UNIVERSITÁ DEGLI STUDI DI UDINE

Dipartimento Politecnico di Ingegneria e Architettura D.P.I.A.

Corso di Laurea Magistrale in
Ingegneria Gestionale

Corso di Applied Statistics

Pacchetto RandomForest R

Studenti:
Matteo Andreutti
Samuele Bovo
Diego Virgili

Docente:
Prof.Ruggero Bellio

Anno accademico 2023/24

Indice

1 INTRODUZIONE **1**

2 DAL BAGGING AL RANDOM FOREST **1**
2.1 Bagging . . . 1
2.1.1 Bootstrapping . . . 1
2.1.2 Aggregation . . . 2
2.2 Random Forest . . . 3
2.2.1 Stima dell'errore, importanza della variabile e altri strumenti del Random Forest . . . 3
2.2.2 Vantaggi e difficoltà del Random Forest . . . 4

3 ALGORITMI E FUNZIONI PACCHETTO RandomForest **6**
3.1 randomForest . . . 6
3.2 predict.randomForest . . . 8
3.3 plot.randomForest . . . 9
3.4 classCenter . . . 9
3.5 combine, getTree, grow . . . 9
3.6 importance, varImpPlot . . . 9
3.7 margin . . . 10
3.8 na.roughfix, rfImpute . . . 10
3.9 outlier . . . 10
3.10 treesize . . . 10
3.11 tuneRF . . . 11
3.12 varUsed . . . 11
3.13 rfcv . . . 11

4 ESEMPI APPLICATIVI **12**
4.1 ESEMPIO APPLICATIVO DI CLASSIFICAZIONE: Titanic dataset . . . 12
4.1.1 Introduzione al dataset . . . 12
4.1.2 RandomForest Package . . . 19
4.2 ESEMPIO APPLICATIVO DI REGRESSIONE: dataset imports85 . . . 27
4.2.1 Introduzione al dataset . . . 27
4.2.2 RandomForest package . . . 31

5 CONCLUSIONI **41**

6 BIBLIOGRAFIA **42**

1 INTRODUZIONE

Con il termine Random Forest si intende una delle tecniche più utilizzate nella sfera dell'ensemble learning (apprendimento automatico), per la classificazione/regressione di modelli con variabili esplicative multiple. L'ensemble learning, o ensemble method, è un approccio in cui vengono utilizzati molteplici modelli o algoritmi ("buliding blocks") per ottenere una migliore prestazione predittiva rispetto a quella ottenuta dagli stessi modelli applicati singolarmente, i quali sono anche chiamati "weak learners". Il Random Forest – che opera nell'ambito dei Decision Trees - utilizza quindi più alberi decisionali per lo stesso problema, combinandoli in modo da ottenere un classificatore più accurato rispetto ai blocchi di partenza. Tale algoritmo, considerato molto versatile e accurato, viene quindi utilizzato per problemi di classificazione e regressione in diversi campi di applicazione: finanza, biologia, bioinformatica e molti altri, fino al machine learning.

2 DAL BAGGING AL RANDOM FOREST

L'algoritmo del Random Forest si basa sulla tecnica del Bagging (che verrà introdotto in questo capitolo), migliorandola con l'introduzione di una leggera modifica per diminuire la correlazione tra i vari alberi ("building blocks"), aumentando così – in generale – l'accuratezza del modello.

2.1 Bagging

Gli alberi decisionali singoli sono spesso caratterizzati da alta varianza. Il Bagging è una procedura che permette di ridurre la varianza di un metodo statistico di apprendimento, e viene spesso utilizzata nel contesto degli alberi decisionali. In particolare questa procedura è composta da due fasi: Bootstrapping + Aggregation (Bagging).

2.1.1 Bootstrapping

La tecnica del Bootstrapping viene utilizzata per ottenere diversi set di addestramento dalla popolazione iniziale, campionando ripetutamente le osservazioni dal set di dati originale. Il campionamento è generato casualmente con sostituzione, il che significa che alcune osservazioni possono ripetersi più di una volta nello stesso set di addestramento, mentre altre possono non comparire. Dunque, da un unico set originale di dimensione n (n osservazioni), si ottengono B set di addestramento di dimensione n. A livello esemplificativo, in figura 1.1. è rappresentata la procedura di Bootstrapping per un piccolo campione di n=3 osservazioni, dal quale vengono generati B set di addestramento (di dimensione n). Ognuno di questi set di addestramento viene utilizzato per generare un modello predittivo - nel nostro caso, un albero decisionale (in figura: esempio per un problema di regressione – la tecnica, comunque, può essere utilizzata anche per i problemi di classificazione).

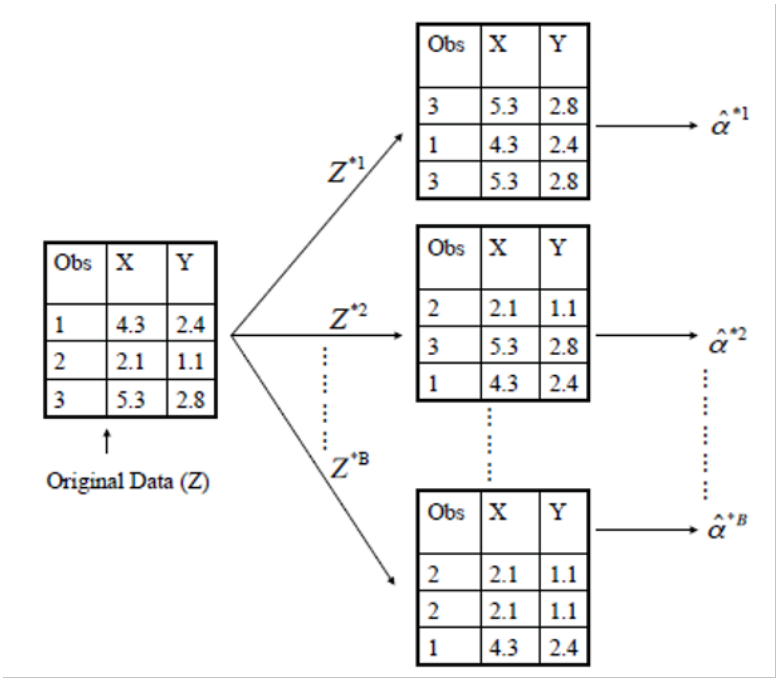


Figura 1.1. Illustrazione grafica della procedura di Bootstrapping in un campione di dimensione n=3 osservazioni. Ogni set di dati generato dal Bootstrap contiene n osservazioni, campionate con sostituzione dal set di dati originale. Ogni set generato (set di addestramento) viene utilizzato per generare un modello predittivo – nel caso del Random Forest: B alberi decisionali sono creati. (Fonte: An introduction to Statistical Learning, by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. Secon Edition)

2.1.2 Aggregation

Per ottenere un modello previsionale singolo dai B alberi costruiti a seguito del Bootstrapping, si procede con l'aggregazione dei risultati (aggregation). Nel caso di problemi di regressione (output Y quantitativo), è sufficiente effettuare una media dei risultati di ogni singolo albero decisionale (singolo modello predittivo - "building block").

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

In cui l'argomento della sommatoria è il modello previsionale (o albero decisionale) `f^*b(x)`, generato dal b-esimo set di dati di addestramento (generato tramite Bootstrapping), con "x" vettore dei predittori. Grazie al Bagging, siamo in grado di ridurre la varianza del modello: ricordiamo infatti che, dato un set di n osservazioni indipendenti, ognuna con varianza `sd^2`, la varianza della media delle osservazioni è data da `sd^2/n`. In sintesi quindi, quello che viene fatto col bagging, è:

- generare B diversi set di addestramento dal set originale, tramite Bootstrapping
- generare B modelli di previsioni (alberi) usando i set di addestramento generati in precedenza
- aggregare i risultati, effettuando una media delle previsioni di ogni modello/albero (e riducendo la varianza)

Nel caso, invece, di problemi di classificazione (output Y qualitativo), come può essere utilizzata la tecnica di Bagging? Le fasi di Bootstrapping e di generazione dei singoli alberi rimangono invariate. Per la fase di aggregazione/previsione – invece – diversi approcci possono essere utilizzati, ma il più comune è quello del "voto di maggioranza": per classificare un nuovo oggetto dato un vettore di input con i predittori x_i, il vettore (oggetto) viene classificato da ogni singolo albero generato nelle fasi precedenti – ogni albero "vota" per una classe e l'algoritmo sceglierà quella con il maggior numero di voti in totale. A livello visivo, in figura 1.2 è riportato uno schema sintetico del processo di Bagging (alla base, come vedremo, del Random Forest).

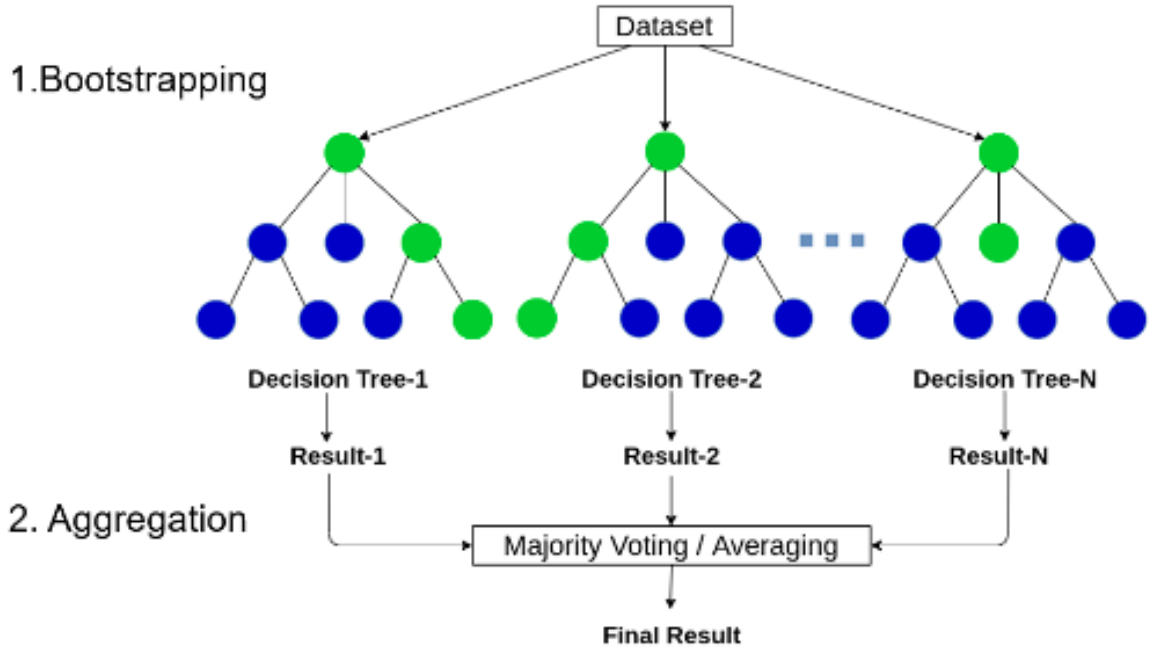


Figura 1.2. Sintesi grafica della procedura di Bagging

2.2 Random Forest

Come nel Bagging, il Random Forest consiste nel generare diversi alberi decisionali a partire dai set di addestramento creati tramite Bootstrapping. Nel caso del Random Forest, tuttavia, vi è l'aggiunta di una leggera modifica che diminuisce la correlazione tra i vari alberi, e quindi aumenta in generale l'accuratezza del metodo: nella costruzione degli alberi decisionali, non tutti i predittori (variabili di input x_i) vengono considerati per la suddivisione degli alberi nei vari rami/nodi. Definendo p il numero totale di variabili di input, ogni albero della foresta valuterà - per ogni nodo - solo m (con `m<<p`) variabili tra le quali scegliere la migliore per la suddivisione del nodo stesso. Ogni suddivisione valuterà quindi m variabili casuali: generalmente il numero m viene definito come `m = sqrt(p)`. In questo modo le previsioni prodotte dai vari alberi nella foresta risulteranno essere decorrelate: effettuare una media (aggregazione) di misure/previsioni decorrelate riduce la varianza della previsione "finale", rendendola meno variabile e quindi, in generale, più affidabile. In sintesi, quindi, la principale differenza tra Bagging e Random Forest risiede nella scelta della grandezza del sottoinsieme delle variabili di input da considerare in ogni nodo: m, numero costante per ogni suddivisione. Per esempio, se un modello Random Forest viene costruito con m=p, il risultato sarà equivalente al Bagging.

2.2.1 Stima dell'errore, importanza della variabile e altri strumenti del Random Forest

2.2.1.1 Stima dell'errore: Out Of Bag error estimation Nel Random Forest non è necessaria la convalida incrociata (cross-validation) o un set di osservazioni apposito per testare il modello ottenuto e per ottenere una stima dell'errore sul set di test. Questa stima viene effettuata automaticamente dall'algoritmo, durante la sua corsa: ogni albero della foresta, infatti, viene costruito utilizzando un campione generato

tramite Bootstrapping, il quale (come visto) non contiene tutte le osservazioni del set di osservazioni originali (alcune osservazioni si ripetono, alcune mancano). In particolare, circa un terzo delle osservazioni originali viene escluso dal processo di generazione del k-esimo albero (osservazioni Out-of-Bag [OOB] per il k-esimo albero). Le OOB vengono quindi classificate all'interno degli alberi (circa un terzo del totale) in cui non sono state prese in considerazione durante la fase di creazione degli stessi. In questo modo le OOB fungono da test e vengono classificate dal modello: comparando la classificazione prevista per le OOB (o il valore previsto) con la vera classe (o vero valore) delle osservazioni stesse, si può effettuare una stima dell'errore del modello (chiamata, appunto, stima dell'errore OOB). Tale stima sarà definita come errore quadratico medio (OOB MSE, Mean squared error) per i problemi di regressione, oppure errore di classificazione per i problemi di classificazione, appunto. Si noti che ognuna delle n osservazioni del set di dati originale è OOB per circa un terzo degli alberi della foresta: tutte le n osservazioni quindi, oltre che generare il modello, hanno anche la funzione di testarlo. E' stato dimostrato che l'errore di previsione della foresta dipende da due elementi: dalla correlazione tra gli alberi nella foresta, direttamente proporzionale all'errore di previsione dalla forza di ogni singolo albero. Un albero con un basso errore previsionale è definito come classificatore forte. Aumentare la forza degli alberi diminuisce l'errore previsionale della foresta Ridurre m riduce sia la correlazione (riducendo l'errore) che la forza (aumentando l'errore): di default, il valore di m è impostato a `sqrt(p)`, ma è possibile ricercare un ottimo del parametro utilizzando la stima dell'errore OOB.

2.2.1.2 Importanza della variabile Il modello Random Forest, come spiegato, migliora l'accuratezza rispetto all'utilizzo di un albero singolo. Tuttavia, tale modello risulta più complicato da interpretare: per quanto riguarda l'importanza delle variabili nei problemi di regressione e classificazione, ad esempio, risulta difficile capire quali variabili di input siano più importanti. Nonostante ciò, è possibile ottenere un riassunto generale dell'importanza delle variabili di input/predittori utilizzando l'indicatore RSS (per i problemi di regressione) oppure l'indice Gini (per i problemi di classificazione). L'argoritmo, infatti, è in grado di calcolare il decremento totale di RSS (o indice Gini) dato dalla suddivisione su una data variabile (considerando l'intera foresta), mediato poi sul numero totale di alberi B.

