# XGBoost

## Report di Statistica

Borelli Roberto  Santi Enrico

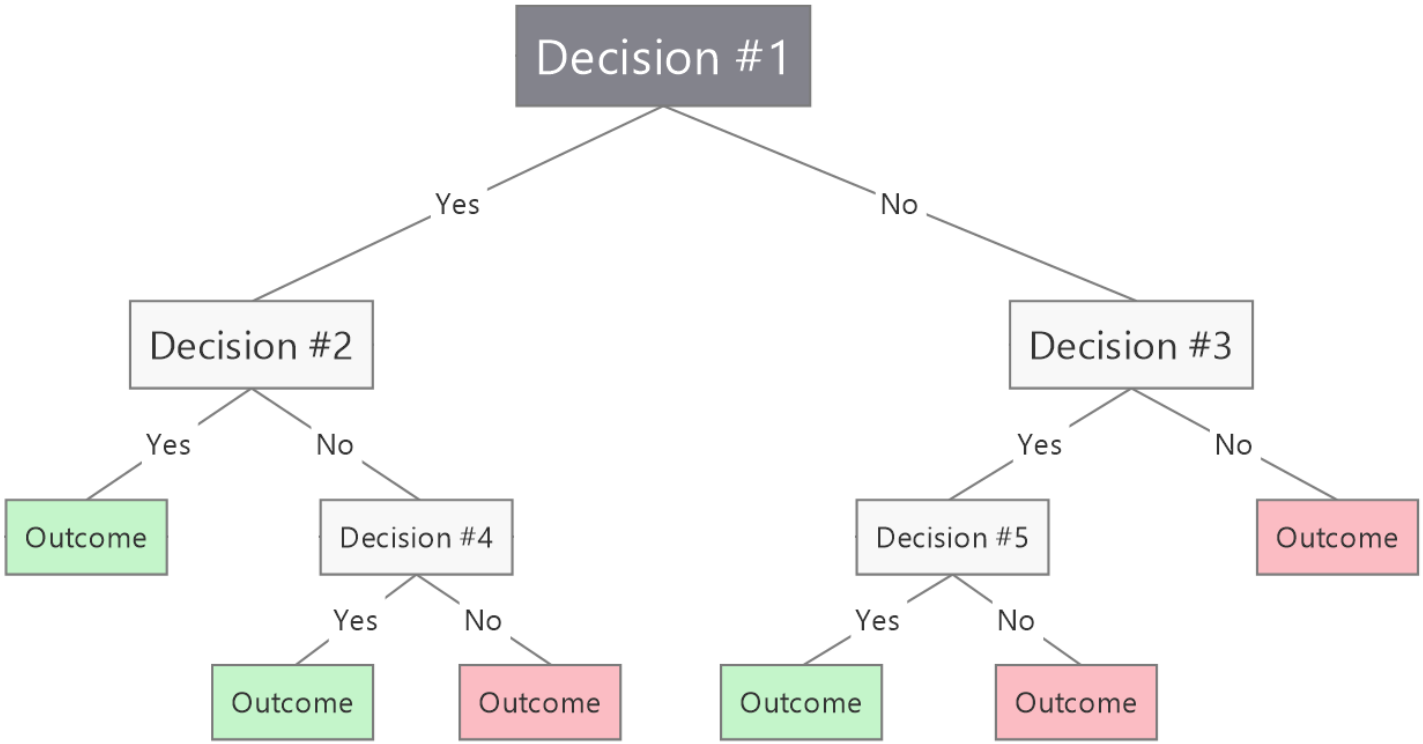
Università degli Studi di Udine

27 gennaio 2023

# Outline

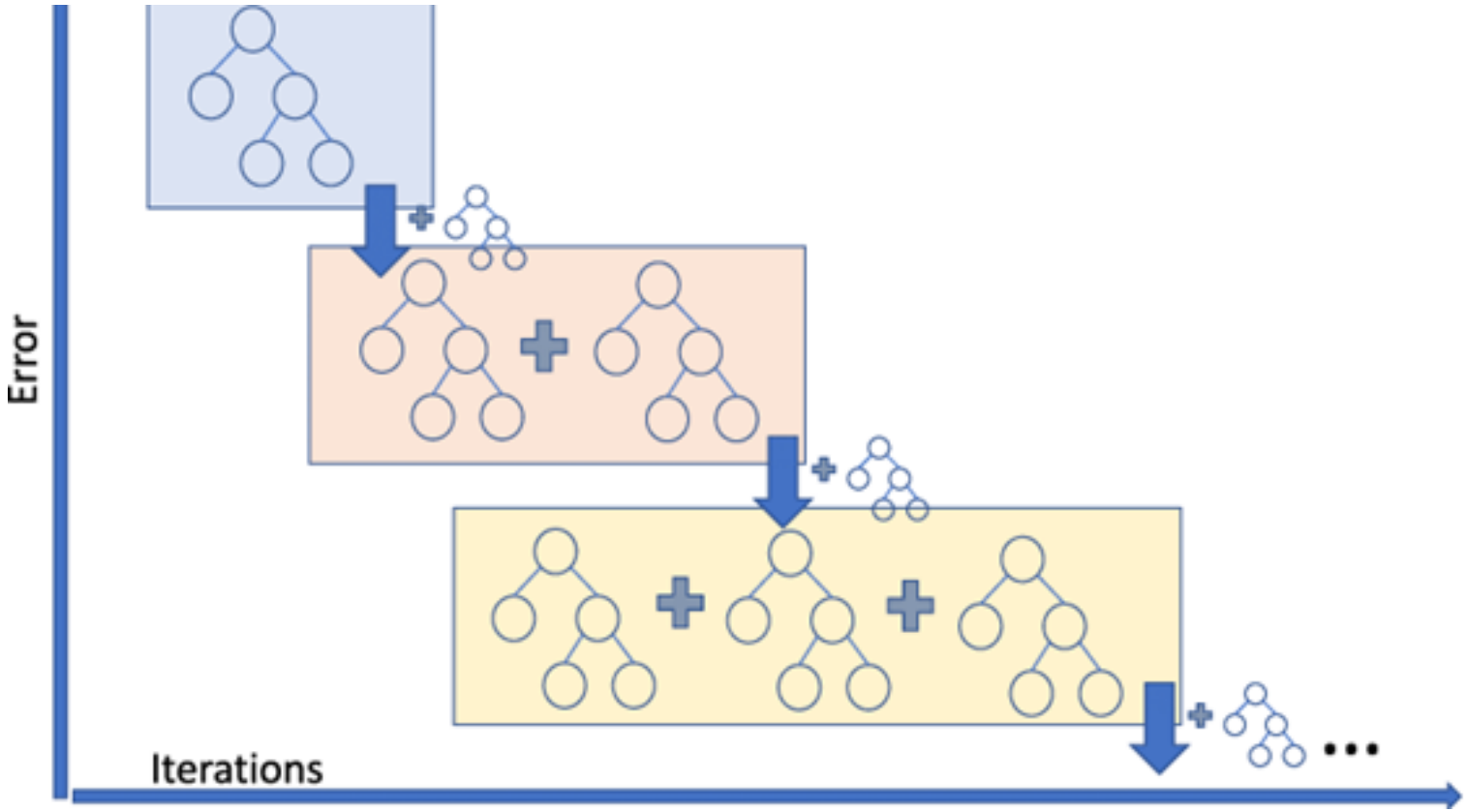
# Presentazione del metodo

# Alberi di decisione



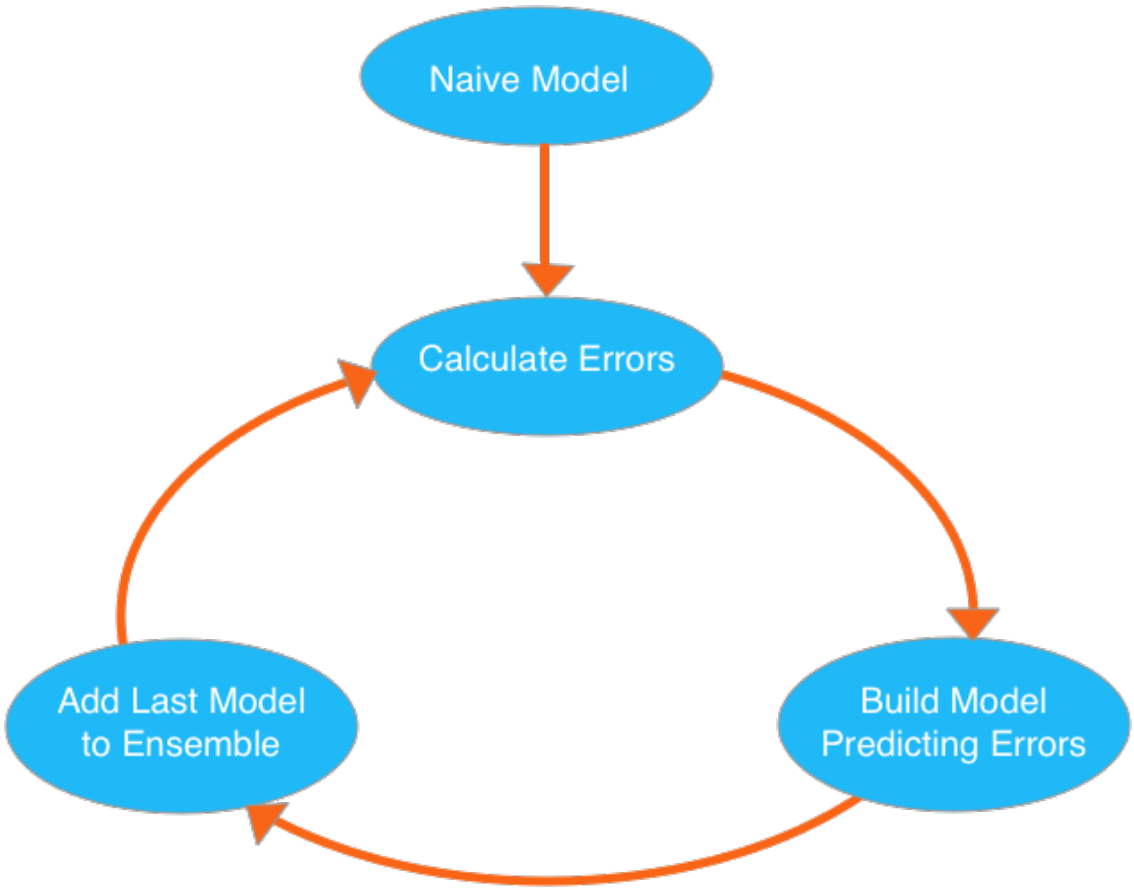

- Nodo interno: identifica una decisione su una variabile esplicativa.
- Foglia: definisce una partizione dello spazio dei regressori.

# Assemblamento




Vengono utilizzati più modelli per creare un predittore forte.

# Boosting (Tree)




**Processo iterativo:**

- Calcolo errori
- Predittore debole: stump
- Rafforzo modello

Figura: L'idea alla base della procedura di boosting

# Gradient Boosting Tree

- Il modello è migliorato: aggiungendo lo stump allenato sui residui precedenti.
- Lo stump è ottenuto usando il gradiente della funzione di costo:

$$Stump_i \leftarrow -\frac{\partial Cost(\hat{f_{i-1}}, y)}{\partial \hat{f_{i-1}}} \tag{1}$$

$$\hat{f_i} \leftarrow \hat{f_{i-1}} + Stump_{i-1} \tag{2}$$

# Gradient Boosting Tree

- Il modello è migliorato: aggiungendo lo stump allenato sui residui precedenti.
- Lo stump è ottenuto usando il gradiente della funzione di costo:

$$Stump_i \leftarrow -\frac{\partial Cost(\hat{f_{i-1}}, y)}{\partial \hat{f_{i-1}}} \tag{1}$$

$$\hat{f_i} \leftarrow \hat{f_{i-1}} + Stump_{i-1} \tag{2}$$

- Per evitare overfitting introduciamo un peso (shrinkage) $\eta$:

$$\hat{f_i} \leftarrow \hat{f_{i-1}} + \eta Stump_{i-1} \tag{3}$$

- La procedura di boosting termina sempre (ciclo enumerativo).

