



CLASSIFICAZIONE DI DUE SPECIE DI PISTACCIO

Conti Sara 837969

Pedemonte Marzia 890369

INDICE

Descrizione del problema

Preparazione e descrizione del dataset

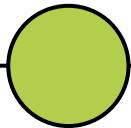
Divisione del dataset in training e test

Implementazione dei modelli competitivi

Comparazione dei risultati

Previsioni

Conclusioni



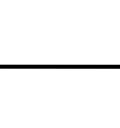
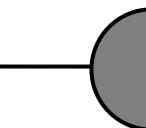
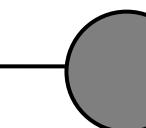
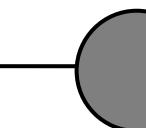
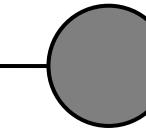
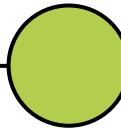
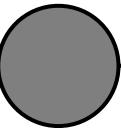
Descrizione Problema

- **OBIETTIVO:** classificare due specie di pistacchio: **Kirmizi** e **Siirt**, le più coltivate in Turchia
 - ✓ frutti più abbondanti
 - ✓ crescono durante tutte le stagioni
- Kirmizi: prodotti della pasticceria, per il colore verde scuro e il sapore/aroma intenso;
- Siirt: snack, per la sua croccantezza e forma tondeggiante.

► Perché sono necessari metodi di classificazione?

Una classificazione efficace consentirebbe di aumentare l'**efficienza** dei processi post-raccolta del prodotto

- aumento della qualità del prodotto finale
- diminuzione del prezzo
- maggiore accessibilità dal consumatore



Descrizione Dataset

■ **VARIABILI:** sono 28 e sono state ottenute attraverso un processo di estrazione di caratteristiche da 2148 immagini. Non sono presenti dati mancanti.

- ▷ **Aspetti morfologici** (Area, Perimetro, Asse maggiore, Asse minore, Diametro equivalente, Area del rettangolo di delimitazione, ...)
- ▷ **Aspetti di forma** (fattori di forma ricavati da alcuni aspetti morfologici, per esempio:
 $\text{Aspect_Ratio} = L/I$)
- ▷ **Caratteristiche delle distribuzioni dei colori primari delle immagini** (Media, Deviazione standard, Skewness, Curtosi dei valori di intensità dei colori primari: rosso, verde e blu)

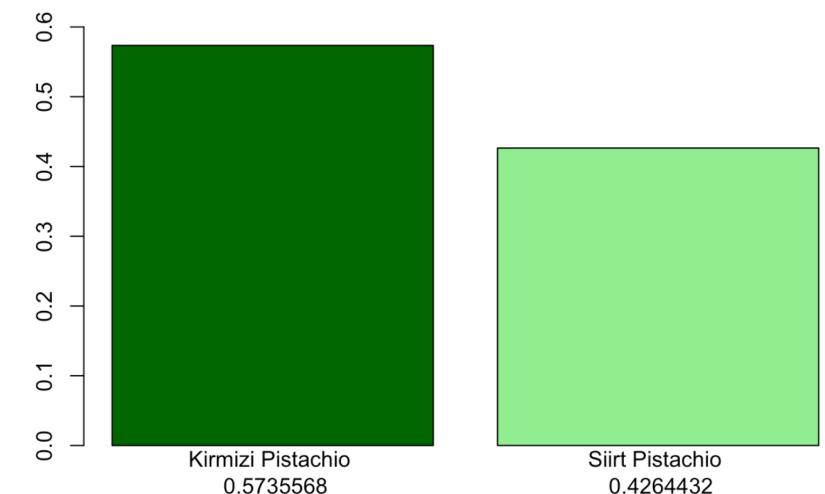
Kirmizi

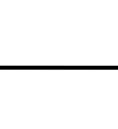
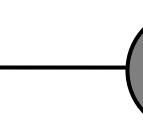
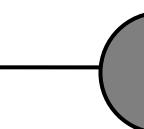
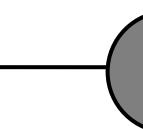
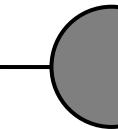
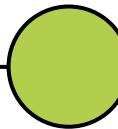
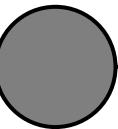


Siirt



i dati sono bilanciati





Descrizione Dataset

COLLINEARITA':

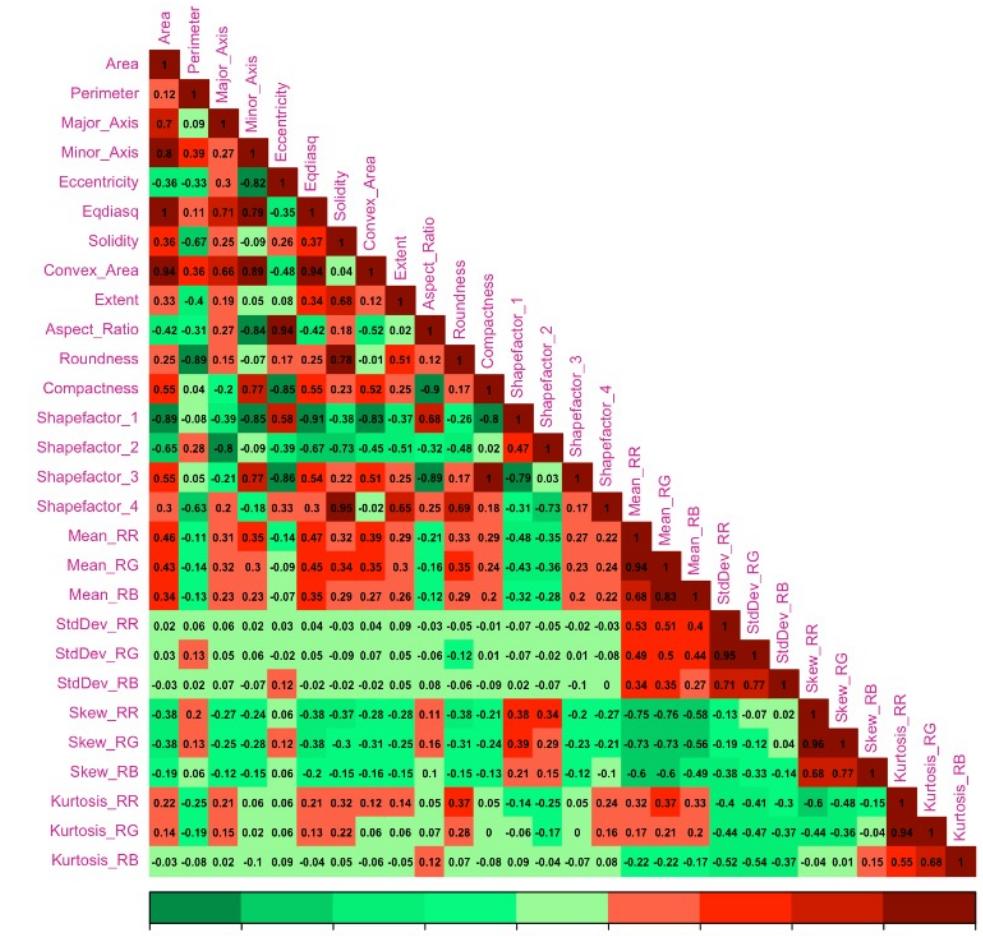
- molte correlazioni elevate: si eliminano variabili fortemente correlate (>0.75) → **12** rimanenti

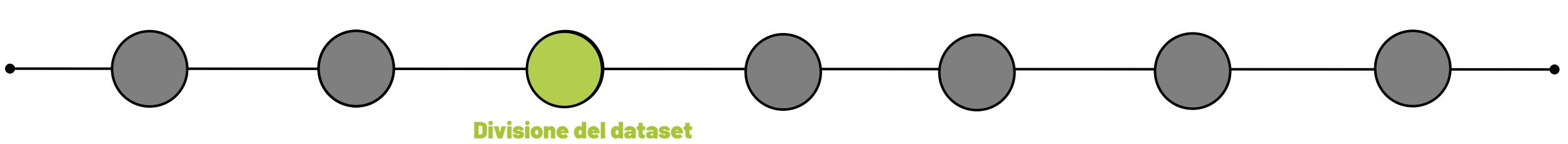
DIFFERENZA TRA MEDIE NELLE DUE SPECIE:

- test di Wilcoxon (no normalità);
- valori medi di **Siirt** più elevati di **Kirmizi**, eccetto *Kurtosis_RB* (pvalue=0.422);

NEAR-ZERO e ZERO VARIANCE:

- quando i valori di una covariata sono molto simili/identici.
- problematica non presente





■ Suddivisione del dataset in:

- dataset di **training** (*train*): 70% dei dati originali;
- dataset di **test** (*test*): 30% dei dati d'origine;

