Data Mining and Machine Learning Lab. Lezione 7 Master in Data Science for Economics, Business and Finance 2018

11.05.18

Marco Frasca

Università degli Studi di Milano

Svantaggi e vantaggi degli alberi di decisione

- Classificatori basati su alberi di decisione hanno il vantaggio di essere semplici, di facile interpretazione ed efficienti a livello computazionale
- Tuttavia, non sono in genere competitivi in termini di qualità delle predizioni con i migliori metodi di apprendimento supervisionati
- Inoltre, se cambiamo di poco i dati di training, l'albero appreso potrebbe cambiare di molto. Cioè manca stabilità. I risultati hanno un'elevata varianza, in genere
- Per ovviare a questi svantaggi sono state proposte diverse strategie, tra cui bagging, random forests, e boosting. Questi metodi costruiscono molteplici alberi che poi vengono combinati in una singola predizione 'consenso'.
- Questo può portare notevoli miglioramenti in termini di accuratezza delle predizioni, a svantaggio di una perdita parziale di interpretabilità del modello

Bagging

- Il *bootstrap aggregation* (*bagging*) è un tecnica generale utilizzata prevalentemente per ridurre la varianza di un metodo di apprendimento statistico o per stimare l'accuratezza di uno stimatore
- Particolarmente utile nel caso degli alberi di decisione
- Idealmente, se si avessero a disposizioni più insiemi di osservazioni, si potrebbe apprendere un albero di decisione per ogni insieme, quindi combinare i risultati (media o voto di maggioranza)
- Tuttavia di solito si dispone di un unico insieme di osservazioni
- L'idea del bootstrap è di estrarre casualmente B diversi sottoinsiemi di osservazioni dall'unico training set, per simulare la variabilità dei dati
- Apprendere una classificatore (o regressore) f_i per ciascun sottoinsieme i seperatamente
- Aggregare (ensemble) le predizioni sulle nuove istanze $\boldsymbol{x}$ di test:

$$f_{bag}(\boldsymbol{x}) = \frac{1}{B}\sum_{i=1}^{B} f_i(\boldsymbol{x})$$

Random Forest – Leo Breiman (2001) “Random Forests”, Machine Learning, 45, 5-32

- È un ensemble costituito da un bagging di alberi di decisione non potati (non-pruned) e complessi, con una scelta casuale di un sottoinsieme delle feature (predittori) ad ogni split
- Metodo non lineare (come gli alberi di decisione) e robusto, bassa varianza e stabilità delle predizioni rispetto al variare dei dati di input
- Migliora le prestazioni di un singolo albero di decisione appreso su tutti i dati
- Perde la facile intepretabilità degli alberi e scala meno bene

Random Forest

- **Addestramento (training) della foresta**. Fissato n_{tree} e il numero di predittori (feature, variabili) m_{try} da scegliere casualmente tra le disponibili, per n_{tree} volte ripete la procedura che segue :
  1. Scegli un sottoinsieme di bootstrap S dai dati di training
  2. Apprendi un albero di decisione su S accurato (`minsplit`$= 1$) in cui ad ogni split solo m_{try} predittori sono scelti (casualmente) come candidati per lo split tra tutti quelli disponibili
  3. Non eseguire pruning dell'albero
  4. Salva l'albero ottenuto
- **Predizione nuove istanze**. Aggrega le predizioni sui nuovi dati come fatto nel bagging, mediante voto di maggioranza delle prezioni fatte dagli n_{tree} alberi (classificazione) o mediante media delle stesse (regressione)
- Di solito m_{try} è posto a $m/3$ per problemi di regressione e $\sqrt{m}$ per problemi di classificazione, dove m è il numero totale di predittori
- Le RF forniscono anche una stima dell'errore di test chiamato *Out Of Bag* (OOB) error, che è la media degli errori di predizione su ogni istanza di training $\boldsymbol{x}_i$ mediante l'aggregazione delle predizioni dei soli alberi che **non** contenevano $\boldsymbol{x}_i$ tra le istanze di training