# Presentazione della libreria

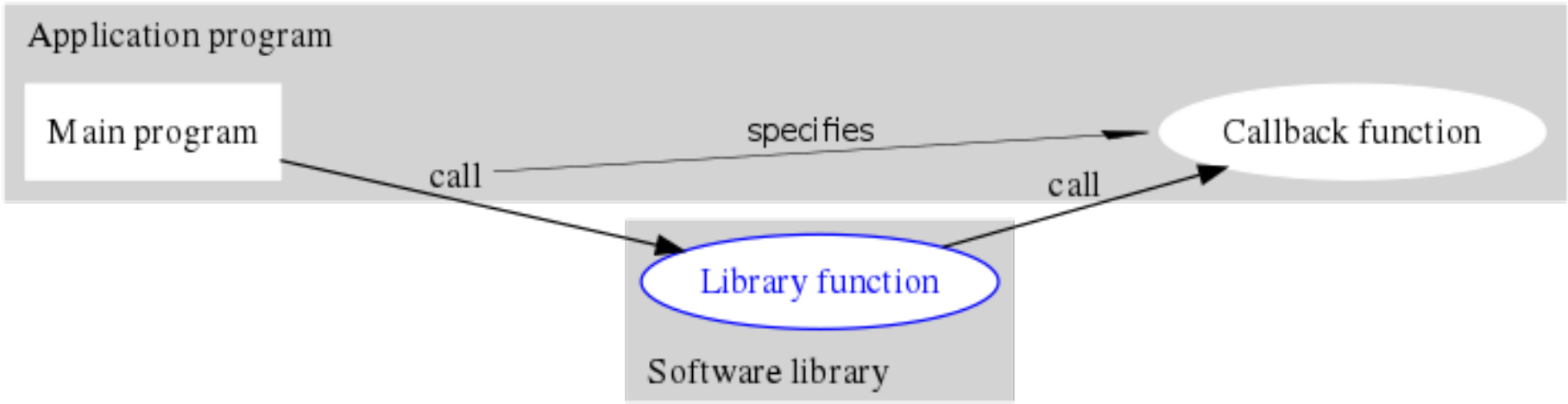
# XGBoost

Libreria (multi piattaforma) che implementa un framework contenente diversi metodi legati alla definizione di modelli (di regressione e classificazione) basati sul gradient boosting.

**Parallelismo:**

- L'ottimizzazione è una feature centrale della libreria.
- Molte funzioni hanno il parametro `nthread`, (vedremo successivamente alcuni test effettuati).
- Pensata per l'analisi di dataset voluminosi.

# Il sistema di Callbacks di XGBoost




Meccanismo che consente di specificare una funzione che verrà chiamata dalla funzione invocata ad un certo punto della sua esecuzione.

Utilizzate ad esempio per capire come il modello stia migliorando ad ogni iterazione di boost.

# Descrizione delle funzioni della libreria I

**Manipolazione dei tipi di dati:**

Possiamo convertire una matrice al tipo di dato `DMatrix` e abbiamo delle funzioni specifiche per il salvataggio/caricamento di modelli.

- `xgb.DMatrix`
- `xgb.save`
- `xgb.load`

# Descrizione delle funzioni della libreria II

## Funzioni principali

- `xgb.train`
  - `data`
  - `eta`
  - `nrounds`
  - `max_depth`
  - `watchlist`

# Descrizione delle funzioni della libreria II

## Funzioni principali

- `xgb.train`
  - `data`
  - `eta`
  - `nrounds`
  - `max_depth`
  - `watchlist`
- `predict.xgb.Booster`

```
predict(model, data_to_predict, iterationrange=c(1,5))
```

# Descrizione delle funzioni della libreria II

## Funzioni principali

- xgb.train
  - data
  - eta
  - nrounds
  - max_depth
  - watchlist
- predict.xgb.Booster

```
predict(model, data_to_predict, iterationrange=c(1,5))
```

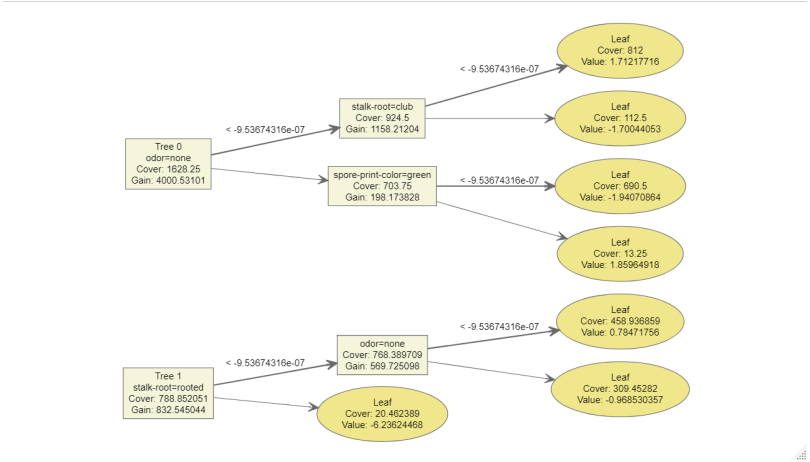
- xgb.cv

```
cv<-xgb.cv(data = dtrain, nrounds = 4, nfold = 5,
           metrics = list("mae","auc"), max_depth = 3,
           eta = 1, objective = "binary:logistic")
print(cv$evaluation_log)
```

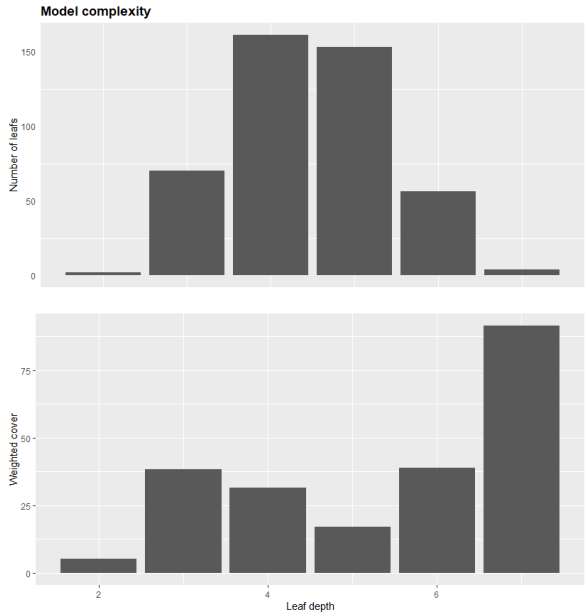
# Descrizione delle funzioni della libreria II