• **CreateDataPartition**: partiziona il dataset e mantiene le frequenze relative nei due dataset creati simili fra loro e a quelle del dataset di partenza.

```
set.seed(123) # Per suddividere sempre nello stesso modo il dataset
cpart <- createDataPartition(y=data$Class,times=1,p=.7)
train <- data[cpart$Resample1,]
test <- data[-cpart$Resample1,]

table(train$Class)/nrow(train)
```

Implementazione di modelli competitivi

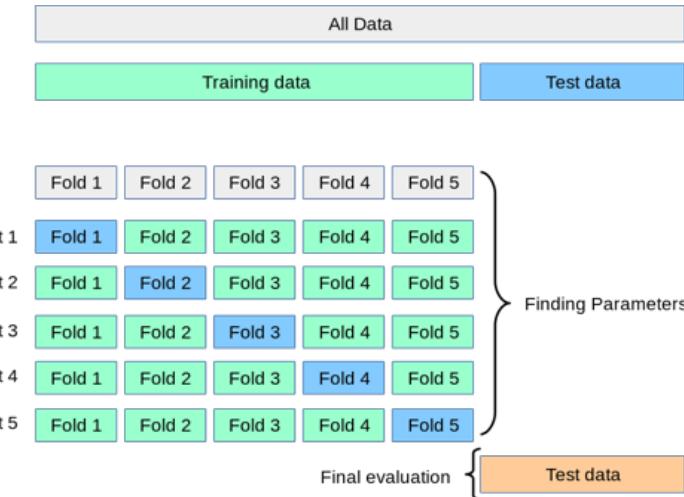
FUNZIONE UTILIZZATA: *train* della libreria ***caret***

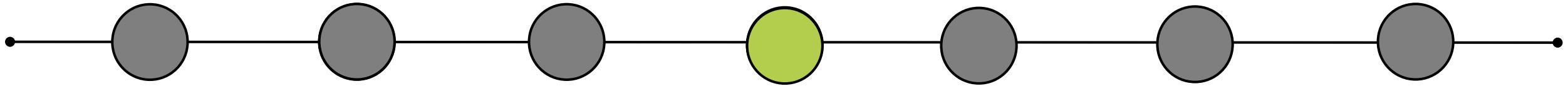
- Vantaggio: permette di addestrare i modelli utilizzando la **k-fold-cross-validation**: metodo di **ricampionamento** in cui si suddivide il dataset di train in diverse porzioni utilizzate in più iterazioni per addestrare e testare il modello.
- Sintassi generale: ***train(formula, data, method, trControl, metric,...)***

specifica la variabile
risposta e le covariate

metodo di
classificazione/
regressione
utilizzato

oggetto che specifica la tecnica
di ricampionamento da utilizzare;
in questo caso: **10-fold cross validation**
trainControl(method = "cv", number = 10)





Implementazione di modelli competitivi

■ TARGET per Caret:

La funzione **train** richiede la creazione di una nuova variabile risposta **r** che assume valori **r0 (Kirmizi)** o **r1 (Siirt)** in base alla specie di pistacchio.

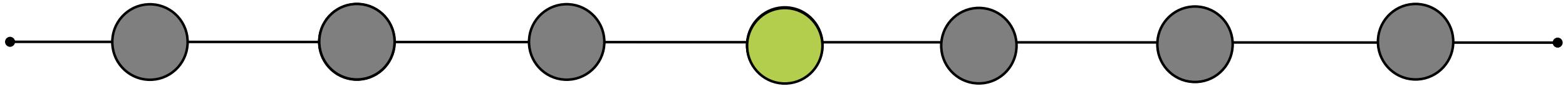
■ Metrica di interesse: **ROC**

- Si vogliono minimizzare sia FP sia FN: non ci sono motivi per massimizzare SE o SP
- **SE=VP/(VP+FN); SP=VN/(VN+FP)**

■ Per valutare i modelli si calcolano sul dataset di test:

- i valori di **AUC**,
- le matrici di confusione con **accuratezza** → associata

definisce la proporzione di previsioni corrette rispetto al numero totale di previsioni $(VP+VN)/(FP+FN+VP+VN)$



Implementazione di modelli competitivi

■ IMPLEMENTAZIONE DEI MODELLI

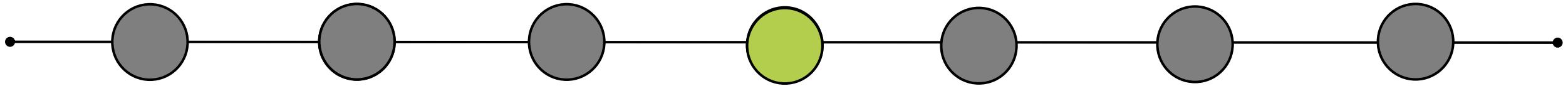
sul dataset di training sono stati addestrati:

1) Modelli che **non** richiedono model selection:

Ridge, Lasso, Elastic Net, Decision Tree, Random Forest;

2) Modelli in cui è **preferibile**: KNN, PLS, Rete Neurale

- una **prima selezione** ottenuta dal modello **LASSO**
- una **seconda selezione** dal modello **TREE**.



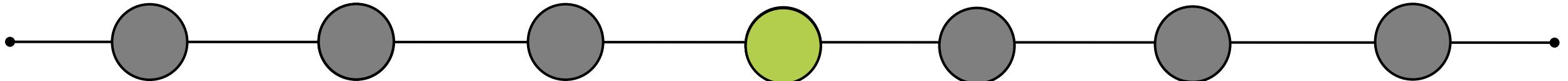
Implementazione di modelli competitivi

1. Modello RIDGE

- Modello di regressione che fa parte dei metodi di **regolarizzazione/regressione penalizzata**.
- Utile quando ci sono molti predittori e/o problemi di multicollinearità
Si cercano i valori dei coefficienti che minimizzano:
 - la somma dei residui al quadrato
 - un **termine di penalità**
è piccolo per valori dei coefficienti ~ zero
→ **restringe** le stime dei coefficienti verso lo zero
- Riduce la dimensione dei coefficienti mantenendo **tutte** le variabili nel modello.

$$RSS + \lambda \sum_{j=1}^p \beta_j^2$$

λ : il valore ottimale viene spesso determinato con **cross-validation**



Implementazione di modelli competitivi

1. Modello RIDGE

```
set.seed(1)
ctrl <- trainControl(method="cv", number=10, classProbs = TRUE,
                      summaryFunction=twoClassSummary)
grid <- expand.grid(.alpha=0, .lambda=seq(0, 1, by = 0.01))
ridge <- train(r~.-class,
               data=train, method="glmnet", metric="ROC",
               trControl=ctrl, na.action=na.pass,
               tuneGrid=grid, preProcess=c("scale", "center"))
```

- **grid**: valori del parametro *lambda* da considerare; *alpha=0* parametro dell'elastic net.
- **metric=ROC**: metrifica utilizzata per selezionare il modello ottimale.
- **preProcess=c(scale,center)** scala e centra le variabili

La funzione valuta qual è il **valore** di λ per cui si ottiene il modello migliore (che massimizza la ROC)
→ ridge\$bestTune: **0.03**

il modello ottenuto, addestrato sul dataset di train, viene **valutato** sul dataset di test

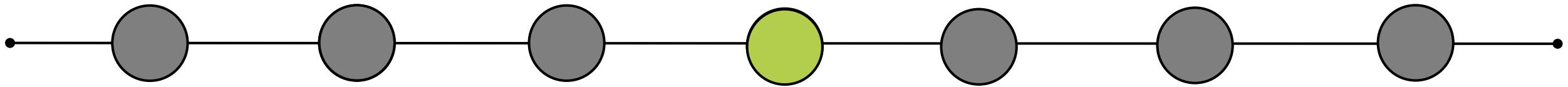
• **Accuracy**=0.90851, **AUC**=0.9613

```
test$pred_ridge <- predict(ridge, test, "prob")[,2]
suppressPackageStartupMessages(library(pROC))
suppressMessages(roc(as.numeric(Class) ~ pred_ridge, data=test))

## Call:
## roc.formula(formula = as.numeric(Class) ~ pred_ridge, data = test)
##
## Data: pred_ridge in 369 controls (as.numeric(Class) 1) < 274 cases
## Area under the curve: 0.9613
```

```
pred_ridge <- predict(ridge, newdata=test)
cm_ridge <- confusionMatrix(pred_ridge, as.factor(test$r))
cm_ridge
```

```
## Confusion Matrix and Statistics
##
##                 Reference
## Prediction      r0     r1
##                r0 338 30
##                r1  31 244
##
##                               Accuracy : 0.9051
##                               95% CI : (0.8798, 0.9267)
## No Information Rate : 0.5739
## P-Value [Acc > NIR] : <0.0000000000000002
##
##                               Kappa : 0.8061
##
## Mcnemar's Test P-Value : 1
##
##                               Sensitivity : 0.9160
##                               Specificity : 0.8905
## Pos Pred Value : 0.9185
## Neg Pred Value : 0.8873
## Prevalence : 0.5739
## Detection Rate : 0.5257
## Detection Prevalence : 0.5723
## Balanced Accuracy : 0.9033
##
## 'Positive' Class : r0
```