2.2.1.3 Prossimità e altri strumenti del Random Forest Come vedremo successivamente, nelle applicazioni in R del pacchetto RandomForest, uno degli strumenti più utili del modello è quello delle prossimità. Dopo aver costruito ogni albero, tutti i dati del nostro set di osservazioni originale vengono trasferiti lungo i B alberi e possono essere calcolate le "similitudini" (prossimità) tra un'osservazione e un'altra: se due osservazioni/casi occupano lo stesso nodo terminale di un k-esimo albero, la loro vicinanza aumenta di uno – questo viene fatto per tutti i B alberi presenti nella foresta. Alla fine della corsa, le distanze/vicinanze vengono normalizzate dividendo per il numero di alberi: si ottiene quindi una matrice NxN (con N numero di osservazioni) al cui interno, per ogni coppia di osservazioni (n,k) viene indicato il valore di prossimità. Tale matrice può essere utilizzata per rimpiazzare valori mancanti in un set di dati/osservazioni, calcolare le distanze tra coppie di casi che possono essere utilizzate nel clustering o nell'individuazione di valori anomali, localizzare gli outlier, generare prototipi ("oggetti di riferimento" che forniscono informazioni sulla relazione tra le variabili e la classificazione) per ogni classe, e molto altro.

2.2.2 Vantaggi e difficoltà del Random Forest

Di seguito un elenco riassuntivo delle caratteristiche principali del modello Random Forest (con alcune comparazioni rispetto alle strutture ad albero decisionale singole).

VANTAGGI:

- Elevata precisione e affidabilità: riduzione della varianza e dell'errore grazie al Bagging e alla selezione casuale delle variabili in un sottoinsieme di dimensioni `m<<sqrt(p)`, per ogni interazione.
- Rischio di overfitting ridotto: le strutture ad albero decisionali (singole) corrono il rischio di overfitting, in quanto tendono ad adattarsi strettamente a tutti i campioni all'interno dei dati di addestramento. Questo rischio viene ridotto nelle foreste, grazie alla presenza di numerosi alberi non correlati, che riducono la varianza complessiva e l'errore di previsione.

- Funziona in modo efficiente su grandi basi di dati e può gestire migliaia di variabili di input senza eliminazione delle stesse
- Può fornire stime sull'importanza delle variabili nella classificazione
- Genera una stima interna dell'errore, grazie all'utilizzo delle osservazioni OOB come dati di test, man mano che la costruzione della foresta procede
- Flessibilità: il Random Forest è utilizzato sia per problemi di regressione che di classificazione
- Ha un metodo efficace per stimare i dati mancanti e mantiene l'accuratezza quando manca una buona parte dei dati
- Possono essere calcolati prototipi, per valutare le relazioni tra le variabili di input e la classificazione

DIFFICOLTA' PRINCIPALI:

- Più complesso: la previsione di una singola struttura ad albero decisionale è più facile da interpretare rispetto a una foresta di alberi.
- Richiede più risorse: poiché gli algoritmi Random Forest elaborano grandi set di dati, richiedono più risorse per l'archiviazione di tali dati.
- Richiede più tempo nell'elaborazione: dal momento che gli algoritmi Random Forest possono gestire grandi set di dati, possono fornire previsioni più accurate, ma possono essere lenti a elaborare i dati perché calcolano i dati per ogni singola struttura ad albero decisionale.

3 ALGORITMI E FUNZIONI PACCHETTO RandomForest

Dopo questa introduzione teorica, iniziamo con l'analisi del pacchetto RandomForest, che implementa tecniche per la creazione, analisi e modifica di modelli RandomForest in ambiente R. Partiamo con l'installazione di tale pacchetto: install.packages("randomForest"). Una volta installato, è necessario importare la libreria per poterlo utilizzare. Digitare quindi il seguente comando: `library(randomForest)`. All'interno del pacchetto RandomForest possiamo trovare diverse funzioni e famiglie di funzioni, tra cui le più importanti sono:

- randomForest
- predict.randomForest
- plot.randomForest
- classCenter
- combine, getTree, grow
- importance , varImpPlot
- margin
- na.roughfix, rfImpute
- outlier
- treesize
- tuneRF
- varUsed
- rfcv

3.1 randomForest

Si tratta della funzione principale del pacchetto: essa implementa l'algoritmo di Breiman per la creazione di un modello Random Forest per la classificazione o la regressione. La funzione può presentare diversi argomenti, e restituisce un oggetto di classe randomForest con diversi componenti utili per lo studio e l'analisi del modello. Gli argomenti principali (input) sono:

- x, formula: data frame o matrice contenente i predittori (variabili di input), oppure una formula che descrive il modello da valutare
- y: il vettore con le variabili di output. Se si tratta di fattori, viene assunta la classificazione, altrimenti viene assunta la regressione. La funzione può essere utilizzata anche senza specificare il vettore delle variabili di output: in questo caso il modello sarà "senza supervisione" (unsupervised) e servirà in sostanza solo per misurare le distanze e le prossimità tra le righe/osservazioni presenti in x
- data: nel caso in cui si utilizzi la formula – per specificare il modello. Si tratta del data frame che contiene le variabili presenti nel modello (e specificate in formula)
- mtry: numero di variabili campionate casualmente come candidate ad ogni suddivisione (si tratta del fattore m introdotto nel secondo capitolo). Se l'argomento mtry non è impostato manualmente, i valori di default sono diversi per la classificazione (`sqrt(p)` , dove p è il numero di variabili in x) e per la regressione (p/3)
- ntree: numero di alberi presenti nella foresta. Valore di default=500. Questo argomento può essere ottimizzato (come anche mtry), in fasi successive alla creazione del modello, per ridurre l'errore di previsione dello stesso
- importance: se impostata su TRUE, durante la creazione della foresta, viene calcolata l'importanza delle variabili di input
- proximity: se impostata su TRUE, calcola il valore di prossimità tra le varie osservazioni presenti in x

Altri argomenti, meno rilevanti ed utilizzati, sono:

- na.action: per specificare l'azione da intraprendere nel caso siano non siano disponibili alcuni valori (NA)

- xtest e ytest: due oggetti come x e y, ma questi – se specificati – sono utilizzati dalla funzione come test per l'accuratezza del modello (oltre al test eseguito di default dalla funzione, quello degli OOB spiegato nel capitolo 2)
- nodesize e maxnodes: per impostare limiti minimi e massimi sulla grandezza degli alberi e numero di nodi

La funzione randomForest presenta altri possibili argomenti, che non riportiamo in questo report in quanto poco utilizzati o poco rilevanti

```
randomForest (x, y=NULL,  xtest=NULL, ytest=NULL, ntree=500,
   mtry=if (!is.null(y) && !is.factor(y))
   max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
   weights=NULL,
   replace=TRUE, classwt=NULL, cutoff, strata,
   sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x)),
   nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
   maxnodes = NULL,
   importance=FALSE, localImp=FALSE, nPerm=1,
   proximity, oob.prox=proximity,
   norm.votes=TRUE, do.trace=FALSE,
   keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,
   keep.inbag=FALSE)
```

ESEMPIO DI FUNZIONE PER LA CLASSE FORMULA – DATABASE "IRIS":

```
randomForest(Species ~ ., data=iris, maxnodes=4, mtry=4, importance=TRUE)
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = iris, maxnodes = 4,      mtry = 4, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 4%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          3        47        0.06
```

ESEMPIO DI FUNZIONE DI DEFAULT (X E Y SPECIFICATI) – "DATABASE IRIS":

```
randomForest(iris[-1], iris[[1]], ntree=101, proximity=TRUE)
```

```
##
## Call:
##  randomForest(x = iris[-1], y = iris[[1]], ntree = 101, proximity = TRUE)
##                Type of random forest: regression
##                      Number of trees: 101
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.1412786
##                     % Var explained: 79.26
```

Come output, quindi, otteniamo un oggetto di tipo randomForest, che presenta numerosi componenti, tra cui i più importanti sono:

- call: la stringa di creazione della foresta
- type: regressione, classificazione o "senza supervisione"
- predicted: i valori previsti dei dati di input, utilizzando gli OOB
- importance: una matrice che, per ogni variabile, riporta la relativa misura di importanza della stessa (misure di importanza come spiegato nel capitolo 2)
- ntree: numero di alberi cresciuti nella foresta
- mtry: numero m di predittori utilizzati per ogni suddivisione
- proximity: se proximity=TRUE, viene calcolata una matrice di prossimità tra le varie osservazione/righe di input
- confusion (solo per classificazione): riporta la matrice di confusione, confrontando i valori di output reali delle osservazioni (y) e i valori predetti delle stesse utilizzati gli OOB. La matrice è utilizzata per calcolare il tasso di errore previsionale del modello
- err.rate (solo per classificazione): vettore dei tassi di errore previsionale. L'i-esimo elemento è il tasso di errore – calcolato con gli OOB – considerando tutti gli alberi fino all'i-esimo compreso. Tale componente risulta utile per ottimizzare il numero di alberi presente nella foresta (e minimizzare l'errore totale del modello stesso)
- mse (solo per regressione): vettore degli errori quadratici medi (come per err.rate, in funzione del numero di alberi)
- rsq (solo per regressione): "pseudo R-squared" `(1-mse)/(Var(y))`
- forest: una lista che contiene tutti gli alberi della foresta

L'oggetto randomForest presenta altri componenti, che non riportiamo in questo report in quanto poco utilizzati o poco rilevanti. Se l'oggetto randomForest viene richiamato in console (una volta creato), si potranno visualizzare velocemente alcune delle componenti descritte sopra: call, type, ntree, mtry, err.rate e confusion (oppure mse e rsq - le ultime due relative all'ntree-esimo albero).

3.2 predict.randomForest

La funzione predict, utilizzata anche per altri modelli di previsione, può essere usata anche con un oggetto randomForest. La funzione permette di prevedere l'output di un set di osservazioni (di test), dato un modello previsionale randomForest. L'oggetto (object) principale della funzione sarà un randomForest; il nuovo set di dati, di cui prevedere le risposte (newdata) dovrà essere specificato (matrice o dataframe) – se non specificato, vengono restituite le previsioni degli OOB del randomForest in oggetto. La funzione ammette anche altri argomenti, che per semplicità non specifichiamo in questo report (poco utilizzati o poco rilevanti).

```
predict(object, newdata)
```

L'output della funzione è, per i problemi di regressioni, un vettore di valori previsti. Per i problemi di classificazione, invece, l'output è un vettore di classi previste.

3.3 plot.randomForest

La funzione plot, applicata ad un modello randomForest, permette di visualizzare sotto forma di grafico i tassi di errore (per i problemi di classificazione) o gli scarti quadratici medi (MSE) in funzione del numero di alberi presenti nella foresta:

```
plot(x, type="l", main=deparse(substitute(x)), ...)
```

con x=oggetto di classe randomForest, type=tipo di grafico, main=titolo del grafico (+ eventuali altri parametri per il grafico [...]).

3.4 classCenter

La funzione classCenter permette di creare prototipi per ogni classe presente nel set di dati considerato.

```
classCenter(x, label, prox)
```

Data una matrice o data frame di input x, le etichette di classe di ogni riga in x (labels), e una matrice di prossimità prox (ricavata ad esempio durante la creazione del modello randomForest), la funzione restituisce una matrice con un prototipo per ogni riga/classe.

3.5 combine, getTree, grow

La funzione `combine(...)` unisce due o più insiemi di alberi in uno solo. Gli argomenti della funzione sono due o più oggetti randomForest che, combinati, restituiscono un unico modello randomForest. Le componenti confusion, err.rate, mse e rsq dell'albero restiuito saranno NULL. La funzione getTree permette di estrarre la struttura di un albero della foresta (sotto forma di matrice), indicando negli argomenti l'oggetto randomForest e il numero dell'albero da estrarre.

```
getTree(rfobj, k=1, labelVar=FALSE)
```

La funzione grow aggiunge un certo numero di alberi (how.many) ad un oggetto random Forest (x)

```
grow(x, how.many)
```

Le componenti confusion, err.rate, mse e rsq dell'albero restiuito saranno NULL (come per combine).

3.6 importance, varImpPlot

La funzione importance, dato un oggetto randomForest come input, estrae le misure di importanza per ogni variabile in termini di: diminuzione media della precisione – mean decrease in accuracy diminuzione media dell'impurità del nodo - mean decrease in node impurity

```
importance(x, type=NULL, class=NULL, scale=TRUE, ...)
```

Se scritta come sopra equivale, in sostanza, a richiamare la componente importance dell'oggetto randomForest, che restituisce una matrice con una riga per variabile ed una colonna per misura di importanza. Con type è possibile specificare quale misura di importanza considerare (una delle due o entrambe), con class c'è la possibilità di indicare una classe specifica (per i problemi di classificazione) secondo cui calcolare l'importanza delle variabili. Oltre che esportare le misure di importanza sotto forma di matrice (con la funzione importance), è possibile direttamente costruire un grafico dotchart delle misure di importanza grazie alla funzione varImpPlot (che, oltre agli argomenti sopra, può anche comprendere il numero di variabili da mostrare [nrow], se mostrarle in ordine di importanza o meno [sort], il nome del grafico [main], ...).

```
varImpPlot(x, sort=TRUE, n.var=min(30, nrow(x$importance)),
     type=NULL, class=NULL, scale=TRUE,
     main=deparse(substitute(x)), ...)
```

3.7 margin

Il margine di un dato punto/osservazione è definito come la proporzione di voti per la classe corretta (nell'intera foresta) meno il massimo tra le proporzioni di voti per le altre classi (solo per problemi di classificazione). Quindi: se un oggetto/punto/osservazione è stato classificato in maniera corretta da tutti gli alberi presenti nella foresta, il suo valore sarà 1; se maggiore di 0, vuol dire che la classificazione è stata corretta; se minore di 0, la classificazione è stata errata (considerando l'approccio di classificazione a "voto di maggioranza"). La funzione: `margin(x, ...)` con x oggetto randomForest – restituisce i margini per tutte le osservazioni del set di dati utilizzato per generare la foresta.

3.8 na.roughfix, rfImpute

Entrambe le funzioni na.roughfix e rfImpute vengono utilizzate per sostituire dati mancanti NA in matrici o data frame. La prima sostituisce il valore mancante NA calcolando la media della colonna in cui è presente il dato (quindi la media dei dati per la variabile considerata). L'argomento è una qualsiasi matrice o data frame, e l'output è il medesimo oggetto con i valori NA sostituiti con la media dei valori della colonna in considerazione (o, nel caso in cui la variabile sia di tipo factor, con la classe/fattore più frequente).

```
na.roughfix(object, ...)
```

La seconda, invece, è più accurata, in quanto utilizza l'algoritmo randomForest e la sua matrice di prossimità per sostituire i valori NA presenti in una matrice o dataframe (x) in maniera più precisa (y è il vettore delle risposte; come per la funzione random forest, è possibile indicare direttamente la formula y~x):

```
rfImpute(x, y, iter=5, ntree=300, ...)
```

In questo modo è possibile ottenere una stima accurata dei valori mancanti NA presenti in una matrice o data frame (che viene restituita dalla funzione con i valori – inizialmente NA - aggiornati).

3.9 outlier

La funzione outlier utilizza la matrice di prossimità di un modello randomForest (x) per verificare la presenza di outlier, calcolando una misura di outlying per ogni osservazione/dato. La funzione è utilizzabile solo per problemi di classificazione.

```
outlier(x, ...)
```

Essa restituisce un vettore con i valori di outlying per ogni osservazione del set di dati.

3.10 treesize

Questa funzione restituisce la grandezza (in termini di numero di nodi) di ogni albero presente nella foresta.

```
treesize(x, terminal=TRUE)
```

x è un oggetto di tipo randomForest, se terminal=TRUE vengono contati solo i nodi terminali (foglie), se terminal=FALSE vengono contati tutti i nodi. La funzione restituisce un vettore col numero di nodi per ogni albero presente nel modello randomForest.