## Funzioni di output

- `xgb.plot.tree`




- `xgb.plot.deepness`



# ❸ Caso di studio

# Il dataset Glass Identification I

214 sample, ognuno con 10 caratteristiche:

1. RI: indice di rifrazione
2. Na: Sodio
3. Mg: Magnesio
4. Al: Alluminio
5. Si: Silicio
6. K: Potassio
7. Ca: Calcio
8. Ba: Bario
9. Fe: Ferro
10. Type: Tipo di vetro

# Il dataset Glass Identification II

7 tipi di vetro:

1. building-window-float
2. building-window-non-float
3. vehicle-window-float
4. vehicle-window-non-float
5. Container
6. Tableware
7. Headlamp

Nessun esempio di categoria 4 nel dataset.

# Il dataset Glass Identification II

7 tipi di vetro:

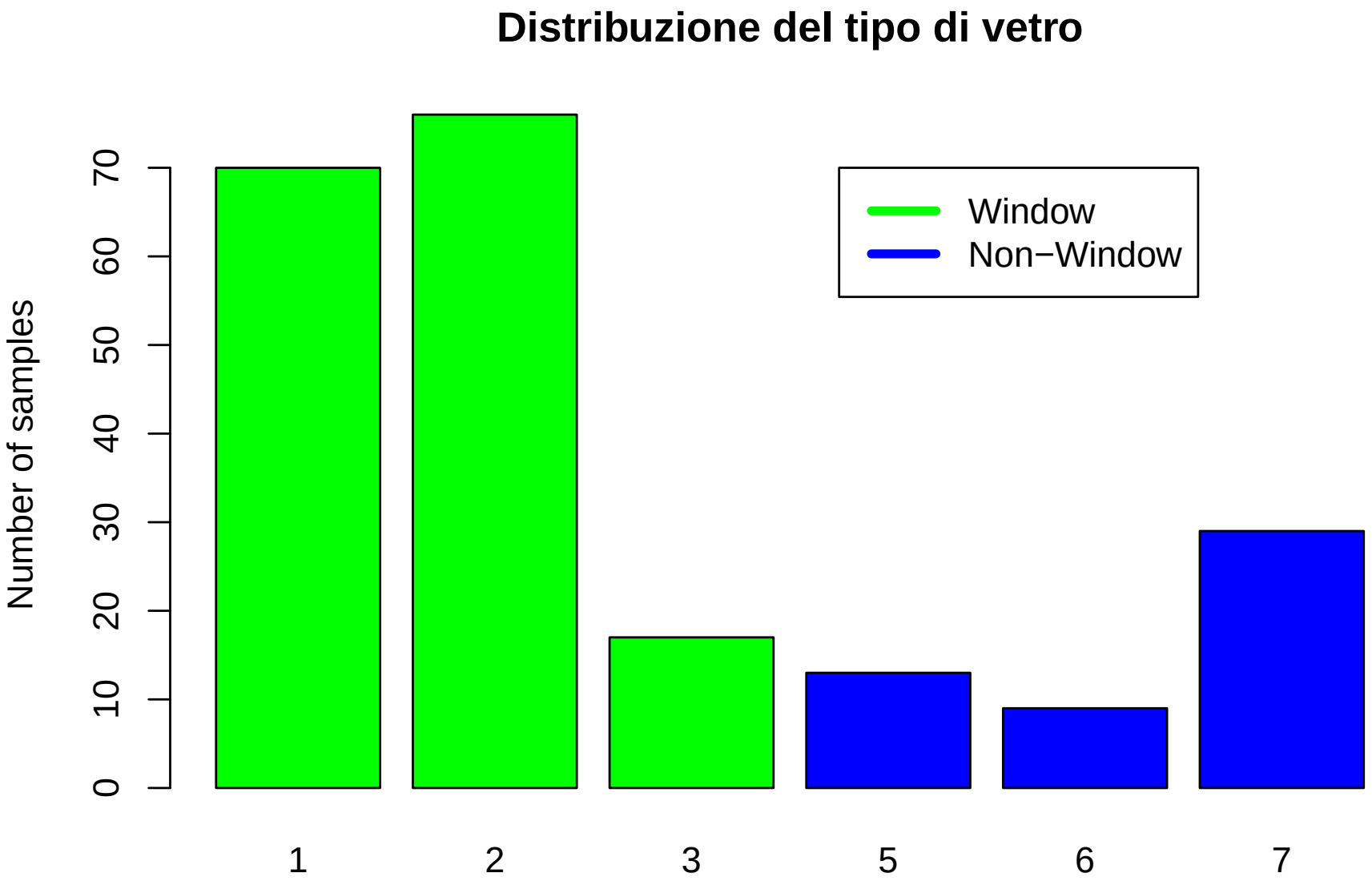
1. building-window-float
2. building-window-non-float
3. vehicle-window-float
4. vehicle-window-non-float
5. Container
6. Tableware
7. Headlamp

Nessun esempio di categoria 4 nel dataset.