Implementazione di modelli competitivi

2. Modello LASSO (Least Absolute Shrinkage and Selection Operator)

- Modello di regressione che fa parte dei metodi di **regressione penalizzata** come il RIDGE, cambia il **termine di penalità**:

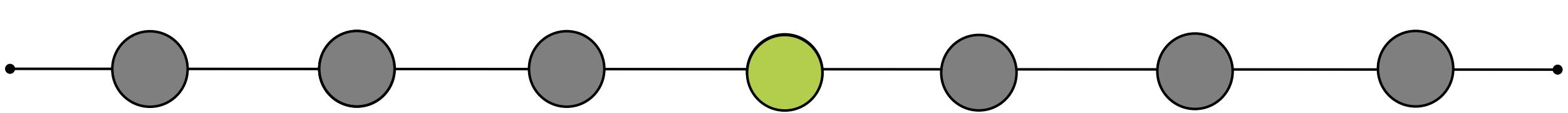
In questo caso alcuni coefficienti possono **azzerarsi** → metodo di **selezione delle variabili**

$$RSS + \lambda \sum_{j=1}^p |\beta_j|$$

```
set.seed(1)
ctrl <- trainControl(method="cv", number=10, classProbs = TRUE,
                      summaryFunction=twoClassSummary)
grid <- expand.grid(.alpha=1, .lambda=seq(0, 1, by = 0.01))
lasso <- train(r~-class,
               data=train, method="glmnet", metric="ROC",
               trControl=ctrl, na.action=na.pass,
               tuneGrid=grid, preProcess=c("scale","center"))
...
```

Il valore di λ che permette di ottenere il modello migliore in termini di ROC è **$\lambda=0$**
→ coincidenza con regressione logistica

- **Accuracy**=0.9082, **AUC**=0.9622 (test)



Implementazione di modelli competitivi

2. LASSO: selezione delle variabili

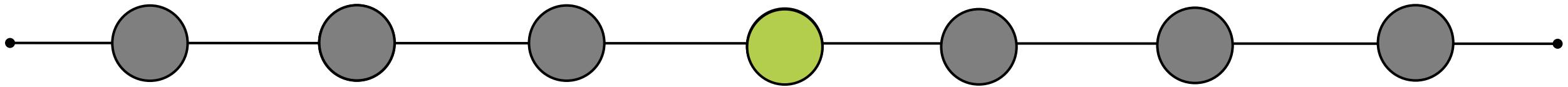
- lamba ottimale 0 → modello logistico per la **selezione** delle variabili.

```
glm <- glm(Class~.-r,  
            data=train, family=binomial(link="logit"))  
drop1(glm, test="LRT")
```

	Df <dbl>	Deviance <dbl>	AIC <dbl>	LRT <dbl>	Pr(>Chi) <dbl>
<none>	NA	823.3867	849.3867	NA	NA
Perimeter	1	825.4889	849.4889	2.102244	0.1470831567151285879103
Major_Axis	1	833.9395	857.9395	10.552827	0.0011601052322537399285
Convex_Area	1	824.9025	848.9025	1.515860	0.2182471218500681708896
Extent	1	830.2723	854.2723	6.885684	0.0086888793817916226692
Shapefactor_3	1	863.6234	887.6234	40.236712	0.0000000002249803922555
Shapefactor_4	1	830.4256	854.4256	7.038922	0.0079757022324939991914
Mean_RB	1	849.5919	873.5919	26.205203	0.0000003069942248384750
StdDev_RR	1	891.1480	915.1480	67.761327	0.00000000000000001845337
StdDev_RB	1	828.5067	852.5067	5.120080	0.0236505324622169367521
Skew_RB	1	875.1627	899.1627	51.776023	0.0000000000006220684825
Kurtosis_RG	1	834.7766	858.7766	11.389891	0.0007384486371390711588
Kurtosis_RB	1	829.4618	853.4618	6.075126	0.0137098557850553835064

1-13 of 13 rows

10 variabili selezionate: Shapefactor_3, Shapefactor_4, Extent, Skew_RB, Kurtosis_RG, Kurtosis_RB, StdDev_RR, StdDev_RB, Mean_RB, Major_Axis.



Implementazione di modelli competitivi

3. Modello ELASTIC NET

■ Metodo di **regressione penalizzata**. Combina i termini di **penalità** dei modelli RIDGE e LASSO superando le criticità di entrambi.

-**RIDGE**: mantiene **tutte** le variabili nel modello e rimane complesso → performance peggiori

-**LASSO**: se seleziona una variabile che è molto correlata con altre, ne mantiene solo una azzerando le altre → riduzione accuratezza e ha un numero massimo di predittori non nulli che può selezionare (n).

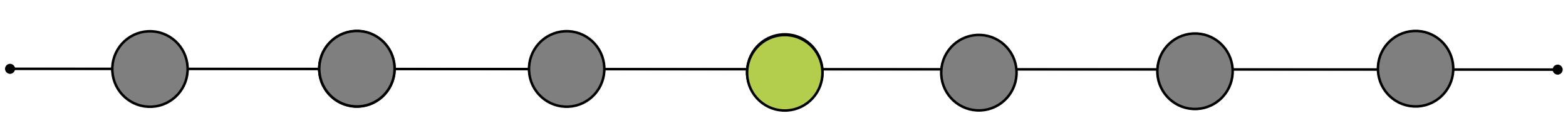
- **ELASTIC NET**: come LASSO: selezione delle variabili. Ma se una variabile selezionata è molto correlata ad altre vengono selezionate anch'esse.

$$RSS + \alpha \lambda \sum_{j=1}^p |\beta_j| + \frac{1-\alpha}{2} \lambda \sum_{j=1}^p \beta_j^2$$

α : il valore ottimale si può trovare con **cross-validation**

$\alpha=0$: penalità RIDGE.

$\alpha=1$: penalità LASSO.



Implementazione di modelli competitivi

3. Modello ELASTIC NET

```
set.seed(1)
ctrl <- trainControl(method="cv", number=10, classProbs = TRUE, summaryFunction=twoClassSummary)
grid <- expand.grid(.alpha=seq(0,1,by=0.01), .lambda=seq(0, 1, by = 0.01))
elastic_all <- train(r~-Class,
                      data=train, method="glmnet", metric="ROC",
                      trControl=ctrl, na.action=na.pass,
                      tuneGrid=grid, preProcess=c("scale", "center"))
```

```
elastic_all$results[elastic_all$results$ROC==max(elastic_all$results$ROC),]
```

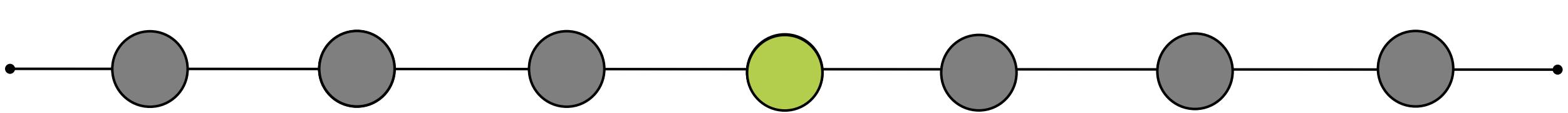
alpha <dbl>	lambda <dbl>	ROC <dbl>	Sens <dbl>	Spec <dbl>	ROCSD <dbl>	SensSD <dbl>	SpecSD <dbl>
3031	0.3	0	0.9501281	0.9037156	0.8380288	0.01768784	0.04294411
1 row							

- **grid:** valori per *lambda* e *alpha*

Modello che massimizza la ROC:
 $\alpha = 0.3, \lambda = 0 \rightarrow$ termine di penalità
azzerato

Il modello ottenuto, addestrato sul dataset di train, viene valutato sul dataset di test:

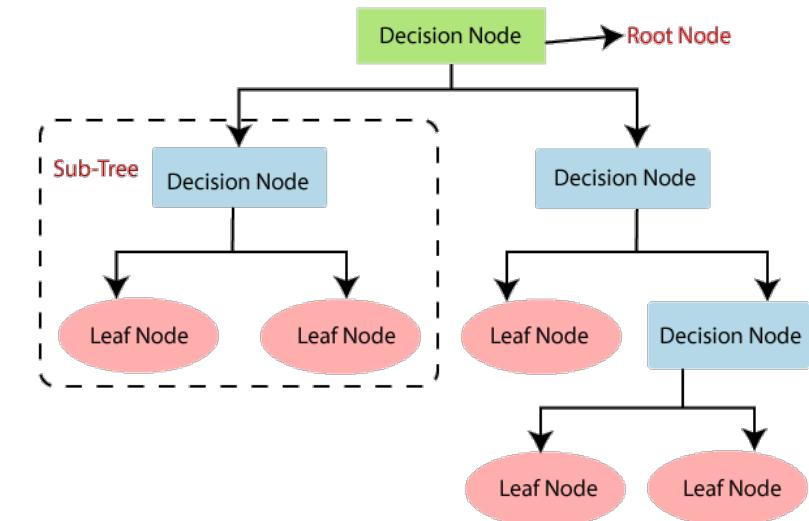
- **Accuracy**= 0.9082
- **AUC**= 0.9621



