Data Mining for Astrophysics

Vittorio Haardt / Riccardo Fossato / Sara Capozio

1/20/2022

La SDSS è una survey che ha osservato diversi oggetti nell'universo scansionando il cielo per anni. Tra questi ci sono stelle (che sono nella nostra galassia), galassie.

objid: nome dell'oggetto nel catalogo della SDSS

RA. and dec: individuano la posizione nel cielo dell'oggetto (nel catalogo sono espresse in gradi).

RA: misura l'est e l'ovest sulla sfera celeste ed assomiglia alla longitudine sulla terra.

Dec: misura il nord e il sud sulla sfera celeste ed assomiglia alla latitudine sulla terra.

mag_u,mag_g,….,mag_z : sono le magnitudini in diverse bande dell'oggetto. La magnitudine (link) è una misura di flusso cioè di quanta luce arriva da una sorgente. Oggetti con magnitudini più alte sono meno luminosi. I pedici _u, _g,… indicano l'intervallo di lunghezza d'onda in cui è stato misurato il flusso (link). Per esempio mag_u è quanta luce proviene da un determinato oggetto in banda u ovvero intorno ai 3500 Angstroms (ultravioletto in pratica).

mag_u: quanta luce proviene da un determinato oggetto in banda u ovvero intorno ai 3580 Angstroms.

mag_g: quanta luce proviene da un determinato oggetto in banda g ovvero intorno ai 4754 Angstroms.

mag_r: quanta luce proviene da un determinato oggetto in banda r ovvero intorno ai 6204 Angstroms.

mag_i: quanta luce proviene da un determinato oggetto in banda i ovvero intorno ai 7698 Angstroms.

mag_z: quanta luce proviene da un determinato oggetto in banda z ovvero intorno ai 9665 Angstroms.

spec_z: indica il redshift della sorgente ed è un modo per indicare la distanza tra noi e quella sorgente. Più il redshift è alto più la sorgente è distante.

u_g_color: è la differenza tra due magnitudini, mag_u-mag_g, in base alla definizione della magnitudine, più u_g_color è grande più l'oggetto emette in banda g.

g_r_color: è la differenza tra due magnitudini, mag_g-mag_r, in base alla definizione della magnitudine, più u_g_color è grande più l'oggetto emette in banda r.

r_i_color: è la differenza tra due magnitudini, mag_r-mag_i, in base alla definizione della magnitudine, più u_g_color è grande più l'oggetto emette in banda i.

i_z_color: è la differenza tra due magnitudini, mag_i-mag_z, in base alla definizione della magnitudine, più u_g_color è grande più l'oggetto emette in banda z.

Il dataset è stato precedentemente bilanciato e snellito, la proporzione totale degli oggetti si dispone in questo modo all'ultimo aggiornamento controllato:

GALAXY 2,541,242 STAR 928,859

Quindi le true priors sono :

π(GALXY) = 0.73 π(STAR) = 0.27

Analisi Preliminari

```
sdss <- read.csv('D:/data_minig/elaborato/sdss_dataset1.csv')
sdss <- sdss[sample(1:nrow(sdss)),]
sdss$class <- as.factor(sdss$class)
sdss <-  sdss[,-1]
```

Importiamo togliamo id e mettiamo class as factor

```
library(funModeling)
status <- df_status(sdss, print_results=F)
status
```

```
##     variable q_zeros p_zeros q_na p_na q_inf p_inf    type unique
## 1         ra       0    0.00    0    0     0     0 numeric  19971
## 2        dec       0    0.00    0    0     0     0 numeric  19997
## 3      mag_u       0    0.00    0    0     0     0 numeric  19741
## 4      mag_g       0    0.00    0    0     0     0 numeric  19734
## 5      mag_r       0    0.00    0    0     0     0 numeric  19702
## 6      mag_i       0    0.00    0    0     0     0 numeric  19674
## 7      mag_z       0    0.00    0    0     0     0 numeric  19653
## 8     spec_z      26    0.13    0    0     0     0 numeric  19963
## 9  u_g_color      18    0.09    0    0     0     0 numeric  19833
## 10 g_r_color      14    0.07    0    0     0     0 numeric  19795
## 11 r_i_color      14    0.07    0    0     0     0 numeric  19665
## 12 i_z_color      14    0.07    0    0     0     0 numeric  19595
## 13     class       0    0.00    0    0     0     0  factor      2
```

Non è presente zero variance e dati mancanti.

```
sapply(sdss, function(x)(sum(is.na(x))))
```

```
##        ra       dec     mag_u     mag_g     mag_r     mag_i     mag_z    spec_z
##         0         0         0         0         0         0         0         0
## u_g_color g_r_color r_i_color i_z_color     class
##         0         0         0         0         0
```

Bilanciamento Dataset

```
prop.table(table(sdss$class))
```

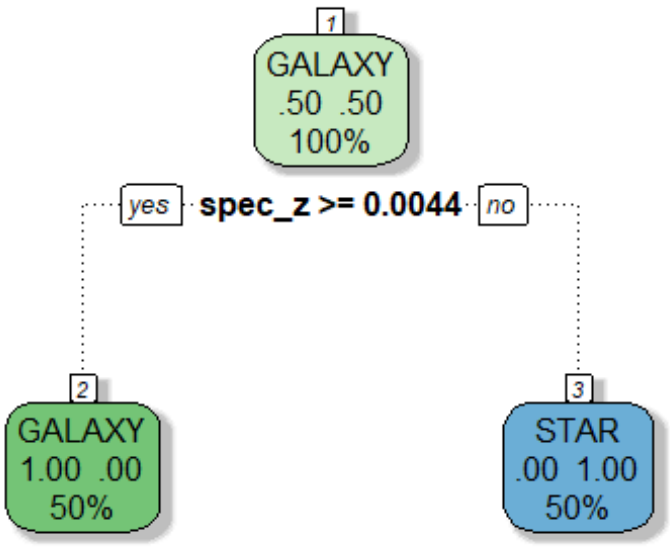
```
##
## GALAXY   STAR
##    0.5    0.5
```

Il dataset e' bilanciato, bisognerà aggiustare le postirior con le vere prior.

Obbiettivo di analisi

La variabile *spec_z* che indica il redshift di per se basterebbe da sola per riuscire a classificare gli oggetti celesti, dato che di fatto per classificare una stella o una galassi bisogna guardare la loro distanza. Se la distanza é maggiore dei limiti della via lattea allora l'oggetto sarà una galassia altrimenti una stella al suo interno. Se si attuasse un'analisi predittiva consideando *spec_z* il modello vincente risulterebbe il seguente.

```
library(caret)
library(rattle)
set.seed(1234)
tree_iniz <- train(class ~ .,
                   data = sdss,
                   method = "rpart",
                   tuneLength = 10)
fancyRpartPlot(tree_iniz$finalModel)
```



Ovvero un albero con un solo split che classifica il 100% delle unitá statistiche nella classe corretta in base alla variabile *spec_z*. Nonostante non venga riportato questo risultato é stato ottenuto con tutti i crismi di una procedura di machine learning.

Quello che si vole trovare é un metodo di classificazione che non usi il redshift, quindi procediamo togliendo la variabile *spec_z* dal nostro dataset.

```
sdss_an <- sdss[,-8]
```

Procediamo quindi con i 4 step.

STEP 1

Training e Validation

Dividiamo il dataset in training (70%) e validation (30%) tenendo il 10% del training come dataset di score.

```
library(caret)
set.seed(1234)
split <- createDataPartition(y=sdss_an$class, p=0.70, list=FALSE)
train <- sdss_an[split,]
test <- sdss_an[-split,]
split <- createDataPartition(y=train$class, p=0.90, list=FALSE)
train <- train[split,]
score <- train[-split,]
```

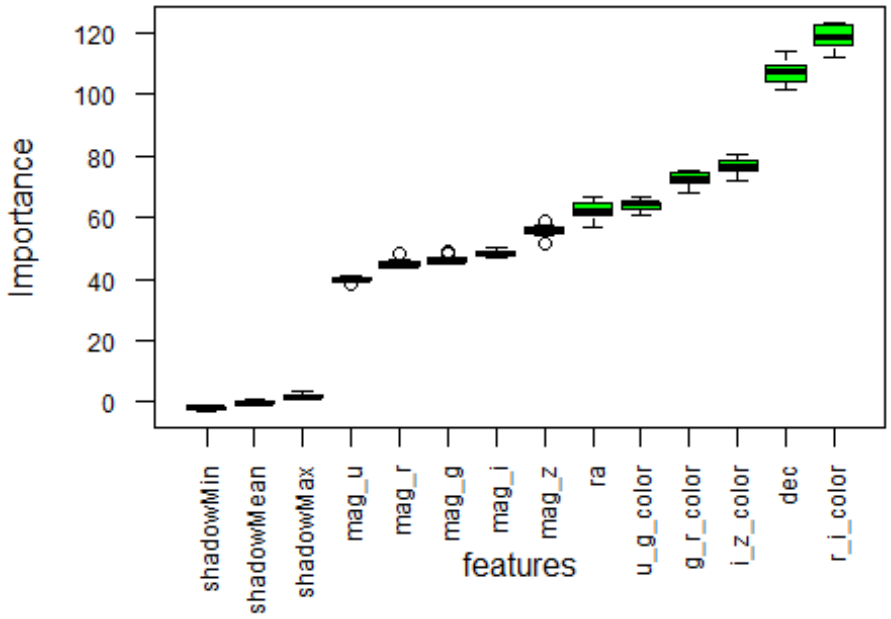
Model Selection sul Training

Faccio una model selection in modo da usare il dataset con solo le variabili selezionata nei modelli che lo richiedono. Per la model selection si sceglie una random forest con metodo Boruta.