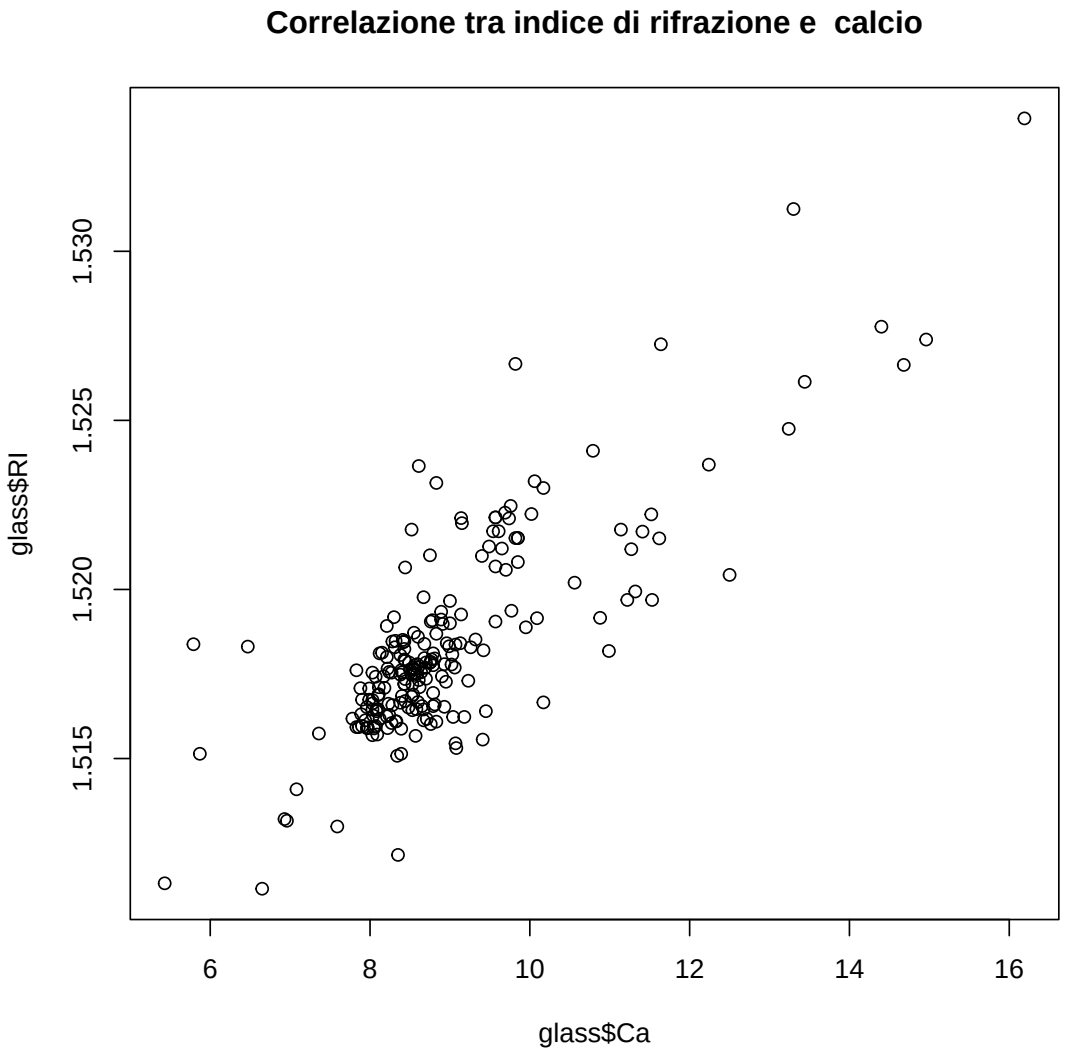
**Obiettivi:**

1. Classificazione binaria Window/Non-Window
2. Classificazione multiclasse

# Analisi esplorativa I



# Analisi esplorativa II




```
cor.test(glass$RI,glass$Ca)
## 0.8104027
```

# Applicazione della libreria I

Per la creazione dei modelli il dataset è stato suddiviso in test e training set. Quest'ultimo consiste di 20 sample estratti casualmente.

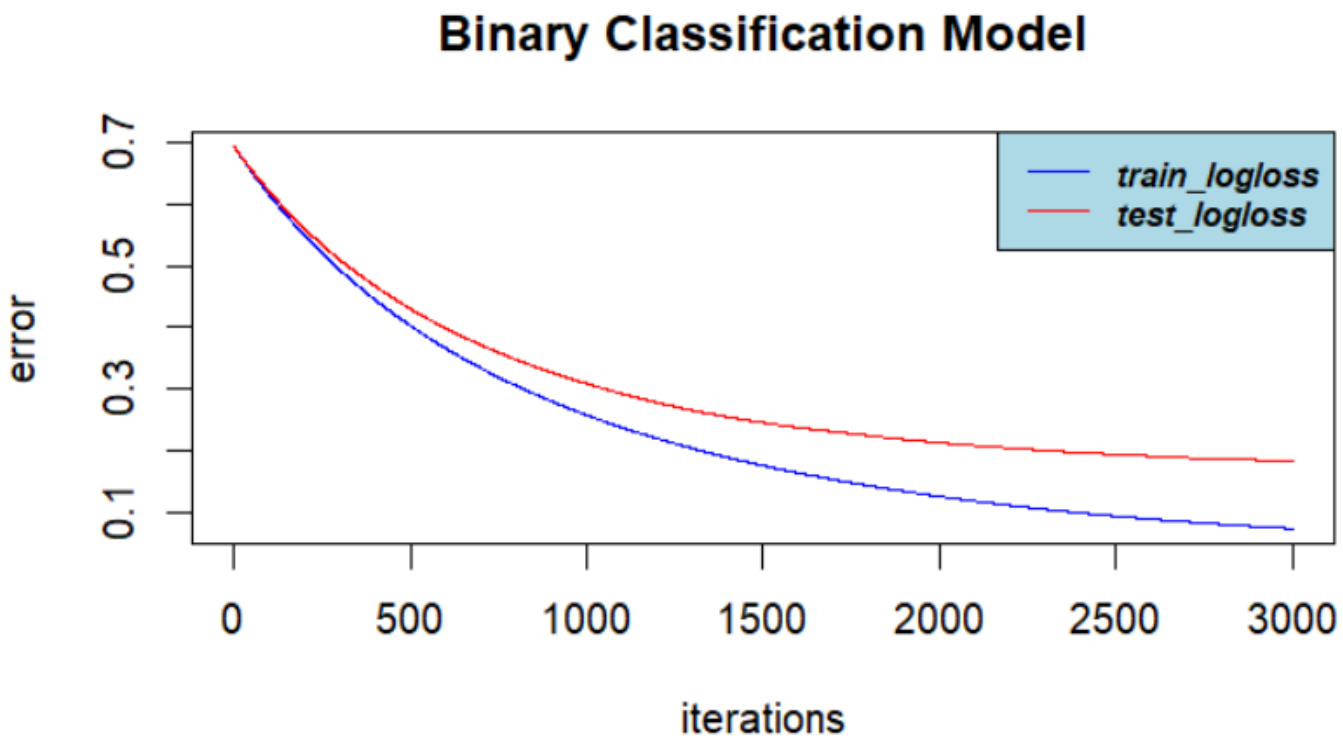
**Obiettivi:**

- Creare due modelli `M` e `M.bin`
  - **M**: per la classificazione multiclasse rispetto a `Type`
  - **M.bin** per la classificazione binaria `isWindow`
- Valutare un numero di iterazioni appropriato per i modelli.
- Valuatare le performance dei modelli, in particolare anche confrontando le performance di `M` utilizzato per predirre i valori di `isWindow` su nuovi sample rispetto ad `M.bin`.