Implementazione di modelli competitivi

4. Modello DECISION TREE

- È caratterizzato da una struttura gerarchica con un **nodo radice**, **nodi decisionali** e **nodi foglie**.
- Costruzione basata su un algoritmo iterativo che individua per ogni nodo la **suddivisione migliore**, individuata, nella classificazione, utilizzando misure di impurità come l'**indice di Gini**/l'entropia.
- **Limiti:** poco robusto e **overfitting** sui dati di training → metodi che aggregano molti alberi (RF, Bagging, Boosting)



4. Modello DECISION TREE

```
set.seed(1)
cvCtrl <- trainControl(method="cv", number=10, search="grid", classProbs = TRUE, summaryFunction=twoClassSummary,
savePredictions=TRUE)
tree <- train(r~-Class,
              data=train, method="rpart", metric="ROC",
              tuneLength=10,
              trControl=cvCtrl, na.action=na.pass)
```

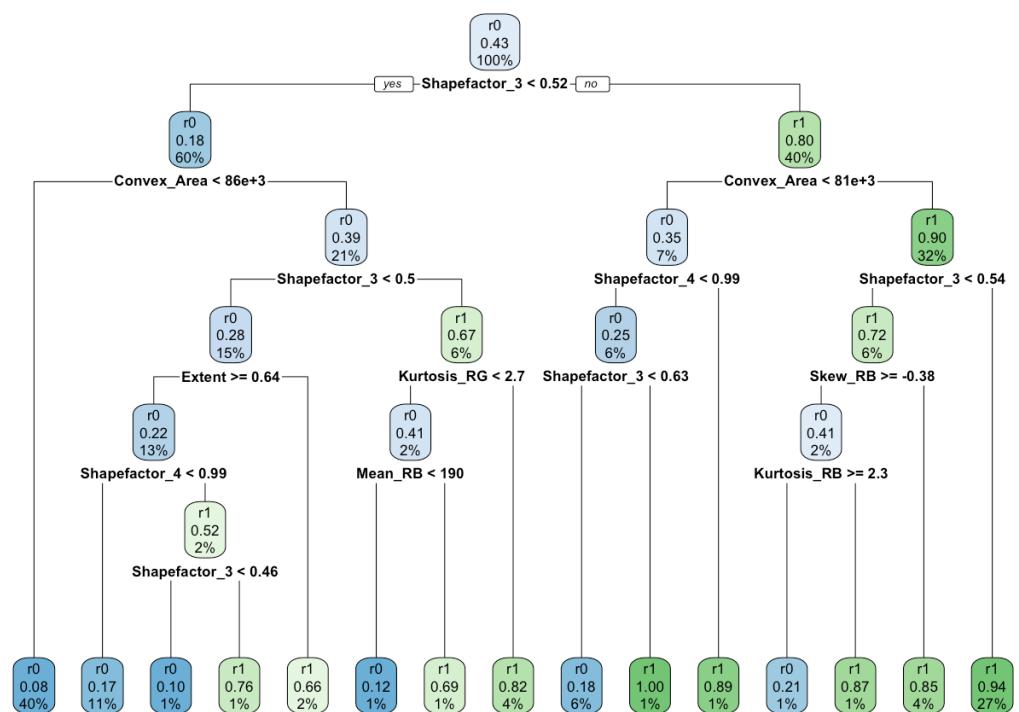
- **tuneLength=10**: valori di **cp** da considerare (riferito alla dimensione dell'albero)
- **method=rpart**: albero decisionale con misura di impurità di default=**indice di Gini**.

La funzione sceglie il valore di cp a cui corrisponde il modello che massimizza la ROC

→ tree\$bestTune: 0.00467

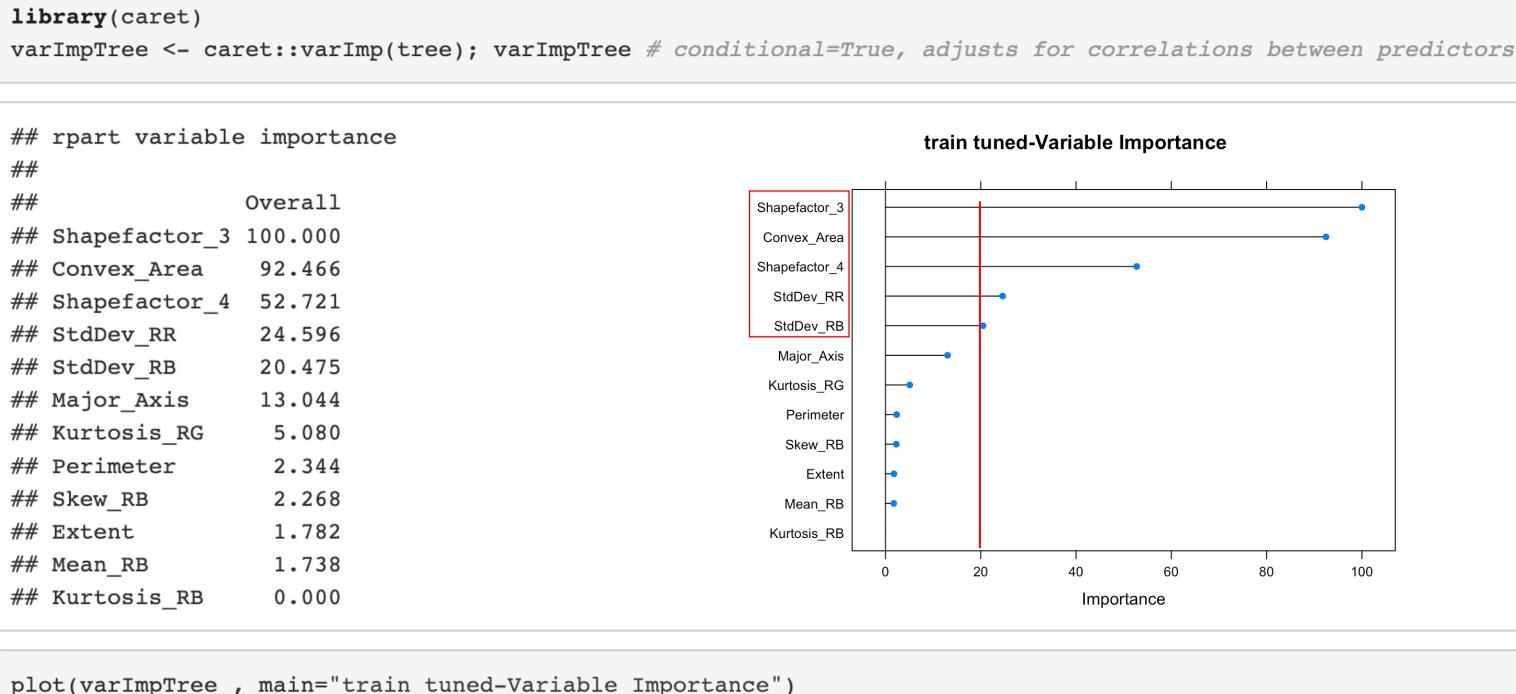
- **Accuracy**= 0.8787, **AUC**= 0.9179

```
suppressPackageStartupMessages(library(rpart.plot))
rpart.plot(tree$finalModel, extra=106)
```



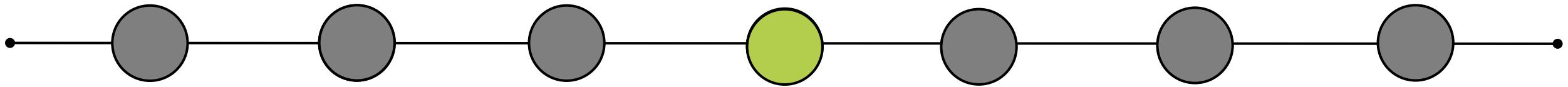
2. TREE: importanza delle variabili

- La variabile più importante è `Shapefactor_3`.