Random Forest con metodo Boruta

Stampiamo le variabili considerate importanti.

```
library(Boruta)
set.seed(123)
boruta.train <- Boruta(class~., data =sdss_an,mcAdj=TRUE)
cat(getSelectedAttributes(boruta.train),sep="\n")

## ra
## dec
## mag_u
## mag_g
## mag_r
## mag_i
## mag_z
## u_g_color
## g_r_color
## r_i_color
## i_z_color

plot(boruta.train, xlab = "features", las=2,cex.axis=0.75)
```



Boruta identifica tutte le variabili come variabili importanti.

Scelta della Metrica

Scegliamo Kappa come metrica per tunare i modelli e scegliere la complessità ottimale.

```
metric = "Kappa"
```

Modelli

Tuniamo i seguenti modelli:

nerest neigh neural network Random Forest Gradient boosting naive bayes lasso tree

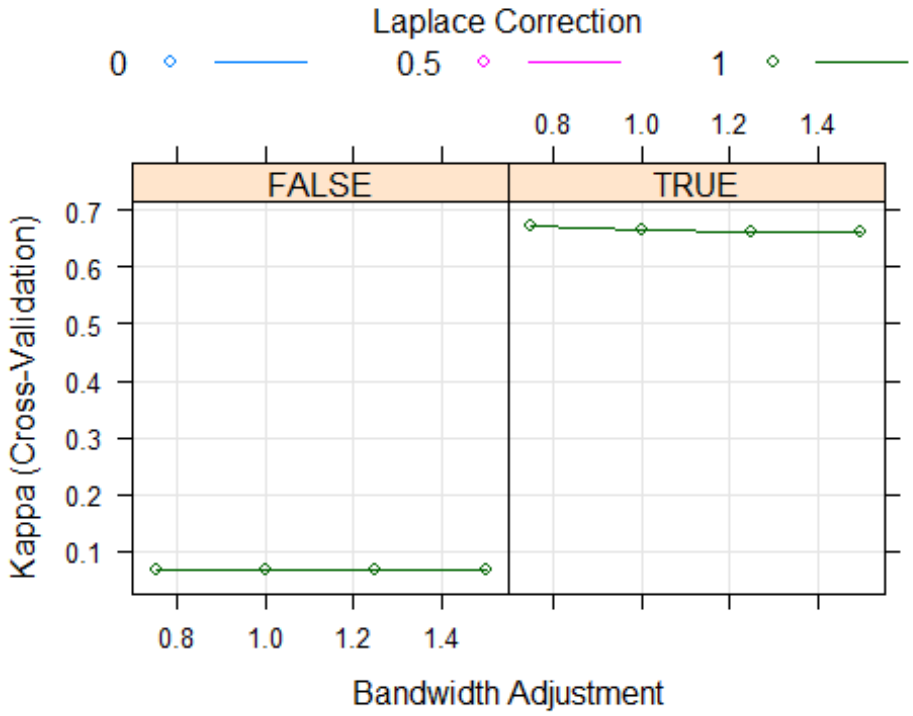
Cross validation per tutti i modelli

```
cv <- trainControl(method= "cv", number=10, search="grid", classProbs = TRUE)
```

Naive Bayes

```
library(caret)
set.seed(1234)
nb_grid <-expand.grid(usekernel=c(TRUE, FALSE),laplace = c(0, 0.5, 1),adjust = c(0
.75, 1, 1.25, 1.5))
naivebayes=train(class~.,
                 data=train,method="naive_bayes",preProcess=c('corr','nzv'),
                 tuneLength=5,tuneGrid=nb_grid,metric=metric,trControl = cv)
naivebayes
```

```
## Naive Bayes
##
## 12600 samples
##    11 predictor
##     2 classes: 'GALAXY', 'STAR'
##
## Pre-processing: remove (4)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 11340, 11340, 11340, 11340, 11340, 11340, ...
## Resampling results across tuning parameters:
##
##   usekernel  laplace  adjust  Accuracy   Kappa
##   FALSE      0.0      0.75    0.5337302  0.06746032
##   FALSE      0.0      1.00    0.5337302  0.06746032
##   FALSE      0.0      1.25    0.5337302  0.06746032
##   FALSE      0.0      1.50    0.5337302  0.06746032
##   FALSE      0.5      0.75    0.5337302  0.06746032
##   FALSE      0.5      1.00    0.5337302  0.06746032
##   FALSE      0.5      1.25    0.5337302  0.06746032
##   FALSE      0.5      1.50    0.5337302  0.06746032
##   FALSE      1.0      0.75    0.5337302  0.06746032
##   FALSE      1.0      1.00    0.5337302  0.06746032
##   FALSE      1.0      1.25    0.5337302  0.06746032
##   FALSE      1.0      1.50    0.5337302  0.06746032
##    TRUE      0.0      0.75    0.8359524  0.67190476
##    TRUE      0.0      1.00    0.8323810  0.66476190
##    TRUE      0.0      1.25    0.8315079  0.66301587
##    TRUE      0.0      1.50    0.8307143  0.66142857
##    TRUE      0.5      0.75    0.8359524  0.67190476
##    TRUE      0.5      1.00    0.8323810  0.66476190
##    TRUE      0.5      1.25    0.8315079  0.66301587
##    TRUE      0.5      1.50    0.8307143  0.66142857
##    TRUE      1.0      0.75    0.8359524  0.67190476
##    TRUE      1.0      1.00    0.8323810  0.66476190
##    TRUE      1.0      1.25    0.8315079  0.66301587
##    TRUE      1.0      1.50    0.8307143  0.66142857
##
## Kappa was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = TRUE
##  and adjust = 0.75.

plot(naivebayes)
```



Il k ottimale è ottenuto usando una funzione Kernel e con un adjustment pari a 0.75. Tre variabili vengono tolte per correlation.

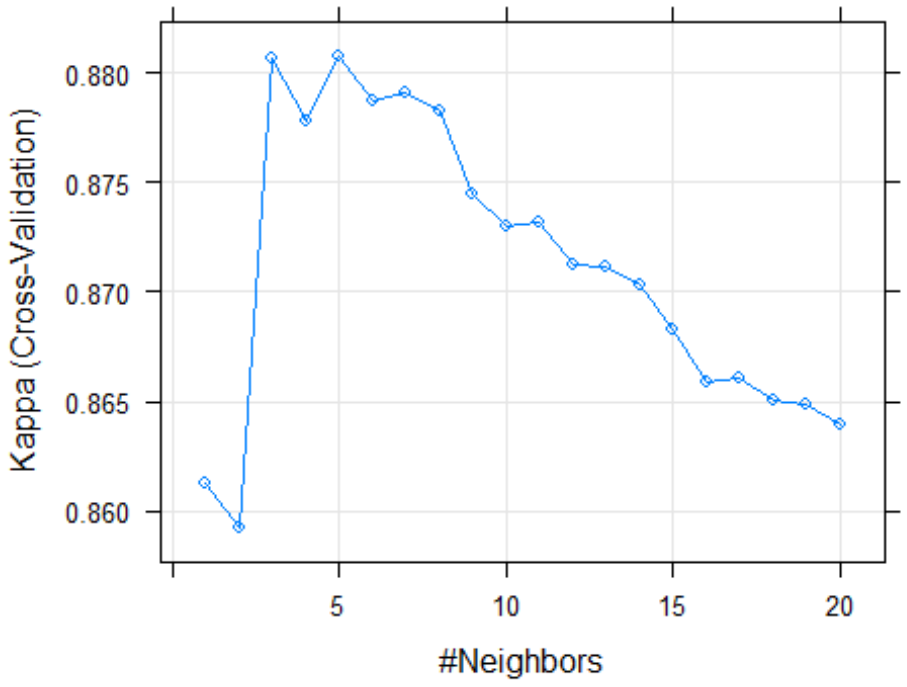
Nearest Neighbor

```
set.seed(1234)
grid_knn <- expand.grid(.k=seq(1,20, by=1)) #numero vicini
knn <- train(class ~., data=train,
                  method = "knn",
                  preProcess = c("center", "scale", "corr", "nzv"),
                  tuneLength = 10,
                  trControl = cv,
                  tuneGrid=grid_knn,
                  metric=metric)
knn
```

```
## k-Nearest Neighbors
##
## 12600 samples
##    11 predictor
##     2 classes: 'GALAXY', 'STAR'
##
## Pre-processing: centered (7), scaled (7), remove (4)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 11340, 11340, 11340, 11340, 11340, 11340, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
```

```
##     1  0.9306349  0.8612698
##     2  0.9296032  0.8592063
##     3  0.9403175  0.8806349
##     4  0.9388889  0.8777778
##     5  0.9403968  0.8807937
##     6  0.9393651  0.8787302
##     7  0.9395238  0.8790476
##     8  0.9391270  0.8782540
##     9  0.9372222  0.8744444
##    10  0.9365079  0.8730159
##    11  0.9365873  0.8731746
##    12  0.9356349  0.8712698
##    13  0.9355556  0.8711111
##    14  0.9351587  0.8703175
##    15  0.9341270  0.8682540
##    16  0.9329365  0.8658730
##    17  0.9330159  0.8660317
##    18  0.9325397  0.8650794
##    19  0.9324603  0.8649206
##    20  0.9319841  0.8639683
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
plot(knn)
```