3.11 tuneRF

La funzione tuneRF viene utilizzata per ottimizzare un modello randomForest: essa permette di diminuire l'errore OOB del randomForest, ricercando il valore di mtry ottimo in tal senso (parametro m).

```
tuneRF(x, y, mtryStart, ntreeTry=50, stepFactor=2, improve=0.05,
   trace=TRUE, plot=TRUE, doBest=FALSE, ...)
```

In cui gli argomenti sono:

- x: matrice o data frame con le variabili di input/predittive
- y: vettore delle risposte
- mtryStart: valore di partenza di mtry. Se non indicato, è di default come per la funzione randomForest
- ntreeTry: numero di alberi da utilizzare ad ogni step, per l'algoritmo di ottimizzazione
- stepFactor: ad ogni interazione, mtry incrementa o decrementa di questo valore
- improve: il miglioramento ad ogni interazione – in termini di errore OOB – deve essere almeno questo affinché il processo di ottimizzazione continui
- trace: se stampare o meno il processo di ricerca
- plot: se porre sotto forma di grafico l'errore OOB in funzione di mtry
- doBest: se creare una foresta con il valore di mtry ottimale trovato, o meno

Se doBest=TRUE, la funzione restituisce un oggetto randomForest ottimizzato con il parametro mtry trovato. Se doBest=FALSE, essa restituisce una matrice con due colonne: nella prima i valori di mtry trovati durante il processo di ottimizzazione, nella seconda il valore di errore OOB per ognuno di questi valori.

3.12 varUsed

Questa funzione permette di verificare quali variabili predittive sono state usate nella foresta. Se by.tree=TRUE, l'informazione viene riportata separatamente per ogni angolo della foresta; se count=TRUE, viene anche conteggiato il numero di volte che la variabile è stata utilizzata nella foresta (o, se by.tree=TRUE, in ogni singolo albero).

```
varUsed(x, by.tree=FALSE, count=TRUE)
```

In base a quanto indicato in by.tree e in count, la funzione può restituire un vettore o una matrice con le informazioni descritte sopra. Nell'esempio (default), la funzione restituisce un vettore con le frequenze di utilizzo delle variabili nell'intera foresta (se by.tree fosse TRUE, la funzione restituirebbe una matrice, con le frequenze indicate per ogni variabile e per ogni albero.

3.13 rfcv

La funzione rfcv sta per "Random Forest Cross Validation" e permette di calcolare le performance predittive dei modelli random forest per mezzo della procedura di cross-validation. Tale funzione, che ha come argomenti principali `trainx` e `trainy` (matrice con i valori relativi ai predittori e variabile dipendente), restituisce gli errori (o MSE) per ogni step / numero di variabili utilizzati, permettendo quindi di individuare il modello con numero di variabili ottimo.

4 ESEMPI APPLICATIVI

4.1 ESEMPIO APPLICATIVO DI CLASSIFICAZIONE: Titanic dataset

4.1.1 Introduzione al dataset

Lo scopo di questo primo esercizio è quello di predire la probabilità di sopravvivenza di ciascuno dei passeggeri a bordo del Titanic in base a determinate caratteristiche e fattori. L'affondamento del Titanic è un evento famoso e continuano a essere pubblicati nuovi libri sull'argomento. Molti fatti ben noti – dalle proporzioni dei passeggeri di prima classe alla politica "donne e bambini prima", e il fatto che tale politica non ebbe del tutto successo nel salvare le donne e i bambini della terza classe – si riflettono nella sopravvivenza dipendente dalle tariffe per le varie classi di passeggeri.

Questi dati furono originariamente raccolti dal British Board of Trade durante le indagini sull'affondamento. Si noti che non esiste un completo accordo tra le fonti primarie sul numero esatto di persone a bordo, salvate o disperse.

Kaggle ha creato un dataset in R contenente informazioni riguardo il naufragio. Per ogni passeggero vengono riportate le seguenti informazioni:

- Pclass: rappresenta la classe di appartenenza del passeggero all'interno della nave

o 1 = prima classe

o 2 = seconda classe

o 3 = terza classe

- Sex: sesso
- Age: età
- Sibsp: numero di fratelli/sorelle e coniugi a bordo
- Parch: numero di genitori/figli a bordo
- Fare: cifra pagata dal passeggero per il biglietto
- Embarked: città nella quale i viaggiatori si sono imbarcati

o C = Cherbourg

o Q = Queenstown

o S = Southampton

Per eseguire la simulazione sarà necessario il pacchetto randomForest:

```
library("ggplot2")
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
library("lattice")
```

```
library("randomForest")
```

Una volta caricati i pacchetti possiamo caricare i nostri dataset, utilizzato la funzione read.table. I dati sono stati scaricati dalla pagina web: https://www.kaggle.com/c/titanic/overview

I dati sono stati suddivisi in due gruppi:

- train.csv: set di training utilizzato per creare il modello predittivo; per ogni passeggero vengono fornite informazioni quali la classe, il sesso, l'età, la tariffa pagata e la sopravvivenza.
- test.csv: set utilizzato per verificare l'accuratezza del modello su dei dati non conosciuti. Per questo motivo il test set non contiene informazioni sulla sopravvivenza o meno del passeggero, in quanto sarà compito del nostro algoritmo determinarlo.

A questo punto passiamo alle lettura dei file: Partiamo innanzitutto dalla lettura della tabella di train utilizzando la funzione read.table

```
train <- read.table('train.csv',sep=",",header=TRUE)
```

A questo punto passiamo alla lettura della tabella di test, utilizzando la medesima funzione di lettura read.table

```
test <- read.table('test.csv',sep=",",header=TRUE)
```

La variabile Survived, come visto in precedenza essere binaria, può assumere due valori (0 se la persona è deceduta, 1 se la persona è sopravvissuta); utilizzando la funzione as.factor() trasformiamo la variabile Survived contenuta nel dataset di train in una variabile categorica.

```
train$Survived <- as.factor(train$Survived)
```

Utilizzando la funzione summary() possiamo analizzare alcune interessanti informazioni sui due dataset; per ogni variabile (colonna), otteniamo valore minimo e massimo, media e mediana, oltre che la divisione in quartili. Queste informazioni sono riportate di seguito.

```
summary(train)
```

```
##   PassengerId    Survived     Pclass          Name               Sex
##  Min.   :  1.0   0:549    Min.   :1.000   Length:891         Length:891
##  1st Qu.:223.5   1:342    1st Qu.:2.000   Class :character   Class :character
##  Median :446.0            Median :3.000   Mode  :character   Mode  :character
##  Mean   :446.0            Mean   :2.309
##  3rd Qu.:668.5            3rd Qu.:3.000
##  Max.   :891.0            Max.   :3.000
##
##       Age            SibSp           Parch           Ticket
##  Min.   : 0.42   Min.   :0.000   Min.   :0.0000   Length:891
##  1st Qu.:20.12   1st Qu.:0.000   1st Qu.:0.0000   Class :character
##  Median :28.00   Median :0.000   Median :0.0000   Mode  :character
##  Mean   :29.70   Mean   :0.523   Mean   :0.3816
##  3rd Qu.:38.00   3rd Qu.:1.000   3rd Qu.:0.0000
##  Max.   :80.00   Max.   :8.000   Max.   :6.0000
##  NA's   :177
##       Fare           Cabin             Embarked
##  Min.   :  0.00   Length:891         Length:891
```

```
##  1st Qu.:  7.91   Class :character   Class :character
##  Median : 14.45   Mode  :character   Mode  :character
##  Mean   : 32.20
##  3rd Qu.: 31.00
##  Max.   :512.33
##
```

Come possiamo vedere la variabile Survived è stata convertita in variabile categorica binaria (0 o 1).

Eseguiamo il medesimo procedimento per il dataset di test:

```
summary(test)
```

```
##   PassengerId         Pclass          Name               Sex
##  Min.   : 892.0   Min.   :1.000   Length:418         Length:418
##  1st Qu.: 996.2   1st Qu.:1.000   Class :character   Class :character
##  Median :1100.5   Median :3.000   Mode  :character   Mode  :character
##  Mean   :1100.5   Mean   :2.266
##  3rd Qu.:1204.8   3rd Qu.:3.000
##  Max.   :1309.0   Max.   :3.000
##
##       Age             SibSp            Parch           Ticket
##  Min.   : 0.17   Min.   :0.0000   Min.   :0.0000   Length:418
##  1st Qu.:21.00   1st Qu.:0.0000   1st Qu.:0.0000   Class :character
##  Median :27.00   Median :0.0000   Median :0.0000   Mode  :character
##  Mean   :30.27   Mean   :0.4474   Mean   :0.3923
##  3rd Qu.:39.00   3rd Qu.:1.0000   3rd Qu.:0.0000
##  Max.   :76.00   Max.   :8.0000   Max.   :9.0000
##  NA's   :86
##       Fare            Cabin              Embarked
##  Min.   :  0.000   Length:418         Length:418
##  1st Qu.:  7.896   Class :character   Class :character
##  Median : 14.454   Mode  :character   Mode  :character
##  Mean   : 35.627
##  3rd Qu.: 31.500
##  Max.   :512.329
##  NA's   :1
```

A questo punto disponiamo di entrambi i set di dati; potrebbe essere pertanto interessante effettuare un'analisi comparativa tra i due.

La prinicipale differenza tra il training set e il test set sta nel fatto che il secondo non contiene la colonna "Survived", in quanto sarà compito del nostro modello predittivo stabilire, per ogni persona, se questa sia sopravvisuta o meno al naufragio.

A questo punto possiamo iniziare a costruire il nostro modello. La cosa più importante nella costruzione di un modello sta nello scegliere le variabili più importanti, ovvero, in questo caso, quelle con una più elevata correlazione con la probabilità di sopravvivenza del passeggero. Prima di tutto analizziamo i dati in nostro possesso, partendo con il numero di sopravvisuti.

```
prop.table(table(train$Survived))
```

```
##
##         0         1
## 0.6161616 0.3838384
```

Come possiamo vedere oltre il 60% dei passeggeri a bordo non è sopravvisuta al naufragio. Nel caso supponessimo che tutti i passeggeri fossero morti otterremmo una precisione del 61,6%.

Lo stesso procedimento può essere fatto per le altre variabili del data set; analizziamo ad esempio la Pclass (classe di appartenenza) e il sesso dei passeggeri

```
prop.table(table(train$Pclass))
```

```
##
##         1         2         3
## 0.2424242 0.2065095 0.5510662
```

Circa il 55% dei passeggeri totali a bordo del Titanic si trovava nella terza classe, teoricamente più svantaggiata rispetto alle altre in ottica di probabilità di sopravvivenza.

Analizziamo ora la distribuzione dei sessi all'interno della nave:

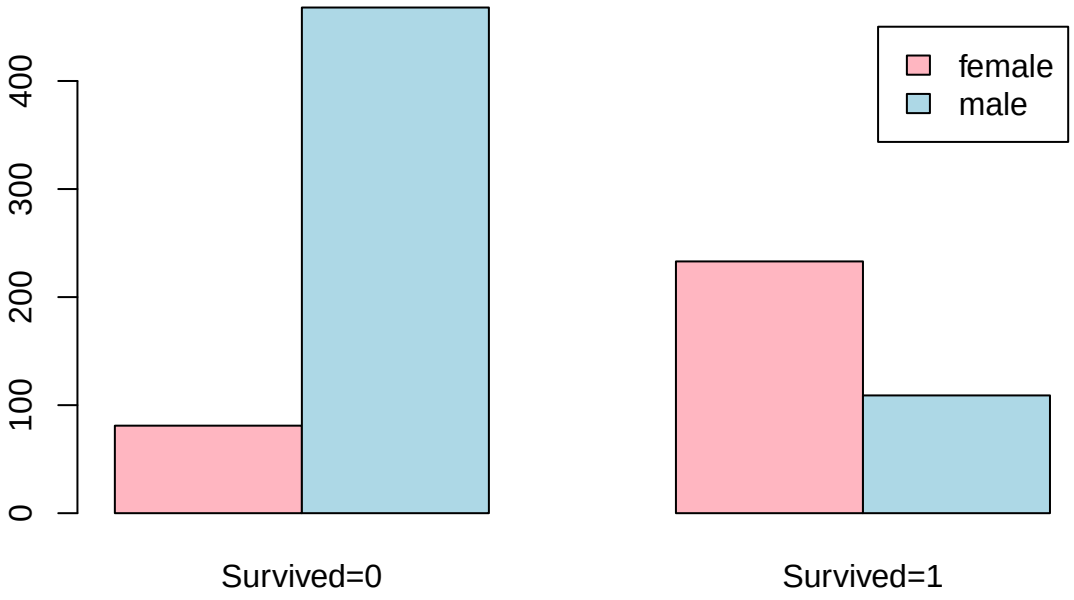
```
prop.table(table(train$Sex))
```

```
##
##   female     male
## 0.352413 0.647587
```

Notiamo come quasi il 65% dei passeggeri fosse di sesso maschile.

Ora, utilizzando dei grafici a barre, analizziamo la presenza di un'eventuale differenza nella probabilità di sopravvivenza tra uomini e donne:

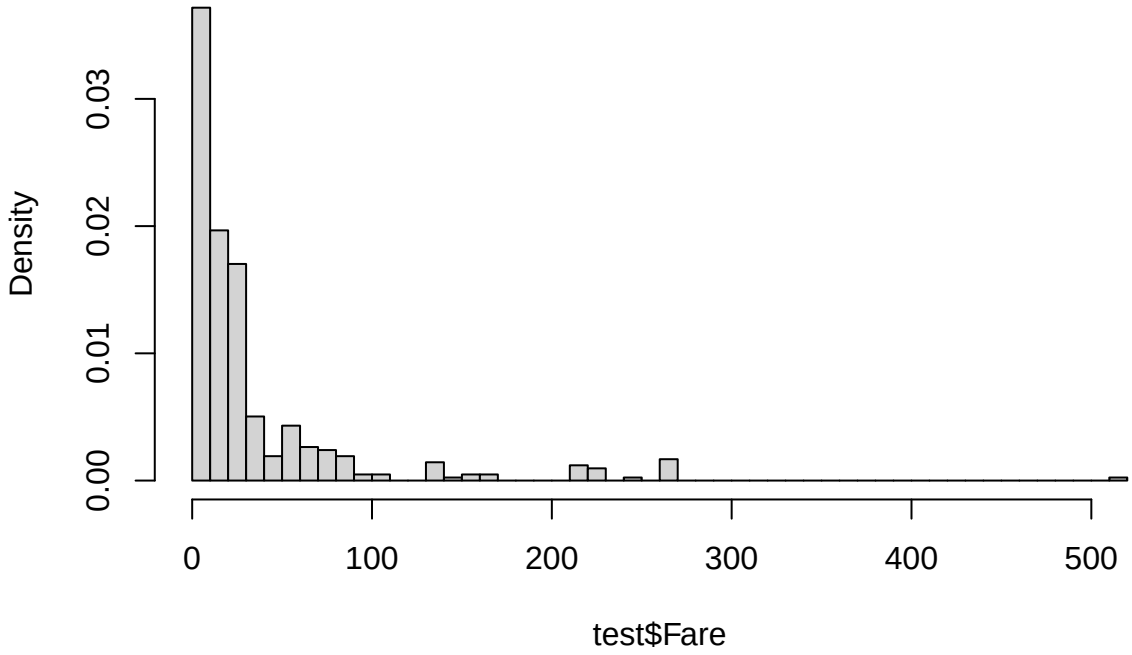
```
barplot(table(train$Sex, train$Survived), col = c("lightpink", "lightblue"),
        beside = TRUE, legend = c("female", "male"),
        names.arg = c("Survived=0", "Survived=1"))
```



Possiamo notare come le probabilità di sopravvivenza sembrino molto più elevate per il sesso femminile. Questo può essere uno spunto importante per la nostra analisi.

Analizziamo ora la Fare, ovvero la tariffa pagata da ciascun passeggero per imbarcarsi sulla nave.