Scelta delle variabili da selezionare: quelle con un'importanza relativa > media=26.4 → solo 3 → si abbassa la **soglia al 20%**

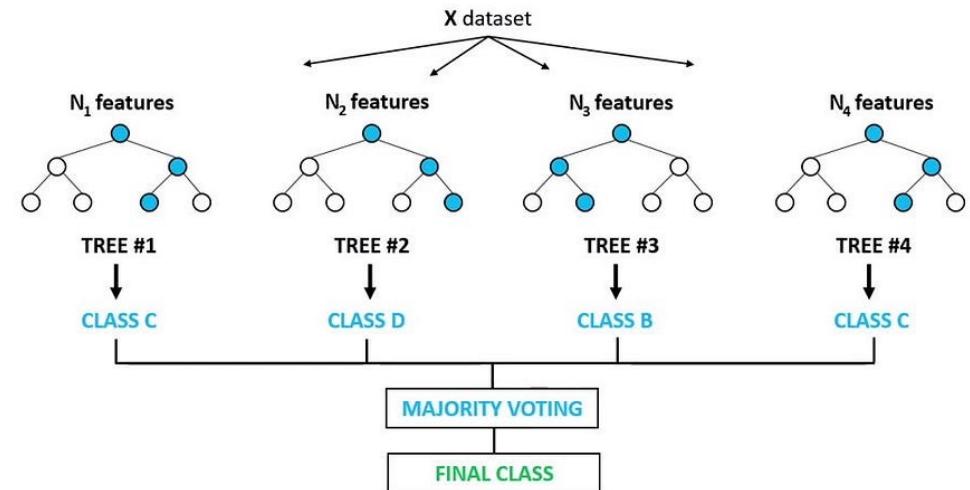
5 variabili selezionate:
Shapefactor_3,
Convex_Area,
Shapefactor_4, StdDev_RR,
StdDev_RB.

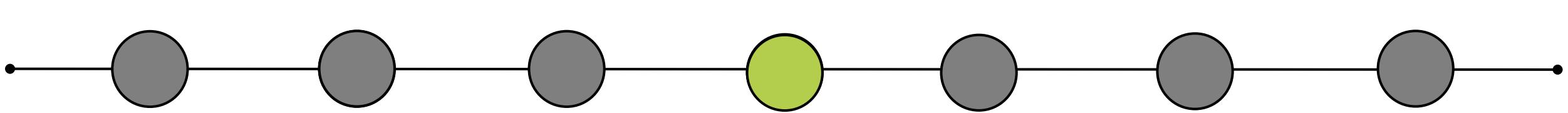


Implementazione di modelli competitivi

5. Modello RANDOM FOREST

- Aggrega molti **alberi decisionali** addestrati su diversi sotto-campioni dei dati di partenza (campioni bootstrap).
- Rispetto al bagging, consente di ottenere alberi non correlati: in ciascun nodo viene considerato un sottoinsieme casuale delle variabili → maggiore diversità
- Nella **classificazione**, l'output consiste nella **classe** selezionata dalla maggior parte degli alberi.





Implementazione di modelli competitivi

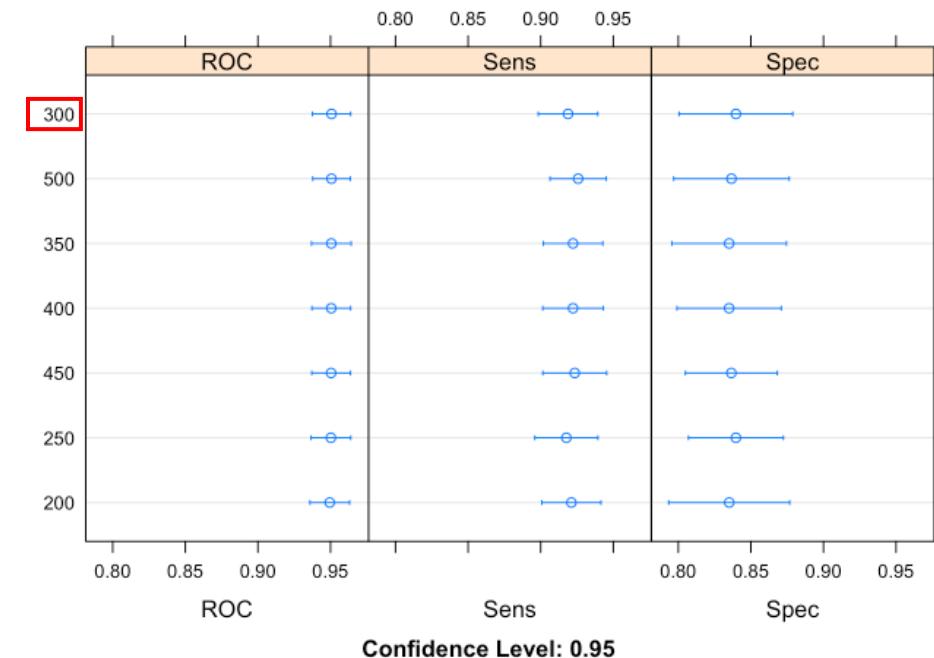
5. RF- ntree migliore

- Selezione numero migliore di alberi **ntree** con un ciclo *for* da 200 a 500 (valore di default)

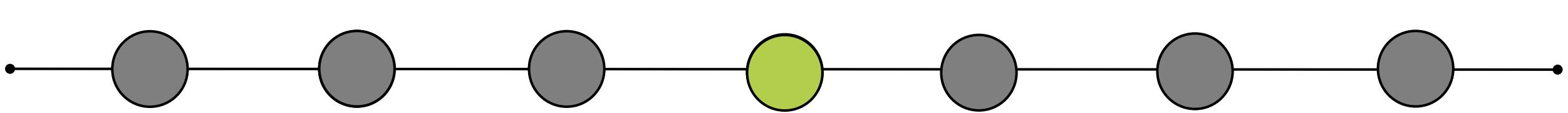
```
set.seed(1)
control <- trainControl(method="cv", number=10, search="grid", summaryFunction=twoClassSummary, classProbs = TRUE)
tunagrid <- expand.grid(.mtry=c(1:6))
modellist <- list()
for (ntree in seq(200,500, by=50)){
  set.seed(123)
  rf <- train(r~-Class,
              data = train,
              method = 'rf',
              metric = 'ROC',
              tuneGrid = tunagrid,
              trControl = control,
              ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- rf
}

results <- resamples(modellist)
s <- summary(results)

dotplot(results)
```



Risultati simili, si sceglie **ntree=300**.



Implementazione di modelli competitivi

5. Modello RANDOM FOREST

```
set.seed(1)
control <- trainControl(method="cv", number=10, search="grid", summaryFunction=twoClassSummary, classProbs = TRUE)
tunegrid <- expand.grid(.mtry=c(1:6))
rf <- train(r~-Class,
           data=train, method="rf", metric="ROC",
           tuneGrid=tunegrid, ntree=300, trControl=control, na.action=na.pass)
```

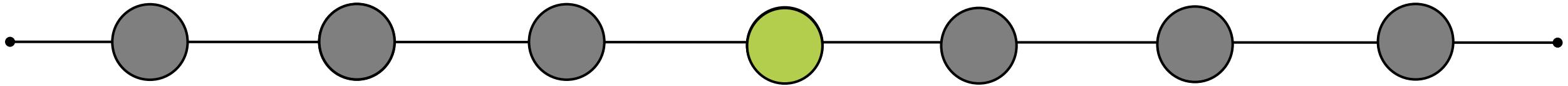
- **grid:** valori del numero di variabili per nodo *mtry* da considerare

il modello ottenuto, addestrato sul dataset di *train*, viene valutato sul dataset di *test*:

- **Accuracy**= 0.916
- **AUC**= 0.9615

La funzione valuta a quale *mtry* corrisponde il modello con valore di ROC più elevata

→ **rf\$bestTune=mtry=2**



Implementazione di modelli competitivi

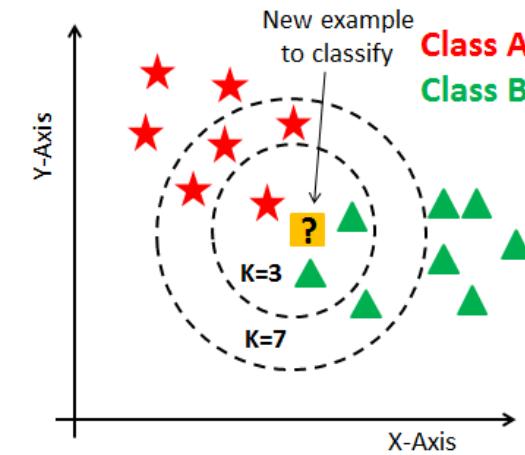
Modelli in cui è preferibile la selezione delle variabili

- Si considerano **due diverse selezioni delle variabili** per i modelli successivi.
 - La **prima**, ottenuta dal **LASSO** e la **seconda** dal **DECISION TREE**.
 - **LASSO – 10 variabili:** Shapefactor_3, Shapefactor_4, Extent, Skew_RB, Kurtosis_RG, Kurtosis_RB, StdDev_RR, StdDev_RB, Mean_RB, Major_Axis
 - **TREE- 5 variabili:** Shapefactor_3, Convex_Area, Shapefactor_4, StdDev_RR, StdDev_RB.
 - Shapefactor_3, Shapefactor_4, StdDev_RR e StdDev_RB: selezionate da **entrambi** i modelli
 - Per le variabili legate alle caratteristiche dei **colori delle immagini**: le due standard deviation (RB e RR) sono comprese nelle due selezioni, mentre **curtosi**, **skewness** e **media** di **RB** sono presenti solo nella prima selezione.
 - Per le variabili riguardanti **aspetti morfologici**: **Extent** e **Major_Axis** sono presenti solo nella prima selezione, mentre **Convex_Area** è presente solo nella seconda.