La complessita finale per il NN che permette di massimizzare la nostra metrica (kappa) è un modello con un numero di vicini uguale a 3. Toglie nel pre processing 3 variabili per correlation.

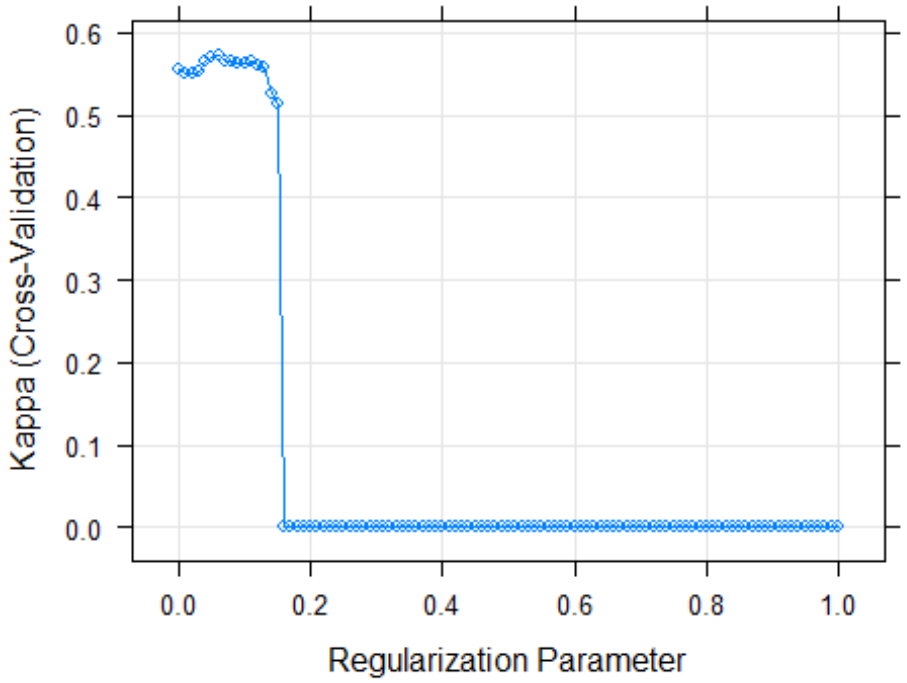
Lasso

```
set.seed(1234)
grid_lasso = expand.grid(.alpha=1,.lambda=seq(0, 1, by = 0.01))
lasso=train(class~.,
            data=train,
            method = "glmnet",
            trControl = cv,
            preProcess='corr',
            tuneLength=10,
            tuneGrid=grid_lasso,
            metric=metric,
            family="binomial")
lasso
```

```
## glmnet
##
## 12600 samples
##    11 predictor
##     2 classes: 'GALAXY', 'STAR'
##
## Pre-processing: remove (4)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 11340, 11340, 11340, 11340, 11340, 11340, ...
## Resampling results across tuning parameters:
##
##   lambda  Accuracy   Kappa
##   0.00    0.7770635  0.5541270
##   0.01    0.7753968  0.5507937
##   0.02    0.7753968  0.5507937
##   0.03    0.7768254  0.5536508
##   0.04    0.7823810  0.5647619
##   0.05    0.7846825  0.5693651
##   0.06    0.7867460  0.5734921
##   0.07    0.7828571  0.5657143
##   0.08    0.7823016  0.5646032
##   0.09    0.7817460  0.5634921
##   0.10    0.7817460  0.5634921
##   0.11    0.7819048  0.5638095
##   0.12    0.7806349  0.5612698
##   0.13    0.7786508  0.5573016
##   0.14    0.7628571  0.5257143
##   0.15    0.7565873  0.5131746
##   0.16    0.5000000  0.0000000
##   0.17    0.5000000  0.0000000
##   0.18    0.5000000  0.0000000
##   0.19    0.5000000  0.0000000
##   0.20    0.5000000  0.0000000
##   0.21    0.5000000  0.0000000
```

```
##   0.22    0.5000000  0.0000000
##   0.23    0.5000000  0.0000000
##   0.24    0.5000000  0.0000000
##   0.25    0.5000000  0.0000000
##   0.26    0.5000000  0.0000000
##   0.27    0.5000000  0.0000000
##   0.28    0.5000000  0.0000000
##   0.29    0.5000000  0.0000000
##   0.30    0.5000000  0.0000000
##   0.31    0.5000000  0.0000000
##   0.32    0.5000000  0.0000000
##   0.33    0.5000000  0.0000000
##   0.34    0.5000000  0.0000000
##   0.35    0.5000000  0.0000000
##   0.36    0.5000000  0.0000000
##   0.37    0.5000000  0.0000000
##   0.38    0.5000000  0.0000000
##   0.39    0.5000000  0.0000000
##   0.40    0.5000000  0.0000000
##   0.41    0.5000000  0.0000000
##   0.42    0.5000000  0.0000000
##   0.43    0.5000000  0.0000000
##   0.44    0.5000000  0.0000000
##   0.45    0.5000000  0.0000000
##   0.46    0.5000000  0.0000000
##   0.47    0.5000000  0.0000000
##   0.48    0.5000000  0.0000000
##   0.49    0.5000000  0.0000000
##   0.50    0.5000000  0.0000000
##   0.51    0.5000000  0.0000000
##   0.52    0.5000000  0.0000000
##   0.53    0.5000000  0.0000000
##   0.54    0.5000000  0.0000000
##   0.55    0.5000000  0.0000000
##   0.56    0.5000000  0.0000000
##   0.57    0.5000000  0.0000000
##   0.58    0.5000000  0.0000000
##   0.59    0.5000000  0.0000000
##   0.60    0.5000000  0.0000000
##   0.61    0.5000000  0.0000000
##   0.62    0.5000000  0.0000000
##   0.63    0.5000000  0.0000000
##   0.64    0.5000000  0.0000000
##   0.65    0.5000000  0.0000000
##   0.66    0.5000000  0.0000000
##   0.67    0.5000000  0.0000000
##   0.68    0.5000000  0.0000000
##   0.69    0.5000000  0.0000000
##   0.70    0.5000000  0.0000000
##   0.71    0.5000000  0.0000000
##   0.72    0.5000000  0.0000000
```

```
##   0.73    0.5000000  0.0000000
##   0.74    0.5000000  0.0000000
##   0.75    0.5000000  0.0000000
##   0.76    0.5000000  0.0000000
##   0.77    0.5000000  0.0000000
##   0.78    0.5000000  0.0000000
##   0.79    0.5000000  0.0000000
##   0.80    0.5000000  0.0000000
##   0.81    0.5000000  0.0000000
##   0.82    0.5000000  0.0000000
##   0.83    0.5000000  0.0000000
##   0.84    0.5000000  0.0000000
##   0.85    0.5000000  0.0000000
##   0.86    0.5000000  0.0000000
##   0.87    0.5000000  0.0000000
##   0.88    0.5000000  0.0000000
##   0.89    0.5000000  0.0000000
##   0.90    0.5000000  0.0000000
##   0.91    0.5000000  0.0000000
##   0.92    0.5000000  0.0000000
##   0.93    0.5000000  0.0000000
##   0.94    0.5000000  0.0000000
##   0.95    0.5000000  0.0000000
##   0.96    0.5000000  0.0000000
##   0.97    0.5000000  0.0000000
##   0.98    0.5000000  0.0000000
##   0.99    0.5000000  0.0000000
##   1.00    0.5000000  0.0000000
##
## Tuning parameter 'alpha' was held constant at a value of 1
## Kappa was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 1 and lambda = 0.06.

plot(lasso)
```



Il lambda ottimale per la nostra metrica è lambda=0. Anche in questo caso sono state tolte tre covariate per correlation.