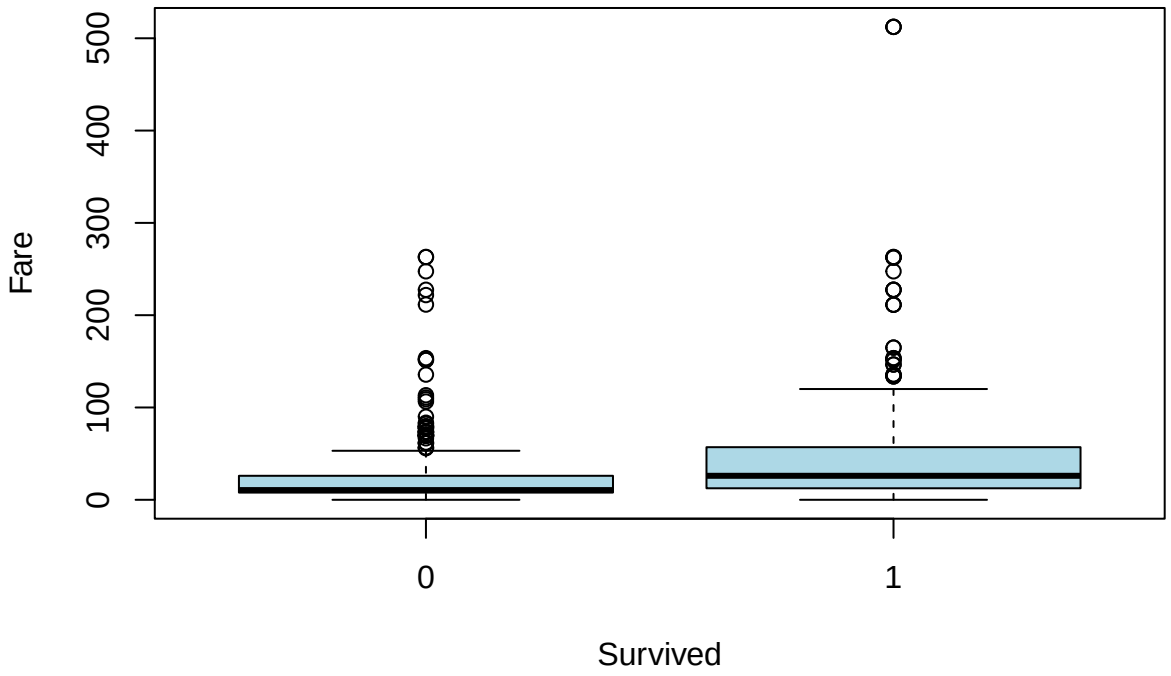
```
hist(test$Fare, main="", probability=TRUE, breaks=50)
```



Dal grafo notiamo come la variabile Fare sia molto concentrata verso valori bassi, con diversi outlier.

Andiamo ora ad analizzare l'impatto della variabile Fare sulla probabilità di sopravvivenza dei passeggeri, costruendo un boxplot che metta in relazione la probabilità di sopravvivenza con la variabile Fare:

```
boxplot(train$Fare ~ train$Survived, data = train, col = "lightblue",
        main = "", xlab = "Survived", ylab = "Fare")
```



Dal grafico precedente possiamo cogliere alcune importanti informazioni; per quanto riguarda le persone

decedute nel naufragio (Survived=0) i valori di Fare (prezzo pagato per il biglietto) sono inferiori rispetto a quelli relativi alle persone sopravvissute al naufragio (Survived=1); l'estremo superiore per le persone decedute si attesta sensibilmente sotto il valore 100, mentre l'estremo superiore per le persone sopravvissute si attesta sensibilmente al di sopra del valore 100. Per un'analisi più approfondita andiamo a studiare i valori delle due mediane:

```
mediana <- tapply(train$Fare, train$Survived, median)
print(paste("Mediana:", mediana))
```

```
## [1] "Mediana: 10.5" "Mediana: 26"
```

Come vediamo per Survived=0 la mediana vale 10.5, mentre per quanto riguarda Survived=1 la mediana vale 26 (più del doppio). L'analisi ci sembra dunque suggerire come al crescere del prezzo del biglietto cresca anche la probabilità di sopravvivenza.

Andiamo ora ad analizzare nel dettaglio i dati contenuti nella variabile Fare:

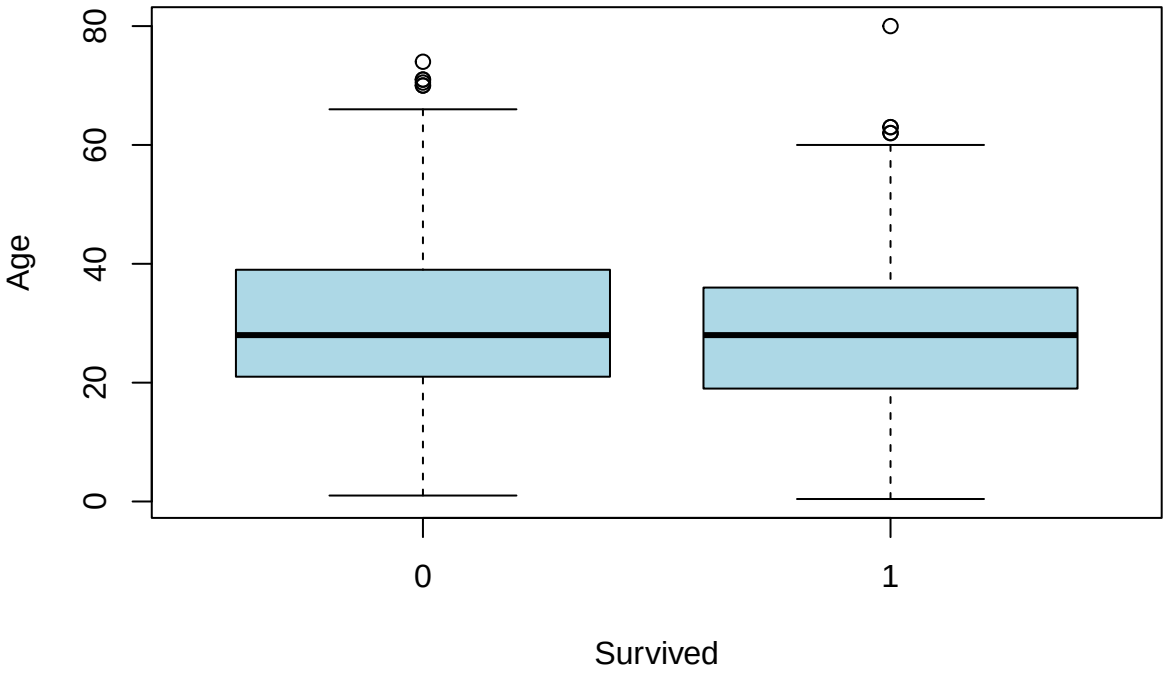
```
summary(train$Fare)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    7.91   14.45   32.20   31.00  512.33
```

Utilizzando la funzione summary su "Fare" notiamo che questa non presenta nessun valore mancante (NA's), pertanto i dati ad esso associati sono completi ed affidabili. "Fare" sembra dunque essere una variabile importante per determinare "Survived" e sarà sicuramente importante all'interno del nostro modello predittivo.

Analizziamo ora la relazione tra età e survived

```
boxplot(train$Age ~ train$Survived, data = train, col = "lightblue",
        main = "", xlab = "Survived", ylab = "Age")
```



Osservando i due boxplot notiamo come questi sembrino disporsi in maniera molto simile per quanto riguarda i due valori di Survived; inoltre le due mediane sembrano essere praticamente corrispondenti. Da queste osservazioni si può dedurre come la variabile Age non sia troppo importante nel determinare la probabilità di sopravvivenza di un passeggero.

Andiamo ora ad analizzare nel dettaglio i dati contenuti nella variabile Age:

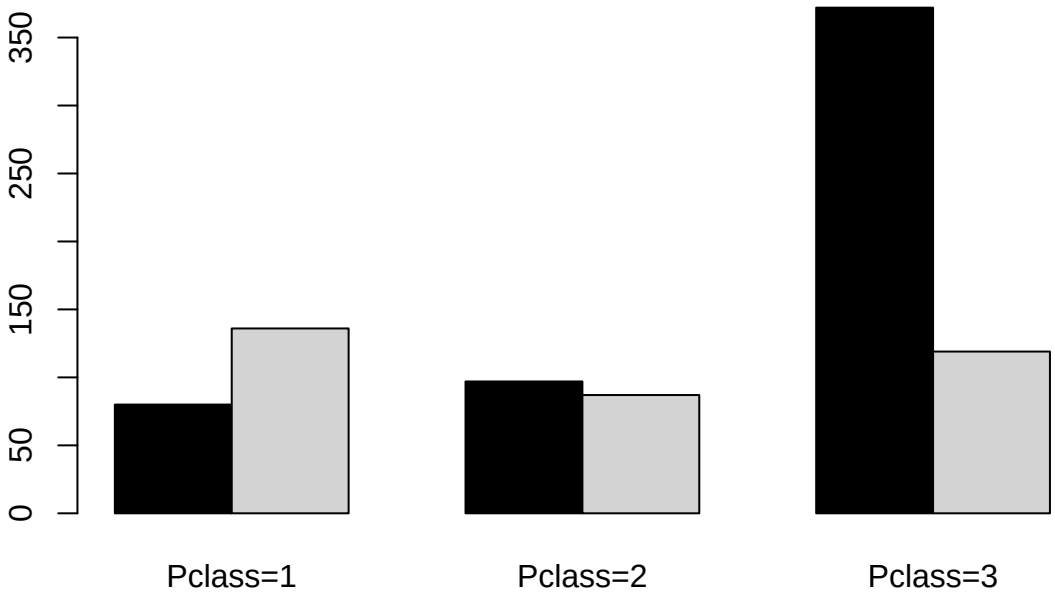
```
summary(train$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.42   20.12   28.00   29.70   38.00   80.00     177
```

Notiamo che l'età contiene un numero molto elevato di NA's (177). Tralasciamo dunque la variabile età in quanto, oltre a non avere grande impatto sulla probabilità di sopravvivenza, contiene anche molti dati mancanti.

Passiamo ora all'analisi dell'impatto della variabile Pclass sulla probabilità di sopravvivenza dei passeggeri. Andiamo a costruire un barplot che metta in relazione la frequenza di sopravvivenza - non sopravvissuti in nero e sopravvissuti in grigio - con la variabile Pclass:

```
barplot(table(train$Survived, train$Pclass), col = c("black", "lightgrey"),
        beside = TRUE, names.arg = c("Pclass=1", "Pclass=2", "Pclass=3"))
```



Possiamo notare una notevole differenza nel tasso di sopravvivenza, per le diverse classi: nel caso di classe 3, infatti, gran parte dei passeggeri NON sopravvive, nel caso di classe 2 la distribuzione è piuttosto simmetrica mentre, per classe 1, buona parte dei passeggeri sopravvive. In sintesi, quindi, i passeggeri che hanno meno probabilità di sopravvivenza sono quelli di classe 3.

Analizziamo nel dettaglio i dati contenuti nella variabile Pclass:

```
summary(train$Pclass)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   2.309   3.000   3.000
```

Utilizzando la funzione summary su "Pclass" notiamo che questa non presenta nessun valore mancante (NA's), pertanto i dati ad esso associati sono completi ed affidabili. "Pclass" sembra dunque essere una variabile importante per determinare "Survived".

4.1.2 RandomForest Package

A questo punto possiamo allenare il nostro modello utilizzando l'algoritmo di RandomForest, e visualizziamo anche le sue caratteristiche principali. Utilizziamo quindi la funzione 'randomForest()' per la creazione della foresta:

```
set.seed(51)
model1 <- randomForest(Survived ~ ., data=train, na.action=na.omit, prox=TRUE)
model1
```

```
##
## Call:
##  randomForest(formula = Survived ~ ., data = train, prox = TRUE,      na.action = na.omit)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 17.79%
## Confusion matrix:
##     0   1 class.error
## 0 375  49   0.1155660
## 1  78 212   0.2689655
```

Come si può notare dall'output, possiamo notare che:

- il modello è di classificazione (il problema, infatti, è binario)
- il numero di alberi presenti nella foresta è quello di default: 500
- il numero di variabili predittive m utilizzate in ogni albero e in ogni split è anch'esso quello di defalut: sqrt(p), per difetto = 3
- l'errore di classificazione - calcolato utilizzando gli OOB - è del 17,79%. Viene riportata nell'output anche la 'Confusion matrix'

Per la creazione di questo modello (model1), abbiamo considerato tutte le variabili presenti nel dataset. Ovviamente, alcune di queste variabili risultano superflue o non rilevanti per la previsione e la costruzione del modello: Name, PassengerId, Ticket ecc. Proviamo a costruire un modello con le variabili che, da analisi preliminare, sembrerebbero essere più impattanti sulla variabile di risposta:

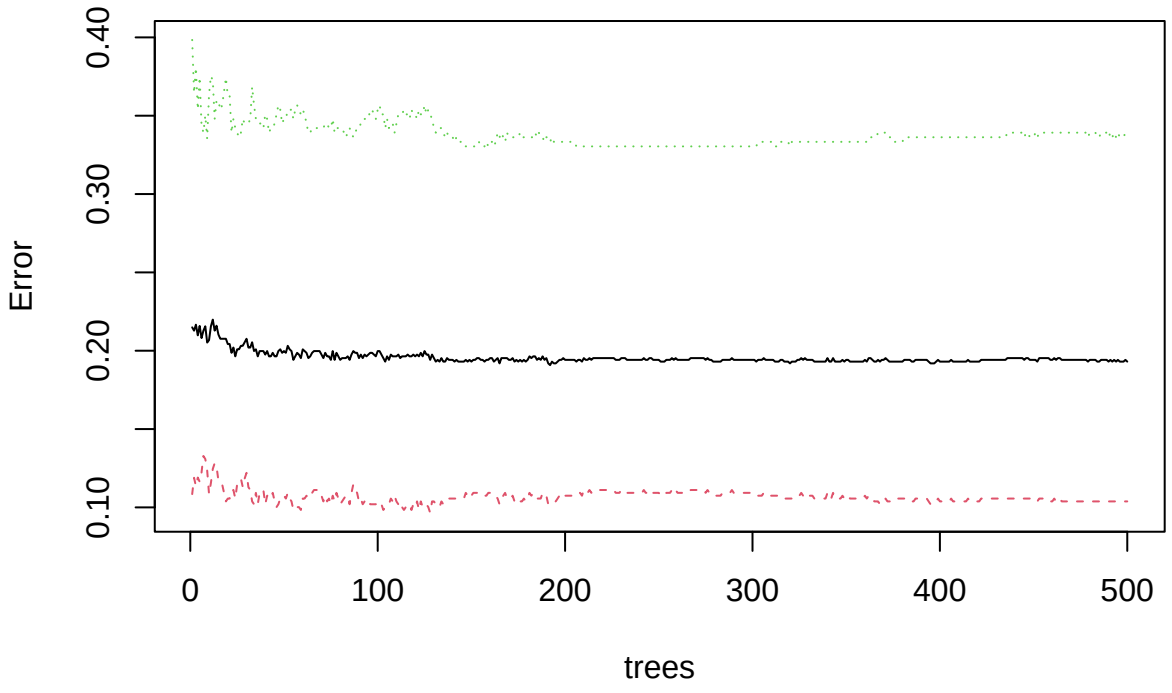
```
set.seed(51)
model2 <- randomForest(Survived ~ Pclass + Sex + Fare, data = train,
                       na.action=na.omit, prox=TRUE)
model2
```

```
##
## Call:
##  randomForest(formula = Survived ~ Pclass + Sex + Fare, data = train,      prox = TRUE, na.action = n
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 19.3%
## Confusion matrix:
##     0   1 class.error
## 0 492  57   0.1038251
## 1 115 227   0.3362573
```

Come si nota dall'output, la stima dell'errore OOB è molto simile a quella del modello con tutte le variabili, a prova del fatto che alcune variabili inizialmente considerate fossero superflue in fase di analisi e non influenti sulla probabilità di sopravvivenza dei passeggeri: possiamo quindi utilizzare quest'ultimo modello per le analisi predittive.