Random Forest - 2

- RF aumenta in maniera contenuta la complessità temporale rispetto agli alberi, dato che ogni singolo albero è appreso solo su un sottoinsieme dei dati e a ogni split solo un sottoinsieme dei predittori viene preso in considerazione. Può gestire dati anche di grande taglia prima di offrire rallentamenti evidenti
- RF non presenta problemi di overfitting all'aumentare del numero di alberi. Questo è dovuto al fatto che solo una piccola porzione dei predittori è usata a ogni split, oltre al fatto che RF predice mediante aggregazione
- RF è più stabile alle variazioni dei dati di input, grazie al bagging
- Gli alberi appresi sono indipendenti (a differenza del boosting)
- Questo rende il metodo facilmente parallelizzabile con hardware appropriato (e.g. più core o processori)

RF in **R**

- Il package *randomForest* fornisce la funzione omonima per apprendere una RF
- Carichiamo il dataset hacide
- Addestriamo una RF su *hacide.train*
- Nell'argomento formula è possibile specificare anche solo la variabile outcome (dipendente)
- Di default viene posto $ntree = 500$ ($ntree$ è n_{tree}) e $mtry = \sqrt{m}$ (m_{try})
- L'argomento `nodesize` (eq. a `minsplit` di `rpart`) è settato al valore che fornisce la complessità maggiore
- La matrice di confusione riporta l'errore per classe e i valori TP, FP, FN, TN

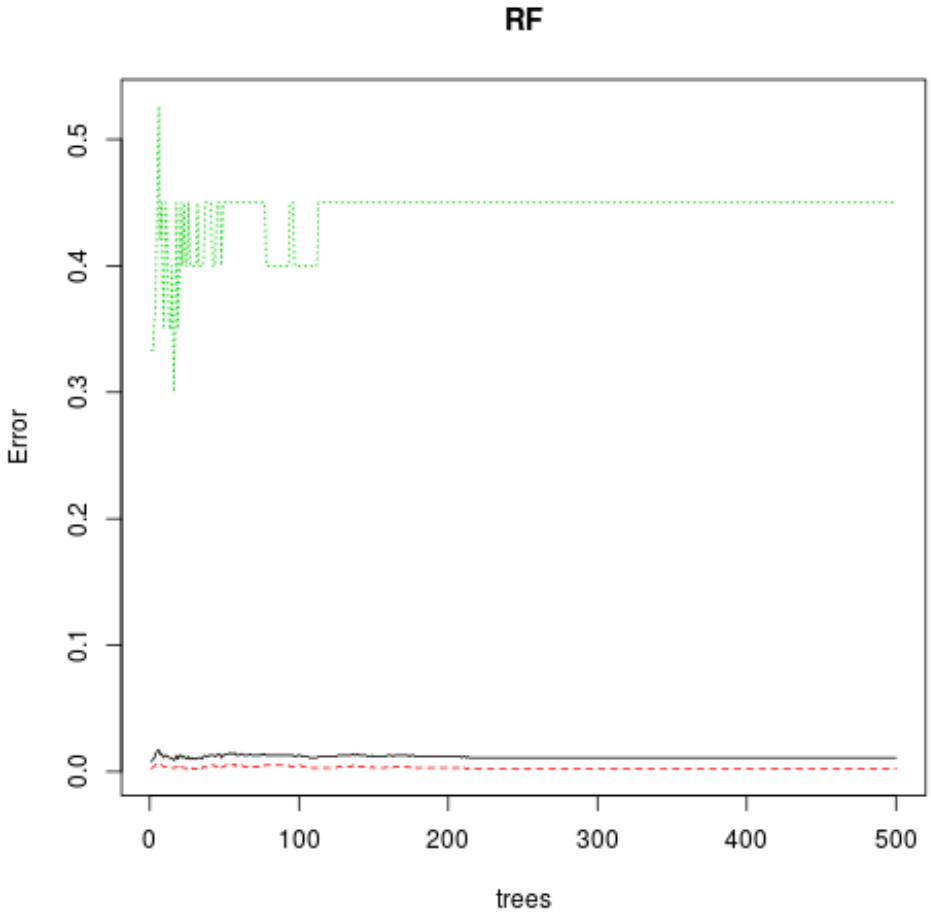
```
library(randomForest)
library(ROSE)
data(hacide)
# se l'outcome variable è un
# factor, di default l'argomento
#  method è settato a "class",
# altrimenti dobbiamo esplicitarlo
RF <- randomForest(cls ~ . ,
         data = hacide.train)
No. of variables tried at
                  each split: 1

 OOB estimate of  error rate: 1.1%
Confusion matrix:
    0  1 class.error
0 978  2 0.002040816
1   9 11 0.450000000
```