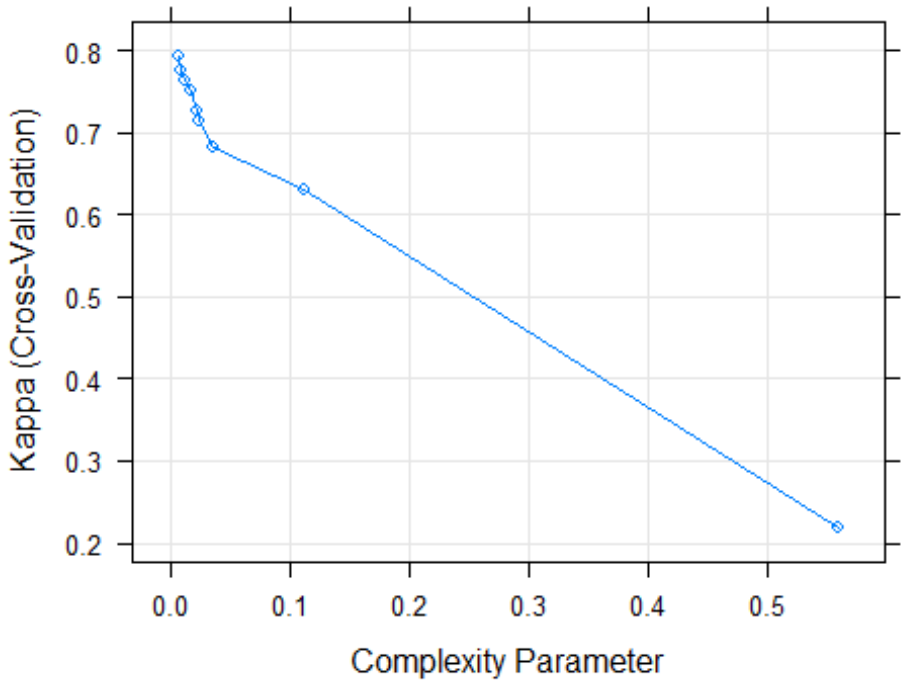
###Tree

```
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 4.0.5

## Loading required package: rpart

set.seed(1234)
tree <- train(class ~ .,
              data = train,
              method = "rpart",
              tuneLength = 10,
              trControl = cv,
              metric=metric,
              minsplit = 5)
tree

## CART
##
## 12600 samples
##    11 predictor
##     2 classes: 'GALAXY', 'STAR'
##
## No pre-processing
```

```
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 11340, 11340, 11340, 11340, 11340, 11340, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.006190476  0.8969048  0.7938095
##   0.006296296  0.8959524  0.7919048
##   0.008730159  0.8883333  0.7766667
##   0.010793651  0.8809524  0.7619048
##   0.016031746  0.8756349  0.7512698
##   0.021269841  0.8625397  0.7250794
##   0.023015873  0.8570635  0.7141270
##   0.035873016  0.8407143  0.6814286
##   0.111111111  0.8147619  0.6295238
##   0.560000000  0.6087302  0.2174603
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.006190476.
```

Il coefficiente di penalizzazione ottimale è pari a 0.0059.

```
plot(tree)
```



Neural Network

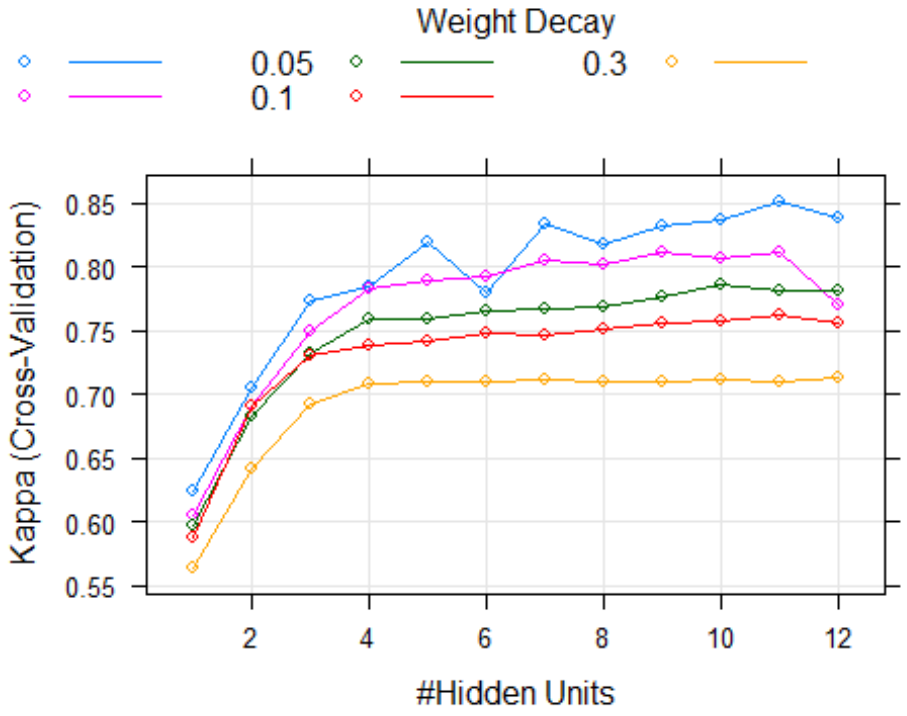
```
set.seed(1234)
grid_nn<-expand.grid(size=1:12,decay=c(0.001,0.01,0.05,.1,.3))
NeuralNetwork <- train(train[-12],
                       train$class,
```

```
                        method = "nnet",
                        preProcess = c("corr", "nzv","range"),
                        tuneLength = 10,
                       metric=metric,
                        tuneGrid=grid_nn,
                        trControl=cv,
                        trace = FALSE,
                        maxit = 300)
NeuralNetwork

## Neural Network
##
## 12600 samples
##    11 predictor
##     2 classes: 'GALAXY', 'STAR'
##
## Pre-processing: re-scaling to [0, 1] (7), remove (4)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 11340, 11340, 11340, 11340, 11340, 11340, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    1    0.001  0.8123810  0.6247619
##    1    0.010  0.8025397  0.6050794
##    1    0.050  0.7985714  0.5971429
##    1    0.100  0.7941270  0.5882540
##    1    0.300  0.7817460  0.5634921
##    2    0.001  0.8523810  0.7047619
##    2    0.010  0.8449206  0.6898413
##    2    0.050  0.8411111  0.6822222
##    2    0.100  0.8455556  0.6911111
##    2    0.300  0.8203968  0.6407937
##    3    0.001  0.8862698  0.7725397
##    3    0.010  0.8749206  0.7498413
##    3    0.050  0.8661111  0.7322222
##    3    0.100  0.8653175  0.7306349
##    3    0.300  0.8460317  0.6920635
##    4    0.001  0.8921429  0.7842857
##    4    0.010  0.8910317  0.7820635
##    4    0.050  0.8793651  0.7587302
##    4    0.100  0.8694444  0.7388889
##    4    0.300  0.8538095  0.7076190
##    5    0.001  0.9098413  0.8196825
##    5    0.010  0.8944444  0.7888889
##    5    0.050  0.8797619  0.7595238
##    5    0.100  0.8709524  0.7419048
##    5    0.300  0.8545238  0.7090476
##    6    0.001  0.8899206  0.7798413
##    6    0.010  0.8961905  0.7923810
##    6    0.050  0.8823016  0.7646032
##    6    0.100  0.8742063  0.7484127
```

```
##    6    0.300  0.8545238  0.7090476
##    7    0.001  0.9163492  0.8326984
##    7    0.010  0.9022222  0.8044444
##    7    0.050  0.8832540  0.7665079
##    7    0.100  0.8732540  0.7465079
##    7    0.300  0.8553175  0.7106349
##    8    0.001  0.9087302  0.8174603
##    8    0.010  0.9011905  0.8023810
##    8    0.050  0.8845238  0.7690476
##    8    0.100  0.8753968  0.7507937
##    8    0.300  0.8548413  0.7096825
##    9    0.001  0.9160317  0.8320635
##    9    0.010  0.9053175  0.8106349
##    9    0.050  0.8880159  0.7760317
##    9    0.100  0.8777778  0.7555556
##    9    0.300  0.8551587  0.7103175
##   10    0.001  0.9183333  0.8366667
##   10    0.010  0.9032540  0.8065079
##   10    0.050  0.8928571  0.7857143
##   10    0.100  0.8789683  0.7579365
##   10    0.300  0.8558730  0.7117460
##   11    0.001  0.9257937  0.8515873
##   11    0.010  0.9057937  0.8115873
##   11    0.050  0.8906349  0.7812698
##   11    0.100  0.8808730  0.7617460
##   11    0.300  0.8549206  0.7098413
##   12    0.001  0.9190476  0.8380952
##   12    0.010  0.8849206  0.7698413
##   12    0.050  0.8908730  0.7817460
##   12    0.100  0.8780159  0.7560317
##   12    0.300  0.8561905  0.7123810
##
## Kappa was used to select the optimal model using the largest value.
## The final values used for the model were size = 11 and decay = 0.001.
```