Prima di fare questo, analizziamo l'andamento dell'errore in funzione del numero di alberi all'interno della fortesta. Utilizziamo la funzione 'plot.randomForest()'.

```
plot(model2, main ="")
```



- In verde: errore di classificazione per la classe "sopravvissuti - 1" (percentuale di falsi negativi).
- In rosso: errore di classificazione per la classe "non sopravvissuti - 0" (percentuale di falsi positivi).
- In nero: stima totale dell'errore OOB.

Qual è il numero minimo di alberi nella foresta, che permette di minimizzare l'errore di classificazione OOB? Lo possiamo scoprire puntualmente con questa funzione:

```
which.min(model2$err.rate[,1])
```

```
## [1] 192
```

Con 192 alberi, la foresta è ottimizzata, con un errore OOB di:

```
model2$err.rate[which.min(model2$err.rate[,1])]
```

```
## [1] 0.1907969
```

Andiamo quindi a creare la nuova foresta ottimizzata:

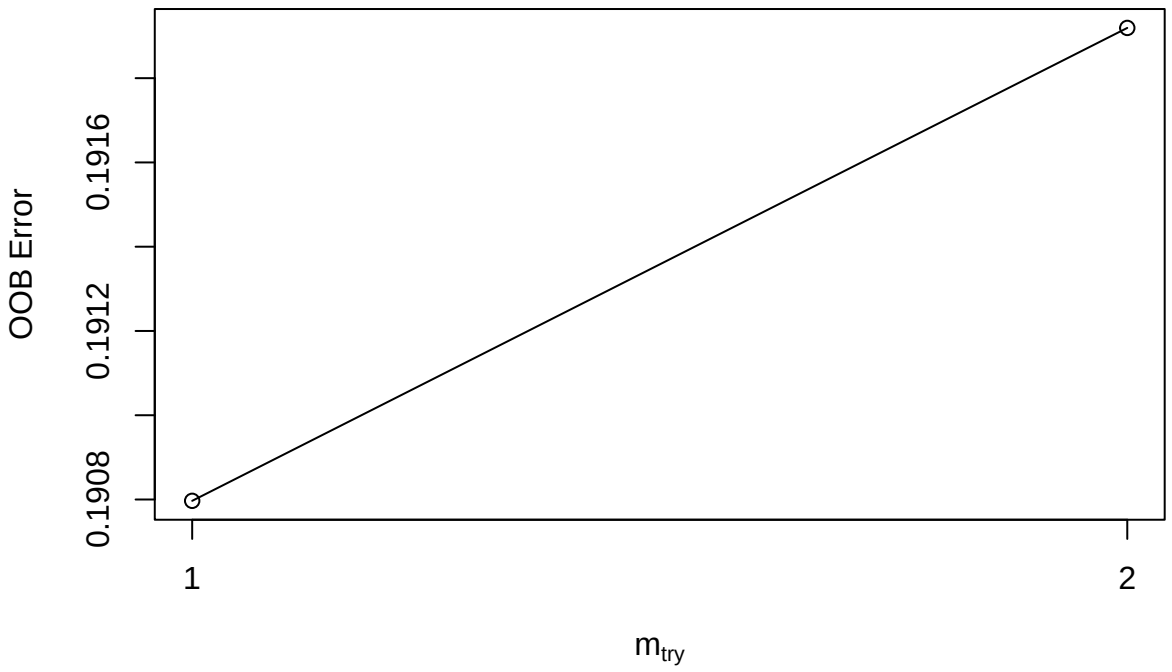
```
set.seed(51)
model_ott <- randomForest(Survived ~ Pclass + Sex + Fare,
                          data = train, ntree=192,
                          na.action=na.omit, prox=TRUE)
model_ott
```

```
##
## Call:
##  randomForest(formula = Survived ~ Pclass + Sex + Fare, data = train,      ntree = 192, prox = TRUE,
##                Type of random forest: classification
##                      Number of trees: 192
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 19.08%
## Confusion matrix:
##     0   1 class.error
## 0 493  56   0.1020036
## 1 114 228   0.3333333
```

Un'altra cosa che possiamo notare dall'output è che il numero di variabili mtry prese in considerazione ad ogni split è 1 (valore di default): è possibile ottimizzare tale parametro? Proviamo con la funzione `tuneRF()`:

```
set.seed(51)
tuneRF(train[,c(3,5,10)], train$Survived, ntree=192)
```

```
## mtry = 1  OOB error = 19.08%
## Searching left ...
## Searching right ...
## mtry = 2     OOB error = 19.19%
## -0.005882353 0.05
```

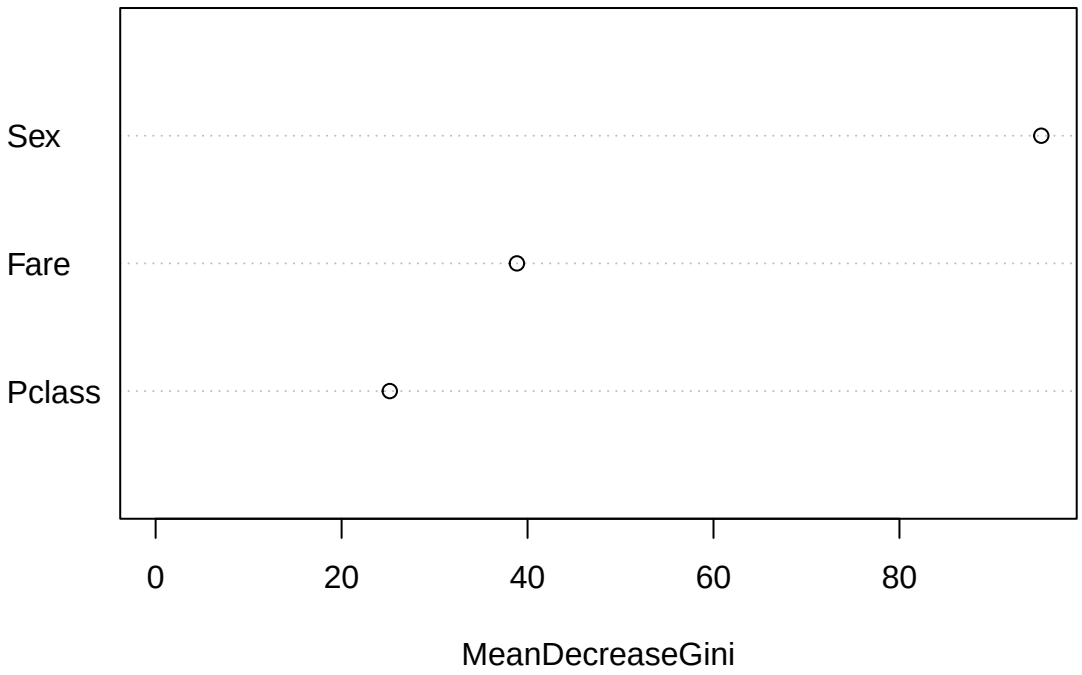



```
##       mtry  OOBError
## 1.OOB    1 0.1907969
## 2.OOB    2 0.1919192
```

Come possiamo notare dall'output, il parametro 1 è il parametro ottimo: dunque non è necessario modificarlo.

Una volta ottenuto il modello ottimizzato finale, risulta interessante - a questo punto - effetuare alcune analisi del dataset e del modello. Iniziamo col capire quali siano le variabili più importanti per le analisi predittive. Utilizziamo la funzione 'varImpPlot()':

```
varImpPlot(model_ott, main ="")
```



Come notato anche visivamente durante le indagini preliminari, il genere dei passeggeri risulta essere la variabile più importante, seguita dal prezzo e dalla classe.

Prima di procedere con le analisi predittive (con il modello ottimizzato), utilizziamo anche la funzione classCenter per creare il "prototipo" della persona sopravvissuta, per dimostrare le potenzialità della funzione 'classCenter()' presente nel pacchetto randomForest:

```
titanic.p <- classCenter(train[,c(3,5,10)], train$Survived, model2$prox)
titanic.p
```

```
##   Pclass Sex      Fare
## 0 "3"    "male"   " 7.8958"
## 1 "2"    "female" " 29.0000"
```

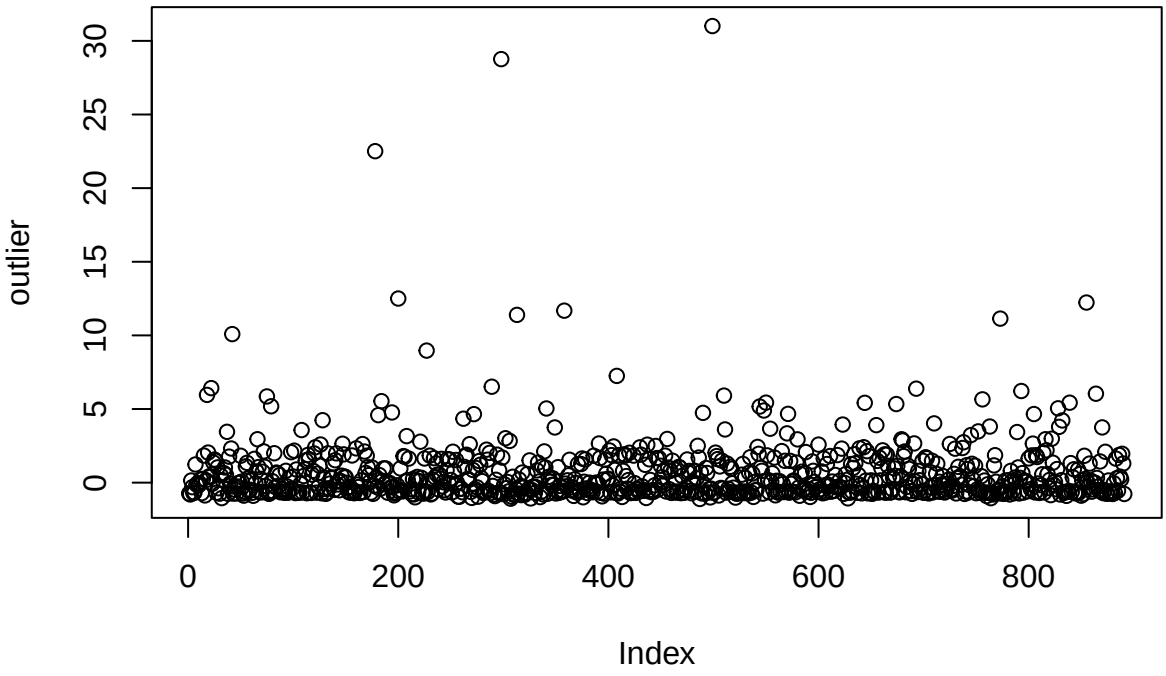
La persona "tipo" che sopravvive all'affondamento della nave è una donna (come ci aspettavamo anche dalle analisi preliminari), presente in classe 2 e che ha pagato 29 euro per il biglietto.

La persona con minor possibilità di sopravvivenza , invece, è un maschio in classe 3 e che ha pagato 7,90 euro per il biglietto.

Come ultima analisi del modello, e del dataset in generale, proviamo ad analizzare la presenza di alcuni outlier o osservazioni non in linea con quanto analizzato precedentemente. Utilizziamo la peculiare funzione del pacchetto chiamata, appunto, 'outlier()'. Tale funzione restituisce un valore di outlying, come spiegato nella parte di teoria (solo per i problemi di classificazione come questo).

Plottiamo quindi tali valori per ogni osservazione:

```
outlier <- outlier(model_ott)
plot(outlier)
```



Si può notare graficamente come alcune osservazioni possano sembrare degli outlier, con valori fino a 30. Ad esempio, l'osservazione con il valore più alto è:

```
outlier[which.max(outlier)]
```

```
##      499
## 31.00707
```

Tale osservazione (numero 499) è la seguente:

```
train[499,]
```

```
##     PassengerId Survived Pclass                                          Name
## 499         499        0      1 Allison, Mrs. Hudson J C (Bessie Waldo Daniels)
##        Sex Age SibSp Parch Ticket   Fare   Cabin Embarked
## 499 female  25     1     2 113781 151.55 C22 C26        S
```

Osservando la riga, si nota come questa evidentemente non sia in linea con quanto affermato precedentemente, e sia quindi un outlier. Si tratta infatti di:

- una donna
- in prima classe
- prezzo del biglietto elevato

Da quanto osservato con le indagini preliminari e grazie al protoripo (classCenter), ci aspetteremmo che questo sia un "profilo tipo" di sopravvissuto; invece, come possiamo vedere, survived=0. Si tratta di un outlier.

Gli esempi sopra sono stati effettuati per mostrare le potenzialità del randomForest e del pacchetto in generale, nel contesto di analisi di un dataset.

Un altro pacchetto utilizzabile per la creazione di modelli randomForest e relative analisi è il pacchetto "caret", utilizzabile anche per altri modelli.

In termini esemplificativi, utilizziamo ora il pacchetto caret per un'analisi comparativa tra un modello randomForest e un modello di regressione logistica, in termini di accuratezza. Iniziamo caricando il pacchetto:

```
library(caret)
```

Creiamo ora un modello Random Forest ottimizzato tramite cross-validation (10-fold cross validation):

```
set.seed(51)
modelloRF <- train(Survived ~ Pclass + Sex + Fare, data = train, method="rf",
                   trControl=trainControl(method="cv", number=10))
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

```
modelloRF
```

```
## Random Forest
##
## 891 samples
##   3 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 802, 803, 801, 802, 802, 802, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8103575  0.5837016
##   3     0.8181716  0.6088334
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 3.
```

L'accuratezza del modello, calcolata con cross-validation, è di circa l'82%.

Ripetiamo ora la procedura per un modello di regressione logistica glm:

```
set.seed(51)
modelloGLM <- train(Survived ~ Pclass + Sex + Fare, data = train, method="glm",
                    trControl=trainControl(method="cv", number=10))
modelloGLM
```

```
## Generalized Linear Model
##
## 891 samples
##   3 predictor
```

```
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 802, 803, 801, 802, 802, 802, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7834031  0.5365933
```

Come possiamo vedere, l'accuratezza del modello di regressione logistica (78% circa) è inferiore rispetto a quella del modello randomForest.

Viene confermato, quindi, quanto spiegato nella parte teorica del report: il modello randomForest - piuttosto semplice da utilizzare - ha anche una buonissima capacità predittiva, superiore a quella di altri modelli.

Procediamo ora - quindi - con alcune analisi previsionali, utilizzando il dataset di test e la funzione 'predict.randomForest()' del pacchetto randomForest. Come possiamo vedere sotto, è possibile utilizzare il modello per prevedere l'esito dell'affondamento della nave per altri profili (newdata):

```
test$Survived <- predict(model_ott, newdata = test)
head(test)
```

```
##   PassengerId Pclass                                          Name    Sex  Age
## 1         892      3                              Kelly, Mr. James   male 34.5
## 2         893      3              Wilkes, Mrs. James (Ellen Needs) female 47.0
## 3         894      2                     Myles, Mr. Thomas Francis   male 62.0
## 4         895      3                              Wirz, Mr. Albert   male 27.0
## 5         896      3 Hirvonen, Mrs. Alexander (Helga E Lindqvist) female 22.0
## 6         897      3                    Svensson, Mr. Johan Cervin   male 14.0
##   SibSp Parch  Ticket    Fare Cabin Embarked Survived
## 1     0     0  330911  7.8292              Q        0
## 2     1     0  363272  7.0000              S        1
## 3     0     0  240276  9.6875              Q        0
## 4     0     0  315154  8.6625              S        0
## 5     1     1 3101298 12.2875              S        1
## 6     0     0    7538  9.2250              S        0
```

Abbiamo quindi aggiunto al dataset di test una colonna con l'esito della previsione: il nostro modello attribuisce per ciascun passeggero un valore pari a 0 (nel caso in cui si prevede che il passeggero sia deceduto) o pari a 1 1 (nel caso in cui il modello preveda che il passeggero sia sopravvissuto).

Per questo dataset(test), non abbiamo a disposizione i valori reali della variabile "Survived"; ci aspettiamo, tuttavia, che questi siano stati previsti dal nostro modello, e dalla funzione `predict()` con un'accuratezza totale attorno all'81% (19% di errore del modello model_ott).

4.2 ESEMPIO APPLICATIVO DI REGRESSIONE: dataset imports85

4.2.1 Introduzione al dataset

Il set di dati imports85 è compreso nel pacchetto randomForest: utilizzeremo proprio questo dataset per presentare un esempio riguardante le potenzialità del pacchetto, applicandolo ad un problema di regressione.

Cominciamo richiamando il pacchetto RandomForest e visualizzando parte del dataset, contenuto nel pacchetto stesso:

```
library(randomForest)
data("imports85")
head(imports85, 4)
```

```
##   symboling normalizedLosses        make fuelType aspiration numOfDoors
## 1         3               NA alfa-romero      gas        std        two
## 2         3               NA alfa-romero      gas        std        two
## 3         1               NA alfa-romero      gas        std        two
## 4         2              164        audi      gas        std       four
##     bodyStyle driveWheels engineLocation wheelBase length width height
## 1 convertible         rwd          front      88.6  168.8  64.1   48.8
## 2 convertible         rwd          front      88.6  168.8  64.1   48.8
## 3   hatchback         rwd          front      94.5  171.2  65.5   52.4
## 4       sedan         fwd          front      99.8  176.6  66.2   54.3
##   curbWeight engineType numOfCylinders engineSize fuelSystem bore stroke
## 1       2548       dohc           four        130       mpfi 3.47   2.68
## 2       2548       dohc           four        130       mpfi 3.47   2.68
## 3       2823       ohcv            six        152       mpfi 2.68   3.47
## 4       2337        ohc           four        109       mpfi 3.19   3.40
##   compressionRatio horsepower peakRpm cityMpg highwayMpg price
## 1                9        111    5000      21         27 13495
## 2                9        111    5000      21         27 16500
## 3                9        154    5000      19         26 16500
## 4               10        102    5500      24         30 13950
```

imports85 è una struttura di dati con 205 casi (righe) e 26 variabili (colonne). Questo set di dati è composto da tre tipi di entità:

- Specifica di un'auto in termini di varie caratteristiche
- Rating di rischio assicurativo assegnato - corrisponde alla misura in cui l'auto è più rischiosa di quanto indichi il suo prezzo. Alle auto viene inizialmente assegnato un simbolo di fattore di rischio associato al prezzo. Poi, se è più rischiosa (o meno rischiosa), il simbolo viene modificato spostandolo verso l'alto (o verso il basso) della scala. Gli attuari chiamano questo processo "simbolizzazione". Un valore di +3 indica che l'auto è rischiosa, -3 che probabilmente è abbastanza sicura
- Perdite d'uso normalizzate in confronto ad altre autovetture - pagamento medio relativo dei sinistri per anno del veicolo assicurato. Questo valore è normalizzato per tutte le auto che rientrano in una particolare classificazione dimensionale (utilitarie a due porte, station wagon, sportive/speciali, ecc.) e rappresenta il danno medio per auto all'anno

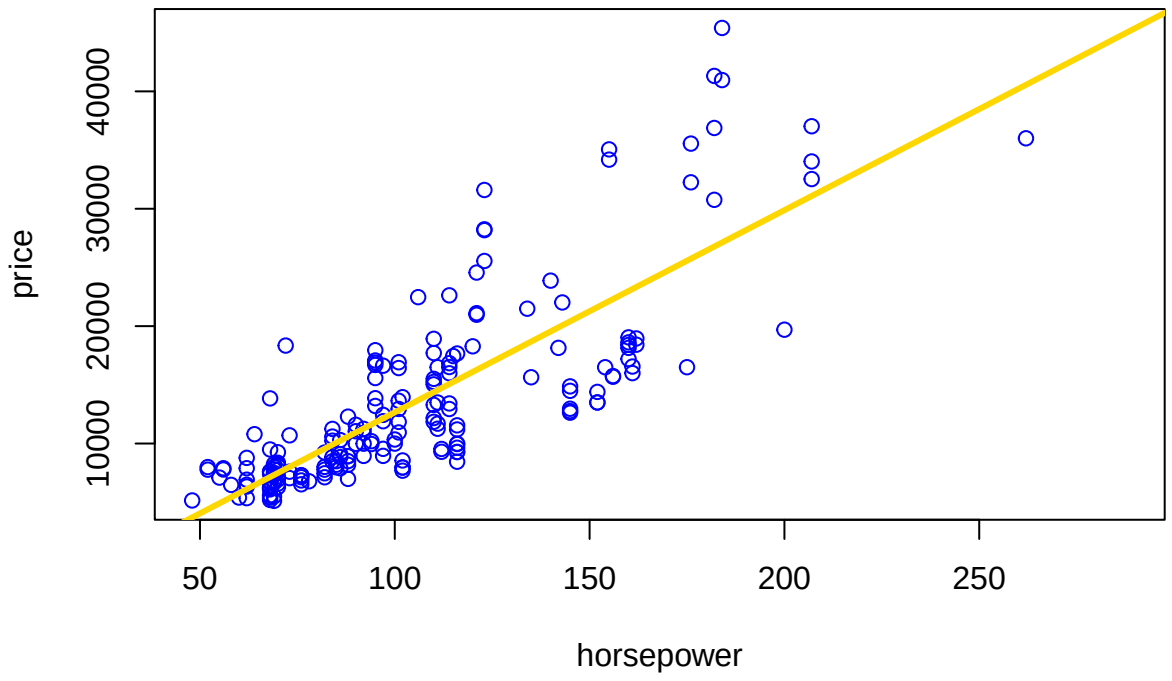
Per questo esempio non prenderemo in conisderazione le ultime due entità: il nostro obiettivo sarà quello di analizzare come varia il prezzo in funzione delle varie caratteristiche tecniche dell'auto. Modifichiamo quindi il dataset rimuovendo le prime due variabili, relative a tematiche assicurative e di ammortamento.

```
auto <- subset(imports85, select = -c(symboling, normalizedLosses))
head(auto, 3)
```

```
##          make fuelType aspiration numOfDoors   bodyStyle driveWheels
## 1 alfa-romero      gas        std        two convertible         rwd
## 2 alfa-romero      gas        std        two convertible         rwd
## 3 alfa-romero      gas        std        two   hatchback         rwd
##   engineLocation wheelBase length width height curbWeight engineType
## 1          front      88.6  168.8  64.1   48.8       2548       dohc
## 2          front      88.6  168.8  64.1   48.8       2548       dohc
## 3          front      94.5  171.2  65.5   52.4       2823       ohcv
##   numOfCylinders engineSize fuelSystem bore stroke compressionRatio horsepower
## 1           four        130       mpfi 3.47   2.68                9        111
## 2           four        130       mpfi 3.47   2.68                9        111
## 3            six        152       mpfi 2.68   3.47                9        154
##   peakRpm cityMpg highwayMpg price
## 1    5000      21         27 13495
## 2    5000      21         27 16500
## 3    5000      19         26 16500
```

Prima di iniziare con il randomForest, è conveniente svolgere alcune analisi preliminari per valutare le relazioni tra il prezzo e alcune altre variabili. Grafico del prezzo in funzione dei cavalli dell'auto, e relativo modello lineare ad una variabile:

```
mod <- with(auto, lm(price ~ horsepower))
with(auto, plot(price ~ horsepower, col = "blue"))
abline(mod, col="gold", lwd=3)
```



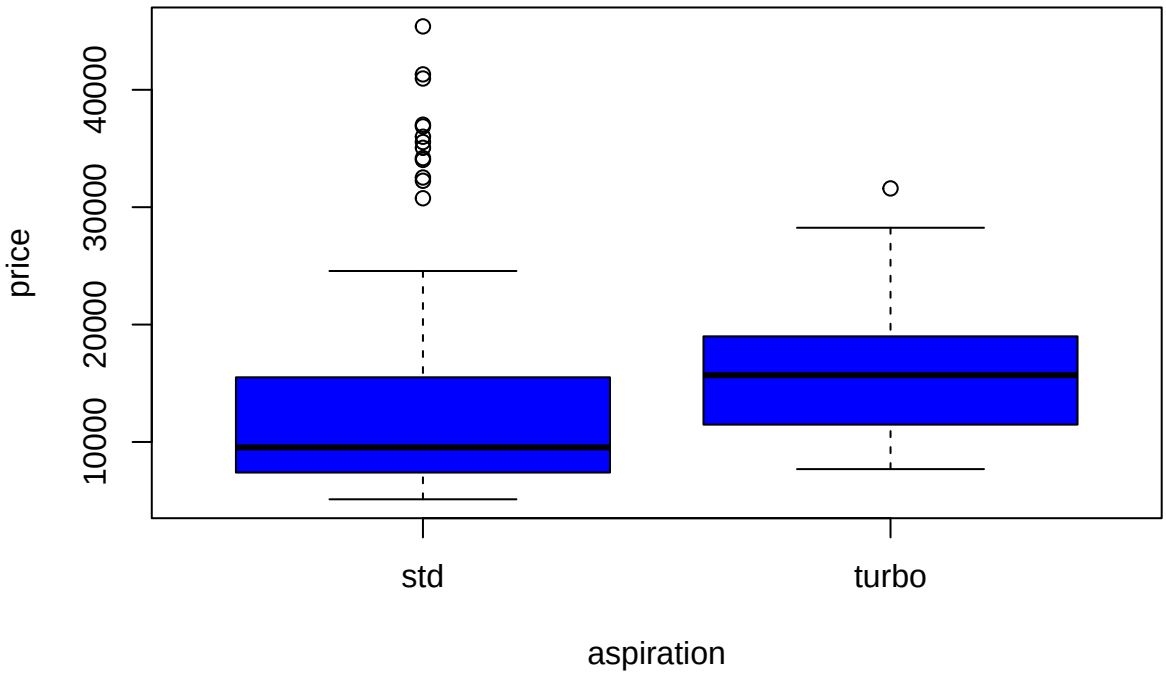
```r
summary(mod)
```

```
##
## Call:
## lm(formula = price ~ horsepower)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10180.1  -2262.0   -471.1   1779.5  18276.2
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4562.175    974.995  -4.679 5.35e-06 ***
## horsepower    172.206      8.866  19.424  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4685 on 197 degrees of freedom
##   (6 observations deleted due to missingness)
## Multiple R-squared:  0.657,  Adjusted R-squared:  0.6552
## F-statistic: 377.3 on 1 and 197 DF,  p-value: < 2.2e-16
```

Sia il grafico che il modello lineare suggeriscono una relazione tra le due variabili.

Grafico del prezzo in funzione della tipologia di aspirazione, e relativo modello aov per valutare la variabilità tra le due classi:

```r
mod2 <- with(auto, aov(price ~ aspiration))
with(auto, plot(price ~ aspiration, col ="blue"))
```



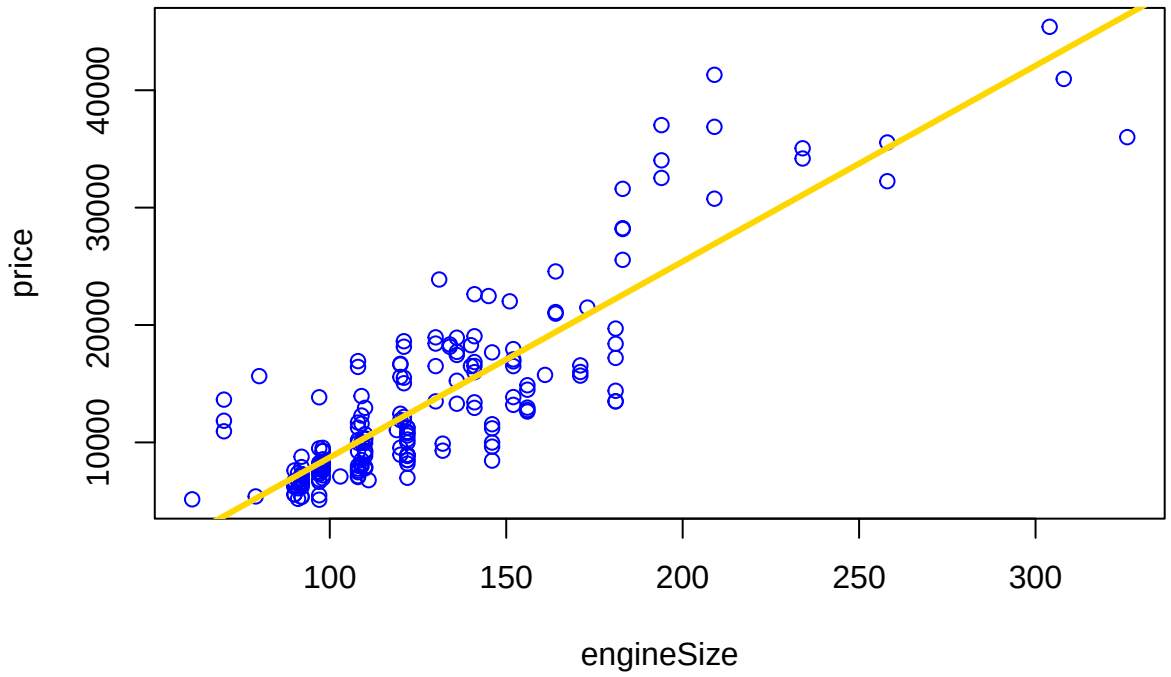
```
summary(mod2)
```

```
##              Df    Sum Sq   Mean Sq F value Pr(>F)
## aspiration    1 4.073e+08 407335500   6.631 0.0107 *
## Residuals   199 1.222e+10  61426318
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 4 observations deleted due to missingness
```

Il prezzo sembrerebbe dipendere anche dalla classe di aspirazione - con un prezzo più alto per aspirazione Turbo.

Grafico del prezzo in funzione delle dimensioni del motore, e relativo modello lineare ad una variabile:

```
mod3 <- with(auto, lm(price ~ engineSize))
with(auto, plot(price ~ engineSize, col = "blue"))
abline(mod3, col="gold", lwd=3)
```




```
summary(mod3)
```

```
##
## Call:
## lm(formula = price ~ engineSize)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10433.0  -2249.4   -469.8   1370.6  14404.6
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -7963.339    884.835   -9.00   <2e-16 ***
## engineSize    166.860      6.629   25.17   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3895 on 199 degrees of freedom
##   (4 observations deleted due to missingness)
## Multiple R-squared:  0.761,  Adjusted R-squared:  0.7598
## F-statistic: 633.5 on 1 and 199 DF,  p-value: < 2.2e-16
```

Come ci si aspetterebbe, anche le dimensioni del motore sembrano avere una relazione forte con il prezzo dell'auto.

Non procediamo, ora, con l'analisi delle altre variabili: sicuramente, diverse altre saranno più o meno correlate alla variabile dipendente prezzo. Per effettuare ulteriori conclusioni riguardo importanza della variabili, prezzo ecc. procediamo nel prossimo paragrafo con la costruzione e analisi del modello randomForest, utilizzando il pacchetto spiegato in precedenza.

4.2.2 RandomForest package

Per una migliore visione d'insieme del dataset da analizzare, visualizzeremo la struttura dell'insieme di dati con la chiamata alla funzione `str()`:

```
str(auto)
```

```
## 'data.frame':    205 obs. of  24 variables:
##  $ make            : Factor w/ 22 levels "alfa-romero",..: 1 1 1 2 2 2 2 2 2 2 ...
##  $ fuelType        : Factor w/ 2 levels "diesel","gas": 2 2 2 2 2 2 2 2 2 2 ...
##  $ aspiration      : Factor w/ 2 levels "std","turbo": 1 1 1 1 1 1 1 1 2 2 ...
##  $ numOfDoors      : Factor w/ 2 levels "four","two": 2 2 2 1 1 2 1 1 1 2 ...
##  $ bodyStyle       : Factor w/ 5 levels "convertible",..: 1 1 3 4 4 4 4 5 4 3 ...
##  $ driveWheels     : Factor w/ 3 levels "4wd","fwd","rwd": 3 3 3 2 1 2 2 2 2 1 ...
##  $ engineLocation  : Factor w/ 2 levels "front","rear": 1 1 1 1 1 1 1 1 1 1 ...
##  $ wheelBase       : num  88.6 88.6 94.5 99.8 99.4 ...
##  $ length          : num  169 169 171 177 177 ...
##  $ width           : num  64.1 64.1 65.5 66.2 66.4 66.3 71.4 71.4 71.4 67.9 ...
##  $ height          : num  48.8 48.8 52.4 54.3 54.3 53.1 55.7 55.7 55.9 52 ...
##  $ curbWeight      : int  2548 2548 2823 2337 2824 2507 2844 2954 3086 3053 ...
##  $ engineType      : Factor w/ 7 levels "dohc","dohcv",..: 1 1 6 4 4 4 4 4 4 4 ...
##  $ numOfCylinders  : Ord.factor w/ 7 levels "two"<"three"<..: 3 3 5 3 4 4 4 4 4 4 ...
##  $ engineSize      : int  130 130 152 109 136 136 136 136 131 131 ...
##  $ fuelSystem      : Factor w/ 8 levels "1bbl","2bbl",..: 6 6 6 6 6 6 6 6 6 6 ...
##  $ bore            : num  3.47 3.47 2.68 3.19 3.19 3.19 3.19 3.19 3.13 3.13 ...
##  $ stroke          : num  2.68 2.68 3.47 3.4 3.4 3.4 3.4 3.4 3.4 3.4 ...
##  $ compressionRatio: num  9 9 9 10 8 8.5 8.5 8.5 8.3 7 ...
##  $ horsepower      : int  111 111 154 102 115 110 110 110 140 160 ...
##  $ peakRpm         : int  5000 5000 5000 5500 5500 5500 5500 5500 5500 5500 ...
##  $ cityMpg         : int  21 21 19 24 18 19 19 19 17 16 ...
##  $ highwayMpg      : int  27 27 26 30 22 25 25 25 20 22 ...
##  $ price           : int  13495 16500 16500 13950 17450 15250 17710 18920 23875 NA ...
```

Il dataset ha alcuni valori NA (mancanti o sconosciuti). Prima di procedere con la creazione del modello randomForest, andiamo ad analizzare il numero totale di valori NA.

```
sum(!complete.cases(auto))
```

```
## [1] 12
```

Questo set di dati ha 12 righe con valori mancanti.

Possiamo sostituire questi 12 valori con le medie delle relative variabili, permettendo al modello di agire in maniera accurata nonostante gli NA.

Per fare questo, possiamo utilizzare proprio una funzione del pacchetto randomForest: `na.roughfix`, che restituisce per ogni valore NA, la media dei valori della relativa variabile.

```
auto2 <- na.roughfix(auto)
sum(!complete.cases(auto2))
```

```
## [1] 0
```

Come possiamo verificare con la funzione sopra, non ci sono più valori NA. Per una stima più accurata, è possibile utilizzare anche le matrici di prossimità, già richiamate nei primi due capitoli. In questo caso è possibile utilizzare la funzione `rfImpute`, che utilizza proprio il random forest e la sua matrice di prossimità per sostituire i valori NA. Rimandiamo alla parte teorica per il suo utilizzo: per questo dataset utilizziamo solamente il `na.roughfix`, in quanto i valori NA non sono numerosi e un'approssimazione con le medie è sufficiente.

Adattiamo quindi un modello Random Forest in R, utilizzando la funzione specifica `randomForest()`. Lo scopo, come speigato nella fase preliminare, è quello di prevedere il prezzo delle auto, date come variabili predittive tutte le caratteristiche tecniche delle stesse.

```
set.seed(51)
model <- randomForest(
  formula = price ~ .,
  data = auto2
)
```

Analizziamo le caratteristiche della foresta appena creata:

```
model
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = auto2)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##           Mean of squared residuals: 6961706
##                     % Var explained: 88.73
```

Da questa sintesi possiamo già osservare alcuni parametri importanti della foresta appena creata:

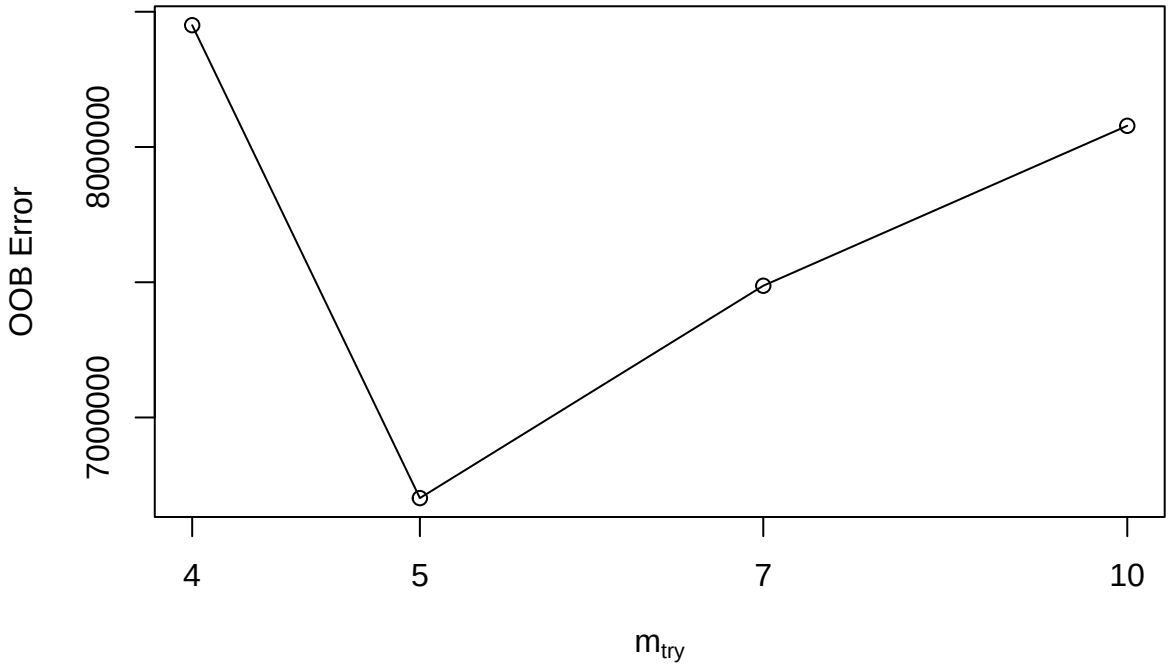
- la classe, ovviamente, di regressione
- il numero di alberi presenti nella foresta è quello di default: 500
- il numero di variabili considerate in ogni albero e ad ogni split è anch'esso quello di default: p/3, arrotondato per difetto = 7
- il modello spiega il 88,73% della variabilità nel prezzo, con un MSE di 6961706

Prima di valutare il modello, l'importanza delle variabili ed altri fattori, vediamo come possiamo migliorare il nostro modello, diminuendo l'errore. Qual è il numero di variabili predittive ottimo da considerare ad ogni albero e nodo?

Possiamo utilizzare la funzione del pacchetto randomForest: tuneRF, che permette di plottare l'errore OOB in funzione del numero di predittori utilizzati per ogni split. Con `doBest=TRUE`, inoltre, la funzione crea un modello ottimizzato con il numero ottimo di predittori mtry. Scegliamo di creare quindi un nuovo modello ottimizzato, che abbia 500 alberi (come il primo):

```
set.seed(51)
model_tuned <- tuneRF(
  x=auto2[,-24],
  y=auto2$price,
  stepFactor = 1.5,
  improve = 0.01,
  trace=TRUE,
  doBest=TRUE
)
```

```
## mtry = 7  OOB error = 7487022
## Searching left ...
## mtry = 5     OOB error = 6702015
## 0.104849 0.01
## mtry = 4     OOB error = 8450319
## -0.2608624 0.01
## Searching right ...
## mtry = 10    OOB error = 8078364
## -0.2053634 0.01
```



Il numero ottimo di variabili ad ogni split sembra essere 5, valore che minimizza l'errore OOB, Out Of Bag.

Vediamo quindi le caratteristiche del nuovo modello ottimizzato:

```
model_tuned
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = res[which.min(res[, 2]), 1])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 6923741
##                     % Var explained: 88.79
```

Un'altro modo per "ottimizzare" la foresta, e renderla anche più snella, poteva essere quello di valutare, tramite la funzione `plot.randomForest`, quale fosse il numero di alberi ottimo; ovvero quale fosse il numero di alberi nella foresta che andasse a minimizzare l'mse.

Verifichiamo questa possibilità di analisi qui di seguito, riprendendo il modello originale `model`. Calcoliamo il numero di alberi ottimo, ovvero quello che minimizza l'mse:

```
which.min(model$mse)
```

```
## [1] 4
```

Il miglior RMSE (Root of Mean Quadratic Error) per il modello è quindi calcolato come

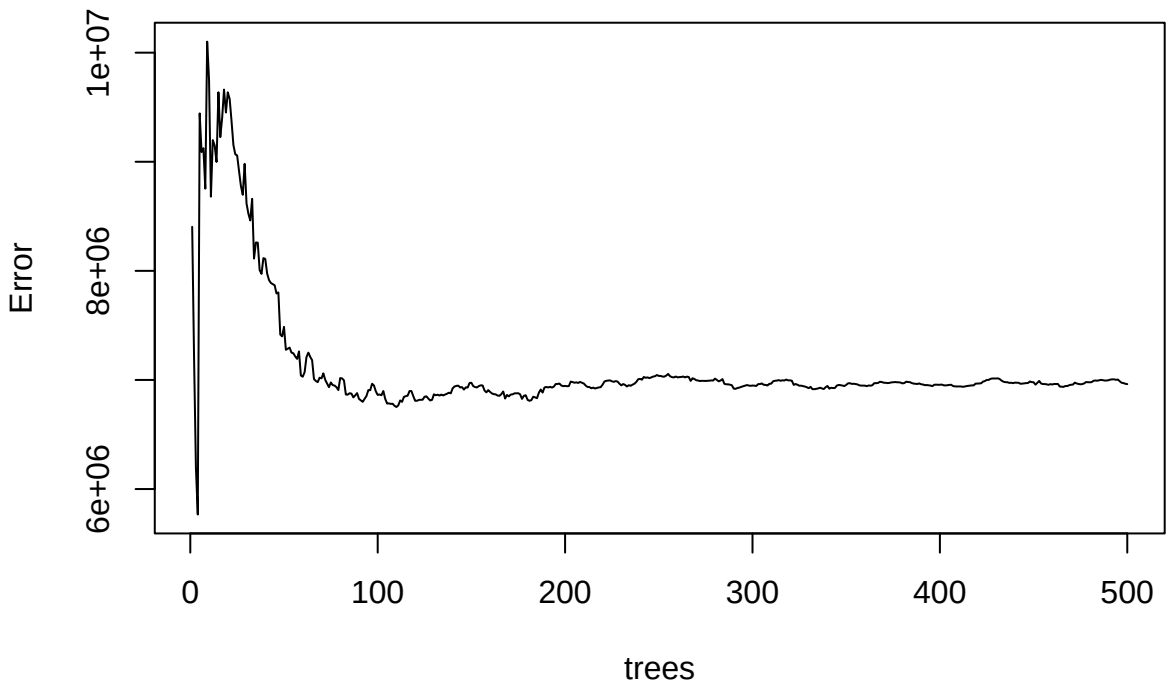
```
sqrt(model$mse[which.min(model$mse)])
```

```
## [1] 2401.343
```

Dall'output si può notare che il modello che produce l'errore quadratico medio (MSE) più basso ha utilizzato 4 alberi (e 7 variabili). Con questo numero di alberi, l'errore di previsione RMSE è di circa 2401 euro.

Utilizzando il grafico seguente, è possibile analizzare visivamente quanto appena affermato. Con `plot(model)`, infatti, è possibile osservare la relazione tra mse e numero di alberi nella foresta (presente nella teoria `plot.randomForest`).

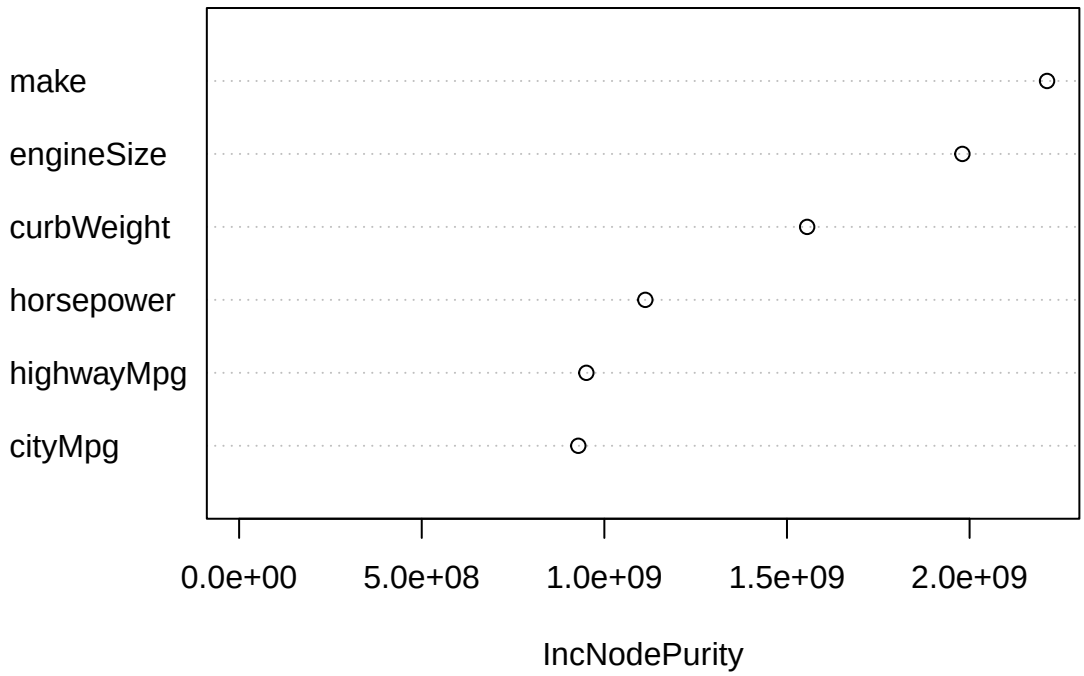
```
plot(model, main="")
```



Nel proseguio delle analisi, comunque, non utilizzeremo il modello con 4 alberi, bensì quello "ottimizzato", generato con la funzione `tuneRF`: `model_tuned` : le performance sono simili.

Una delle analisi più significative da effettuare è sicuramente quella relativa all'importanza delle variabili predittive. Utilizziamo la funzione `varImpPlot` analizzata nella parte teorica del report; mostriamo in grafico solo le 6 variabili più importanti:

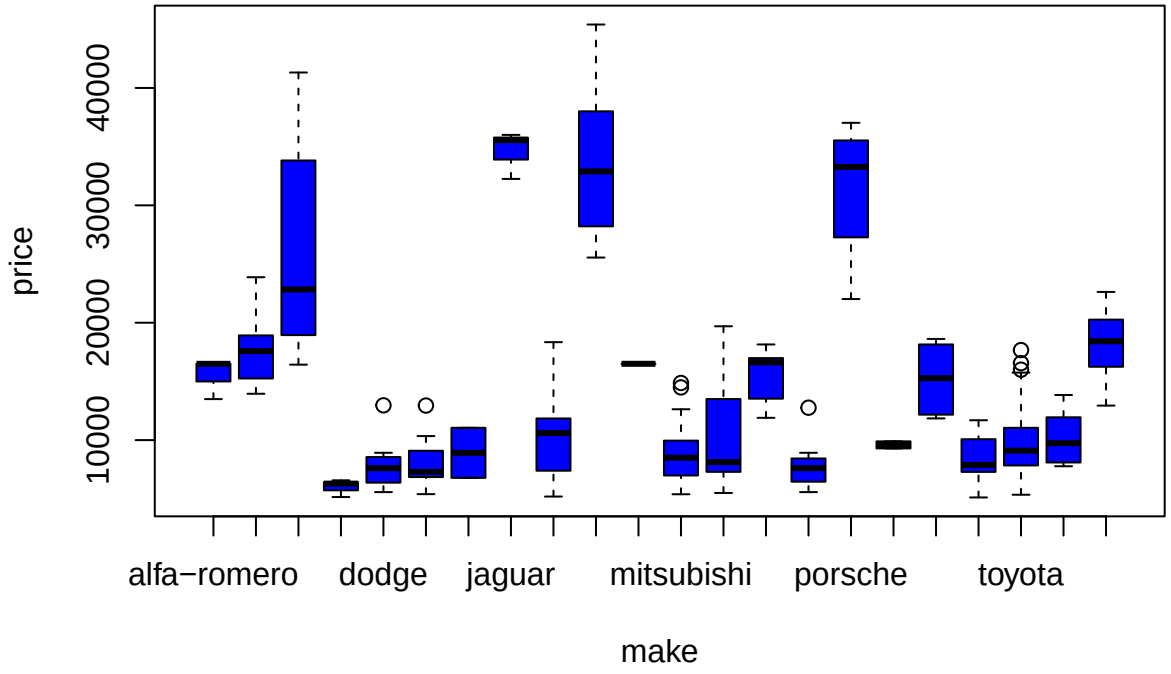
```
varImpPlot(model_tuned, n.var=6, main="")
```



IncNodePurity si riferisce alla funzione di perdita con cui vengono scelte le migliori suddivisioni. La funzione di perdita è MSE per la regressione e Gini-impurity per la classificazione. Le variabili più utili consentono di ottenere aumenti più elevati della purezza dei nodi, cioè di trovare uno split che abbia un'alta "varianza" inter-nodo e una piccola "varianza" intra-nodo. Riassumendo, per ogni suddivisione è possibile calcolare quanto questa riduca l'impurità dei nodi (per gli alberi di regressione, la differenza tra RSS (Somma dei quadrati residui) prima e dopo la divisione). La somma viene fatta su tutti gli alberi per quella variabile. Le variabili con maggiore valore di IncNodePurity, quindi, sono le più importanti.

E' molto interessante notare, quindi, come la variabile "produttore" (`make`) sia la più importante: è una conclusione molto interessante a livello di marketing, in quanto fa capire come il driver principale del prezzo sia il brand, il produttore dell'auto stessa e la sua immagine. Dopodichè, abbiamo la grandezza del motore (relazione forte, come suggerito durante le indagini preliminare) e le altre caratteristiche tecniche visibili nel grafico precedente.