Implementazione di modelli competitivi

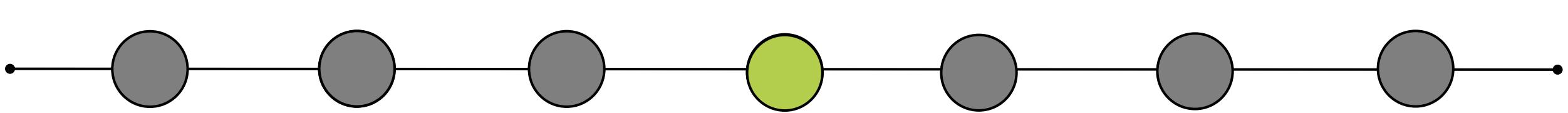
6. Modello KNN

- Algoritmo utilizzato sia nei problemi di regressione che di classificazione:
- **Classificazione**: un'osservazione è classificata in base ai suoi “vicini più vicini”. Considerate le k unità più vicine ad ogni osservazione, l'unità è assegnata alla classe più frequente fra le classi di queste k unità
- Si basa sulla **distanza** → rendere variabili a varianza unitaria
- Svantaggio: **alto costo computazionale** (calcolo delle distanze tra **tutti** i punti del dataset)
- **Scelta di k**: un valore troppo basso può portare ad un modello troppo sensibile alla presenza di outlier e in generale overfitting



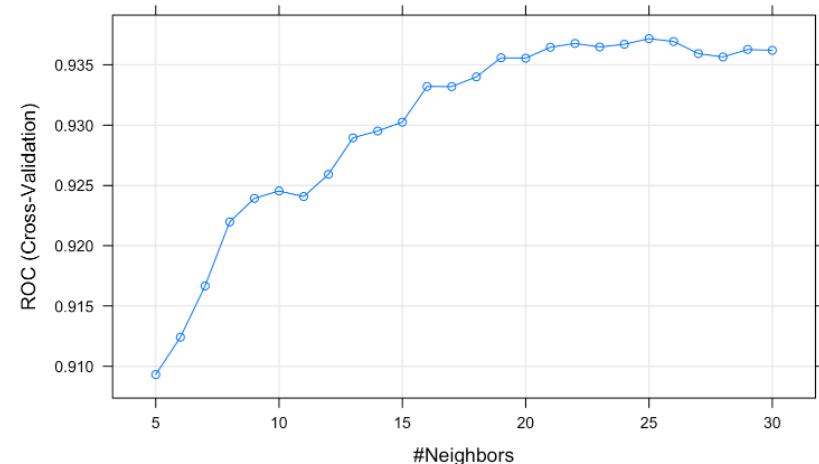
Nella funzione *train*:

- **grid <- expand.grid(k=seq(5,30))**: valori per il parametro k
- **preProcess=c("scale", "corr", "nzv")**: standardizzazione (scale), rimozione delle variabili altamente correlate (corr) e con varianza nulla o quasi (nzv).
- **method="knn"**: la distanza di default è quella Euclidea.

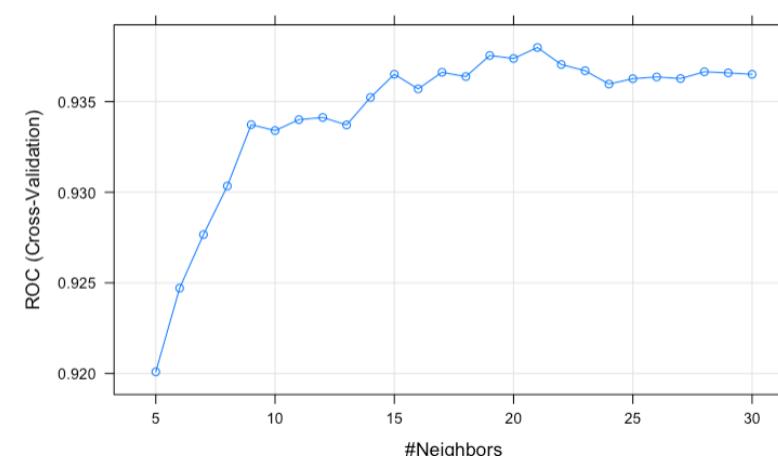


Implementazione di modelli competitivi

6.1 Modello KNN - MS LASSO

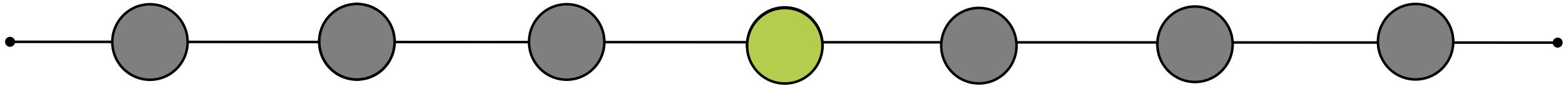


6.2 Modello KNN - MS TREE



- Al crescere di k la ROC cresce, ma da k=25 inizia a decrescere → si sceglie **k=25**
- Accuracy**=0.8616
- AUC**=0.9486

- Al crescere di k la ROC cresce, ma da k=21 inizia a decrescere → si sceglie **k=21**
- Accuracy**=0.8802
- AUC**=0.9492



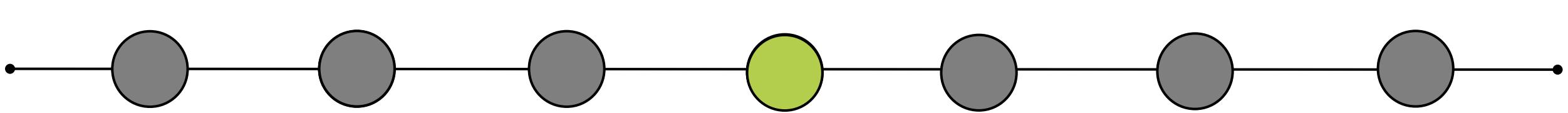
Implementazione di modelli competitivi

7. Modello PLS

- Versione supervisionata della **PCA** (Principal Component Analysis)
- Tecnica inizialmente progettata per la regressione che sfrutta un approccio di riduzione della dimensionalità individuando componenti ortogonali (combinazioni lineari delle covariate) che spieghino quanta più varianza possibile della risposta
- Può essere utilizzata anche per la classificazione
- E' sensibile alla scelta del numero di componenti da includere nel modello

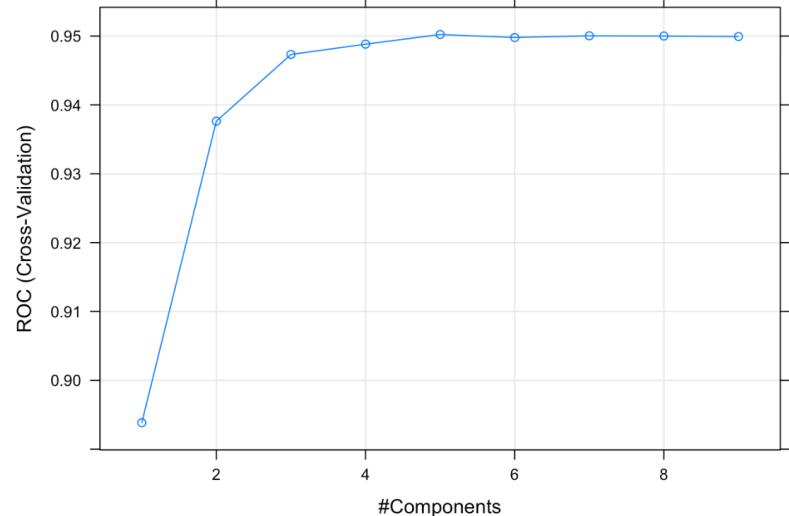
Nella funzione *train*:

- **tuneLength=5:**
valori per il numero delle componenti
- **method="pls"**



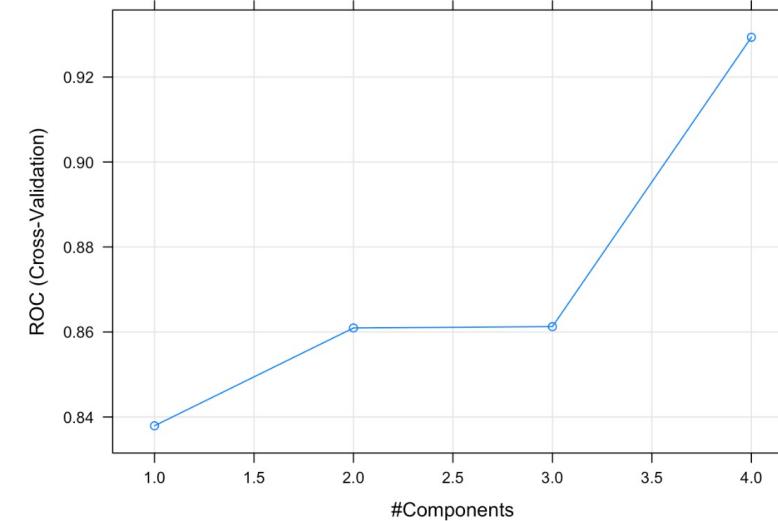
Implementazione di modelli competitivi

7.1 Modello PLS- MS LASSO

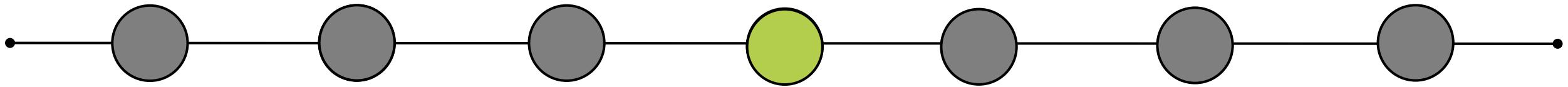


- Al crescere di **ncomp** la ROC cresce, ma da **ncomp=5** si stabilizza → si sceglie **ncomp=5**
- **Accuracy**= 0.9067
- **AUC**= 0.9589

