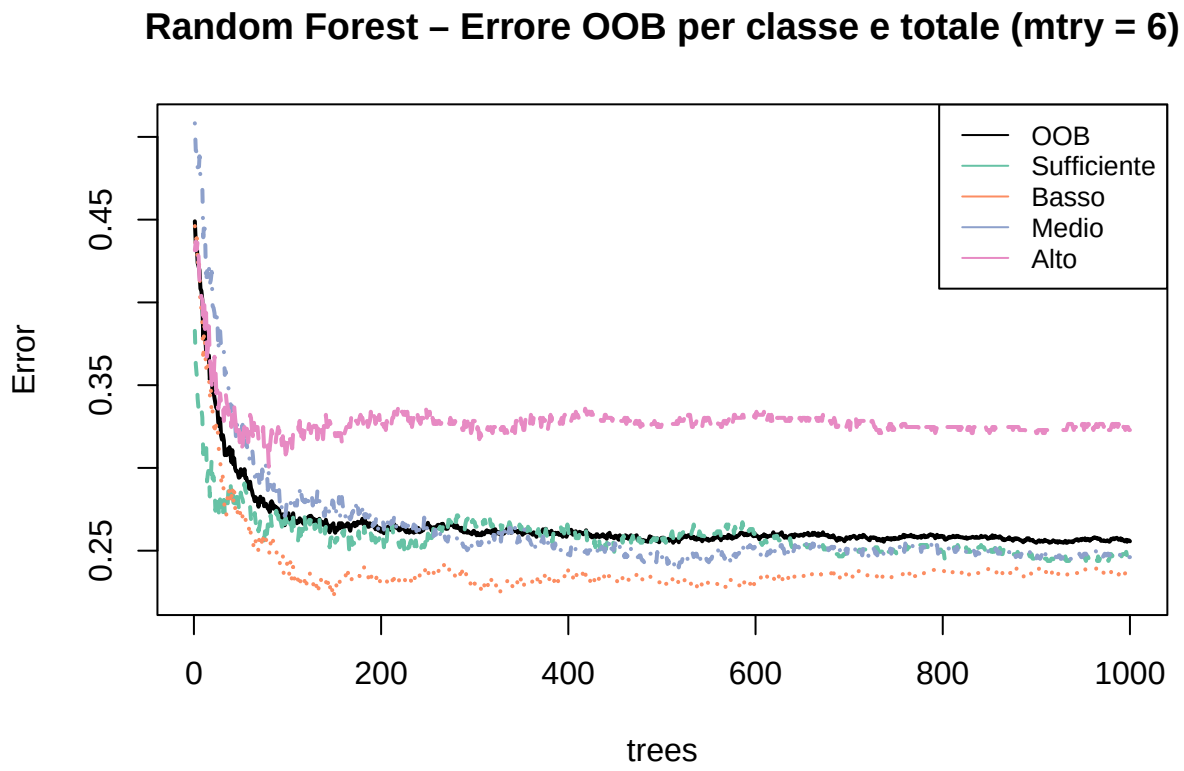
```
##
## Call:
##  randomForest(formula = Categorical_Exam_Score ~ ., data = train_2,      ntree = 1000, mtry = mtry)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 25.57%
## Confusion matrix:
##             Sufficiente Basso Medio Alto class.error
```

```
## Sufficiente          546   180     0    0   0.2479339
## Basso                 88   821   165    0   0.2355680
## Medio                  0   179   726   58   0.2461059
## Alto                   4    12   159  367   0.3228782
```

```r
# Verifica: 5 colonne nell'err.rate (4 classi + OOB)
colnames(rf_2$err.rate)
```

```
## [1] "OOB"         "Sufficiente" "Basso"       "Medio"       "Alto"
```

```r
# [1] "OOB" "Sufficiente" "Basso" "Medio" "Alto"

# Palette: nero per OOB, colori pastello per le classi
colori <- c("black", brewer.pal(4, "Set2"))

# Plot
plot(rf_2,
     col = colori,
     lwd = 2,
     main = "Random Forest - Errore OOB per classe e totale (mtry = 6)")

# Legenda
legend("topright",
       legend = colnames(rf_2$err.rate),
       col = colori,
       lty = 1,
       cex = 0.8)
```

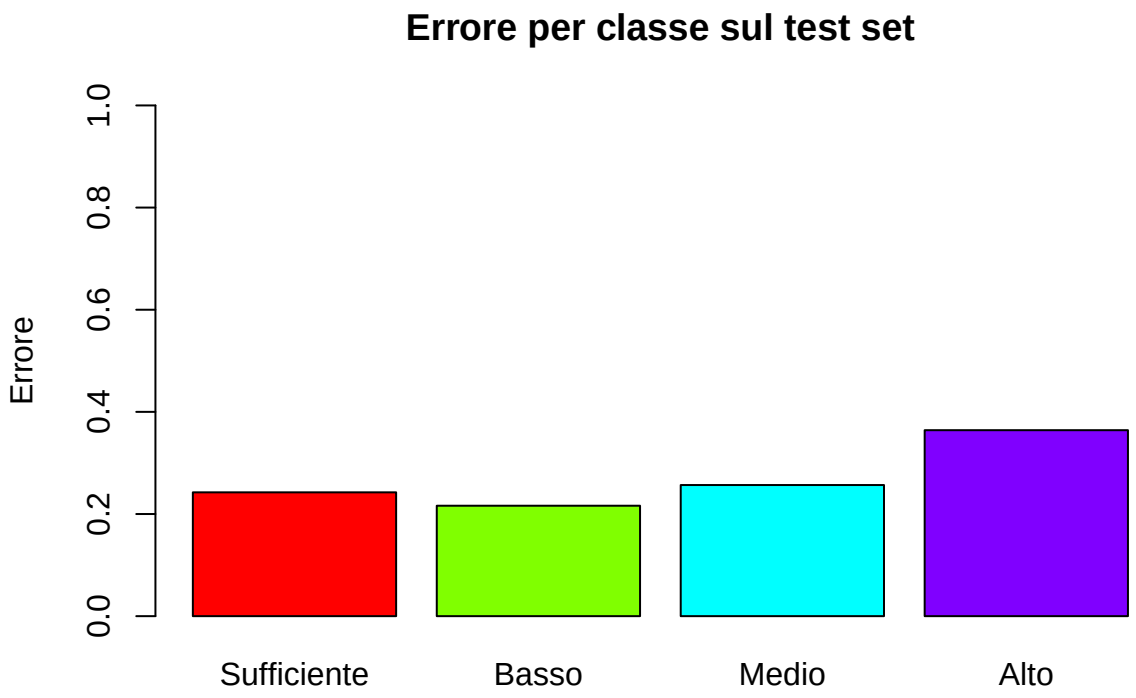


Dai risultati ottenuti si osserva una chiara riduzione dell'errore OOB dopo il raggruppamento delle classi. Una conseguenza rilevante di questa aggregazione è che il numero di simulazioni (ovvero di alberi) necessario per stabilizzare l'errore OOB si è ridotto rispetto alla configurazione iniziale, pur mantenendo invariato il valore del parametro mtry. Sebbene il bilanciamento delle classi sia stato ottenuto attraverso la ricodifica dei punteggi, permane una maggiore difficoltà del modello nel classificare correttamente la classe "alto", che anche dopo il raggruppamento resta la meno rappresentata. Il suo tasso di errore di classificazione si conferma infatti il più elevato tra tutte le categorie. Infine, confrontando la matrice di confusione e i livelli di accuratezza per classe sul test set, emerge un miglioramento netto della performance complessiva: l'accuratezza del modello risulta aumentata. Questo dimostra come una corretta aggregazione delle categorie possa contribuire a semplificare il compito di classificazione e migliorare la capacità predittiva dell'algoritmo.

