Predire la Sopravvivenza sul Titanic con gli Alberi Decisionali

Elia Mercatanti

Università degli Studi di Firenze
Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Magistrale in Informatica - Data Science

Contest MASL, Dicembre 2020

L'Affondamento del Titanic

- Il 15 aprile 1912, durante il suo viaggio inaugurale, considerato "inaffondabile", affondò dopo essersi scontrato con un iceberg.
- Scarse scialuppe di salvataggio, provocando la morte di 1502 su 2224 passeggeri e membri dell'equipaggio.
- Sembra che alcuni gruppi di persone fossero più propensi a sopravvivere rispetto ad altri.

Scopo del Progetto

- Rispondere alla domanda: "Che tipo di persone avevano maggiori probabilità di sopravvivere?".
- Predire quali passeggeri del Titanic sono sopravvissuti o meno.
- Uso degli alberi di decisione in R. Utilizzando diversi pacchetti e confrontando i risultati ottenuti.

Dataset

I dati sono stati suddivisi in due gruppi:

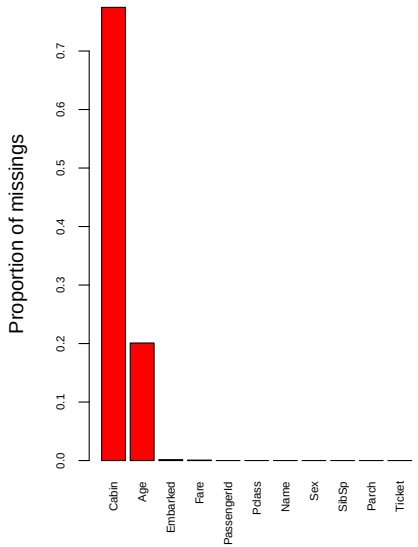
- Set di **training** (*titanic-train.csv*). Contiene la variabile "*Survived*", variabile di risposta. Per addestrare e valutare i modelli, diviso appositamente.
- Set di **test** (*titanic-test.csv*). Mancante della variabile "*Survived*". Per analisi del dataset.
- **Osservazioni**: 891 (train set) + 418 (test set).
- **Variabili**: 12 in totale che descrivono alcune caratteristiche dei passeggeri del Titanic.

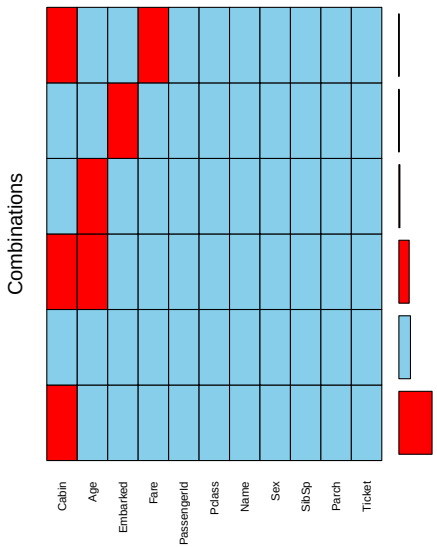
PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925		S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S

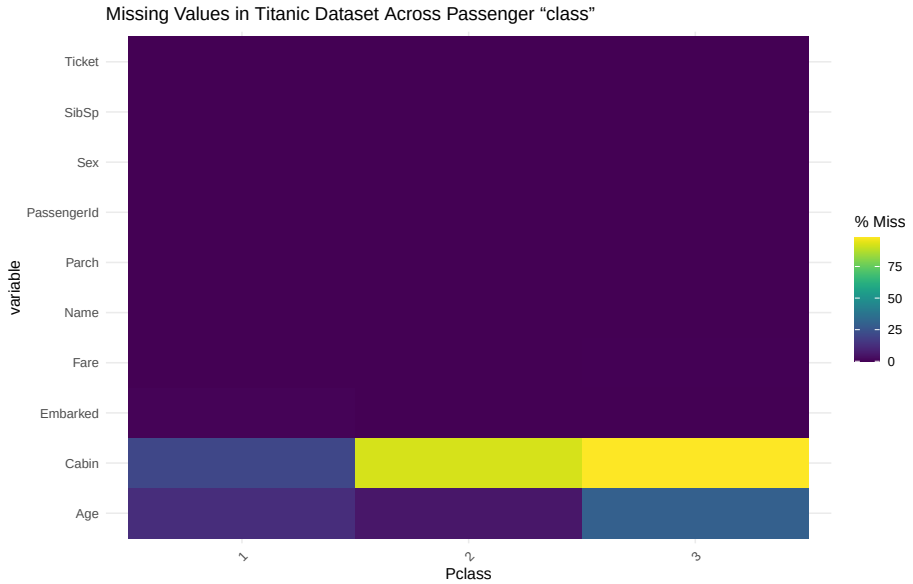
Presentazione del Dataset del Titanic

- **Survival**: indica se il passeggero è sopravvissuto o meno (0 = No, 1 = Si). Rappresenta la nostra variabile di risposta.
- **PassengerId**: id assegnato al passeggero.
- **Name**: nome, cognome e titolo del passeggero.
- **Pclass**: classe del biglietto (1 = 1st, 2 = 2nd, 3 = 3rd).
- **Sex**: sesso del passeggero.
- **Age**: età in anni del passeggero.
- **Sibsp**: numero di fratelli/coniugi a bordo del Titanic.
- **Parch**: numero di genitori/figli a bordo del Titanic.
- **Ticket**: numero del biglietto.
- **Fare**: tariffa del passeggero.
- **Cabin**: numero della cabina.
- **Embarked**: porto di imbarcazione (C = Cherbourg, Q = Queenstown, S = Southampton).

Gestione Dati Mancanti





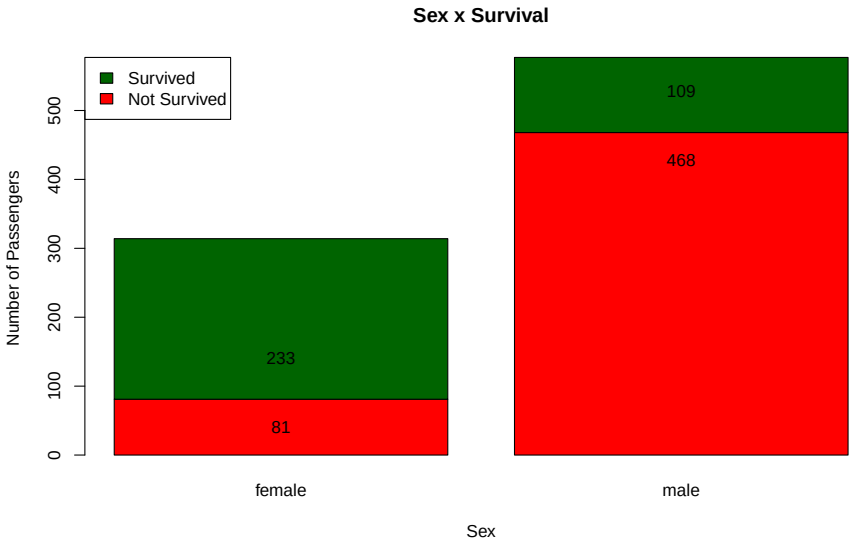


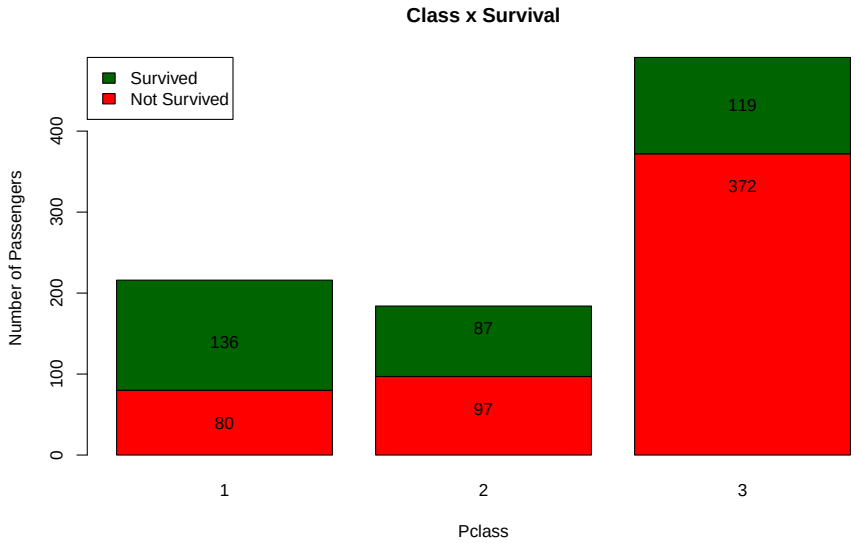
- Uniamo i dati di train e test per analizzare meglio il dataset.
- **Cabin**: troppi valori mancanti, rischia di alterare un modello. Viene dunque rimossa.
- **Age**: probabilmente uno dei fattori più importanti. Sfruttiamo l'età mediane tra Sex e Pclass per stimarne i valori mancanti.