7.2 Modello PLS- MS TREE



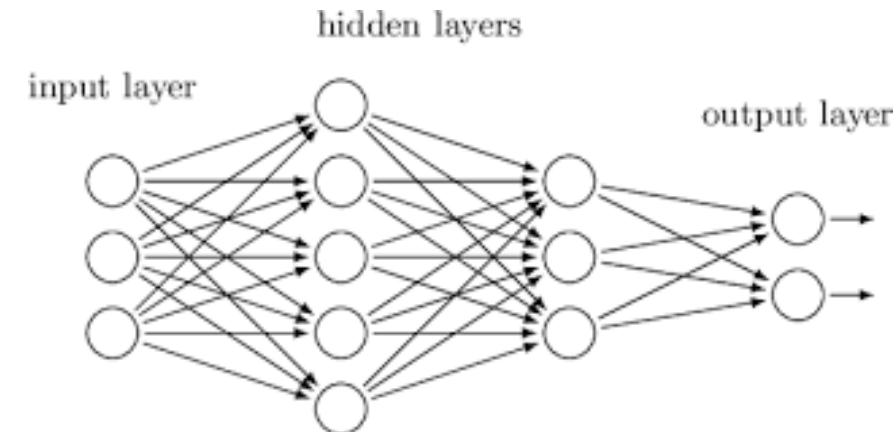
- Il valore di **ncomp** che massimizza la ROC è 4 → si sceglie **ncomp=4**
- **Accuracy**= 0.8802
- **AUC**= 0.9458



Implementazione di modelli competitivi

8. Modello RETE NEURALE

- Regressione e classificazione, ispirata dal funzionamento dei **neuroni**.
- È costituita da **nodi interconnessi** i cui output vengono pesati e combinati in una funzione lineare e diventano l'input dei nodi dello strato successivo in base ad una **funzione di attivazione**.
- È composta da almeno 3 strati: uno **strato di input**, uno **nascosto**, ed uno **di output**.
- Addestramento: trovare i **pesi ottimali** che minimizzano una **funzione di costo**.

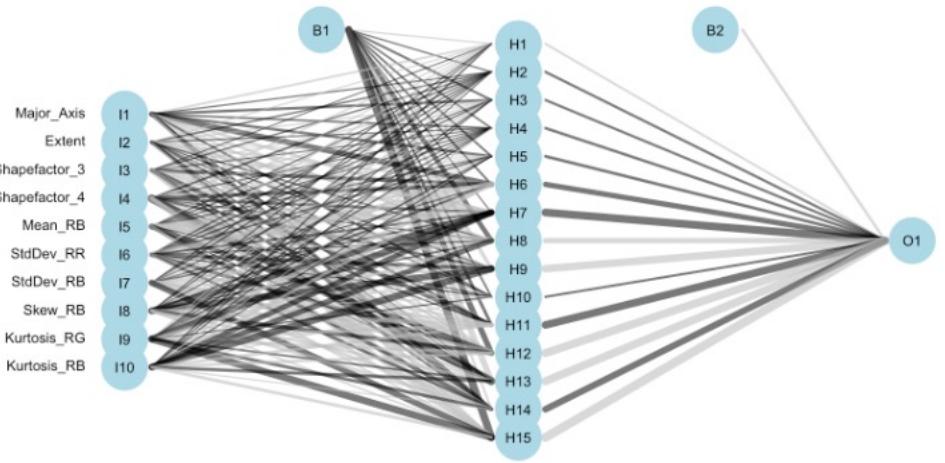


Nella funzione *train*:

- **grid <- expand.grid(.size = c(5, 10, 15), .decay = c(0.1, 0.01, 0.001)):** size (numero di neuroni nello strato nascosto) e decay (controlla la regolarizzazione applicata durante l'addestramento del modello).
- **method="nnet"**: funzione di attivazione: tangente iperbolica.

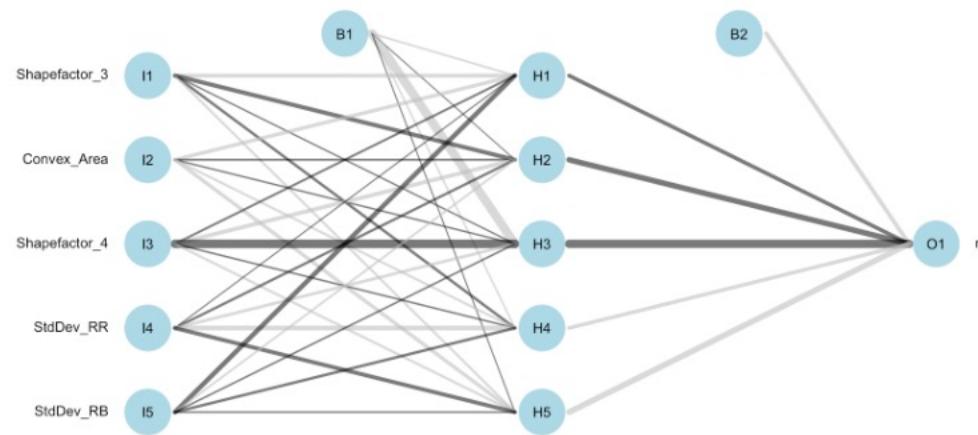


8.1 Modello Nnet - MS LASSO

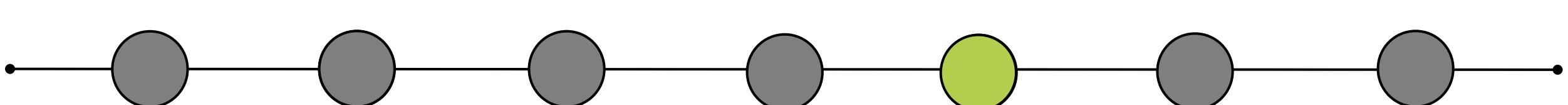


- I parametri che massimizzano la ROC sono **size=15** e **decay=0.01**
- **Accuracy**= 0.9114
- **AUC**= 0.97

8.2 Modello Nnet- MS TREE

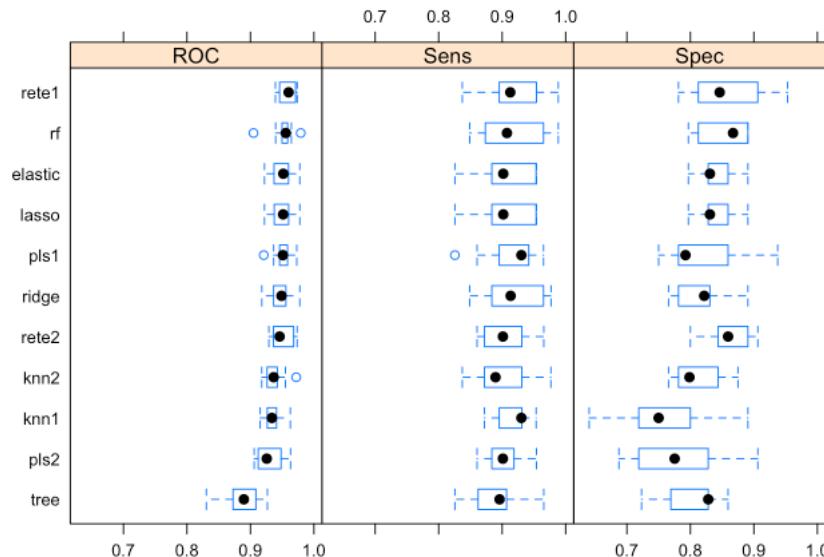


- I parametri che massimizzano la ROC sono **size=5** e **decay=0.01**
- **Accuracy**= 0.8849
- **AUC**= 0.9577