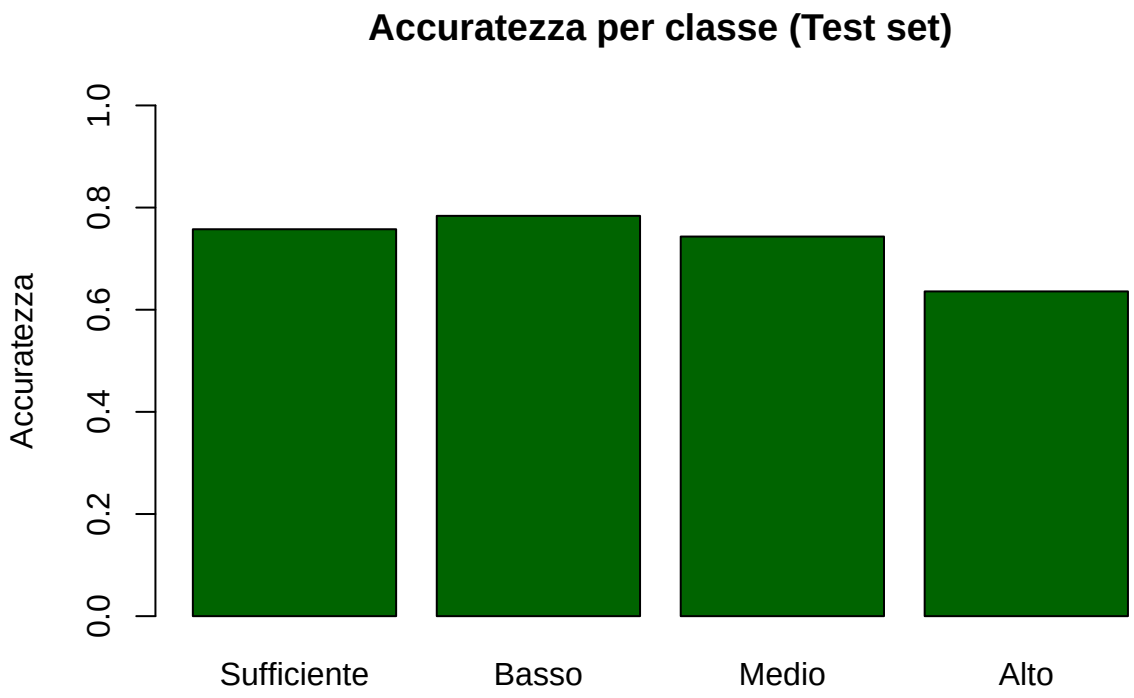
```
# Predizione sul test set
rf_2p <- predict(rf_2, newdata = test_2)

# Tabella di frequenza predetta
table(rf_2p)
```

```
## rf_2p
## Sufficiente       Basso       Medio        Alto
##         638        1215        1047         402
```

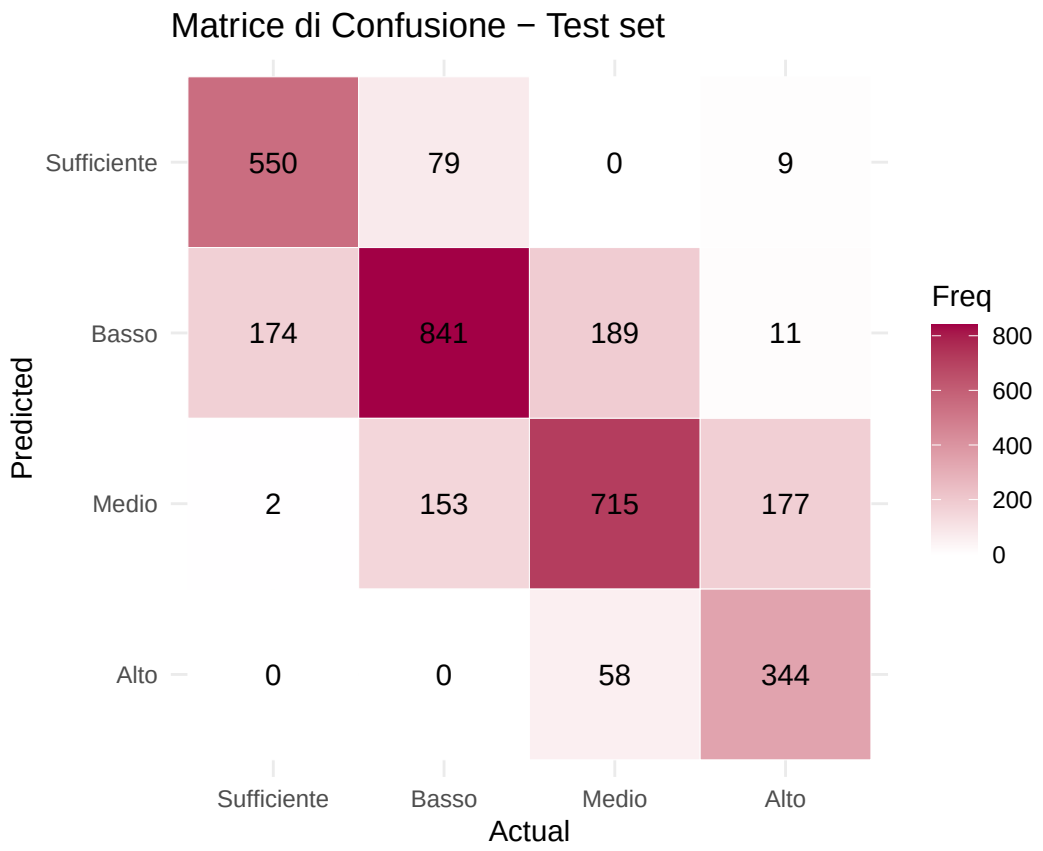
```
# Confusion matrix (Predetto vs Osservato)
tab2 <- table(Predicted = rf_2p, Actual = test_2$Categorical_Exam_Score)

# Errore per classe (1 - accuracy per colonna)
err_by_class2 <- 1 - diag(prop.table(tab2, 2))

# Grafico barre errore per classe
barplot(err_by_class2,
        col = rainbow(length(err_by_class2)),
        main = "Errore per classe sul test set",
        ylab = "Errore",
        ylim = c(0, 1))
```




```
accuratezza_classe2 <- diag(prop.table(tab2, 2))
barplot(accuratezza_classe2,
        main = "Accuratezza per classe (Test set)",
        col = "darkgreen",
        ylab = "Accuratezza",
        ylim = c(0, 1))
```



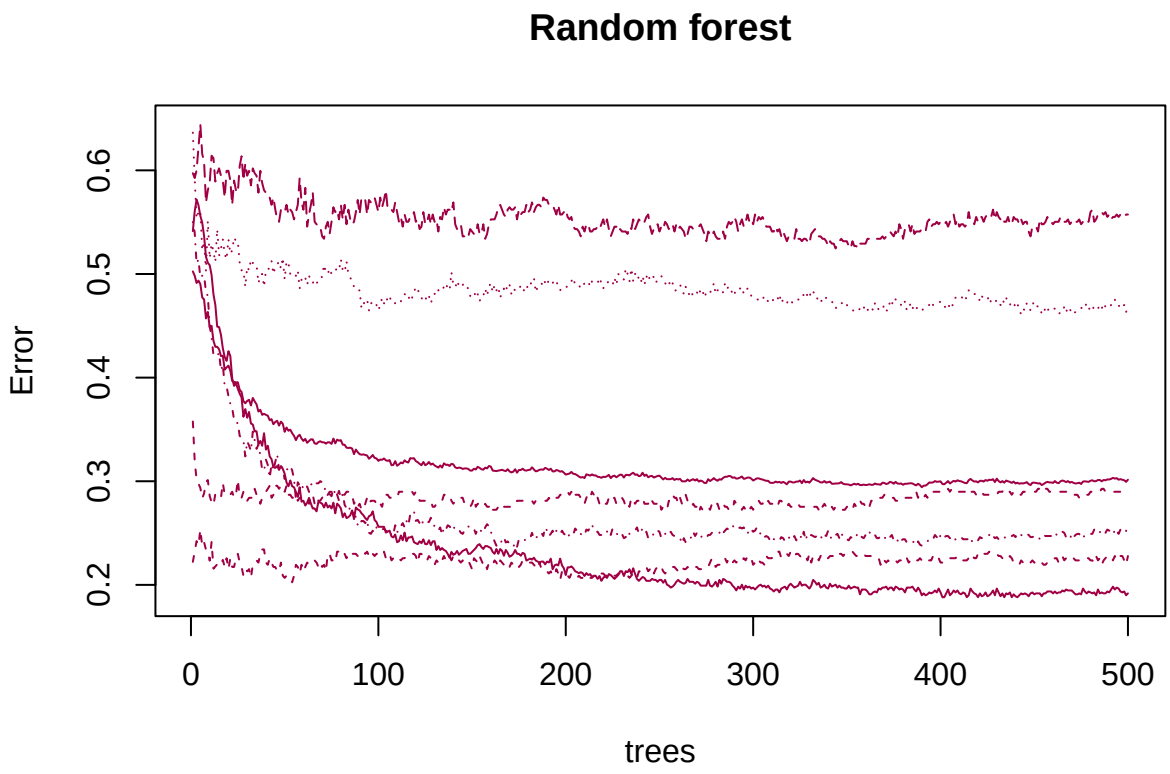

```r
# Heatmap della confusion matrix
conf_df2=as.data.frame(tab2)
conf_df2 <- conf_df2 %>%
  mutate(is_correct = Predicted == Actual)

# Heatmap migliorata
ggplot(conf_df2, aes(x = Actual, y = Predicted)) +
  geom_tile(aes(fill = Freq), color = "white") +
  geom_text(aes(label = Freq), vjust = 0.5) +
  scale_fill_gradient(low = "white", high = "#A20045") +
  scale_y_discrete(limits = rev) +
  theme_minimal() +
  labs(title = "Matrice di Confusione - Test set") +
  coord_fixed()
```