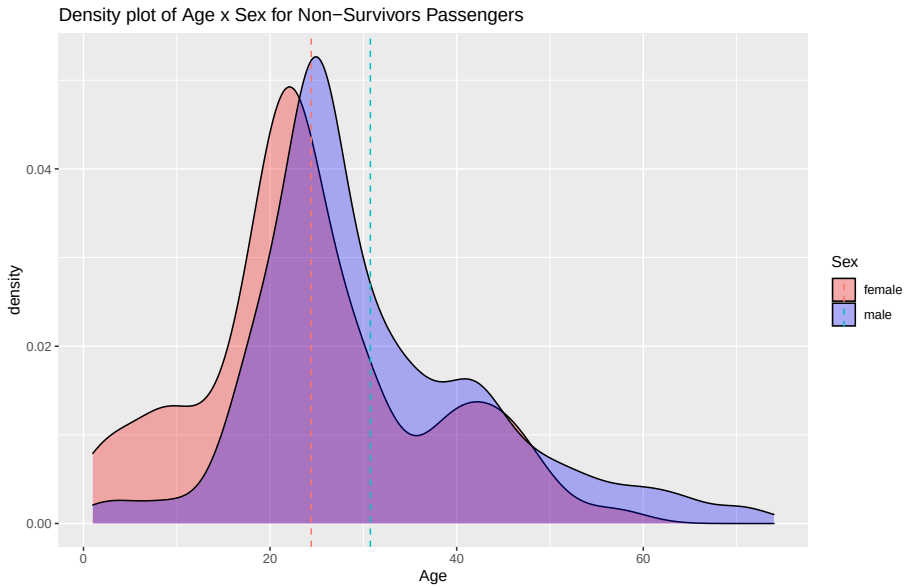
```
# Iterate over Sex (male or female) and Pclass (1, 2, 3) to calculate guessed values of
      Age for the six combinations.
for (i in 1:nlevels(titanic_full$Sex)) {
  for (j in 1:length(unique(titanic_full$Pclass))) {
    guess_age <- round(median(subset(titanic_full, Sex == titanic_full$Sex[i] & Pclass ==
          j, select=c(Age))$Age, na.rm = TRUE))
    titanic_full$Age[titanic_full$Sex == titanic_full$Sex[i] & titanic_full$Pclass == j &
          is.na(titanic_full$Age)] <- guess_age
  }
}
```

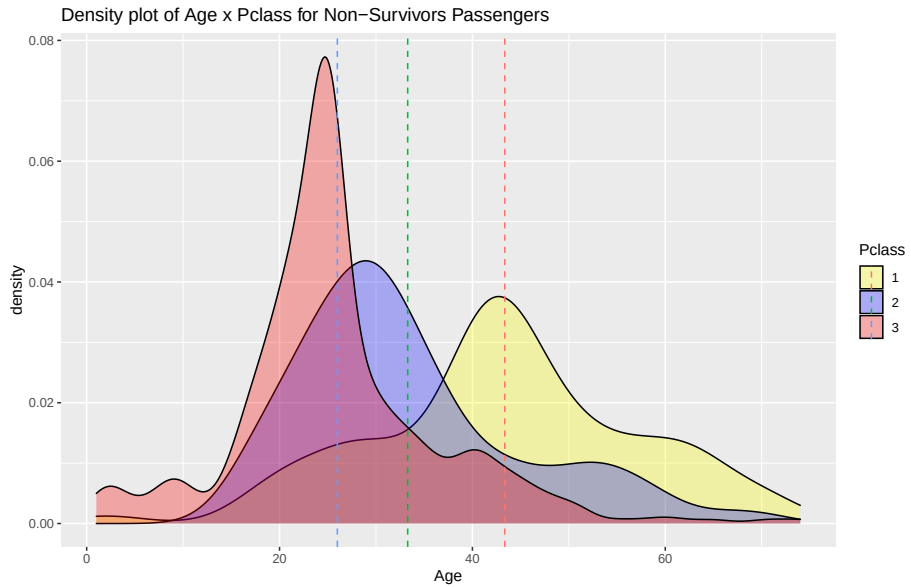
- **Embarked**: ha due soli valori mancanti. Li riempiamo semplicemente con il porto più comune (Southampton, S).
- **Fare**: l'unico valore mancante possiamo sostituirlo con la mediana.