```
with(auto, plot(price ~ make, col ="blue"))
```




```
summary(lm(price ~ make, data=auto))
```

```
##
## Call:
## lm(formula = price ~ make, data = auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9688.7 -2131.5  -354.4  1409.0 15196.2
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       15498.333   2191.253   7.073 3.30e-11 ***
## makeaudi           2360.833   2683.726   0.880  0.38021
## makebmw           10620.417   2569.472   4.133 5.49e-05 ***
## makechevrolet     -9491.333   3098.900  -3.063  0.00253 **
```

```
## makedodge         -7622.889   2530.241  -3.013  0.00296 **
## makehonda         -7313.641   2430.977  -3.009  0.00300 **
## makeisuzu         -6581.833   3464.676  -1.900  0.05908 .
## makejaguar        19101.667   3098.900   6.164 4.58e-09 ***
## makemazda         -4845.451   2376.748  -2.039  0.04295 *
## makemercedes-benz 18148.667   2569.472   7.063 3.48e-11 ***
## makemercury        1004.667   4382.507   0.229  0.81894
## makemitsubishi    -6258.564   2430.977  -2.575  0.01085 *
## makenissan        -5082.667   2366.824  -2.147  0.03310 *
## makepeugot           -9.242   2472.067  -0.004  0.99702
## makeplymouth      -7534.905   2619.049  -2.877  0.00450 **
## makeporsche       15902.167   2898.756   5.486 1.39e-07 ***
## makerenault       -5903.333   3464.676  -1.704  0.09014 .
## makesaab           -275.000   2683.726  -0.102  0.91850
## makesubaru        -6957.083   2449.896  -2.840  0.00504 **
## maketoyota        -5612.521   2291.668  -2.449  0.01528 *
## makevolkswagen    -5420.833   2449.896  -2.213  0.02818 *
## makevolvo          2564.848   2472.067   1.038  0.30089
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3795 on 179 degrees of freedom
##   (4 observations deleted due to missingness)
## Multiple R-squared:  0.7959, Adjusted R-squared:  0.7719
## F-statistic: 33.23 on 21 and 179 DF,  p-value: < 2.2e-16
```

Analizzando il boxplot e il modello lm possiamo notare che il marchio jaguar raggiunge il prezzo maggiore, seguito da porsche, mercedes e bmw. Queste ultime due case automobolistiche, in particolare, presentano un'alta variabilità interquartile (IQR) dei dati, che è proporzionale alla dispersione dei dati nella parte centrale della distribuzione.

Come vediamo dai grafici e dalle considerazioni sopra, alcune caratteristiche sono ampiamente più importanti di altre nell'analisi e previsione della variabile dipendente "price". Quindi, ci poniamo la seguente domanda: era davvero necessario prendere in considerazione tutte 23 le variabili predittive per ottenere un modello accurato?

Lo possiamo scoprire grazie alla funzione del pacchetto `rfcv`, spiegata nella parte di teoria:

```
set.seed(51)
rfcv (trainx=auto2[,-24],
  trainy=auto2$price)[2]
```

```
## $error.cv
##       23       12        6        3        1
## 12386191 12754849 13540787 15366379 25174012
```

Ad esempio, costruendo il modello con le sole 6 variabili più importanti (mostrate nel grafico sopra) - anzichè 23 - l'errore MSE non aumenta di molto (9,32%). Continueremo, comunque, ad utilizzare il modello `model_tuned` costruito in precedenza. Nel prossimo paragrafo analizzeremo le performance predittive del modello, con alcune considerazioni.

Arrivati a questo punto, è possibile utilizzare il modello di random forest aggiustato per fare previsioni su nuove osservazioni, calcolando il prezzo date le varie caratteristiche tecniche. Per effettuare una prova e valutare visivamente la bontà del nostro modello, effettuiamo delle previsioni dei prezzi delle auto in imports85 con gli OOB.

Utilizziamo quindi la funzione `predict.randomForest()` spiegata nella parte di teoria senza specificare un nuovo dataset `newdata`. Abbiamo quindi i valori di prezzo previsti per le osservazioni utilizzate come training data (valori ottenuti utilizzando gli OOB):

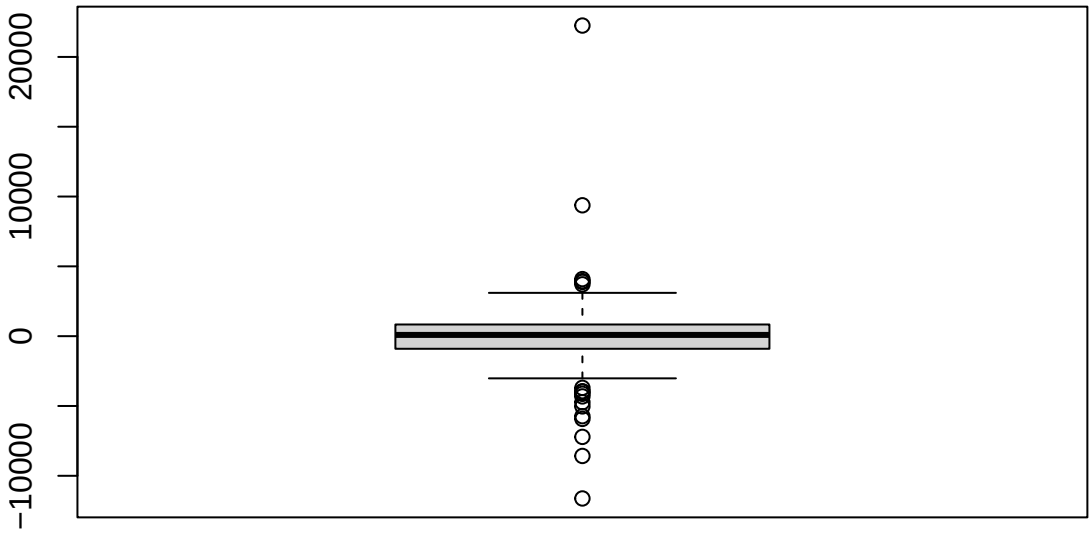
```
predict(model_tuned)
```

```
##         1         2         3         4         5         6         7         8
## 15998.066 14015.464 15754.638 11806.977 14559.945 14957.431 18902.050 18188.391
##         9        10        11        12        13        14        15        16
## 18844.053 19668.466 16336.734 16025.490 20028.866 20571.497 20863.459 33861.129
##        17        18        19        20        21        22        23        24
## 29691.414 34455.391  6738.957  7820.107  8387.445  5957.802  6188.107  8700.888
##        25        26        27        28        29        30        31        32
##  6586.035  7179.847  6744.415  8647.556 10165.225 13666.378  6368.335  6733.179
##        33        34        35        36        37        38        39        40
##  6436.865  7039.803  6862.868  7507.342  7545.623  8878.268  8396.268  9681.761
##        41        42        43        44        45        46        47        48
##  9355.456 11325.166  9779.531  9663.867  7949.650  8104.556 11672.061 34300.504
##        49        50        51        52        53        54        55        56
## 32848.908 32978.090  6698.777  6332.491  6010.201  7196.218  6889.630 13015.603
##        57        58        59        60        61        62        63        64
## 12549.009 11888.682 13919.835 10331.496 10325.371  9583.556  9599.373 10783.114
##        65        66        67        68        69        70        71        72
##  9656.981 15361.291 12419.045 28636.607 26824.082 26994.159 27285.486 34996.895
##        73        74        75        76        77        78        79        80
## 30926.615 38340.221 36819.329 17005.863  6104.414  6434.769  6426.258  8309.576
##        81        82        83        84        85        86        87        88
##  9476.223  8333.551 14227.770 13465.692 13682.986  8676.774  8244.631  9570.121
##        89        90        91        92        93        94        95        96
##  9599.861  7291.900  7876.426  6837.534  7049.585  7732.987  6794.495  7109.406
##        97        98        99       100       101       102       103       104
##  6958.269  7418.709  7146.998  9530.930  9109.948 15563.041 15336.373 15734.073
##       105       106       107       108       109       110       111       112
## 18088.097 19095.544 16397.099 15986.571 16913.284 16321.816 17799.656 15305.872
##       113       114       115       116       117       118       119       120
## 15671.711 14918.118 16660.780 13907.749 15245.955 16349.863  6060.046  8403.451
##       121       122       123       124       125       126       127       128
##  6558.732  7238.817  7290.019 10005.384 14320.834 18038.894 33372.011 32811.580
##       129       130       131       132       133       134       135       136
## 29814.437 32541.555 11581.590 11243.942 14104.142 15015.777 14080.235 14020.003
##       137       138       139       140       141       142       143       144
## 16856.978 16618.516  7307.169  7632.614  7554.237  7991.162  7906.830  9639.227
##       145       146       147       148       149       150       151       152
## 10086.127 10450.600  8428.687 10174.034  9491.239 12290.551  6533.238  6412.918
##       153       154       155       156       157       158       159       160
##  6540.195  7677.366  7909.484 10222.585  7655.593  7683.544  7982.330  8010.993
##       161       162       163       164       165       166       167       168
##  7526.933  8024.562  7839.967  8238.559  8128.667  9927.706 10002.838 10796.037
##       169       170       171       172       173       174       175       176
## 10500.595 10790.661 11238.512 12096.691 11936.767 10464.835 11232.364 10785.281
##       177       178       179       180       181       182       183       184
## 10126.528 10343.585 15973.291 16300.929 16478.833 16512.858  8306.057  8599.239
##       185       186       187       188       189       190       191       192
##  8305.719  8555.033  8645.467  8576.700  9784.181  9500.912  9575.646 14497.723
##       193       194       195       196       197       198       199       200
```

```
## 11168.953 12147.722 15710.361 15728.216 15010.882 14832.410 18393.892 18093.067
##       201       202       203       204       205
## 17950.490 20003.555 19660.103 18525.669 17890.332
```

Vediamo la differenza tra i valori previsti e quelli reali:

```
boxplot (predict(model_tuned) - auto2[24])
```



Si può notare - semplicemente dal grafico sopra - come la differenza tra quanto previsto con gli OOB e quanto effettivamente presente nel dataset sia distribuita attorno allo 0 (con la presenza di alcune differenze "anomale" - probabilmente outlier, da stabilire con ulteriori analisi).

In sintesi, per il modello creato, il RMSE (root of mean square error - errore medio) è:

```
sqrt(model_tuned$mse[which.min(model_tuned$mse)])
```

```
## [1] 2624.683
```

Come fatto per l'esempio precedente (quello sul Titanic), confrontiamo con il pacchetto caret le performance tra il modello random forest (in questo caso di regressione) e un modello lineare semplice, ottimizzati con cross validation (10 fold):

```
set.seed(51)
modelloLM <- train(price ~ ., data = auto2, method="lm",
                   trControl=trainControl(method="cv", number=10))
```

```
modelloLM
```

```
## Linear Regression
##
## 205 samples
##  23 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 184, 185, 185, 184, 184, 185, ...
## Resampling results:
##
```

```
##   RMSE      Rsquared   MAE
##   3238.025  0.7840503  2058.043
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
set.seed(51)
modelloRF <- train(price ~ ., data = auto2, method="rf",
                   trControl=trainControl(method="cv", number=10))
modelloRF
```

```
## Random Forest
##
## 205 samples
##  23 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 184, 185, 185, 184, 184, 185, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    2    3192.293  0.8600386  2158.569
##   32    2585.604  0.8632068  1609.170
##   63    2569.634  0.8651900  1619.831
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 63.
```

Anche in questo caso (problema di regressione), il modello random forest sembra avere un'accuratezza migliore rispetto al modello lineare (RMSE maggiore per il modello lm).

5 CONCLUSIONI

In questo report abbiamo analizzato il pacchetto randomForest da un punto di vista teorico e da un punto di vista pratico, portando un esempio riguardante un problema di classificazione e un altro riguardante un problema di regressione.

Come suggerito dalla teoria, il modello è in grado di analizzare in maniera efficace ed efficiente datasets di vario tipo, con un'accuratezza migliore rispetto ad alcuni modelli sostitutivi.

Abbiamo ritenuto superflue, nei nostri esempi, alcune funzioni del pacchetto (come getTree, varUsed ecc.). L'utilizzo di tali funzioni, ad ogni modo, risulta piuttosto intuitivo: rimandiamo in tal senso alla lettura della parte teorica.

6 BIBLIOGRAFIA

- Breiman and Cutler's Random Forest for Classsification and Regression - Package 'randomForest' - Version 4.7-1.1 - 2022-01-24
- An Introduction to Statistical Learning - Second Edition - G. James, D. Witten, T. Hastie, R. Tibshirani, June 21, 2023
- https://www.stat.berkeley.edu/users/breiman/RandomForests/
- https://www.ibm.com/topics/random-forest
- https://en.wikipedia.org/wiki/Random_forest
- Dispense e appunti del corso di Apllied Statistics, A.A. 2023/24, Prof. Ruggero Bellio
- TITANIC: https://www.kaggle.com/datasets/yasserh/titanic-dataset
- IMPORTS85: https://www.rdocumentation.org/packages/randomForest/versions/4.7-1.1/topics/imports85