RF in **R**

- Tracciamo il grafico dell'errore rispetto al numero di alberi, man mano che la foresta cresce

```
plot(RF)
```

- Viene riportato l'errore (assoluto) di test stimato (linea nera), quello sulla classe negativa (rosso tratteggiato) e quello sulla classe positiva (verde)
- Come si vede 500 alberi sono troppi, ne bastavano meno di 100 in questo caso
- Vediamo sul test set



RF in **R**

- Prediciamo le istanze in *hacide.test* e calcoliamo le prestazioni
- Come notate le prestazioni sono basse, confrontate con quelle di alberi di decisione e kNN
- Questo può essere dovuto al basso numero di predittori (2) e al fatto che il dataset è sbilanciato
- Infatti a ogni split viene sempre valutata una sola variabile (troncamento della radice quadrata di 2)
- Riduciamo il numero di alberi
- La nostra impressione risulta verificata, l'errore non aumenta pur usando meno alberi

```
predRF <- predict(RF,hacide.train)
source("perf.R")
perfRF <- do.perf(hacide.test$cls,
          predRF)
print(round(perfRF, 3))
   A Prec  Rec    F
0.88 0.25 0.25 0.25
RF2 <- randomForest(cls ~ . ,
       data = hacide.train, ntree=75)
print(RF2)
  Type of random forest: classification
                Number of trees: 75
  No.of variables tried at each split: 1

  OOB estimate of  error rate: 1.3%
  Confusion matrix:
    0  1 class.error
  0 976  4 0.004081633
  1   9 11 0.450000000

predRF2 <- predict(RF2,hacide.train)
perfRF2 <- do.perf(hacide.test$cls, predRF2)

print(round(perfRF2, 3))
   A Prec  Rec    F
0.88 0.25 0.25 0.25
```