Esplorazione e Visualizzazione dei Dati

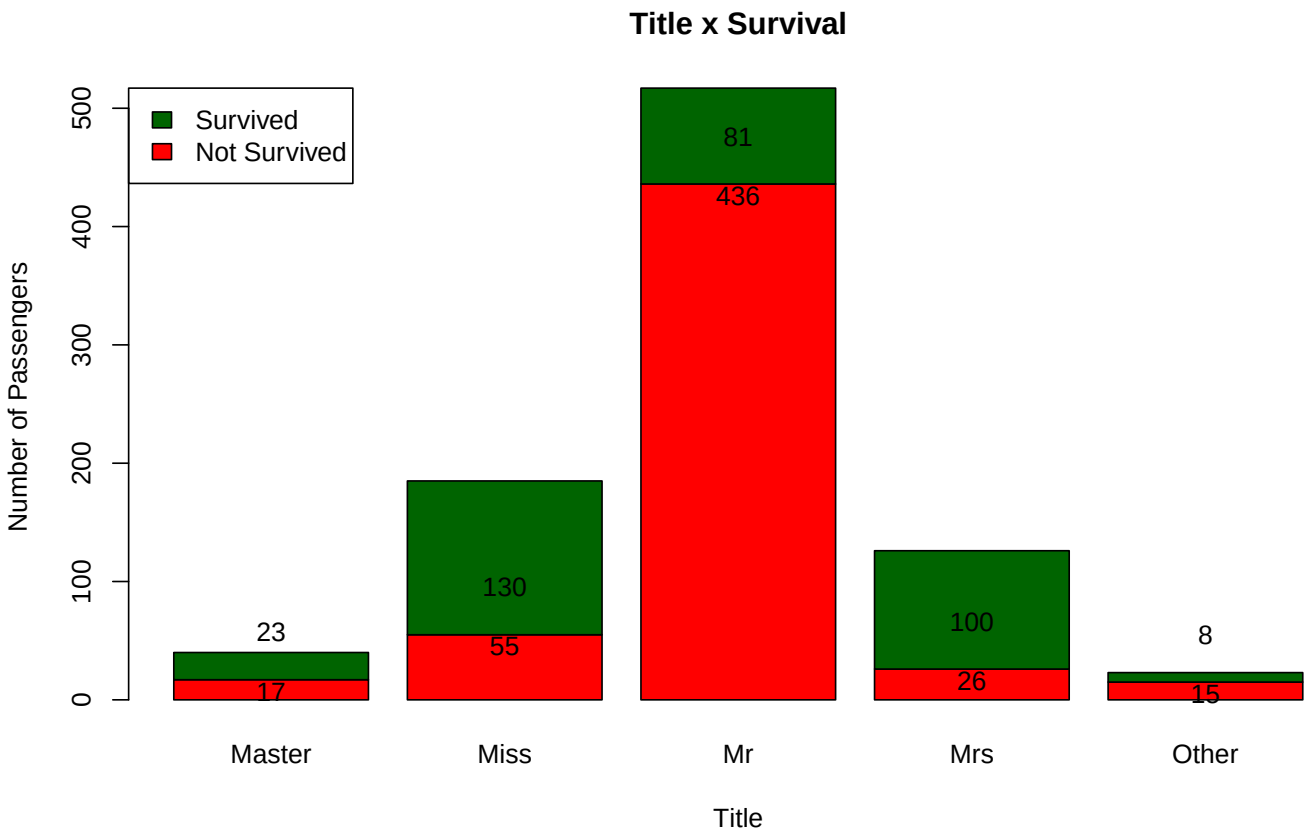






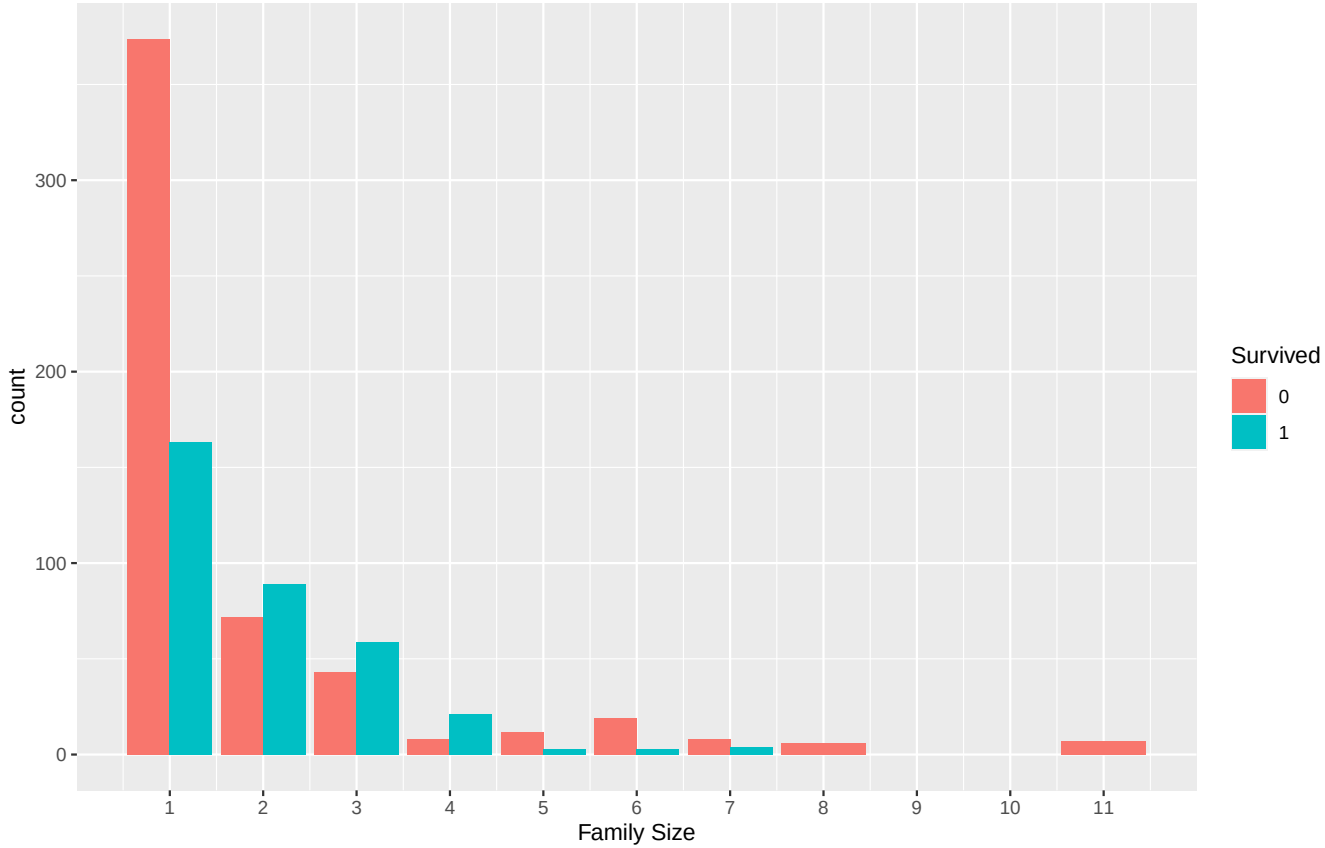


- **Title**: Utilizzato per capire il grado nella società, utile nella classificazione. Trasmette gli effetti interattivi tra Age e Sex.
  - Possiamo raggruppare i titoli in "MR", "MISS", "MRS", "MASTER" e "Other" per i più rari.



Feature Extraction - Family Size e IsAlone

- **FamilySize**: la dimensione della famiglia imbarcata potrebbe essere un fattore interessante (**SibSp** + **Parch** + 1).



- **IsAlone**: possiamo vedere che le persone single hanno un più alto tasso di "morte" rispetto alle altre persone.

- **Fare**: per rendere le tariffe più facili da analizzare, ed essendo continua, le raccogliamo in 4 gruppi di dimensioni omogenee (1 - **[0,7.9]**, 2 - **(7.9,14.5]**, 3 - **(14.5,31.3]**, 4 - **(31.3,512]**).
- Infine eliminiamo le variabili che non ci servono più e non contengono più informazioni rilevanti (*PassengerId*, *Name*, *SibSp*, *Parch*, *Ticket*).

Di seguito viene mostrato il dataset finale che utilizzeremo.

Survived	Pclass	Sex	Age	Fare	Embarked	Title	FamilySize	IsAlone
0	3	male	22	1	S	Mr	2	0
1	1	female	38	4	C	Mrs	2	0
1	3	female	26	2	S	Miss	1	1
1	1	female	35	4	S	Mrs	2	0
0	3	male	35	2	S	Mr	1	1

- Utilizzano un albero per passare dalle osservazioni su un elemento (**rami**) alle conclusioni sulla variabile obiettivo dell'elemento (**foglie**).
- **Classificazione**: le foglie rappresentano etichette di classe, i rami le congiunzioni di caratteristiche che portano a classi.

In questo progetto utilizzeremo quattro tipologie di alberi di decisione:

- **Classification And Regression Tree** (**CART**): partizionamento ricorsivo dello spazio dei predittori, nel quale ad ogni elemento viene assegnata una classe di predizione.
- **Conditional Inference Trees**: approccio statistico che utilizza test non parametrici come criteri di split.
- **Evolutionary Learning of Globally Optimal Trees** (**evtree**).
- **Bootstrap Aggregated** (**Bagged**): costruiscono più alberi decisionali ricampionando ripetutamente i dati di training con reimmissione e le previsioni sono effettuate con voto di maggioranza o con media dei risultati su tutti gli alberi.

Alberi Decisionali - Vantaggi e Limitazioni

Vantaggi:

- Semplici da capire e interpretare.
- Sono in grado di gestire dati sia numerici che categorici.
- Richiedono poca preparazione dei dati.
- Funzionano bene con set di dati di grandi dimensioni.
- Rispecchiano il processo decisionale umano più da vicino.
- Robusti contro la collinearità.
- La gerarchia degli attributi riflette l'importanza di essi.

Limitazioni:

- Gli alberi possono essere poco robusti.
- L'apprendimento di un albero ottimale è NP-completo.
- Gli albero decisionali possono andare in *Overfitting*.

Obiettivo

Ottenere una segmentazione gerarchica di un insieme di unità statistiche mediante l'individuazione di "regole" che sfruttano la relazione esistente tra classe e predittori.

- Le regole basate sui valori delle variabili vengono selezionate per ottenere il migliore **split** per differenziare le osservazioni in base alla variabile dipendente.
- Una regola divide un nodo in due **nodi figli**, lo stesso processo viene applicato a ciascun nodo "figlio" (ricorsivamente).
- Lo splitting si interrompe quando non è possibile ottenere più un guadagno (Gain) o viene soddisfatto un criterio di arresto.

Ogni osservazione rientra in un unico **nodo terminale** definito in modo univoco da un insieme di regole.

Valutazione degli Split

- Favorire l'**omogeneità** interna dei nodi ed **eterogeneità** esterna dei nodi figli.
- Lo scopo è **minimizzare** l'impurezza dei nodi figli calcolata attraverso:
  - Indice di **Entropia**: $-\sum_{i=1}^{C} p_i * log_2(p_i)$
  - Indice di **Gini**: $\sum_{i=1}^{C} p_i * (1 - p_i)$
- Impurezza $= 0$, il valore della variabile di risposta è lo stesso per tutte le osservazioni all'interno del nodo.

Assegnazione delle Classi ai Nodi

- Ci sono varie regole che possono essere utilizzate per farlo.
- Il nodo è etichettato con una modalità della variabile risposta.
- Al nodo viene assegnata la classe più frequente in esso.

Criteri di Arresto

- Numero di osservazioni minimo per ogni nodo.
- Purezza dei nodi maggiore di un valore fissato.
- Profondità dell'albero limitata.
- In ogni nodo abbiamo valori appartenenti ad una sola classe.

Potatura

- Facciamo crescere totalmente l'albero per poi potarlo successivamente.
- Utilizziamo metodi per individuare il numero di rami che ci garantiscano il minore *classification error rate* e la minima complessità dell'albero.
- Ad esempio con una *K-fold cross validation*.

Obiettivo

Migliorare la stabilità e l'accuratezza, riduce la varianza ed aiuta ad evitare l'overfitting.

- Dato un training set D di dimensione n, il bagging genera m nuovi training set D_i, ciascuno di dimensione n', campionando da D in modo uniforme e con reimmissione.
- Alcune osservazioni possono essere ripetute in ogni D_i.
- Questo tipo di campionamento è noto come bootstrap.
- Successivamente, m modelli vengono addestrati utilizzando gli m campioni di bootstrap e combinati calcolando la media degli output (Regressione) oppure per votazione (Classificazione).

Predire la Sopravvivenza sul Titanic

- Andiamo a predire la sopravvivenza sul Titanic utilizzando:
  - **Classification trees**: usando i pacchetti *rpart* e *tree* (con Gini e Entropia).
  - **Conditional inference trees**: usando il pacchetto *party*.
  - **Evolutionary learning of globally optimal trees**: usando il pacchetto *evtree*.
  - **Bootstrap aggregating**: usando il pacchetto *randomForest*.
- Confrontando i risultati ottenuti.
- Siamo interessati a predire i valori della variabile **Survived**.
- Dividiamo in maniera casuale le osservazioni in due insiemi, uno di training (80%) e uno di testing (20%).