La rete neurale presenta 10 neuroni nascosti con un decay pari a 0.001.

```
plot(NeuralNetwork)
```



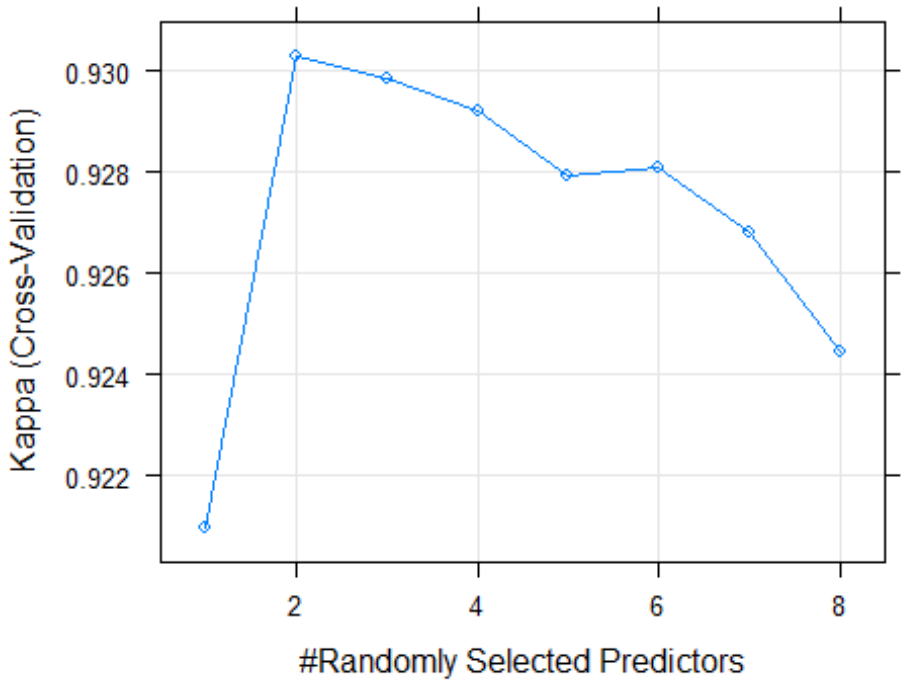
Random Forest

```
set.seed(1234)
grid_rf <- expand.grid(.mtry=seq(1,8, by=1))
Random_Forest <- train(class~.,
                       data=train,
                       method="rf",
                       metric=metric,
                       tuneGrid=grid_rf,
                       ntree=250,
                       trControl=cv)
Random_Forest

## Random Forest
##
## 12600 samples
##    11 predictor
##     2 classes: 'GALAXY', 'STAR'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 11340, 11340, 11340, 11340, 11340, 11340, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   1     0.9604762  0.9209524
##   2     0.9651587  0.9303175
##   3     0.9649206  0.9298413
```

```
##   4     0.9646032  0.9292063
##   5     0.9639683  0.9279365
##   6     0.9640476  0.9280952
##   7     0.9634127  0.9268254
##   8     0.9622222  0.9244444
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Il numero ottimale di covariate considerate in ogni albero è pari a 5.

```
plot(Random_Forest)
```



Gradient Boosting

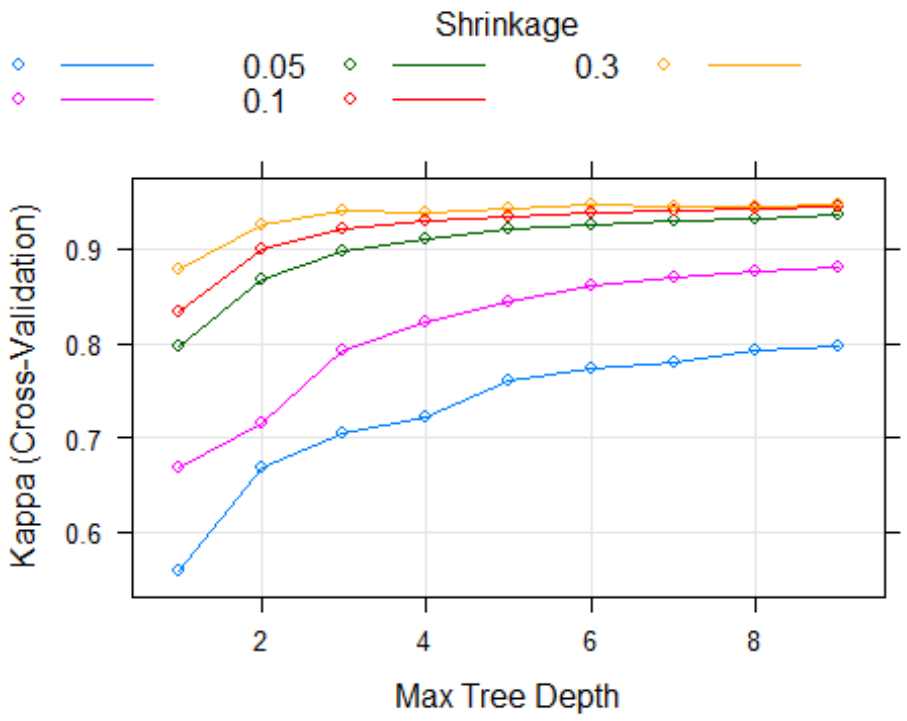
```
set.seed(1234)
grid_gbm <- expand.grid(interaction.depth=c(1,2,3,4,5,6,7,8,9),
                        n.trees=250,
                        shrinkage=c(0.001,0.01,0.05,.1,.3),
                        n.minobsinnode=20)
gbm <- train(class ~ .,
             data = train,
             method = "gbm",
             tuneGrid=grid_gbm,
             metric=metric,
             trControl=cv,
             verbose = FALSE)
gbm
```

```
## Stochastic Gradient Boosting
##
## 12600 samples
##    11 predictor
##     2 classes: 'GALAXY', 'STAR'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 11340, 11340, 11340, 11340, 11340, 11340, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  Accuracy   Kappa
##   0.001      1                  0.7794444  0.5588889
##   0.001      2                  0.8344444  0.6688889
##   0.001      3                  0.8520635  0.7041270
##   0.001      4                  0.8609524  0.7219048
##   0.001      5                  0.8800794  0.7601587
##   0.001      6                  0.8872222  0.7744444
##   0.001      7                  0.8902381  0.7804762
##   0.001      8                  0.8964286  0.7928571
##   0.001      9                  0.8989683  0.7979365
##   0.010      1                  0.8342063  0.6684127
##   0.010      2                  0.8579365  0.7158730
##   0.010      3                  0.8963492  0.7926984
##   0.010      4                  0.9109524  0.8219048
##   0.010      5                  0.9223016  0.8446032
##   0.010      6                  0.9305556  0.8611111
##   0.010      7                  0.9350000  0.8700000
##   0.010      8                  0.9379365  0.8758730
##   0.010      9                  0.9407937  0.8815873
##   0.050      1                  0.8983333  0.7966667
##   0.050      2                  0.9342063  0.8684127
##   0.050      3                  0.9492063  0.8984127
##   0.050      4                  0.9548413  0.9096825
##   0.050      5                  0.9604762  0.9209524
##   0.050      6                  0.9629365  0.9258730
##   0.050      7                  0.9645238  0.9290476
##   0.050      8                  0.9658730  0.9317460
##   0.050      9                  0.9677778  0.9355556
##   0.100      1                  0.9162698  0.8325397
##   0.100      2                  0.9503175  0.9006349
##   0.100      3                  0.9602381  0.9204762
##   0.100      4                  0.9647619  0.9295238
##   0.100      5                  0.9667460  0.9334921
##   0.100      6                  0.9687302  0.9374603
##   0.100      7                  0.9706349  0.9412698
##   0.100      8                  0.9713492  0.9426984
##   0.100      9                  0.9719841  0.9439683
##   0.300      1                  0.9388095  0.8776190
##   0.300      2                  0.9623016  0.9246032
##   0.300      3                  0.9699206  0.9398413
```