# Applicazione della libreria II

Creiamo i modelli e tramite CV valutiamo le performance:

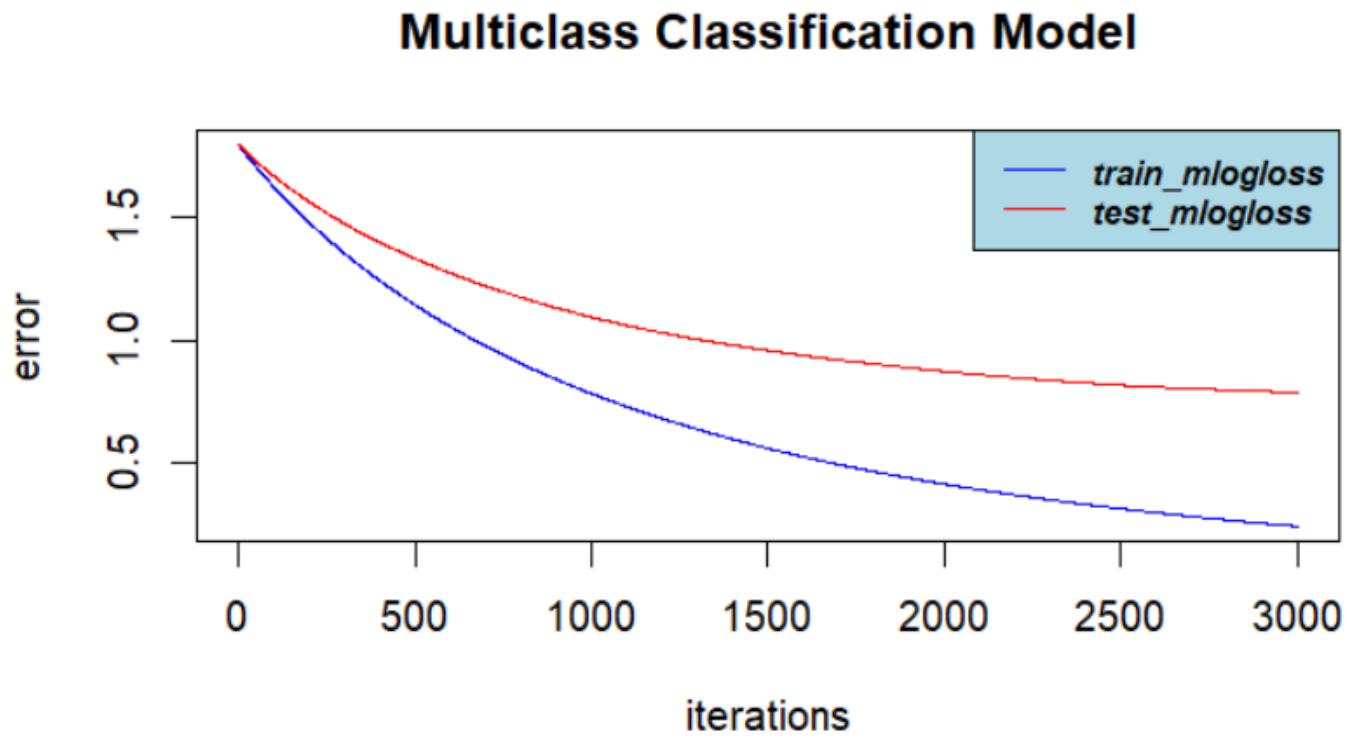
```
M.bin = xgb.train(nrounds=3000, params = par.bin, watchlist=
wl.bin. data = dtrain.bin)
M = xgb.train(nrounds=3000, params = par, watchlist=wl, data
= dtrain)
```



# Applicazione della libreria III

Gli errori di previsione sembrano stabilizzarsi rispettivamente:

- Dopo 1500 iterazioni nel caso multiclasse
- Dopo 1700 iterazioni nel caso binario




Empiricamente sono stati anche confrontati tali modelli con quelli ottenuti con più iterazioni (2000 e 2200).

# Risultati ottenuti I

Le matrici di confusione riferite al classificatore binario e multiclasse:

```
          Reference
Prediction  0  1
         0  4  1
         1  2 13

               Accuracy : 0.85
```

```
          Reference
Prediction 1 2 3 5 6 7
         1 6 1 0 0 0 0
         2 0 2 0 1 0 0
         3 1 0 2 1 0 0
         5 0 0 0 1 0 0
         6 0 0 0 0 2 0
         7 0 0 0 0 0 0

Overall Statistics

               Accuracy : 0.7647
```

Per quanto riguarda la sensitivity e specifitivity si hanno:

- 0.666 ed 0.928 per il modello binario.
- per il modello multiclasse la sensitivity varia molto a seconda delle classi (da 0.333 a 1.0).