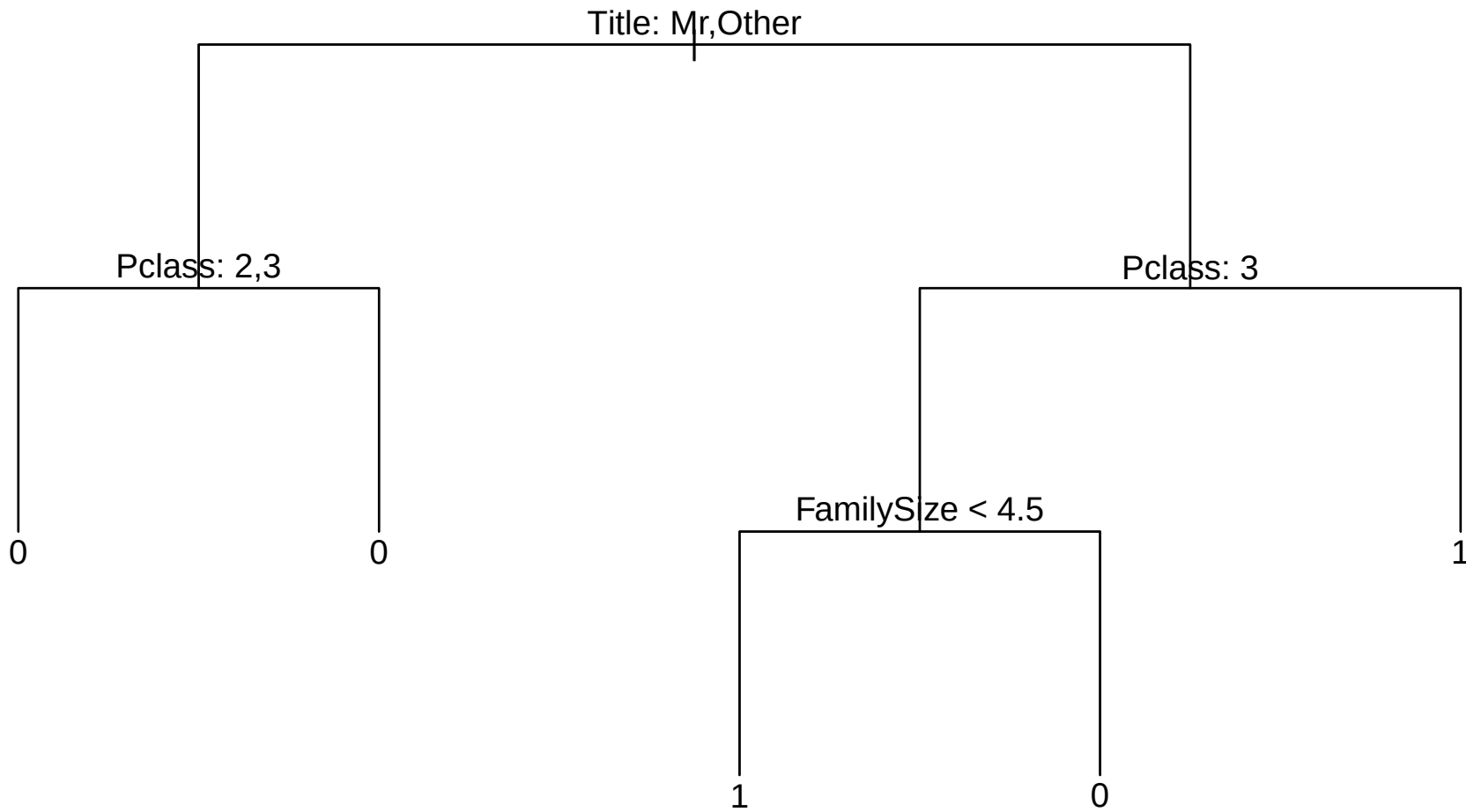
```
# Split the training set to get an approximation of the models performance
split = sample.split(titanic$Survived, SplitRatio = 0.8)
train = subset(titanic, split == TRUE)
test = subset(titanic, split == FALSE)

# Outcome variable
dependent_var <- "Survived"
# Predictor variables Model
independent_vars <- c("Pclass", "Sex", "Age", "Fare", "Embarked", "Title", "FamilySize", "
      IsAlone")

# Create the formula string
formula <- paste(dependent_var, "~", paste(independent_vars, collapse = " + "))
```

Caratteristiche Principali

- Partizionamento ricorsivo binario.
- Variabili numeriche divise in $X < a$ e $X > b$, i livelli delle variabili fattore sono divisi in due gruppi non vuoti.
- Viene scelto lo split che massimizza la riduzione dell'impurità.
- Profondità limitata a 31, le variabili fattore limitate a 32 livelli.
- Split di default con Entropia. In alternativa Gini.
- Regola di stop dell'albero (default):
  - Numero minimo di osservazioni da inserire in un nodo figlio: 5.
  - La dimensione minima di ogni nodo devi essere pari a 10.
  - Entropia all'interno del nodo $\leq 1\%$ di quella del nodo radice.
- Pruning scegliendo la dimensione finale dell'albero, minimizzando il *cross validation error*.

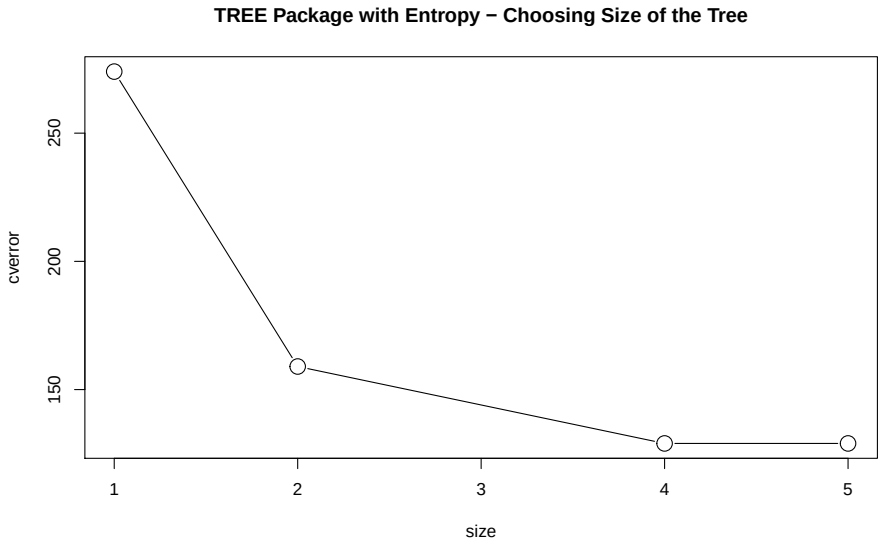
Pacchetto *tree* con Entropia - Costruzione dell'Albero

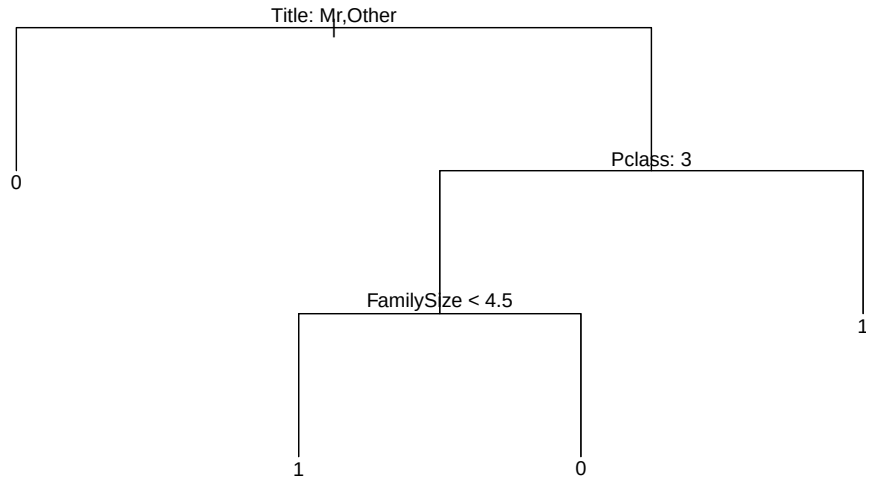
```
# TREE with entropy
tree_model <- tree(formula, data = train, split = "deviance")
plot(tree_model, type = "uniform", main = "TREE Package with Entropy")
text(tree_model, pretty=5, cex=1)
```



Pacchetto *tree* con Entropia - Pruning

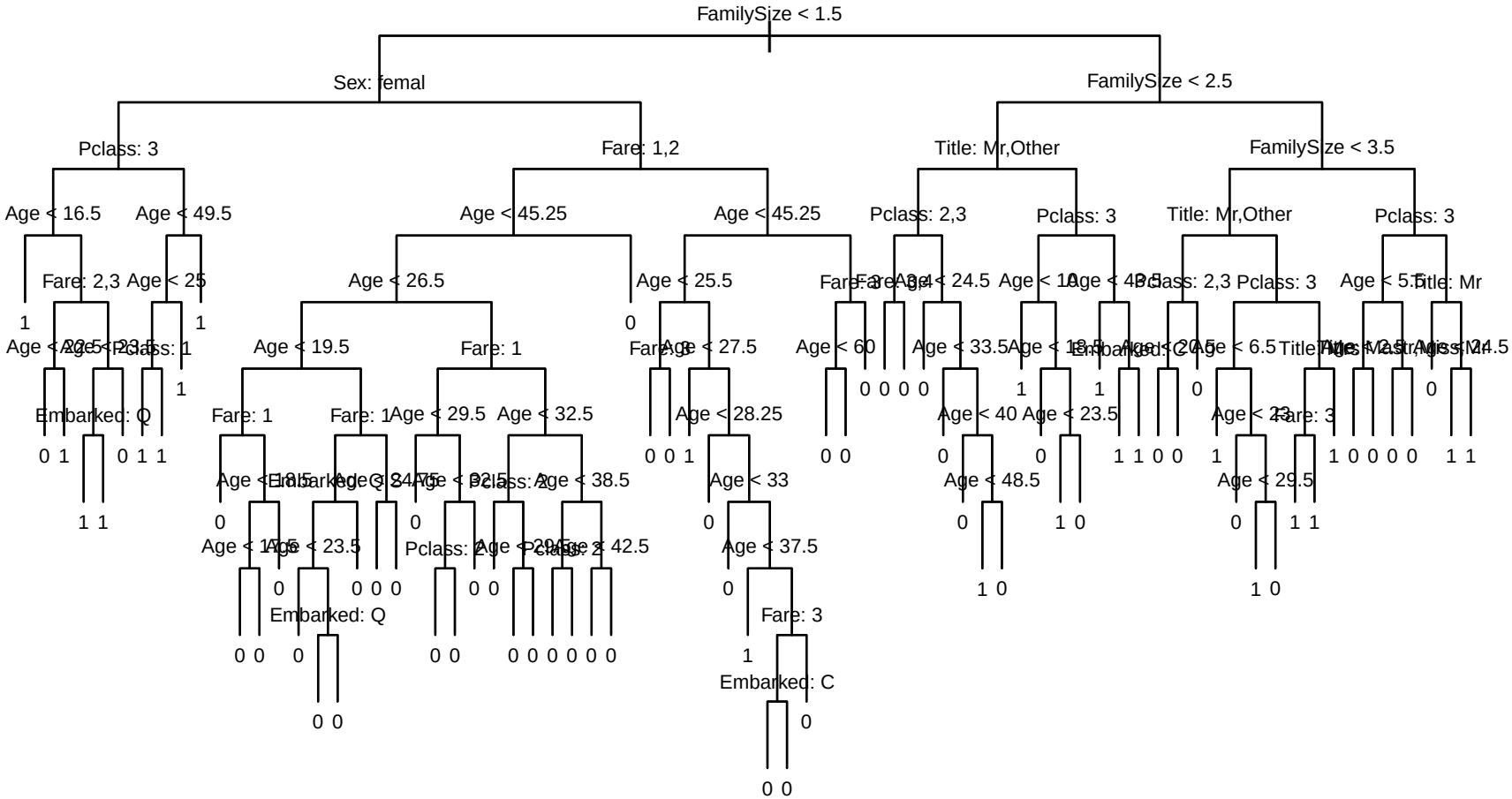
```
# Search best size (leaf nodes) for the tree
tree_cv <- cv.tree(tree_model, K = 10, FUN = prune.misclass)
tree_cv
plot(tree_cv$size, tree_cv$dev, type="b", xlab="size", ylab="cverror", main="TREE Package with
      Entropy - Choosing Size of the Tree", cex=2)
best_size <- tail(tree_cv$size[which(tree_cv$dev==min(tree_cv$dev))], n=1)

# Prune tree
pruned_tree <- prune.misclass(tree_model, best=best_size)
plot(pruned_tree, type = "uniform", main="TREE Package with Entropy - Pruned")
text(pruned_tree, pretty=5, cex=1)
```