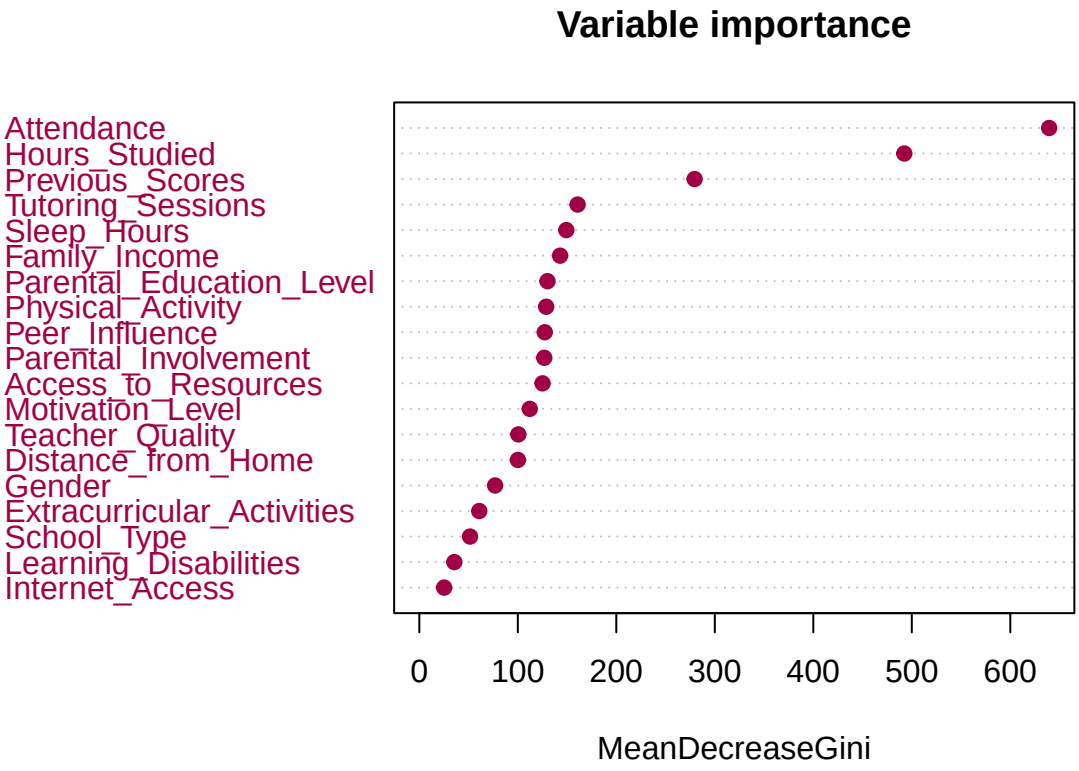
Random Forest sul dataset augmented

Di seguito presentiamo i risultati ottenuti fittando un algoritmo di random forest sul dataset augmented:

```
p <- NCOL(train_augmented) - 1


set.seed(11)

rf_1_aug <- randomForest(Categorical_Exam_Score ~ . , data = train_augmented)
plot(rf_1_aug, col="#A20045", main="Random forest")
```




```
varImpPlot(rf_1_aug, main="Variable importance", pch = 19, color="#A20045")
```



Come ci potevamo aspettare nel coso delle prediizoni sul dataset di train, il modello sembra migliorato.

```
pred = predict(rf_1_aug)
confusionMatrix(pred, train_augmented$Categorical_Exam_Score)
```

```
## Confusion Matrix and Statistics
##
##                    Reference
## Prediction          Alto Basso Medio Medio-Alto Medio-Basso Quasi-Sufficiente
##   Alto               244     0     0          9           2                 0
##   Basso                5   301     1          0          54               103
##   Medio               12     1   722        225         149                 0
##   Medio-Alto          74     0    31        189           0                 0
##   Medio-Basso         10   257   209          4         868                 5
##   Quasi-Sufficiente    0     9     0          0           1               366
##
## Overall Statistics
##
##                Accuracy : 0.6985
##                  95% CI : (0.6837, 0.713)
##     No Information Rate : 0.2789
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6163
##
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: Alto Class: Basso Class: Medio Class: Medio-Alto
## Sensitivity              0.70725      0.52993       0.7497           0.44262
## Specificity              0.99686      0.95035       0.8660           0.96933
## Pos Pred Value           0.95686      0.64871       0.6510           0.64286
## Neg Pred Value           0.97191      0.92117       0.9121           0.93309
## Prevalence               0.08959      0.14749       0.2501           0.11088
## Detection Rate           0.06336      0.07816       0.1875           0.04908
## Detection Prevalence     0.06622      0.12049       0.2880           0.07634
## Balanced Accuracy        0.85205      0.74014       0.8079           0.70598
##                      Class: Medio-Basso Class: Quasi-Sufficiente
## Sensitivity                      0.8082                  0.77215
## Specificity                      0.8254                  0.99704
## Pos Pred Value                   0.6415                  0.97340
## Neg Pred Value                   0.9175                  0.96892
## Prevalence                       0.2789                  0.12308
## Detection Rate                   0.2254                  0.09504
## Detection Prevalence             0.3513                  0.09764
## Balanced Accuracy                0.8168                  0.88460
```

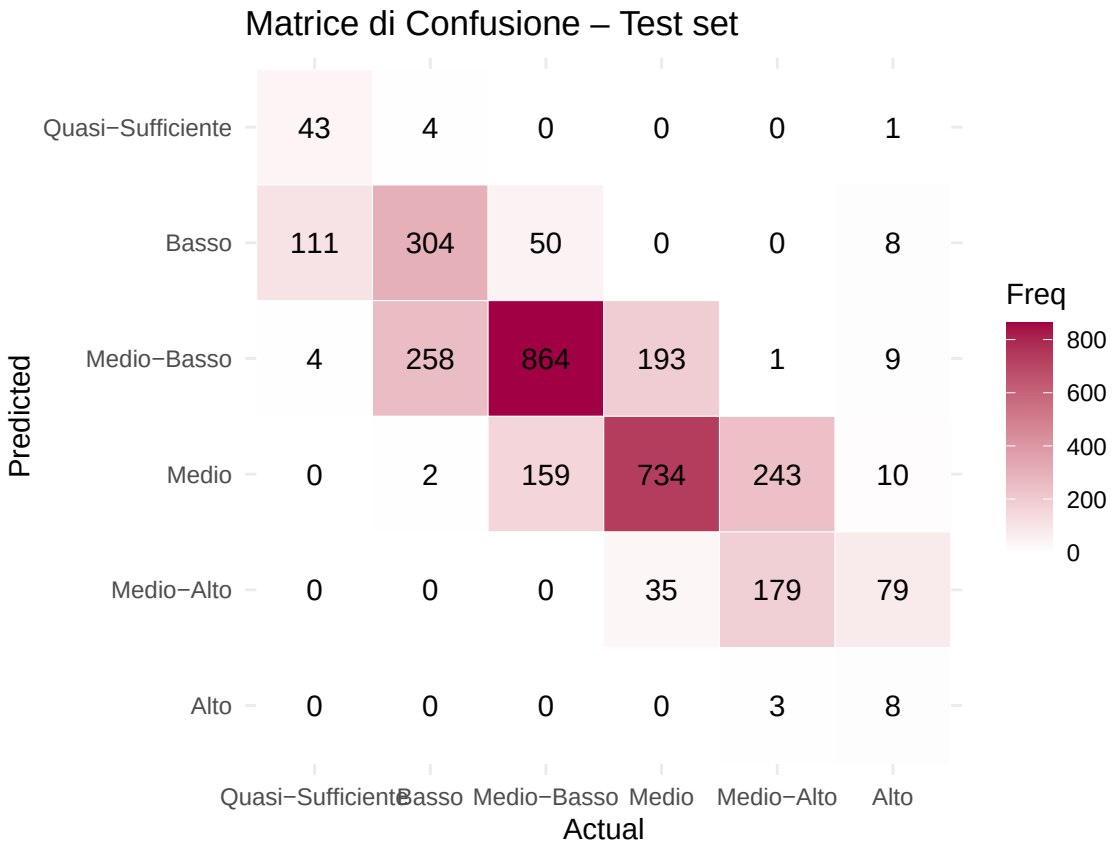
Dalla matrice di confusione e dal barplot dell'accuratezza emerge chiaramente che il modello addestrato sui dati aumentati non solo non migliora le prestazioni sul test set, ma mostra addirittura un peggioramento rispetto ai modelli analoghi esaminati in precedenza, segnalando una ridotta capacità di generalizzazione.

```
rf_preds <- predict(rf_1_aug, newdata = test)
rf_preds <- factor(rf_preds,
                   levels = levels(test$Categorical_Exam_Score))

cm_caret <- confusionMatrix(rf_preds, test$Categorical_Exam_Score)
print(cm_caret)
```