```
##   0.300      4                   0.9693651  0.9387302
##   0.300      5                   0.9715873  0.9431746
##   0.300      6                   0.9733333  0.9466667
##   0.300      7                   0.9727778  0.9455556
##   0.300      8                   0.9721429  0.9442857
##   0.300      9                   0.9739683  0.9479365
##
## Tuning parameter 'n.trees' was held constant at a value of 250
## Tuning
##  parameter 'n.minobsinnode' was held constant at a value of 20
## Kappa was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 250, interaction.depth =
##  9, shrinkage = 0.3 and n.minobsinnode = 20.
```

Il gbm usa 250 alberi con massima profondità uguale 9 e numero di nodi terminali pari a 20 e parametro di shrinkage pari a 0.3.

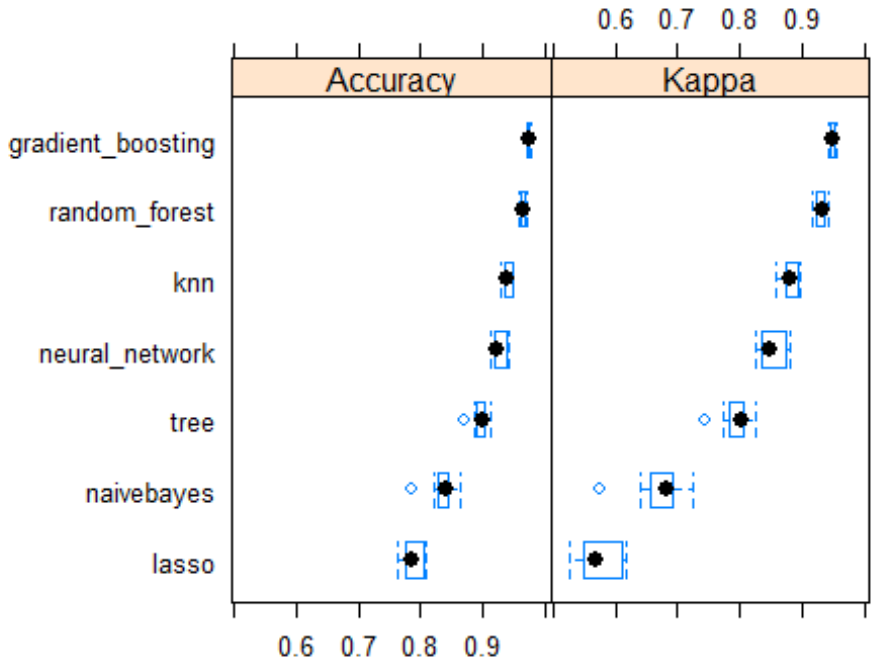
```
plot(gbm)
```



##Confronto metriche cross validate

```
library(caret)
results<-resamples(list(gradient_boosting=gbm,
                        naivebayes= naivebayes,
                        lasso=lasso,
                        random_forest=Random_Forest,
                        neural_network=NeuralNetwork,
                        knn=knn,
```

```
                        tree=tree))
bwplot(results)
```



I modelli migliori sembrerebbero il gradient boosting e la random forest, seguiti da knn, neural network e l'albero terminando con lasso e naivebajes. Questo garfico non corrisponde al vero confronto modelli perchè valutato sul dataset di training e per una sola soglia.

#STEP 2

##Scoring modelli sul validation

Calcoliamo le posterior

```
test$p_net = predict(NeuralNetwork, test, "prob")[,1]
test$p_lasso = predict(lasso, test, "prob")[,1]
test$p_rf = predict(Random_Forest, test, "prob")[,1]
test$p_knn = predict(knn, test, "prob")[,1]
test$p_tree = predict(tree, test, "prob")[,1]
test$p_nvby = predict(naivebayes, test, "prob")[,1]
test$p_gbm = predict(gbm, test, "prob")[,1]
```

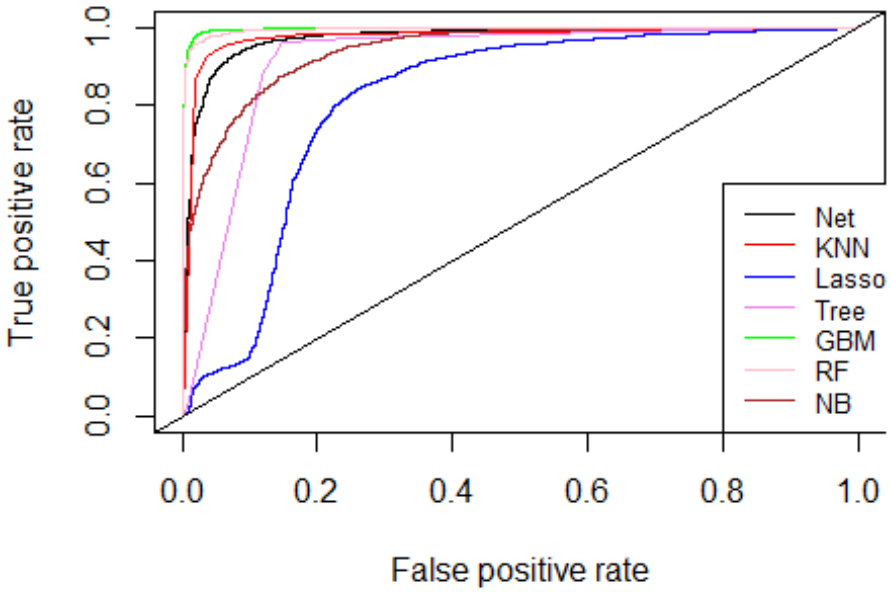
Aggiustiamo le posterior considerando le true prior: π(GALXY) = 0.73 π(STAR) = 0.27

```
test_f <- test
pg_rp <- 0.73/0.5
ps_rs <- 0.27/0.5
test["p_net"] <-  test$p_net * pg_rp / (test$p_net * pg_rp + (1-test$p_net) *ps_rs
)
test["p_lasso"] <-  test$p_lasso * pg_rp / (test$p_lasso * pg_rp + (1-test$p_lasso
```

```
) *ps_rs)
test["p_rf"] <-  test$p_rf * pg_rp / (test$p_rf * pg_rp + (1-test$p_rf) *ps_rs)
test["p_knn"] <-  test$p_knn * pg_rp / (test$p_knn * pg_rp + (1-test$p_knn) *ps_rs
)
test["p_tree"] <-  test$p_tree * pg_rp / (test$p_tree * pg_rp + (1-test$p_tree) *p
s_rs)
test["p_nvby"] <-  test$p_nvby * pg_rp / (test$p_nvby * pg_rp + (1-test$p_nvby) *p
s_rs)
test["p_gbm"] <-  test$p_gbm * pg_rp / (test$p_gbm * pg_rp + (1-test$p_gbm) *ps_rs
)
```

Valutazione curve ROC

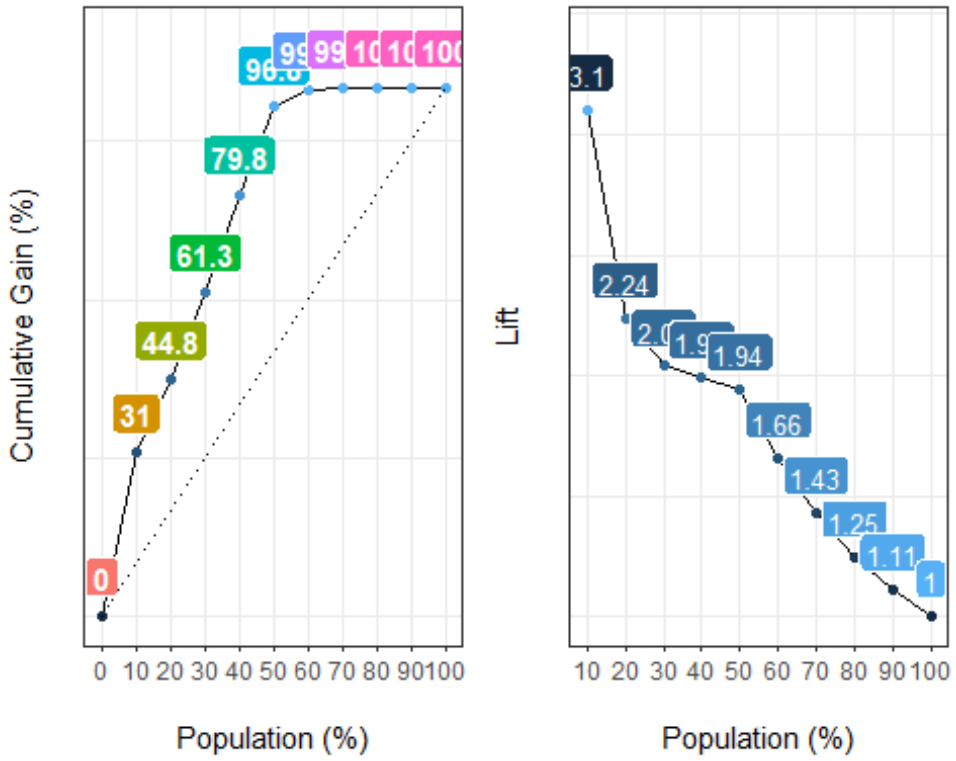
```
test$y <- test$class
test$y <- ifelse(test$y == "GALAXY", 1, 0)

library(ROCR)