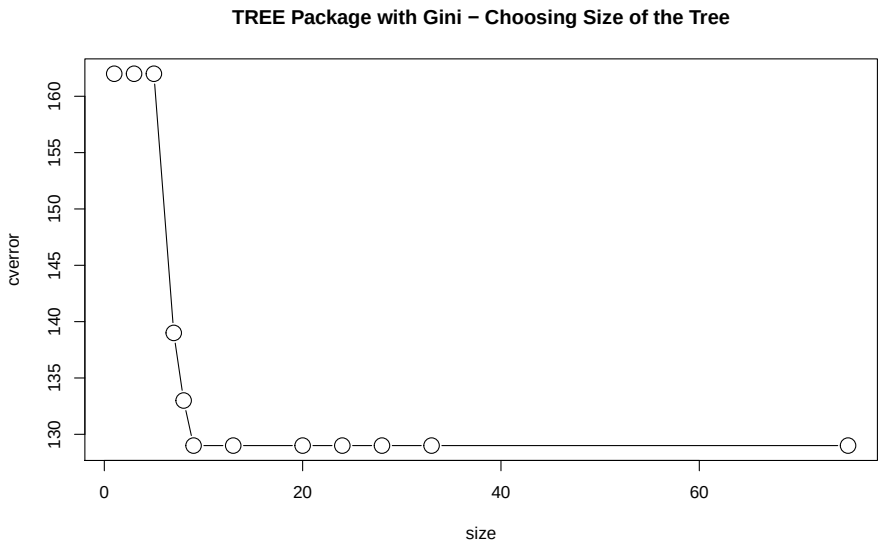
Pacchetto *tree* con Gini - Costruzione dell'Albero

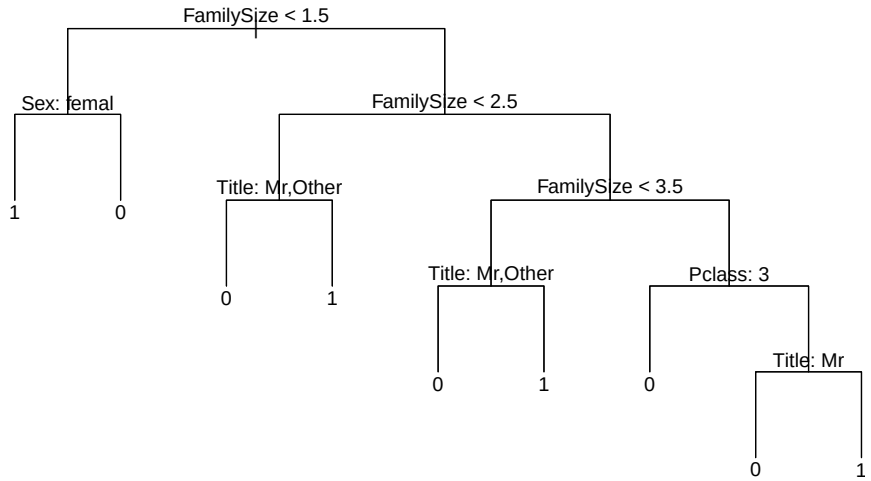
```
# Decision tree with Gini Index
tree_model <- tree(formula, data = train, split = "gini")
plot(tree_model, type = "uniform",  main="TREE Package with Gini")
text(tree_model, pretty=5, cex=0.6)
```



Pacchetto *tree* con Gini - Pruning

```r
# Search best size (leaf nodes) for the tree
tree_cv <- cv.tree(tree_model, K = 10, FUN = prune.misclass)
tree_cv
plot(tree_cv$size, tree_cv$dev, type="b", xlab="size", ylab="cverror", main="TREE Package with
      Gini - Choosing Size of the Tree", cex=2)
best_size <- tail(tree_cv$size[which(tree_cv$dev==min(tree_cv$dev))], n=1)

# Prune tree
pruned_tree <- prune.misclass(tree_model, best=best_size)
plot(pruned_tree, type = "uniform", main="TREE Package with Gini - Pruned")
text(pruned_tree, pretty=5, cex=1)
```



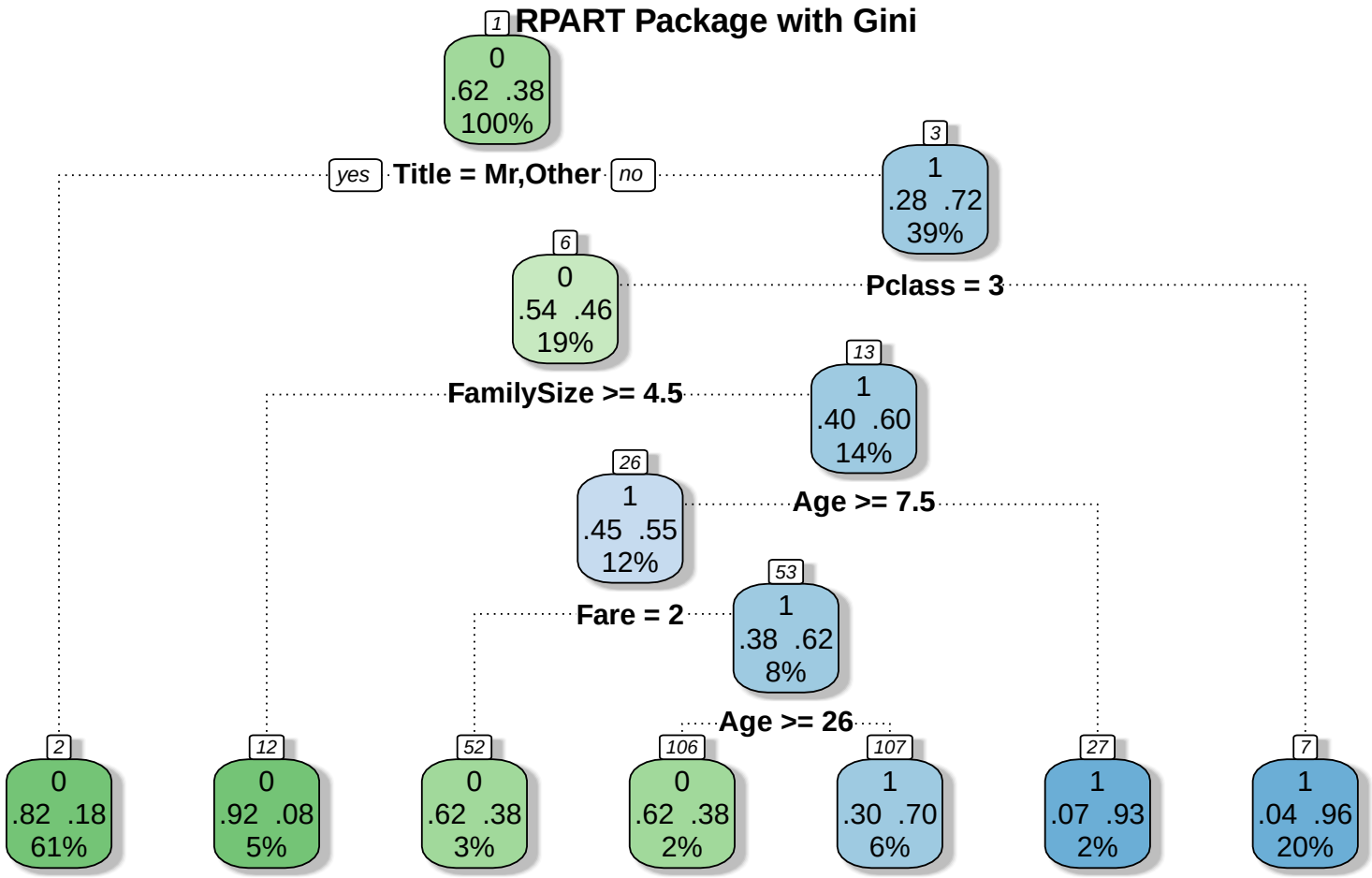


Caratteristiche Principali

- Reimplementazione moderna del pacchetto tree.
- Più efficiente, molte più parti vengono eseguite in codice C.
- Maggior controllo per gestire eventuali valori mancanti.
- Offre maggiore flessibilità durante la crescita degli alberi.
- Vengono offerti circa 9 parametri per la modellazione.
- Split di default con Gini. In alternativa Entropia.
- La dimensione minima di ogni nodo deve essere 20 (default).
- La profondità massima dell'albero è pari a 30 (default).
- Algoritmo greedy, come il precedente.