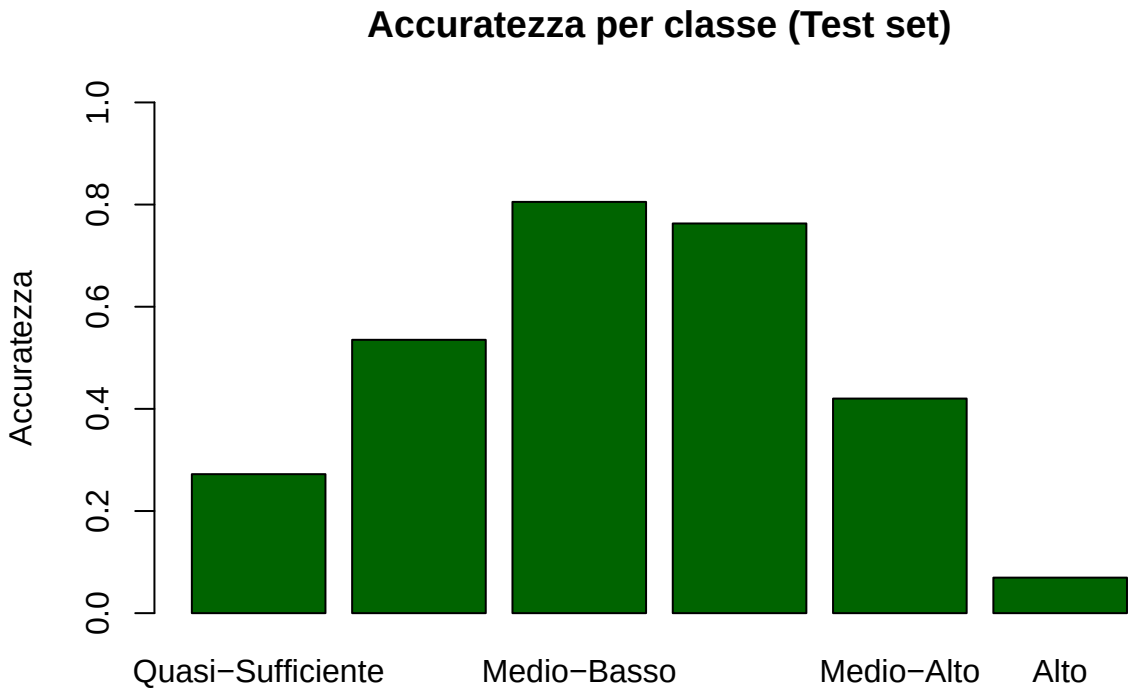
```
## Confusion Matrix and Statistics
##
##                    Reference
## Prediction          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente                43     4           0     0          0    1
##   Basso                           111   304          50     0          0    8
##   Medio-Basso                       4   258         864   193          1    9
##   Medio                             0     2         159   734        243   10
##   Medio-Alto                        0     0           0    35        179   79
##   Alto                              0     0           0     0          3    8
##
## Overall Statistics
##
##                Accuracy : 0.6457
##                  95% CI : (0.6291, 0.662)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5153
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                           0.27215      0.53521             0.8052
## Specificity                           0.99841      0.93819             0.7914
## Pos Pred Value                        0.89583      0.64271             0.6501
## Neg Pred Value                        0.96466      0.90668             0.8941
## Prevalence                            0.04785      0.17202             0.3250
## Detection Rate                        0.01302      0.09207             0.2617
## Detection Prevalence                  0.01454      0.14325             0.4025
## Balanced Accuracy                     0.63528      0.73670             0.7983
##                      Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                0.7630           0.42019    0.069565
## Specificity                0.8231           0.96036    0.999059
## Pos Pred Value             0.6394           0.61092    0.727273
## Neg Pred Value             0.8942           0.91791    0.967487
## Prevalence                 0.2913           0.12901    0.034827
## Detection Rate             0.2223           0.05421    0.002423
## Detection Prevalence       0.3477           0.08873    0.003331
## Balanced Accuracy          0.7930           0.69027    0.534312
```

```r
conf_mat <- table(Predicted = rf_preds, Actual = test$Categorical_Exam_Score)
conf_df  <- as.data.frame(conf_mat)

conf_df <- conf_df %>%
  mutate(
    Actual    = factor(Actual,    levels = levels(test$Categorical_Exam_Score)),
    Predicted = factor(Predicted, levels = levels(test$Categorical_Exam_Score))
  )


ggplot(conf_df, aes(x = Actual, y = Predicted, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), vjust = 0.5) +
  scale_fill_gradient(low = "white", high = "#A20045") +
  scale_x_discrete(limits = levels(test$Categorical_Exam_Score)) +
  scale_y_discrete(limits = rev(levels(test$Categorical_Exam_Score))) +
  theme_minimal() +
  labs(title = "Matrice di Confusione - Test set") +
  coord_fixed()
```



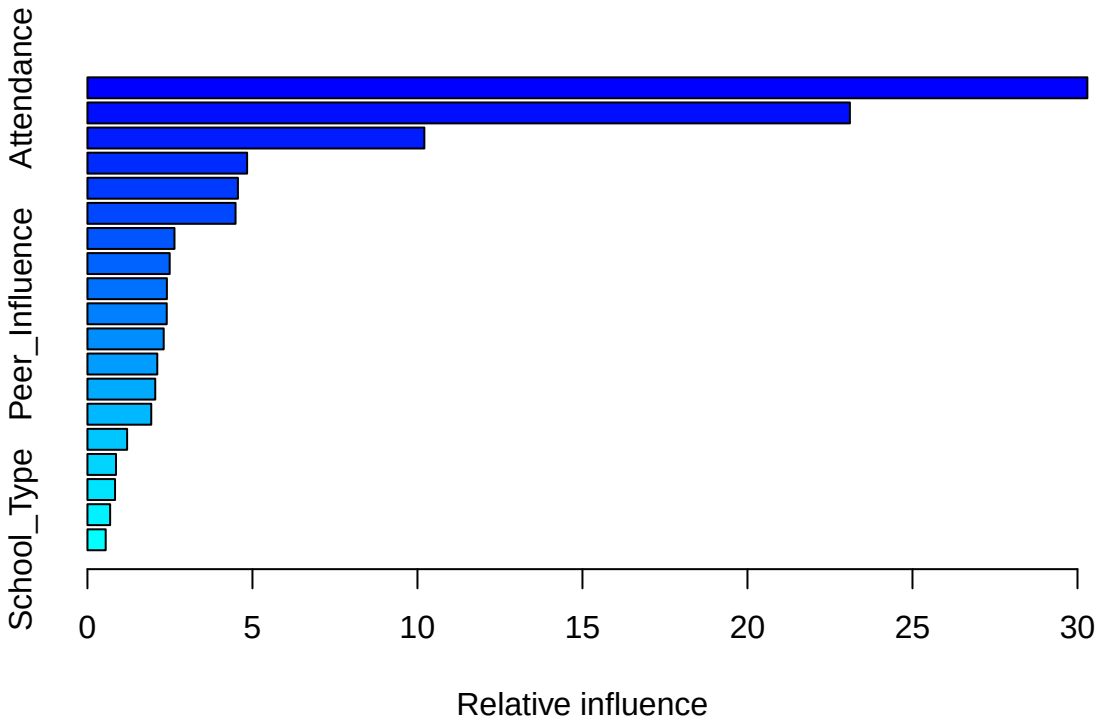

```r
accuratezza_classe <- diag(prop.table(conf_mat, 2))
accuratezza_classe <- accuratezza_classe[levels(test$Categorical_Exam_Score)]

barplot(
  accuratezza_classe,
  names.arg = levels(test$Categorical_Exam_Score),
  main      = "Accuratezza per classe (Test set)",
  ylab      = "Accuratezza",
  ylim      = c(0, 1),
  col       = "darkgreen"
)
```



Boosting

Vediamo ora come si comporta l'algoritmo di Boosting sul dateaset 1 contenente 6 classi. Si è deciso di testare solamente questo algoritmo sul dataset con più classi quindi con una maggiore difficoltà nella capacità di previsione in quanto ci si aspetta che questo sia più performante e che quindi possa darci molte più informazione nella capacità di previsione con una maggior numero di classi da prevedere. Inizialmente si è proceduti con un modello base con 2000 alberi, con massimo 4 rami per albero e un learning rate indicato dalla variabile di shrinkage paria a 0.1. Il parametro di shrinkage non è stato cambiato per tutta la trattazione in quanto 0.1 rappresenta il valore massimo da cui partire e già da questo l'algoritmo impiega molto tempo per essere eseguito, per cui effettuare un fine tuning risulterebbe troppo difficoltoso. Effettuando il summary del modello possiamo osservare sia la tabella con i valori di importanza relativi a ogni variabile sia un grafico a barre ottenuto da esse. Come visto anche per il modello precedente al primo posso possiamo trovare la variabile Attendance, e a seguire il numero di ore studiate.

```
set.seed(123)

boost.1 <- gbm(Categorical_Exam_Score ~ ., data = train,
               distribution = "multinomial", n.trees = 2000,
               interaction.depth = 4, shrinkage=0.1)
summary(boost.1)
```