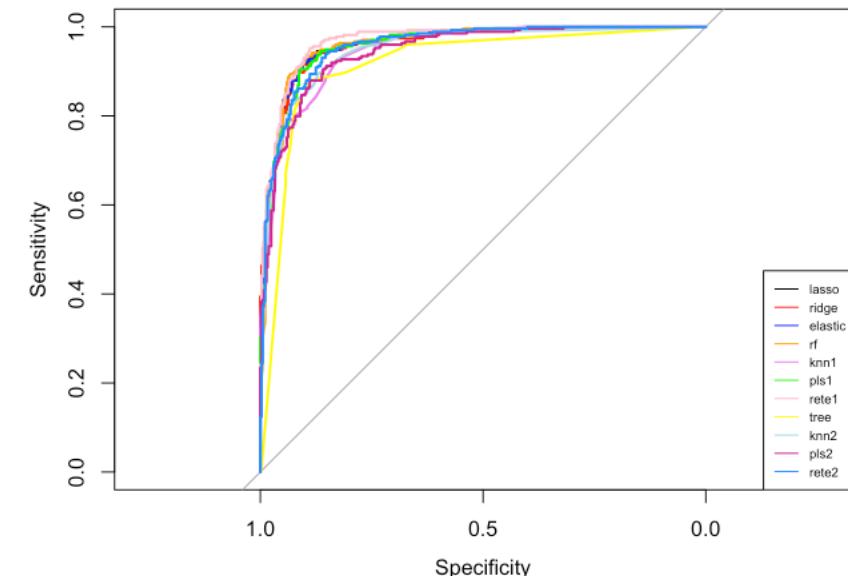
Comparazione dei risultati

```
results <- resamples(list(ridge=ridge, lasso=lasso, elastic=elastic, rf=rf, knn1=knn1, pls1=pls1, rete1=rete1, tree=tree, knn2=knn2, pls2=pls2, rete2=rete2))
#summary(results, metric="ROC")
bwplot(results)
```

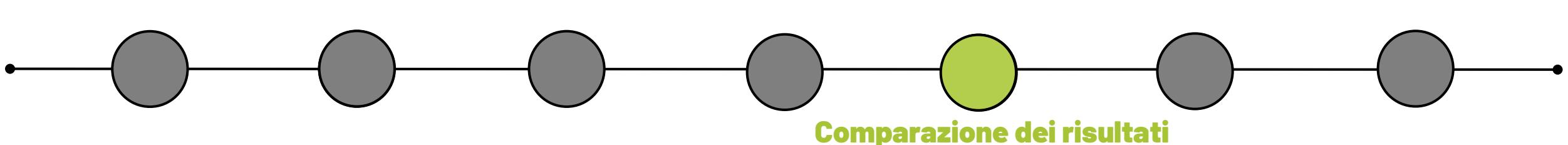


- Molti modelli **simili** fra loro che massimizzano la ROC adeguatamente
- Modello migliore: **rete1**, seguita da **random forest, lasso**
- Non performano bene: **pls2** e il **tree**.

Rappresentazione grafica delle curve ROC



- La maggior parte delle curve ROC si sovrappone



ROC e ACCURACY

	Modello <chr>	ROC <dbl>	Accuracy <dbl>
4	RF	0.9615	0.9160
7	RETE1	0.9700	0.9114
1	LASSO	0.9622	0.9082
3	ELASTIC	0.9621	0.9082
6	PLS1	0.9589	0.9067
2	RIDGE	0.9613	0.9051
11	RETE2	0.9577	0.8849
9	KNN2	0.9492	0.8802
10	PLS2	0.9458	0.8802
8	TREE	0.9179	0.8787
5	KNN1	0.9486	0.8616

11 rows

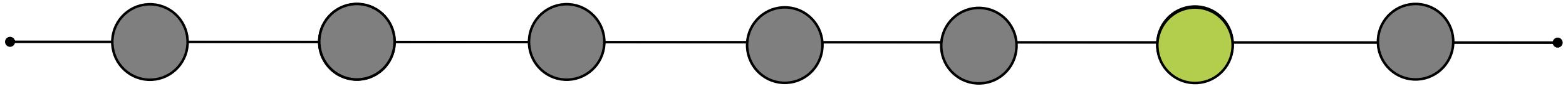
- **Rete1, Random Forest e Lasso:** modelli migliori in termini di **ROC, ACCURACY**

Confronto modelli con MS

	Modello <chr>	ROC <dbl>	Accuracy <dbl>
5	KNN1	0.9486	0.8616
9	KNN2	0.9492	0.8802
6	PLS1	0.9589	0.9067
10	PLS2	0.9458	0.8802
7	RETE1	0.9700	0.9114
11	RETE2	0.9577	0.8849

6 rows

- Prestazioni modelli con MS –Lasso migliori di quelle dei modelli con MS - Decision Tree, ma sono comunque simili



Previsioni

■ Previsioni sul dataset di test nei 3 modelli migliori

```
set.seed(1)
previsione_retel <- predict(retel, test, "prob") #probabilità
previsione_retel$R <- predict(retel, test) #r0 o r1
head(previsione_retel)
```

	r0 <dbl>	r1 R <dbl> <fct>
1	0.99976062	0.0002393767 r0
3	0.99887411	0.0011258870 r0
4	0.05675251	0.9432474888 r1
7	0.34819782	0.6518021789 r1
12	0.98588305	0.0141169515 r0
14	0.99925016	0.0007498407 r0

6 rows

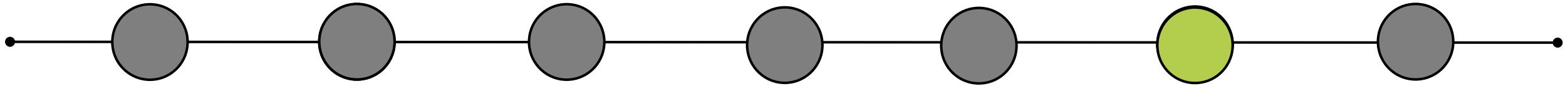
■ Tabella: numero di pistacchi classificati correttamente

```
confronto <- rbind(table(previsione_retel$prev_retel01),table(previsione_rf$prev_rf01),table(previsione_lasso$prev_lasso01))
rownames(confronto) <- c("Rete - Lasso", "Random Forest", "Lasso"); confronto
```

```
##          0   1
## Rete - Lasso 57 586
## Random Forest 54 589
## Lasso        59 584
```

Risultati simili:

- **Rete Neurale:** classifica correttamente 586 pistacchi (91%)
- **Random Forest:** 589 (**92%**)
- **Lasso:** 584 (91%)



Pattern di risposta

- per osservare la distribuzione delle previsioni in base ai 3 modelli migliori:

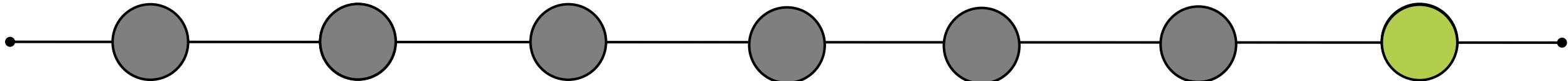
Rete1-Rf-Lasso <fct>	Frequenza assoluta <int>
8 1-1-1	552
1 0-0-0	31
4 0-1-1	17
6 1-0-1	14
7 1-1-0	12
5 1-0-0	8
3 0-1-0	8
2 0-0-1	1

1-8 of 8 rows

- la **Rete** sbaglia in 26 casi (17+8+1);
 - la **Random Forest** sbaglia in 23 casi (14+8+1);
 - il **Lasso** sbaglia in 28 casi (12+8+8);
- RF**: modello che prevede i dati nel modo migliore

- **8** pattern di risposta:

- **1-1-1**: in 552 casi **tutti** i 3 modelli classificano **correttamente** le osservazioni
- **0-0-0**: in 31 casi **tutti** i 3 modelli **sbagliono** a classificare le osservazioni
- **0-1-1**: in 17 casi la **Rete sbaglia** nel classificare le osservazioni
- **1-0-1**: in 14 casi la **Random Forest sbaglia**
- **1-1-0**: in 12 casi il **Lasso sbaglia**
- **1-0-0**: in 8 casi **solo** la **Rete** classifica correttamente
- **0-1-0**: in 8 casi **solo** la **RF** classifica correttamente
- **0-0-1**: in 1 caso **solo** il **Lasso** classifica correttamente



Conclusioni

- Tre modelli **migliori**: **Random Forest, Rete Neurale – MS Lasso, Lasso** → prestazioni **simili**, ma **RF e Rete Neurale** modelli **computazionalmente** più **onerosi**;
- Quale preferire?
 - In presenza di **scarse risorse** a disposizione e di **tecniche poco all'avanguardia** in fase di classificazione, potrebbe essere **preferibile** utilizzare un **modello** con buone performance, ma **meno complesso**.
 - In un contesto in cui il **numero di pistacchi** da analizzare è molto più **elevato** di quello considerato, potrebbe essere **preferibile** utilizzare **modelli più complessi**
- Utilizzando un **modello** con **buone prestazioni** si può **raggiungere l'obiettivo** di **semplificare** il processo di **classificazione, diminuendo le risorse economiche** e il **tempo** impiegato e quindi il **prezzo** del prodotto finale.