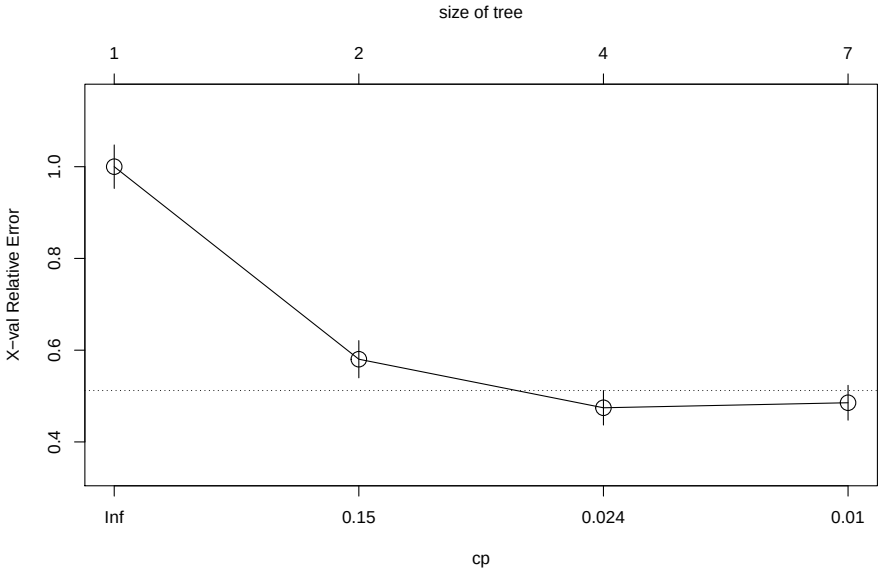
Pacchetto *rpart* con Gini - Costruzione dell'Albero

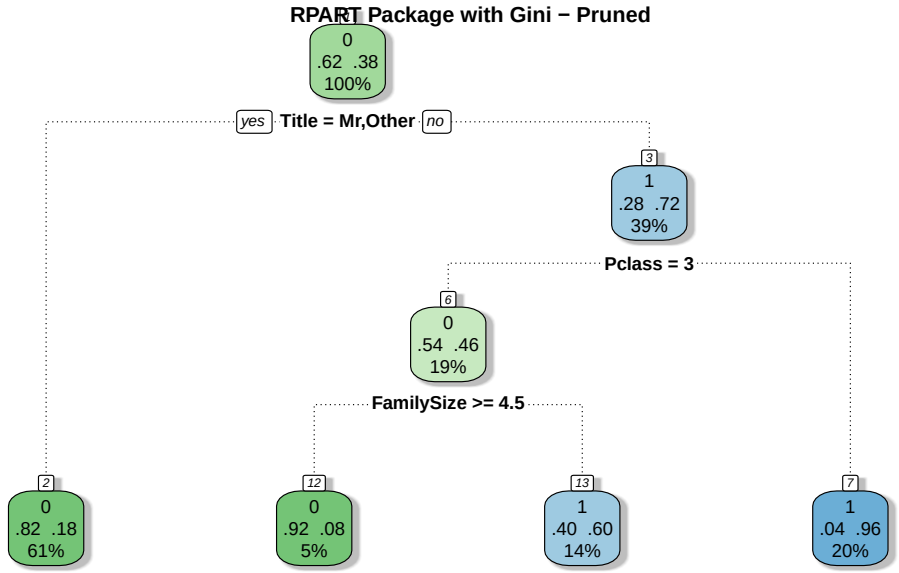
```
# RPART tree with Gini
rtree_model <- rpart(formula, data = train, method = "class", parms = list(split='gini'))
fancyRpartPlot(rtree_model, main="RPART Package with Gini", sub = "")
```



Pacchetto *rpart* con Gini - Pruning

```
# Search best index
plotcp(rtree_model, cex=2)

# Pruning the tree
pruned_rtree <- prune.rpart(rtree_model, cp=rtree_model$cptable[which.min(rtree_model$cptable[,"
      xerror"]),"CP"])
fancyRpartPlot(pruned_rtree, main="RPART Package with Gini - Pruned", sub = "")
```





Algoritmo

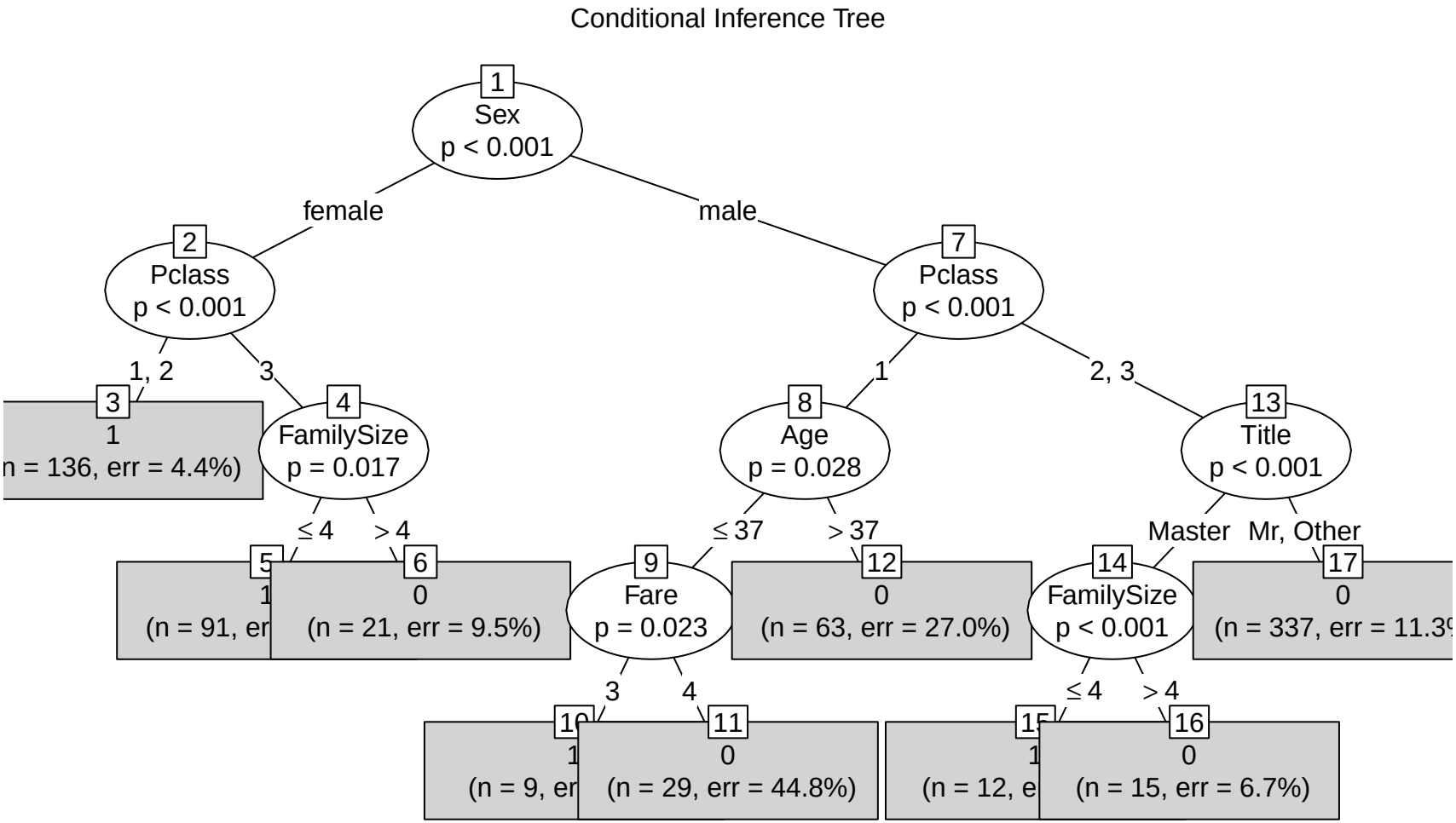
1. Viene effettuato un test di ipotesi nulla di indipendenza tra ogni variabile di input e la variabile risposta.
   - Stop se l'ipotesi non può essere rifiutata, altrimenti seleziona la variabile con più forte associazione alla variabile di risposta.
   - Misurata da un *p-value* per test di ipotesi nulla parziale.
2. Split binario nella variabile di input selezionata.
3. Ripete in modo ricorsivo i passaggi 1) e 2).

Caratteristiche Principali

- Split effettuato quando il criterio (1 - *p-value*) supera il valore minimo fornito (default mincriterion $= 0.95$, dunque $\alpha = 0.05$)
- Garantisce la crescita nella giusta dimensione, niente pruning.
- Dimensione minima di ogni nodo 20. Profondità massima ∞.