## Warning: package 'ROCR' was built under R version 4.0.5

net_roc <- performance(prediction(test$p_net, test$y), measure = "tpr", x.measure
= "fpr")
nvby_roc <- performance(prediction(test$p_nvby, test$y), measure = "tpr", x.measur
e = "fpr")
knn_roc <- performance(prediction(test$p_knn, test$y), measure = "tpr", x.measure
= "fpr")
lasso_roc <- performance(prediction(test$p_lasso, test$y), measure = "tpr", x.meas
ure = "fpr")
tree_roc <- performance(prediction(test$p_tree, test$y), measure = "tpr", x.measur
e = "fpr")
gmb_roc <- performance(prediction(test$p_gbm, test$y), measure = "tpr", x.measure
= "fpr")
rf_roc <- performance(prediction(test$p_rf, test$y), measure = "tpr", x.measure =
"fpr")

plot(net_roc)
plot(knn_roc,add=T,col="red")
plot(lasso_roc,add=T,col="blue")
plot(tree_roc,add=T,col="violet")
plot(nvby_roc,add=T,col="brown")
plot(gmb_roc,add=T,col="green")
plot(rf_roc,add=T,col="pink")
abline(a=0,b=1)
legend(0.8,0.6, legend=c("Net", "KNN","Lasso", "Tree", "GBM", "RF","NB"),
       col=c("black","red","blue", "violet", "green", "pink","brown"),lty=1, cex=0
.8)
```



I modelli vincenti risultano Random Forest e Gradient Boosting. Dato che le curve si intersecano, valutiamo il modello vincente confrontando le curve Lift.

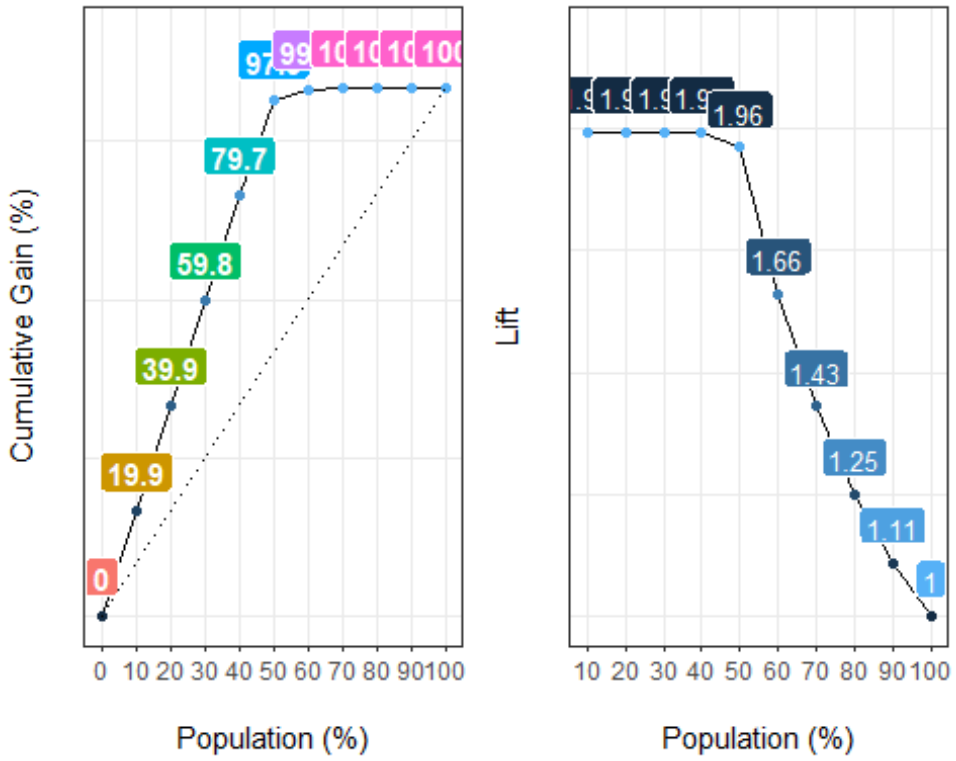
Valutazione curve Lift

```
library(funModeling)
gain_lift(data = test, score = 'p_rf', target = 'class')
```




```
##    Population   Gain Lift Score.Point
## 1          10  30.97 3.10  0.99851681
## 2          20  44.80 2.24  0.99552783
## 3          30  61.33 2.04  0.98792081
## 4          40  79.80 1.99  0.96220693
## 5          50  96.80 1.94  0.72364603
## 6          60  99.70 1.66  0.21503586
## 7          70  99.93 1.43  0.07225679
## 8          80 100.00 1.25  0.02133879
## 9          90 100.00 1.11  0.00000000
## 10        100 100.00 1.00  0.00000000
```

```
gain_lift(data = test, score = 'p_gbm', target = 'class')
```




```
##    Population   Gain Lift      Score.Point
## 1          10  19.90 1.99 0.99975124781814
## 2          20  39.90 1.99 0.99945904468050
## 3          30  59.83 1.99 0.99874024276336
## 4          40  79.73 1.99 0.99593615996317
## 5          50  97.90 1.96 0.82612459807688
## 6          60  99.87 1.66 0.01662735305643
## 7          70 100.00 1.43 0.00247795515872
## 8          80 100.00 1.25 0.00058838107368
## 9          90 100.00 1.11 0.00013934482003
## 10        100 100.00 1.00 0.00000003153231
```

##Modello Vincente

Confronto Il modello vincente e' la Random Forest poichè vince il confronto puntuale con il Gradietn Boosting.

STEP 3

##Studio della soglia

Valutiamo il Kappa per ogni soglia per scegliere quella ottimale.

```
library(dplyr)
thresholds <- seq(from = 0, to = 1, by = 0.01)
prop_table <- data.frame(threshold = thresholds, prop_true_G = NA,  prop_true_S =
NA, true_G = NA,  true_S = NA ,fn_G=NA)
```

```
for (threshold in thresholds) {
  pred <- ifelse(test$p_rf > threshold, "GALAXY", "STAR")
  pred_t <- ifelse(pred == test$class, TRUE, FALSE)

  group <- data.frame(test, "pred" = pred_t) %>%
    group_by(class, pred) %>%
    dplyr::summarise(n = n())

  group_G <- filter(group, class == "GALAXY")

  true_G=sum(filter(group_G, pred == TRUE)$n)
  prop_G <- sum(filter(group_G, pred == TRUE)$n) / sum(group_G$n)

  prop_table[prop_table$threshold == threshold, "prop_true_G"] <- prop_G
  prop_table[prop_table$threshold == threshold, "true_G"] <- true_G

  fn_G=sum(filter(group_G, pred == FALSE)$n)
  prop_table[prop_table$threshold == threshold, "fn_G"] <- fn_G


  group_S <- filter(group, class == "STAR")

  true_S=sum(filter(group_S, pred == TRUE)$n)
  prop_S <- sum(filter(group_S, pred == TRUE)$n) / sum(group_S$n)

  prop_table[prop_table$threshold == threshold, "prop_true_S"] <- prop_S
  prop_table[prop_table$threshold == threshold, "true_S"] <- true_S
  fn_S=sum(filter(group_S, pred == FALSE)$n)
  prop_table[prop_table$threshold == threshold, "fn_S"] <- fn_S
}
```

Calcoliamo il Kappa:

```
prop_table$N <- prop_table$true_G + prop_table$true_S + prop_table$fn_G + prop_tab
le$fn_S
prop_table$acc_fin <- (prop_table$true_G+prop_table$true_S)/prop_table$N
prop_table$expTG <- (prop_table$true_G+prop_table$fn_S)*(prop_table$true_G+prop_ta
ble$fn_G)/prop_table$N
prop_table$expTS <- (prop_table$fn_S+prop_table$true_S)*(prop_table$fn_G+prop_tabl
e$true_S)/prop_table$N
prop_table$expFG <- (prop_table$true_G+prop_table$fn_G)*(prop_table$fn_G+prop_tabl
e$true_S)/prop_table$N
prop_table$expFS <- (prop_table$true_G+prop_table$fn_S)*(prop_table$fn_S+prop_tabl
e$true_S)/prop_table$N
prop_table$expacc <- (prop_table$expTG + prop_table$expTS)/prop_table$N
prop_table$k <- (prop_table$acc_fin - prop_table$expacc)/(1 - prop_table$expacc)
prop_table[seq(1,101, by=10),c(1,15)]
```

```
##     threshold         k
## 1         0.0 0.2570000
## 11        0.1 0.6480000
## 21        0.2 0.7830000
## 31        0.3 0.8446667
## 41        0.4 0.8840000
## 51        0.5 0.9113333
## 61        0.6 0.9266667
## 71        0.7 0.9366667
## 81        0.8 0.9353333
## 91        0.9 0.8983333
## 101       1.0 0.0000000
```

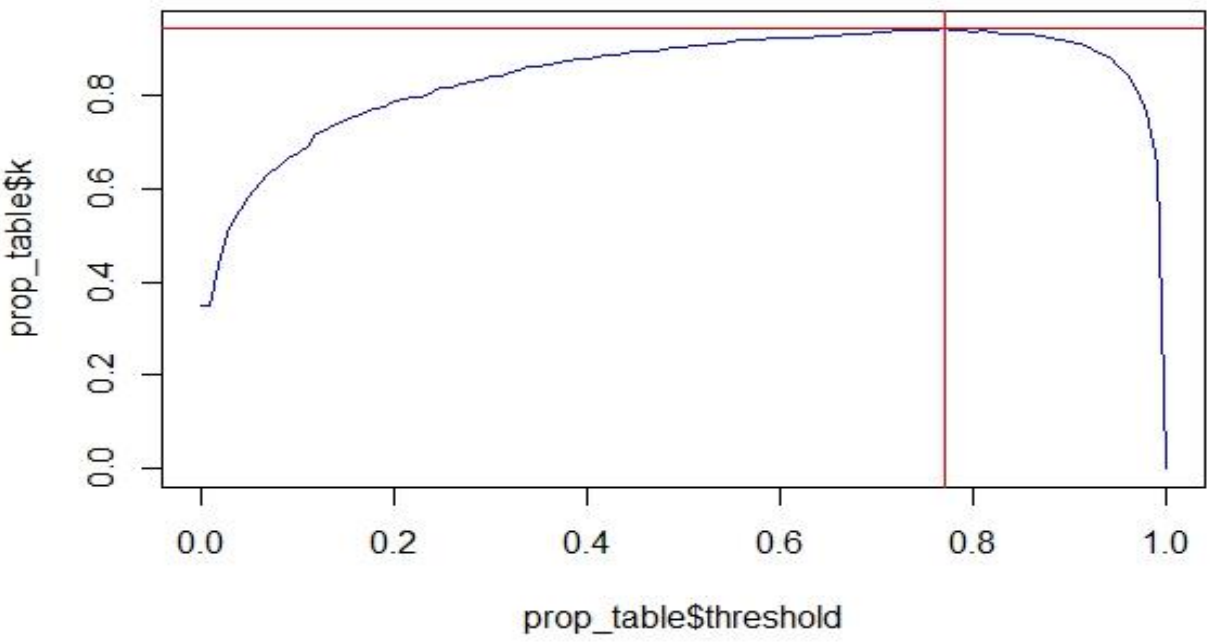
```
max_k <- lapply(list(prop_table$k), function(x) x[which.max(abs(x))])
threshold_fin<-prop_table[which(prop_table$k == max_k), ]$threshold
k_fin<-prop_table[which(prop_table$k == max_k), ]$k
threshold_fin[1]
```

```
## [1] 0.77
```

```
k_fin[1]
```

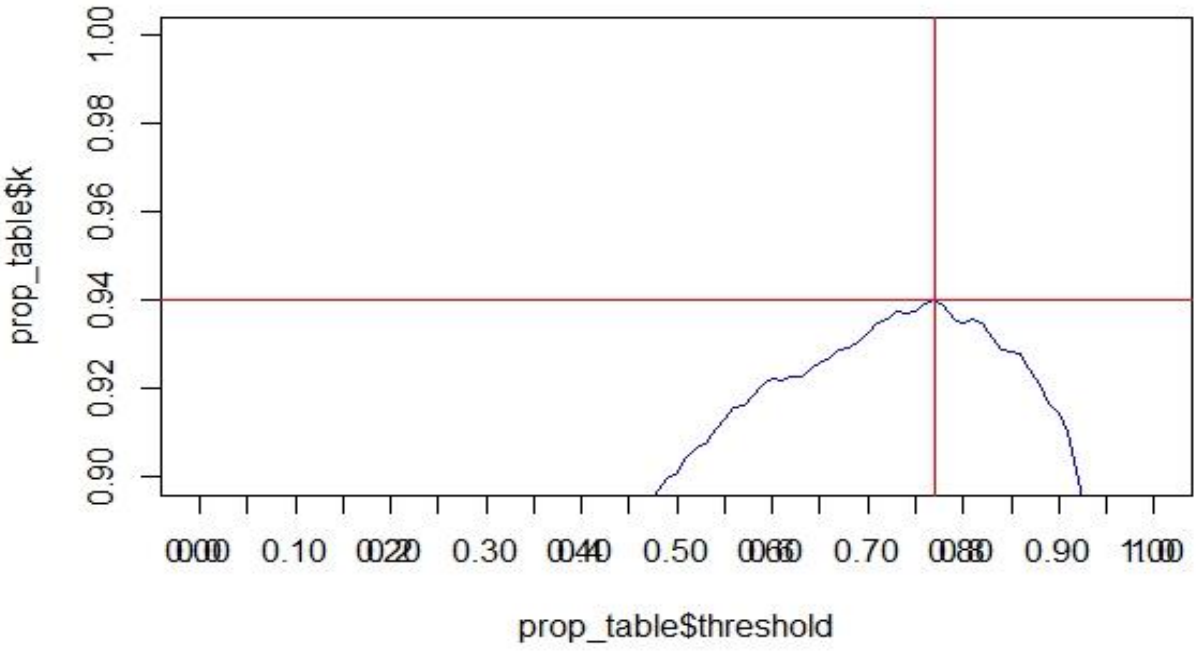
```
## [1] 0.938
```

```
plot(prop_table$threshold, prop_table$k, type='l', col='blue')
abline(h = k_fin[1], col='red')
abline(v = threshold_fin[1], col='red')
```



zoom

```
plot(prop_table$threshold, prop_table$k, col='blue', ylim=c(0.9,1), type='l')
axis(side=1, at=seq(0.0, 1.0, by=0.05))
abline(h = k_fin[1], col='red')
abline(v = threshold_fin[1], col='red')
```



Si osserva che il Kappa massimo pari a 0.94 corrisponde ad una soglia pari a 0.77.

Matrice di confusione con la soglia scelta

```{r}
test$pred <- ifelse(test$p_rf > threshold_fin[1], "GALAXY", "STAR")
table1<-table(actual=test$class, test$pred)
table1
```

```
actual   GALAXY STAR
  GALAXY   2902   98
  STAR       82 2918
```

Calcoliamo le seguenti metriche:

```{r}
TP1 <- (table(actual=test$class, test$pred)[1,1])
TP<-TP1*pg_rp
TN1 <- (table(actual=test$class, test$pred)[2,2])
TN<-TN1*ps_rs
FP1 <- (table(actual=test$class, test$pred)[2,1])
FP<-FP1*ps_rs
FN1 <- (table(actual=test$class, test$pred)[1,2])
FN<-FN1*pg_rp
N  <- TP + TN + FP + FN
acc_fin <- (TP+TN)/N
tpr <- TP/(TP+FN)
tnr <-  TN/(FP+TN)
print(c('Accuracy : ', round(acc_fin, 2)))
print(c('Kappa : ', k_fin[1]))
print(c('Specificity : ', round(tpr, 2)))
print(c('Sensitivity  : ', round(tnr, 2)))
```

```
[1] "Accuracy : " "0.97"
[1] "Kappa : " "0.94"
[1] "Specificity : " "0.97"
[1] "Sensitivity  : " "0.97"
```

STEP 4

Scoring nuovi dati

Usiamo il modello con la soglia scelta per stimare il target previsto sui dati di score.

```
score$prob <-  predict(Random_Forest, score, type = "prob")[,1]
score$pred_y=ifelse(score$prob > threshold_fin[1], "GALAXY","STAR")
table(score$pred_y)

##
## GALAXY   STAR
##    613    656
```