# Risultati ottenuti II

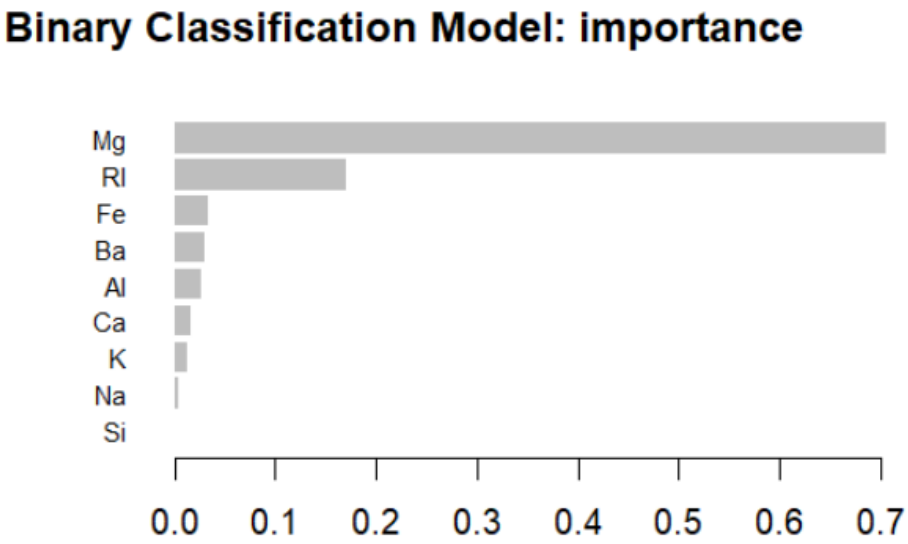
Provando a verificare il comportamento del modello **M** (multiclasse) nel caso di predizione della variabile:

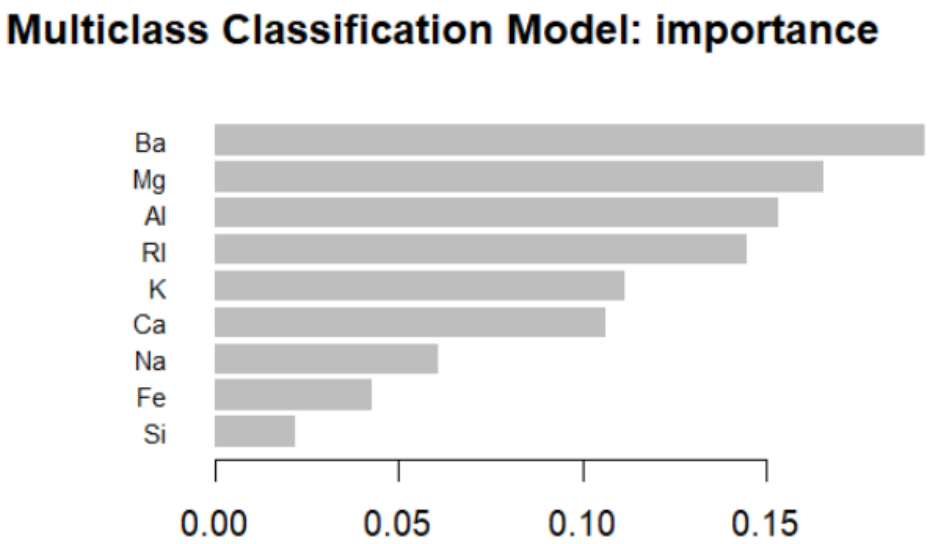
- Accuracy: 0.8.
- Anche la sensitivity e specifitivity sono minori rispetto a quelle del modello **M.bin**.

# Risultati ottenuti II

Provando a verificare il comportamento del modello **M** (multiclasse) nel caso di predizione della variabile:

- Accuracy: 0.8.
- Anche la sensitivity e specifitivity sono minori rispetto a quelle del modello **M.bin**.



# Approfondimento su `xgb.create.features` I

- Abbiamo applicato la funzione per creare un nuovo dataset aumentato

```
glass.aug = xgb.create.features(model=M, glass)
```

- Abbiamo diviso il dataset in test set e training set secondo la stessa modalità presentata in precedenza

# Approfondimento su `xgb.create.features` I

- Abbiamo applicato la funzione per creare un nuovo dataset aumentato

```
glass.aug = xgb.create.features(model=M, glass)
```

- Abbiamo diviso il dataset in test set e training set secondo la stessa modalità presentata in precedenza
- Abbiamo allenato il modello sul nuovo training set

```
M.aug = xgb.train(nrounds=1500, ...,data = dtrain.aug)
```

# Approfondimento su `xgb.create.features` II

- Abbiamo eseguito gli stessi test delle performance sul test set (anch'esso aumentato al nuovo numero di features)

# Approfondimento su `xgb.create.features` II

- Abbiamo eseguito gli stessi test delle performance sul test set (anch'esso aumentato al nuovo numero di features)

**Risultati:**

- Stesse performance in classificazione binaria
- Performance leggermente peggiori nella classificazione multiclasse
- Dataset di circa 70k features

# Approfondimento su `xgb.create.features` II

- Abbiamo eseguito gli stessi test delle performance sul test set (anch'esso aumentato al nuovo numero di features)