Pacchetto *party* - Costruzione dell'Albero

```
# Conditional Inference Tree - Party
party_tree_model <- ctree(as.formula(formula), data = train)
plot(party_tree_model, type = "simple", main="Conditional Inference Tree")
```



Caratteristiche Principali

- Algoritmo evolutivo per l'apprendimento ottimale a livello globale di alberi di classificazione e regressione in R.
- Compiti intensivi completamente eseguiti in C++.

Algorithm 1: Algoritmo *evtree*

1. Inizializzazione della popolazione.
2. Valutazione di ogni individuo.

while *La condizione di terminazione non è soddisfatta* **do**

   a. Seleziona i genitori.
   b. Modifica gli individui selezionati tramite operatori di variazione.
   c. Valuta le nuove soluzioni.
   d. Seleziona i sopravvissuti per la prossima generazione.

Pacchetto *evtree* - Dettagli dell'Algoritmo

Inizializzazione:

- Regola di split generata casualmente in un insieme di alberi, nel *root node*.

Selezione del Genitore:

- Ogni albero viene selezionato una sola volta per essere modificato.

Operatori di Variazione:

- Quattro operatori di mutazione e un operatore di crossover.
- In ogni fase di modifica, uno casualmente per ogni albero.

Funzione di Valutazione:

- Rappresenta i requisiti a cui la popolazione dovrebbe adattarsi.
- Massimizzare la precisione e minimizzare la complessità.

Selezione dei Sopravvissuti:

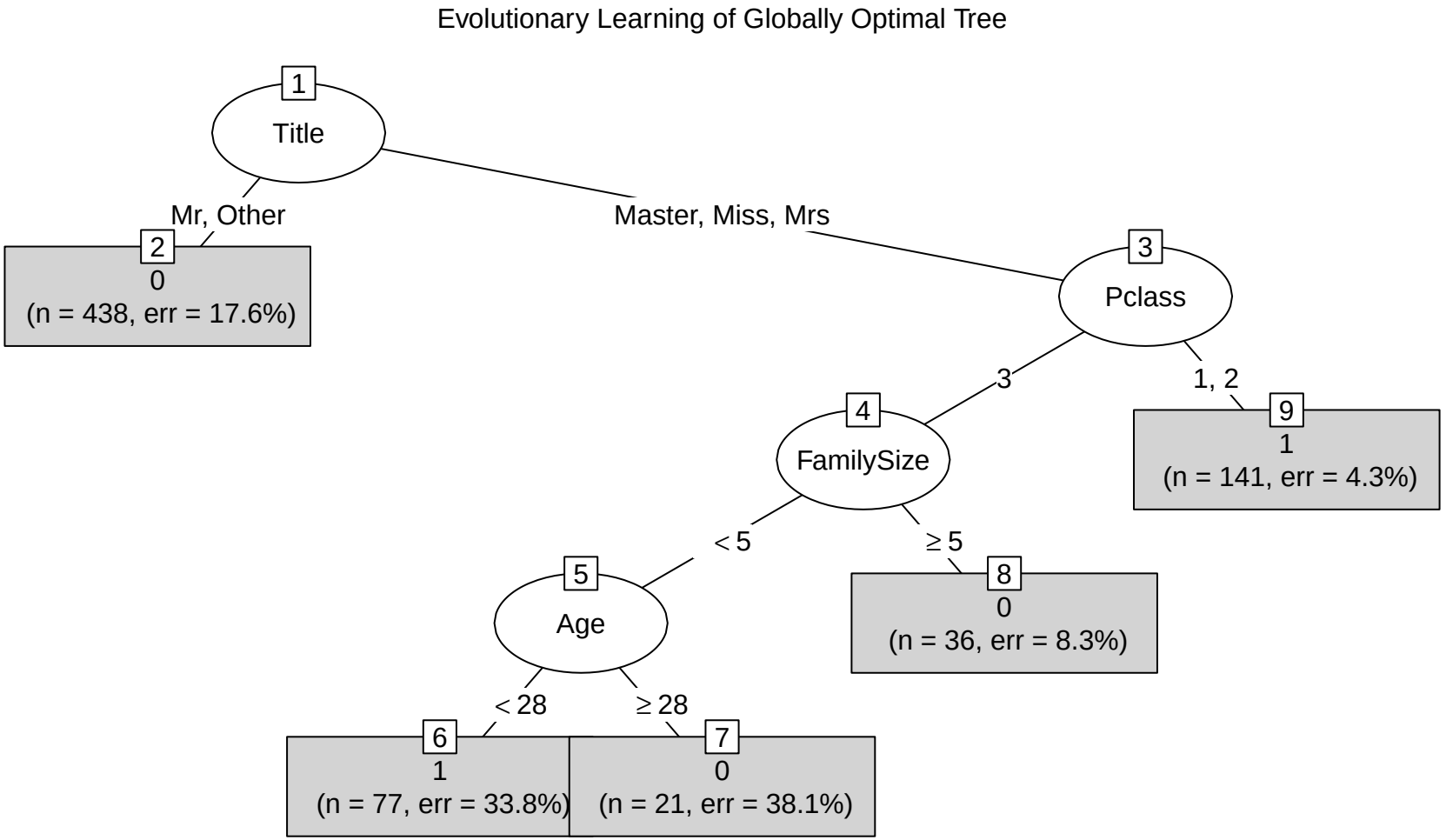
- Solo la soluzione prima della modifica o quella dopo viene mantenuta.

Terminazione:

- La qualità del 5% dei migliori alberi si stabilizza per 100 iterazioni, ma non prima di 1000.
- Se non converge termina dopo un numero di iterazioni dato in input.
- Output: albero con la massima qualità in base alla funzione di valutazione.

Pacchetto *evtree* - Costruzione dell'Albero

```
# Evolutionary Learning of Globally Optimal Trees
ev_tree_model <- evtree(formula, data = train)
plot(ev_tree_model, type = "simple", main="Evolutionary Learning of Globally Optimal Tree")
```



- Il bagging riduce l'overfitting, ma non risolve completamente il problema.
- Un piccolo numero di predittori tende a dominare sugli altri.
- Selezionati nei primi split, influenzano le forme degli alberi nella foresta.
- Aumentano correlazioni tra gli alberi, ostacolano riduzione della varianza.

Caratteristiche Principali

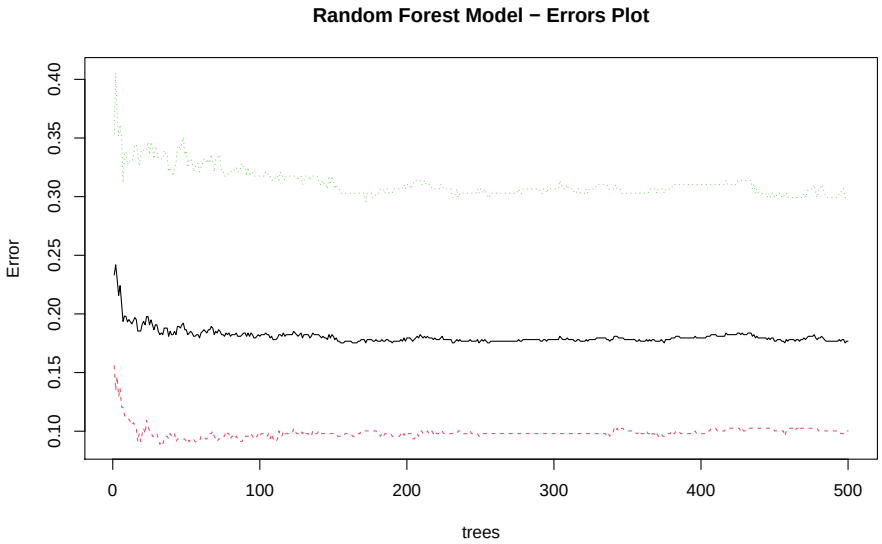
- Aggira il problema usando un sottoinsieme casuale di variabili in ogni split.
- Evita sovrarappresentazione variabili dominanti. Foresta più diversificata.
- *Out-of-bag* (OOB), errore di previsione medio su ciascun campione di addestramento X_i, utilizzando solo alberi che non hanno X_i nel loro campione di bootstrap.
- Valuta le previsioni su quelle osservazioni che non sono state utilizzate nella costruzione del modello. Test set non necessario.

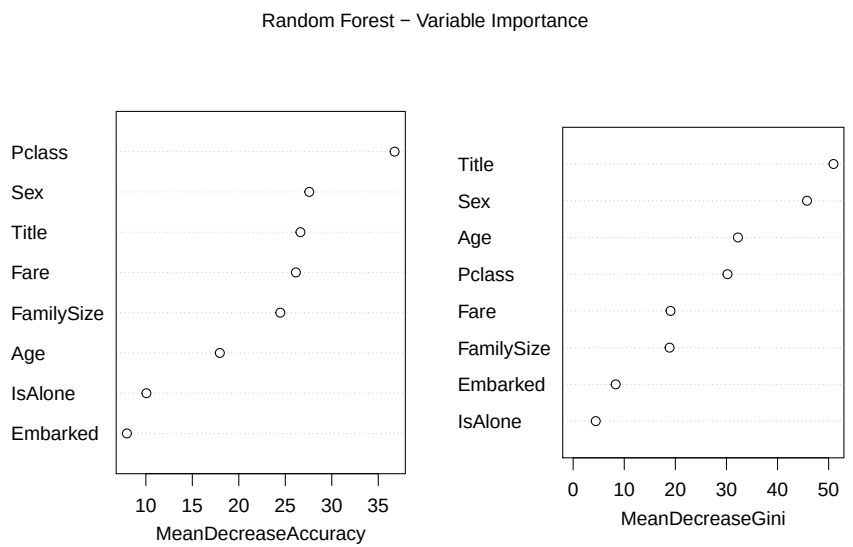
Default:

- N. di alberi (**ntree**) = 500, n. di variabili per lo split (**mtry**) = $\sqrt{n.\ predittori}$ per classificazione o $mtry = \frac{(n.\ predittori)}{3}$ per regressione.