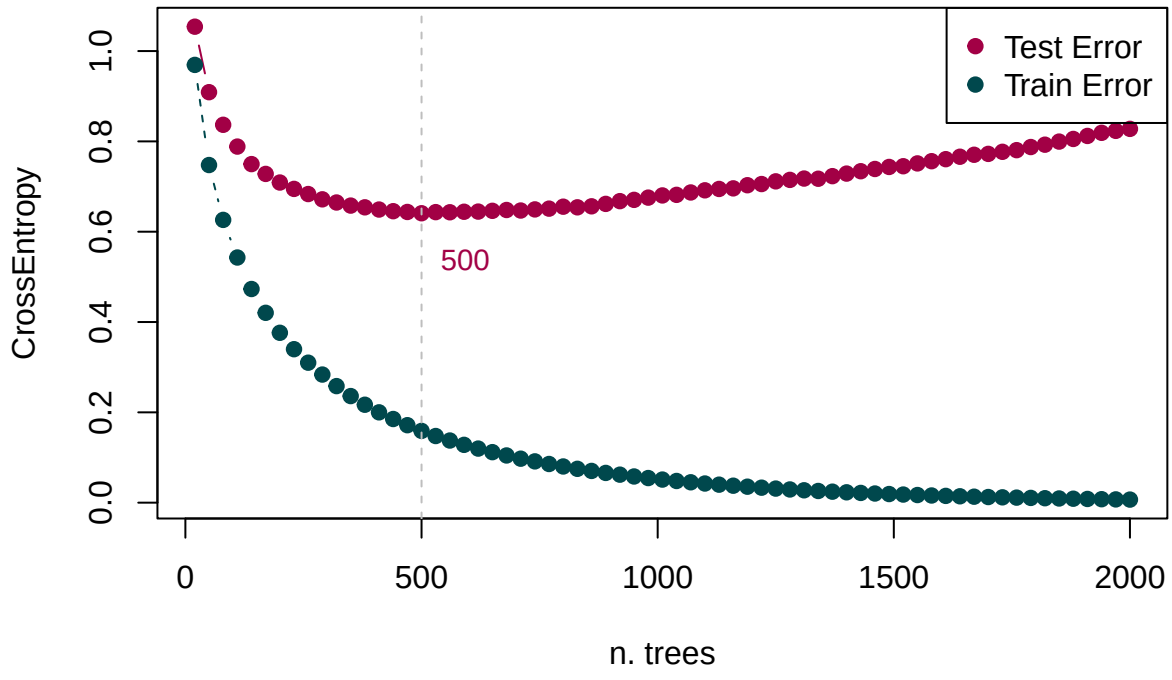

```
##                                                  var    rel.inf
## Attendance                                Attendance 30.2991804
## Hours_Studied                          Hours_Studied 23.1036236
## Previous_Scores                      Previous_Scores 10.2086155
## Tutoring_Sessions                  Tutoring_Sessions  4.8388459
## Access_to_Resources              Access_to_Resources  4.5601296
## Parental_Involvement            Parental_Involvement  4.4876980
## Parental_Education_Level    Parental_Education_Level  2.6370557
## Family_Income                          Family_Income  2.4920223
## Motivation_Level                    Motivation_Level  2.4075285
## Peer_Influence                        Peer_Influence  2.4034831
## Distance_from_Home                Distance_from_Home  2.3112820
## Sleep_Hours                              Sleep_Hours  2.1161358
## Physical_Activity                  Physical_Activity  2.0551103
## Teacher_Quality                      Teacher_Quality  1.9339373
## Extracurricular_Activities Extracurricular_Activities  1.2022736
## Learning_Disabilities          Learning_Disabilities  0.8657044
## Internet_Access                      Internet_Access  0.8364042
## Gender                                        Gender  0.6875161
## School_Type                              School_Type  0.5534537
```

Dal primo modello è stato possibile calcolare l'errore sia sul dataset di training che su quello di test. Per farlo, è stato necessario trasformare i risultati previsti, originariamente espressi come logaritmi delle probabilità, in probabilità vere e proprie comprese tra 0 e 1 per ciascuna classe, assicurandosi che la loro somma fosse pari a 1. A tal fine è stata implementata la funzione softmax. Poiché le etichette erano rappresentate in formato one-hot encoding, per il calcolo dell'errore è stata utilizzata la funzione di perdita cross entropy, confrontando i valori predetti con quelli attesi. Per ciascun numero di alberi (da 20 a 2000, con incrementi di

30), è stata calcolata la media dell'errore su tutte le previsioni, sia per il dataset di training che per quello di test. I risultati sono stati poi visualizzati nel grafico riportato alla fine del blocco di codice. Dal grafico si osserva che l'errore sul test set (curva rossa) raggiunge un minimo attorno ai 500 alberi, per poi aumentare progressivamente. Al contrario, l'errore sul training set (curva nera) mostra un andamento decrescente e monotono, segno che il modello continua ad apprendere anche oltre i 500 alberi. Questo comportamento evidenzia un chiaro caso di overfitting: il modello si adatta sempre meglio ai dati di training, ma a scapito della sua capacità di generalizzazione sui dati di test.

```
# Calcola l'errore di tipo CrossEntropy
calculate_rowwise_error <- function(real_matrix, predicted_matrix) {
  real_matrix <- as.matrix(real_matrix)
  predicted_matrix <- as.matrix(predicted_matrix)
  errors <- -rowSums(real_matrix * log(predicted_matrix))
  return(errors)
}

# Funzione softmax
softmax <- function(logits) {
  stable_logits <- logits - max(logits) # Stabilità numerica
  exp_logits <- exp(stable_logits)
  return(exp_logits / sum(exp_logits))
}

n.trees.seq <- seq(from = 20, to = 2000, by = 30)

test_matrix_onehot <- model.matrix(~ test$Categorical_Exam_Score - 1)
train_matrix_onehot <- model.matrix(~ train$Categorical_Exam_Score - 1)

Yhat_logits_array_test <- predict(boost.1,
                            newdata = test,
                            n.trees = n.trees.seq
                            )

Yhat_logits_array_train <- predict(boost.1, n.trees = n.trees.seq)



error_matrix_test <- matrix(NA, nrow = 1, ncol = length(n.trees.seq),
                        dimnames = list("MeanCrossEntropy",
                                        paste0("nTrees_", n.trees.seq)))
error_matrix_train <- matrix(NA, nrow = 1, ncol = length(n.trees.seq),
                        dimnames = list("MeanCrossEntropy",
                                        paste0("nTrees_", n.trees.seq)))

for (i in 1:length(n.trees.seq)) {
  current_n_trees <- n.trees.seq[i]

  current_logits_test <- Yhat_logits_array_test[,,i]
  current_probs_test <- t(apply(current_logits_test, 1, softmax))

  rowwise_errors_test <- calculate_rowwise_error(test_matrix_onehot,
                                                 current_probs_test)
  mean_error_test <- mean(rowwise_errors_test)
  error_matrix_test[1, i] <- mean_error_test
```