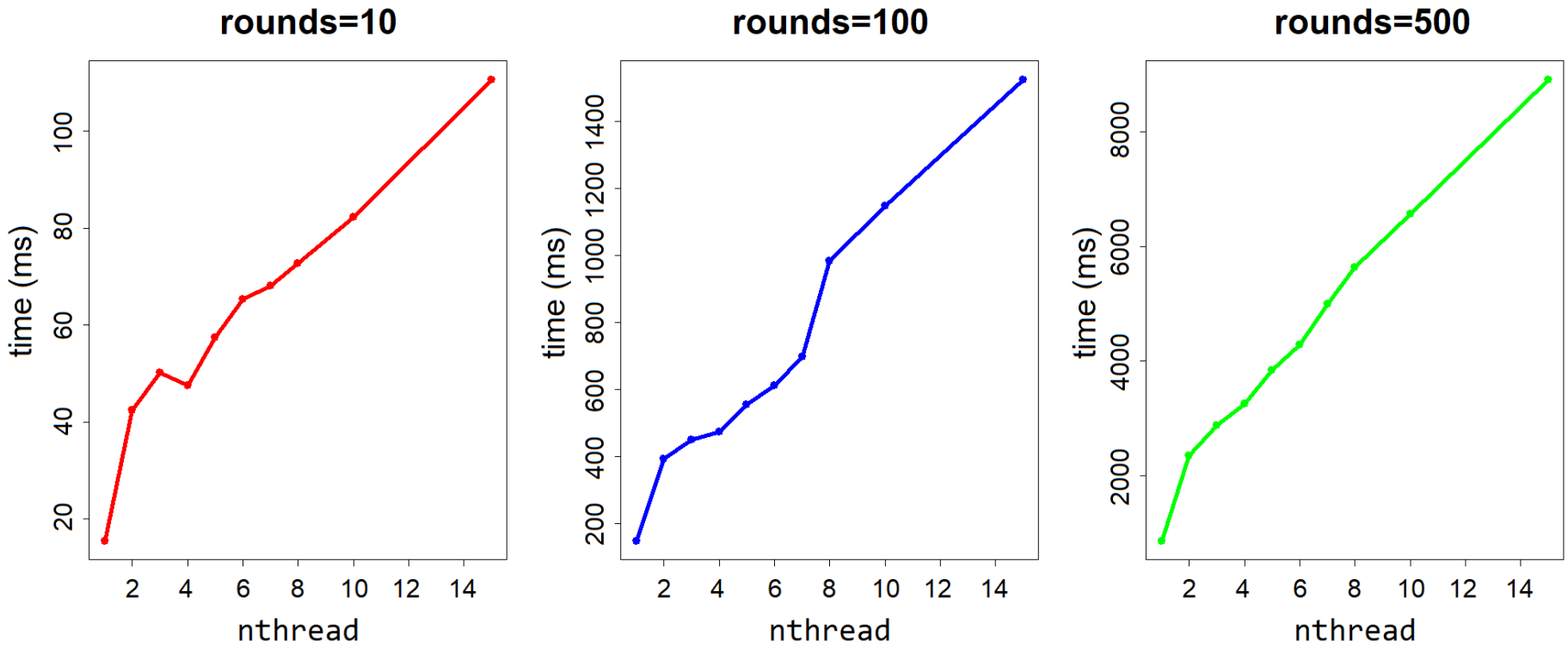
**Risultati:**

- Stesse performance in classificazione binaria
- Performance leggermente peggiori nella classificazione multiclasse
- Dataset di circa 70k features

**Test implementabili:**

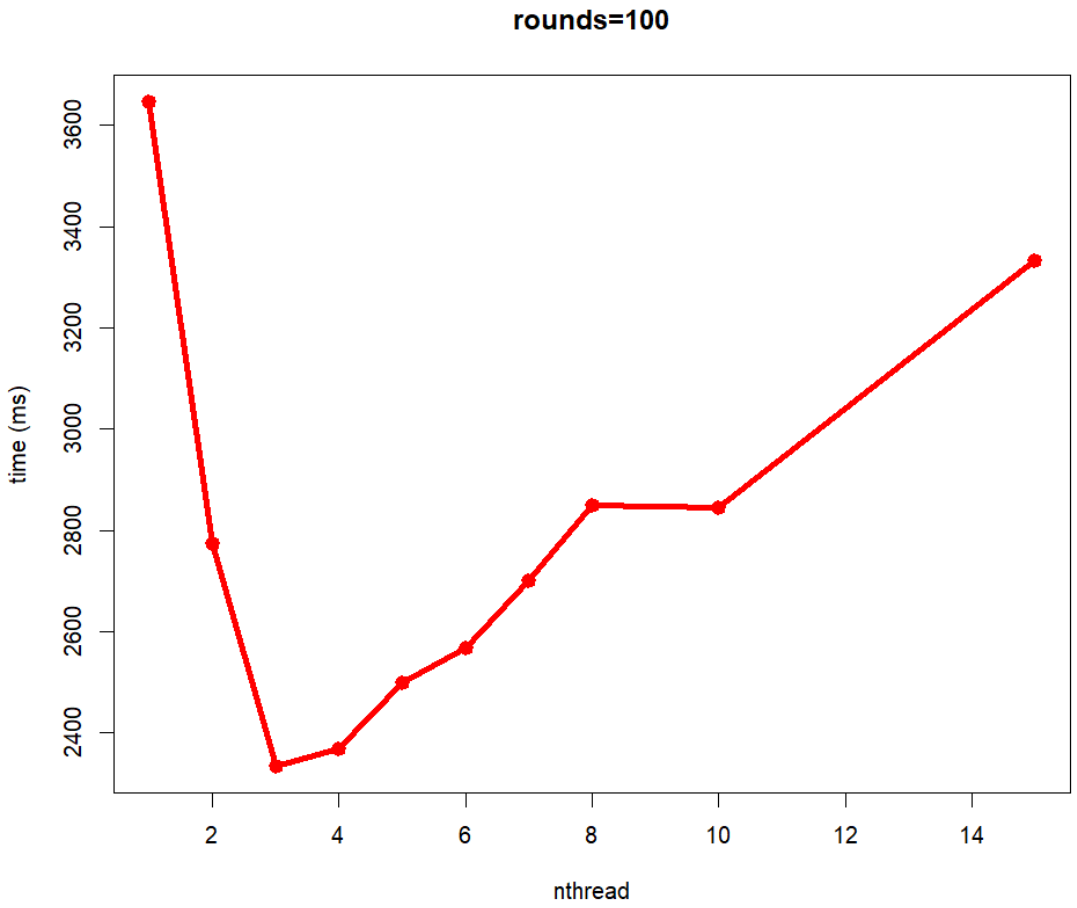
- provare a creare il dataset aumentato a partire da un modello piccolo

# Approfondimento sul multithreading I




Con il dataset originale, anche all'aumento del numero di iterazioni nel definire il modello (10, 100 e 500), il numero di thread peggiora le tempistiche di allenamento.

# Approfondimento sul multithreading II




Con un dataset più voluminoso (5000 sample) il numero di thread utilizzati influisce sulla velocità di training del modello.

# 4 Conclusioni

# Sviluppi futuri

In relazione alla libreria:

- Testare performance multithread in relazione al numero di features
- Testare performance della funzione `xgb.create.features` in più situazioni

# Sviluppi futuri

In relazione alla libreria:

- Testare performance multithread in relazione al numero di features
- Testare performance della funzione `xgb.create.features` in più situazioni

In relazione al dataset:

- Confronto sul glass dataset dei gradient boosting tree con gli altri modelli (usati sul glass dataset) visti in letteratura
- Utilizzare la PCA sul dataset

Grazie per l'attenzione!