Pacchetto *randomForest*

```r
# Random Forest Model
random_forest_model <- randomForest(as.formula(formula), data = train, importance=TRUE)
print(random_forest_model)

# Plot errors
plot(random_forest_model, main = "Random Forest Model - Errors Plot")

# Plot variables importance
importance(random_forest_model)
varImpPlot(random_forest_model, main = "Random Forest - Variable Importance", cex=1.2)
```

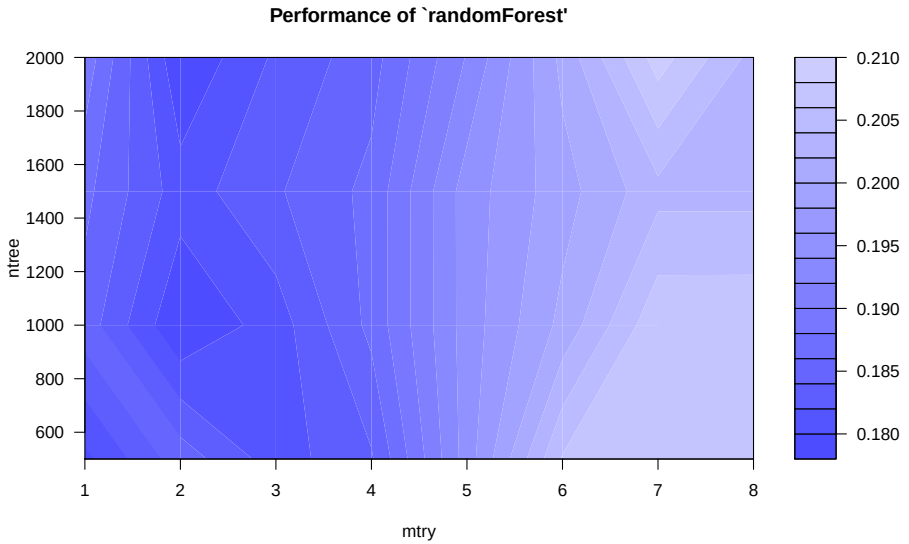


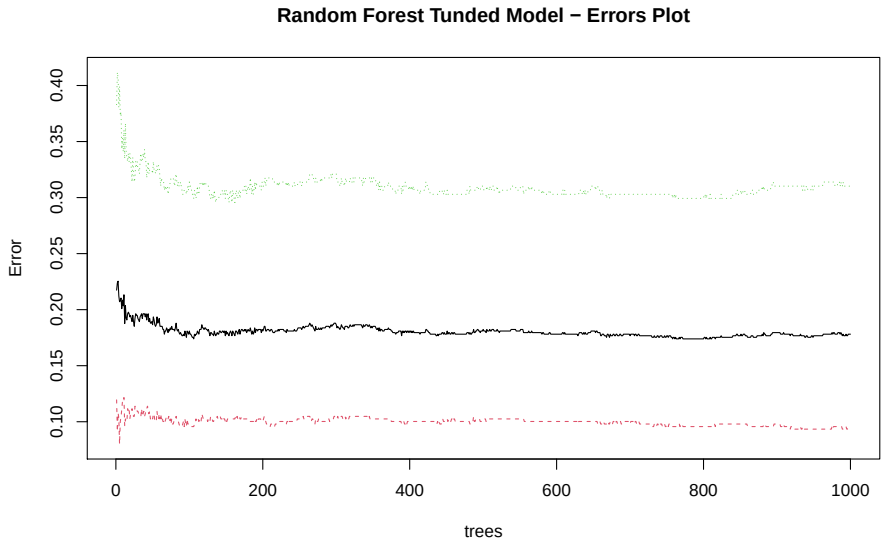


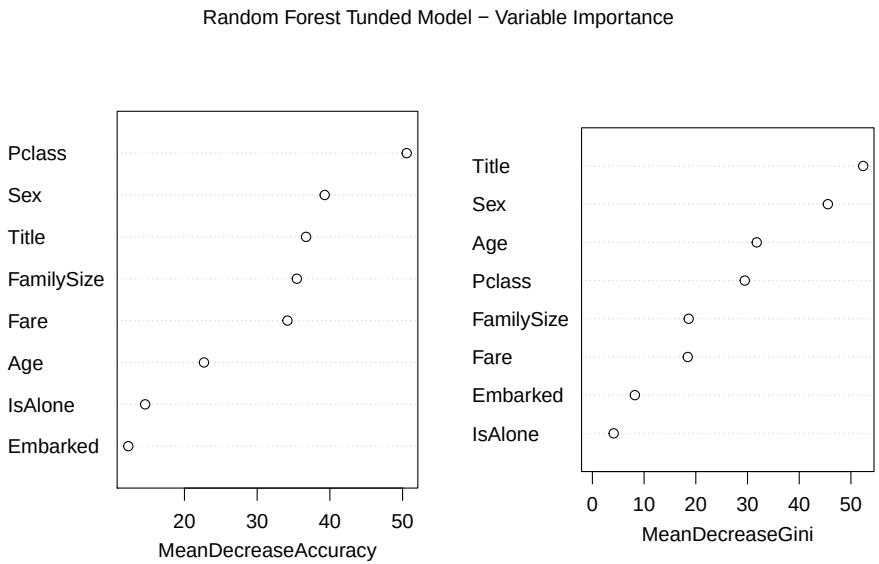
- Troppi alberi possono essere eccessivi per la qualità del modello, necessità di calibrare anche il n. di variabili campionate casualmente per lo split.
- Cerchiamo quali sono i valori ottimali per i due parametri principali eseguendo una *grid search* (*ntree* tra 500 e 2000 in incrementi di 500).

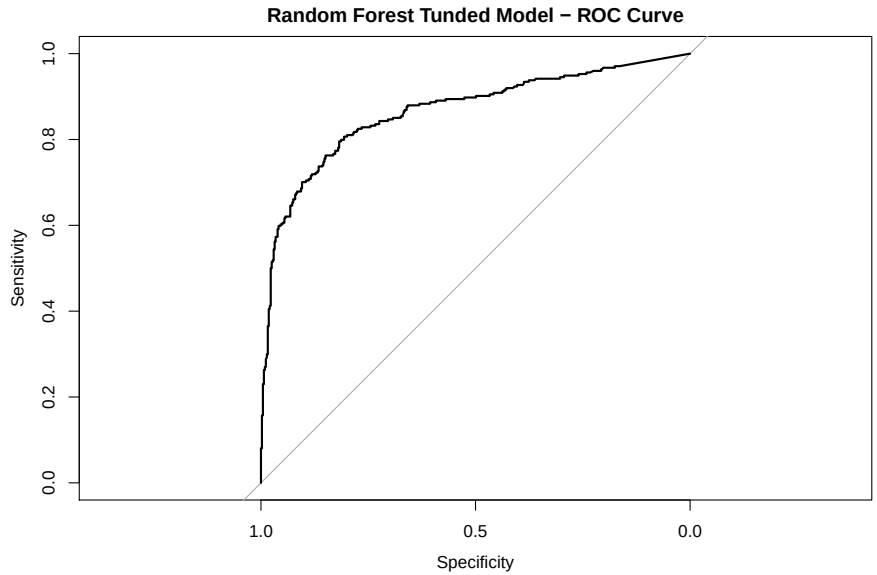
```
# Random Forest with tuned parameters
tuned_random_forest <- tune(randomForest, train.x = as.formula(formula), data = train,
      validation.x = test, ranges = list(mtry = 1:(ncol(train)-1), ntree = seq(500, 2000, 500)),
       importance=TRUE)
plot(tuned_random_forest)

# Search best model
best_model_random_forest <- tuned_random_forest$best.model
tuned_random_forest$best.parameters
print(best_model_random_forest)

# Plot errors
plot(best_model_random_forest, main = "Random Forest Tunded Model - Errors Plot")

# Variable importance
importance(best_model_random_forest)
varImpPlot(best_model_random_forest, main = "Random Forest Tunded Model - Variable Importance",
      cex=1.2)
```

Pacchetto *randomForest* - Risultati del Tuning dei Parametri









Confronto tra Pacchetti - Matrici di Confusione

tree - Entropia		
	Not Survived	Survived
Not Survived	98	13
Survived	12	55
Accuracy	85.95%	

tree - Gini		
	Not Survived	Survived
Not Survived	99	19
Survived	11	49
Accuracy	83.15%	

rpart - Gini		
	Not Survived	Survived
Not Survived	98	13
Survived	12	55
Accuracy	85.95%	

party		
	Not Survived	Survived
Not Survived	95	13
Survived	15	55
Accuracy	84.27%	

evtree		
	Not Survived	Survived
Not Survived	100	13
Survived	10	55
Accuracy	87.08%	

randomForest		
	Not Survived	Survived
Not Survived	98	14
Survived	12	54
Accuracy	85.39%	

Grazie dell'Attenzione