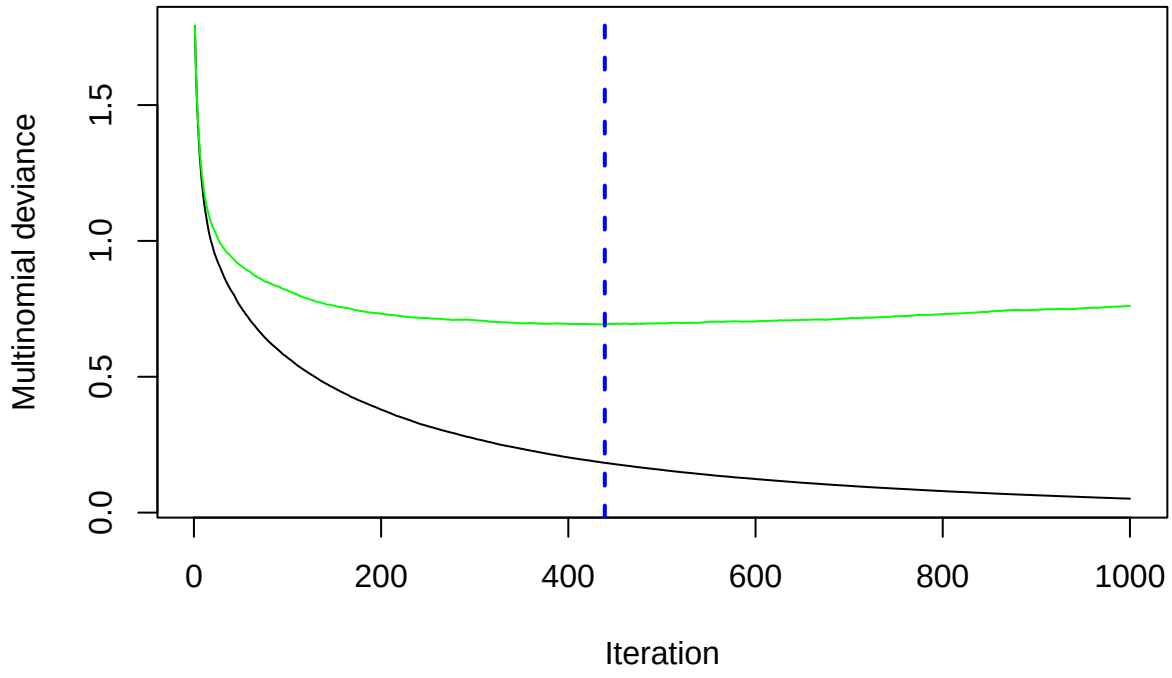
```
  current_logits_train <- Yhat_logits_array_train[,,i]
  current_probs_train <- t(apply(current_logits_train, 1, softmax))

  rowwise_errors_train <- calculate_rowwise_error(train_matrix_onehot,
                                                  current_probs_train)
  mean_error_train <- mean(rowwise_errors_train)
  error_matrix_train[1, i] <- mean_error_train

}


best_n_trees <- n.trees.seq[which.min(error_matrix_test[1, ])]

matplot(n.trees.seq, cbind(error_matrix_test[1, ], error_matrix_train[1, ]),
        pch=19 , col=c("#A20045","#00484D"),
        type="b",ylab="CrossEntropy",xlab="n. trees")
abline(v = best_n_trees, col = "gray", lty = "dashed", lwd = 1)
text(x = best_n_trees, y = mean(par("usr")[3:4]), labels = best_n_trees,
     pos = 4, col = "#A20045", cex = 0.9)
legend("topright",legend=c("Test Error","Train Error"),pch=19,
       col=c("#A20045","#00484D"))
```



Successivamente è stato addestrato un nuovo modello con parametri analoghi a quelli del primo, ad eccezione del numero di alberi, ridotto a 1000. Su questo secondo modello di boosting è stata utilizzata la funzione perf della libreria gbm, che consente di individuare graficamente il numero ottimale di alberi tramite validazione incrociata, minimizzando l'errore. Il risultato è un grafico simile a quello precedente, ma in questo caso viene

evidenziato il punto corrispondente al numero ottimale di alberi. Tale valore, salvato nella variabile best.nt, è pari a 439 molto vicino a quanto osservato in precedenza utilizzando il dataset di test. Ricordiamo che il valore di 439 alberi è ottenuto attraverso la cross validation per tanto è meno affidabile di quello precedente basato sul dataset di test in quanto la suddivisione per la valutazione avviene sullo stesso dataset di train.

```
set.seed(123)
boost.2 <- gbm(Categorical_Exam_Score ~ ., data = train,
               distribution = "multinomial", n.trees = 1000,
               interaction.depth = 4, shrinkage=0.1, cv.folds = 5)

best.nt = gbm.perf(boost.2, method = "cv", plot.it = TRUE)
```




```
best.nt
```

```
## [1] 439
```

Successivamente è stata avviata una procedura di fine-tuning dei parametri, con l'obiettivo di individuare non solo il numero ottimale di alberi, ma anche il valore ideale del parametro depth. Utilizzando il codice mostrato, sono stati testati tutti i valori di depth compresi tra 1 e 7. Per ciascun valore, sono stati calcolati sia l'errore sul dataset di training e su quello di test, sia il numero ottimale di alberi tramite validazione incrociata. Il risultato è una matrice contenente tutte le combinazioni testate, con i relativi valori di errore e numero di alberi ottimale. Da questa matrice è stato possibile individuare la combinazione di parametri che minimizza l'errore sul dataset di test. In basso è riportata la tabella con i risultati ottenuti.

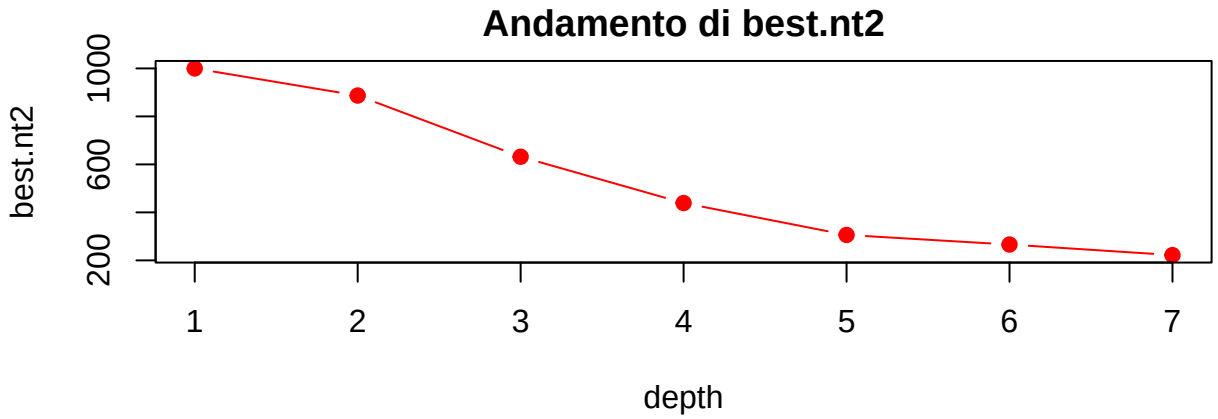
```
myd <- 1:7
test_matrix_onehot <- model.matrix(~ test$Categorical_Exam_Score - 1)
train_matrix_onehot <- model.matrix(~ train$Categorical_Exam_Score - 1)
```

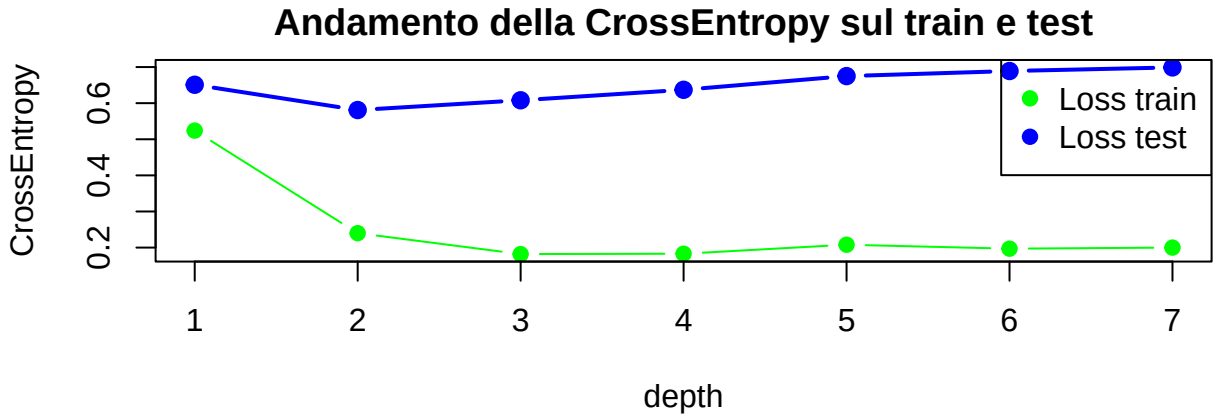
```
myEval <- sapply(myd , function(x) {
  set.seed(123)
  boost.3 <- gbm(Categorical_Exam_Score ~ ., data = train,
                 distribution = "multinomial", n.trees = 1000,
                 interaction.depth = x, shrinkage=0.1, cv.folds = 5)

  # Numero ottimale di alberi
  best.nt2 <- gbm.perf(boost.3, method = "cv", plot.it = FALSE)


  Yhat_test <- predict(boost.3, newdata = test,n.trees = best.nt2)
  Yhat_test = t(apply(Yhat_test, 1, softmax))

  Yhat_train <- predict(boost.3, n.trees = best.nt2)
  Yhat_train = t(apply(Yhat_train, 1, softmax))

  ### Crossentropy
  loss_train = mean(calculate_rowwise_error(train_matrix_onehot, Yhat_train))
  loss_test = mean(calculate_rowwise_error(test_matrix_onehot, Yhat_test))

  return(c(as.integer(best.nt2), round(loss_train,3), round(loss_test,3)))}
  )

myEval
```

```
##          [,1]    [,2]    [,3]    [,4]    [,5]    [,6]    [,7]
## [1,] 1000.000 887.000 632.000 439.000 306.000 266.000 222.000
## [2,]    0.524   0.240   0.182   0.183   0.208   0.197   0.200
## [3,]    0.651   0.581   0.608   0.637   0.675   0.689   0.699
```

Per una migliore visualizzazione dei risultati, sono stati realizzati due grafici. Il primo mostra come varia il numero ottimale di alberi determinato tramite validazione incrociata in funzione del parametro depth. Mentre il secondo mostra l'errore del modello calcolato sul train e test. In basso sono riportati i valori ottimali di depth e di numero di alberi che minimizzano l'errore.

```
myEval_invertita <- t(myEval)

par(mfrow = c(2, 1), mar = c(4, 4, 2, 1))

# Primo grafico: best.nt2
plot(1:7, myEval_invertita[,1], type = "b", pch = 19, col = "red",
     xlab = "depth", ylab = "best.nt2",
     main = "Andamento di best.nt2")

plot(1:7, myEval_invertita[,2], type = "b", pch = 19, col = "green",
     xlab = "depth", ylab = "CrossEntropy",
     main = "Andamento della CrossEntropy sul train e test",
     ylim = range(c(myEval_invertita[,2], myEval_invertita[,3])))
lines(1:7, myEval_invertita[,3], type = "b", pch = 19, col = "blue", lwd = 2)

# Legenda
legend("topright",
       legend = c("Loss train", "Loss test"),
       col    = c("green", "blue"),
       pch    = 19)
```






```
indice_minimo <- which.min(myEval_invertita[,3])
best_nt= myEval_invertita[,1][indice_minimo]
best_depth =  indice_minimo
best_depth
```

```
## [1] 2
```

```
best_nt
```

```
## [1] 887
```

Si può osservare come il numero ottimale di alberi tenda a diminuire all'aumentare del parametro depth. Questo comportamento è probabilmente dovuto al fatto che valori elevati di depth aumentano la complessità del modello, rendendolo più soggetto a overfitting. Di conseguenza, un numero inferiore di alberi risulta sufficiente (e talvolta necessario) per contrastare questo effetto e mantenere la generalizzazione. Il secondo grafico mostra l'andamento dell'errore del modello al variare del parametro depth, distinguendo tra il dataset di test (curva blu) e quello di training (curva verde). Per ogni valore di depth, è stato utilizzato il corrispondente numero ottimale di alberi trovato in precedenza. Dal grafico emerge che l'errore sul dataset di test raggiunge il minimo per depth = 2, per poi aumentare, indicando un peggioramento della capacità di generalizzazione. Al contrario, come già osservato nei casi precedenti, la loss sul dataset di training decresce in modo monotono, suggerendo un crescente overfitting con l'aumentare della profondità. Tuttavia, è importante notare che mentre il depth è stato scelto in base alla minima loss sul dataset di test, il numero ottimale di alberi è stato determinato tramite validazione incrociata. Questo significa che i due valori ottimali non sono perfettamente coerenti tra loro, e il numero di alberi potrebbe non essere il migliore possibile in corrispondenza del depth ottimale. Per evitare di appesantire ulteriormente l'analisi e i tempi di calcolo, si è deciso di non ripetere la procedura di ottimizzazione del numero di alberi specificamente per il valore ottimale di depth. Una tale

operazione avrebbe probabilmente portato a una stima più accurata, ma a fronte di una maggiore complessità computazionale.

Utilizzando i parametri ottimali identificati in precedenza, è stato addestrato il modello finale di boosting, denominato boost.final.

```
set.seed(123)
boost.final <- gbm(Categorical_Exam_Score ~ ., data = train,
                   distribution = "multinomial", n.trees = best_nt,
                   interaction.depth = best_depth, shrinkage=0.1, cv.folds = 5)
```

Da questo modello è stata ricavata la matrice di confusione sia sul dataset di train che di test, e sia altri parametri di interesse statistico come l'accuracy, Specificity e la Balanced Accuracy. Dai risulatati possiamo osservare come il modella abbia un valore di accuracy molto più elevato sul dataset di train che su quello di test, anche se su quest ultimo raggiunge comunque un accuratezza di quasi l'80%, indicando una discreta capacità predittiva nonostante l'overfitting. Per quanto riguarda Specificity e la Balanced Accuracy queste risultano molto variabili fra le classi. Nel caso della classe quasi sufficiente i valori sembrano discreti mentre per la classe Altro queste assumono valore abbastanza bassi.

Dal modello sono state estratte le matrici di confusione e calcolate le principali metriche: accuracy, specificity e balanced accuracy, sia sul training set sia sul test set. Vediamo alcune considerazioni:

- Training set:
  - Accuracy complessiva: 96,58%
  - Balanced accuracy per classe: da 98,70% (Quasi-Sufficiente) a 88,68% (Alto)
  - Specificity molto elevate per tutte le classi (ad es. 99,94% per Quasi-Sufficiente, 99,97% per Alto)
- Test set:
  - Accuracy complessiva: 79,92%
  - Balanced accuracy per classe: varia da 82,31% (Quasi-Sufficiente) a 67,66% (Alto)
  - Specificity alta anche in test (ad es. 99,43% per Quasi-Sufficiente, 98,81% per Alto)

Nonostante si possa notare dell'overfitting sui dati di train, il modello mantiene quasi l'80% di accuratezza sui dati di test, confermando una discreta capacità predittiva. Tuttavia, le metriche di balanced accuracy e specificity mostrano una forte variabilità tra le classi: La classe Quasi-Sufficiente presenta un'ottima balanced accuracy in training (98,70%), che scende a 82,31% in test, con specificity vicina al 99%.

La classe Alto è invece la più critica: nonostante specificity elevata (98,81%), la sensibilità sui dati di test è solo 36,52%, portando la balanced accuracy al 67,66%.

```
# Effettua le previsioni sul set di training
predictions_train_probs <- predict(boost.final, newdata = train,
                                   n.trees = best_nt, type = "response")
predictions_train <- apply(predictions_train_probs, 1, which.max)
predicted_classes_train <- levels(
  train$Categorical_Exam_Score)[predictions_train]

# Effettua le previsioni sul set di test
predictions_test_probs <- predict(boost.final, newdata = test,
                                  n.trees = best_nt, type = "response")
predictions_test <- apply(predictions_test_probs, 1, which.max)
predicted_classes_test <- levels(test$Categorical_Exam_Score)[predictions_test]

# Confusion matrix sul set di training
confusion_matrix_train <- confusionMatrix(
  factor(predicted_classes_train, levels = levels(train$Categorical_Exam_Score)),
  train$Categorical_Exam_Score)
print("Confusion Matrix - Training Set:")
```

```
## [1] "Confusion Matrix - Training Set:"
```

```
print(confusion_matrix_train)
```

```
## Confusion Matrix and Statistics
##
##                    Reference
## Prediction          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente               154     1           0     0          0    1
##   Basso                             4   545           3     0          0    4
##   Medio-Basso                       0    22        1057    16          0    6
##   Medio                             0     0          14   945         24    5
##   Medio-Alto                        0     0           0     2        402   10
##   Alto                              0     0           0     0          1   89
##
## Overall Statistics
##
##                Accuracy : 0.9658
##                  95% CI : (0.959, 0.9717)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9548
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                           0.97468       0.9595             0.9842
## Specificity                           0.99936       0.9960             0.9803
## Pos Pred Value                        0.98718       0.9802             0.9600
## Neg Pred Value                        0.99873       0.9916             0.9923
## Prevalence                            0.04781       0.1719             0.3250
## Detection Rate                        0.04660       0.1649             0.3198
## Detection Prevalence                  0.04720       0.1682             0.3331
## Balanced Accuracy                     0.98702       0.9777             0.9822
##                      Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                0.9813            0.9415     0.77391
## Specificity                0.9816            0.9958     0.99969
## Pos Pred Value             0.9565            0.9710     0.98889
## Neg Pred Value             0.9922            0.9914     0.99191
## Prevalence                 0.2914            0.1292     0.03480
## Detection Rate             0.2859            0.1216     0.02693
## Detection Prevalence       0.2989            0.1253     0.02723
## Balanced Accuracy          0.9815            0.9686     0.88680
```

```
# Calcola l'accuracy sul set di test
confusion_matrix_test <- confusionMatrix(
  factor(predicted_classes_test, levels = levels(test$Categorical_Exam_Score)),
  test$Categorical_Exam_Score)
print("Confusion Matrix - Test Set:")
```

```
## [1] "Confusion Matrix - Test Set:"
```

```r
print(confusion_matrix_test)
```

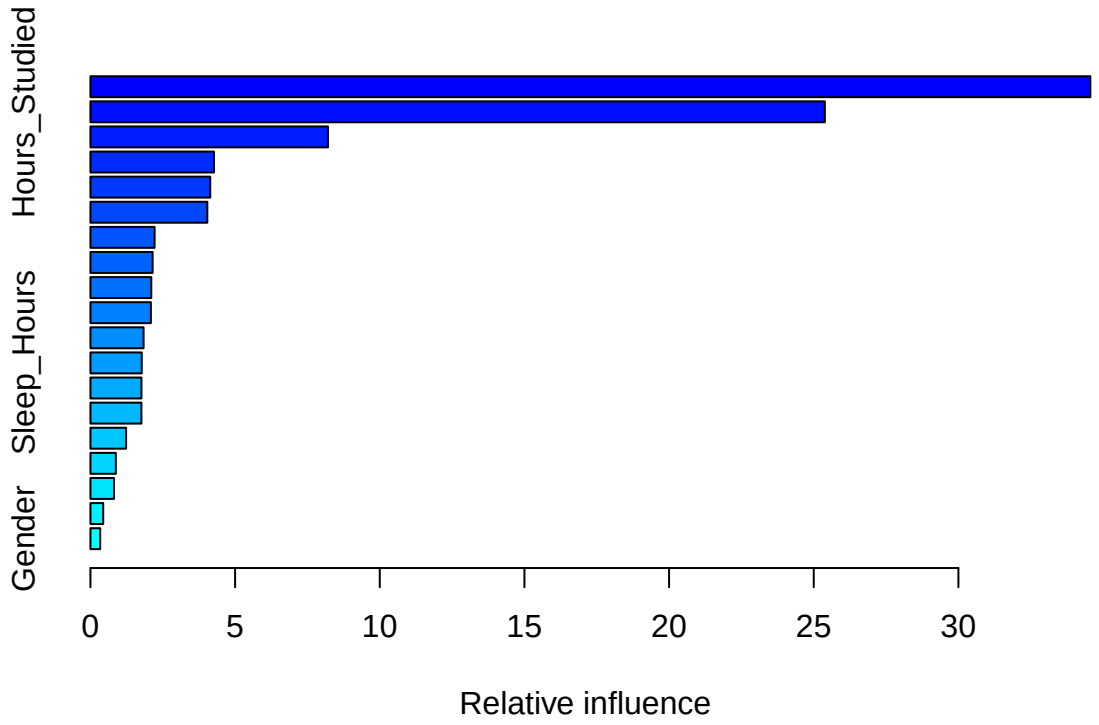
```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction          Quasi-Sufficiente Basso Medio-Basso Medio Medio-Alto Alto
##   Quasi-Sufficiente               103    14           1     0          0    3
##   Basso                            55   447          38     0          0    6
##   Medio-Basso                       0   107         948   101          0   11
##   Medio                             0     0          81   820        119    3
##   Medio-Alto                        0     0           0    36        279   50
##   Alto                              0     0           5     5         28   42
##
## Overall Statistics
##
##                Accuracy : 0.7992
##                  95% CI : (0.7851, 0.8128)
##     No Information Rate : 0.325
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7321
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Quasi-Sufficiente Class: Basso Class: Medio-Basso
## Sensitivity                           0.65190       0.7870             0.8835
## Specificity                           0.99427       0.9638             0.9017
## Pos Pred Value                        0.85124       0.8187             0.8123
## Neg Pred Value                        0.98271       0.9561             0.9415
## Prevalence                            0.04785       0.1720             0.3250
## Detection Rate                        0.03119       0.1354             0.2871
## Detection Prevalence                  0.03664       0.1654             0.3534
## Balanced Accuracy                     0.82309       0.8754             0.8926
##                      Class: Medio Class: Medio-Alto Class: Alto
## Sensitivity                0.8524           0.65493     0.36522
## Specificity                0.9132           0.97010     0.98808
## Pos Pred Value             0.8016           0.76438     0.52500
## Neg Pred Value             0.9377           0.94995     0.97734
## Prevalence                 0.2913           0.12901     0.03483
## Detection Rate             0.2483           0.08449     0.01272
## Detection Prevalence       0.3098           0.11054     0.02423
## Balanced Accuracy          0.8828           0.81251     0.67665
```

Effettuando il summary del modello possiamo osservare l'importanza che il modello ha dato a ciascuna variabile utilizzata. Si può notare che rispetto a prima è crescita l'importanza delle variabili Attendance e Hours_Studied rispetto alla altre. Tenendo conto di queste considerazioni unite a quelle precedenti si può pensare che la difficoltà del modello nel determinare le classi sembra sia da attribuire ai pochi dati nelle classi più lontane dalla media e sia dalla dipendenza lineare che vige tra le variabili più importanti utilizzate dal modello e Exam_Score.

Il summary del modello mette in luce l'importanza attribuita a ciascuna variabile: in particolare, Attendance e Hours_Studied hanno guadagnato peso rispetto alle altre feature e ai valori visti in precedenza. Alla luce di queste evidenze e dei risultati precedenti, si può ipotizzare che le maggiori difficoltà del modello nel classificare

correttamente le classi “più estreme” dipendano da due fattori principali: - Scarsa rappresentatività dei dati per le categorie lontane dalla media, che rende il modello meno accurato nel riconoscerle. - Forte correlazione lineare tra le variabili più influenti (Attendance e Hours_Studied) e l’obiettivo (Exam_Score), elemento che può complicare l’apprendimento di pattern distintivi soprattutto nelle classi meno frequenti.

```
summary(boost.final)
```




```
##                                                  var    rel.inf
## Attendance                                Attendance 34.5603677
## Hours_Studied                          Hours_Studied 25.3830110
## Previous_Scores                      Previous_Scores  8.2101842
## Tutoring_Sessions                  Tutoring_Sessions  4.2694557
## Access_to_Resources              Access_to_Resources  4.1411103
## Parental_Involvement            Parental_Involvement  4.0396447
## Family_Income                          Family_Income  2.2180240
## Parental_Education_Level    Parental_Education_Level  2.1493356
## Peer_Influence                        Peer_Influence  2.1032345
## Distance_from_Home                Distance_from_Home  2.0893412
## Motivation_Level                    Motivation_Level  1.8356807
## Sleep_Hours                              Sleep_Hours  1.7736258
## Physical_Activity                  Physical_Activity  1.7635919
## Teacher_Quality                      Teacher_Quality  1.7612849
## Learning_Disabilities          Learning_Disabilities  1.2322316
## Extracurricular_Activities Extracurricular_Activities  0.8777931
## Internet_Access                      Internet_Access  0.8136140
## School_Type                          School_Type  0.4414407
## Gender                                    Gender  0.3370282
```

Conclusioni

Dall'analisi condotta emergono alcuni elementi chiave. In primo luogo, le variabili che maggiormente influenzano le prestazioni dei modelli sono le ore di studio settimanali, la percentuale di frequenza alle lezioni e i punteggi ottenuti negli esami precedenti; seguono, con un peso più contenuto, fattori quali le sessioni di tutoraggio e le ore sonno.

Per quanto riguarda i modelli Random Forest, senza l'uso di pesi associati a ciascuna classe si riscontra una marcata difficoltà nel riconoscere adeguatamente le classi meno rappresentate, con un'evidente sottostima delle categorie estreme. L'introduzione dei pesi migliora sensibilmente la capacità di classificare queste classi meno rappresentate, anche se non elimina del tutto il problema. Inoltre, passando a una classificazione in quattro fasce, il modello diventa più stabile, l'errore OOB si riduce e si osserva un miglioramento dell'accuratezza. Al contrario, il riequilibrio tramite SMOTE non si è tradotto in un miglioramento delle prestazioni sul dataset di test; anzi, l'ampliamento del campione ha peggiorato la capacità predittiva complessiva.

Il modello CART, seppure apprezzabile per la sua semplicità interpretativa, si è mostrato meno efficace rispetto agli altri algoritmi, tendendo a concentrare le predizioni sulle classi centrali e a trascurare quelle estreme.

L'algoritmo di Boosting ha dimostrato una migliore capacità predittiva, raggiungendo un'accuratezza dell'80% sul dataset di test e mostrando un'ottima capacità di generalizzazione una volta ottimizzati gli iperparametri.

In sintesi, il modello che si è rivelato più promettente nella nostra analisi è il modello di Boosting. Le strategie di gestione dello sbilanciamento, seppur teoricamente efficaci, non hanno apportato benefici significativi nel nostro contesto. Mentre, rimangono da considerare le limitazioni legate alle dipendenze tra variabili, in particolare, modelli come CART soffrono quando si tratta di catturare relazioni lineari o prossime alla linearità, come evidenziato nel nostro caso di studio.