Dataset *PimaIndiansBiabetes*

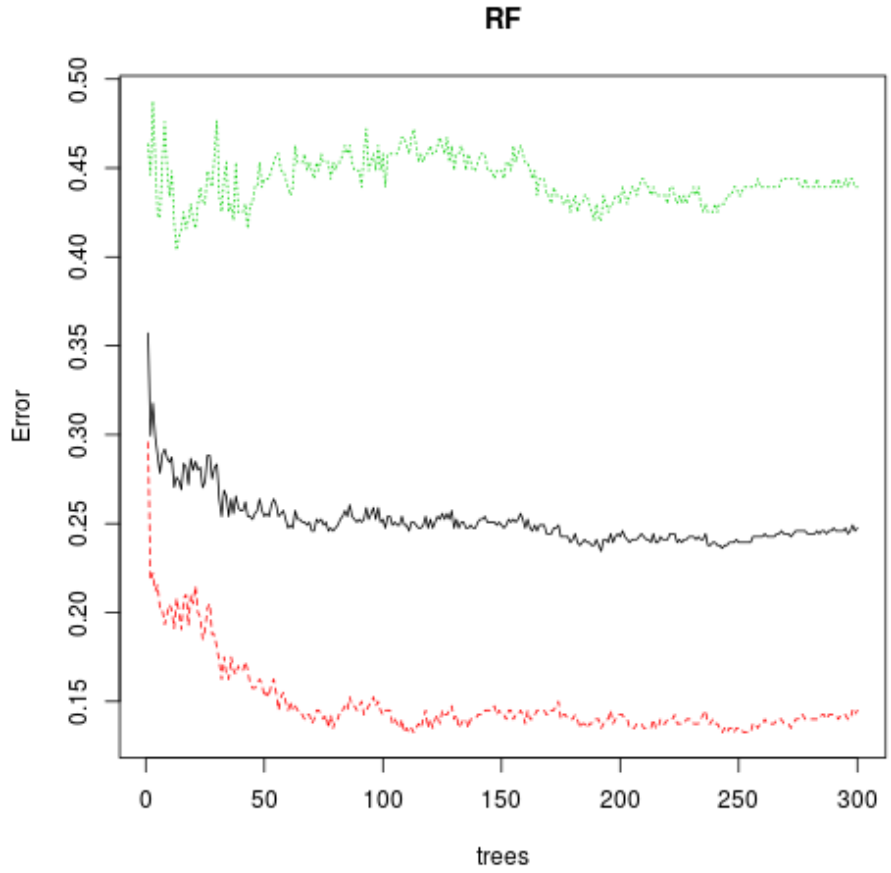
- Carichiamo un dataset con più predittori e un po' più complesso
- Il dataset *PimaIndiansDiabetes*, del package *mlbench*, contiene 768 osservazioni e 8 predittori, l'obiettivo è predire se un paziente ha (o può sviluppare) il diabete o meno a partire dai suoi dati clinici
- La variabile `diabetes` è quella dipendente e denota la presenza o meno di diabete
- Per informazioni sui predittori usare l'help.
- Scegliamo gli insiemi di training e test casualmente in maniera stratificata
- Addestriamo sia una RF che un albero di decisione
- Notiamo che la RF comincia a predire meglio dell'albero di decisione. In genere più aumenta la complessità dei dati, più questa differenza tende a crescere

Confronto RF DT sui dati *PimaIndiansBiabetes*

```
library(mlbench)
data(PimaIndiansDiabetes);   data <- PimaIndiansDiabetes
dim(data)
[1] 768   9
table(data$diabetes)
 neg pos
 500 268
n <- nrow(data); niter <- 50
RF.perf <- DT.perf <- rep(0, 4);
names(RF.perf) <- names(DT.perf) <- c("A", "P", "R", "F")
pos <- which(data$diabetes == "pos"); neg <- setdiff(1:n, pos);
n.pos <- length(pos);   n.neg <- length(neg)
num.lab.diab <- sapply(data[,"diabetes"], function(x){return(ifelse(x=="pos", 1, -1))})
for(i in 1:niter){
  train <- c(sample(pos, floor(n.pos * 0.8)), sample(neg, floor(n.neg * 0.8)))
  test <- setdiff(1:n, train)
  RF <- randomForest(diabetes ~ . , data = data[train,])
  predRF <- predict(RF, data[test,])
  num.pred.RF <- sapply(predRF, function(x){return(ifelse(x=="pos", 1, -1))})
  RF.perf <- RF.perf + do.perf.numeric(num.lab.diab[test], num.pred.RF)
  DT <- rpart(diabetes ~ . , data = data[train,])
  predDT <- predict(DT, newdata=data[test,], type="class")
  num.pred.DT <- sapply(predDT, function(x){return(ifelse(x=="pos", 1, -1))})
  DT.perf <- DT.perf + do.perf.numeric(num.lab.diab[test],  num.pred.DT)
}
print(round(RF.perf/niter, 3))
    A     P     R     F
0.760 0.689 0.580 0.627
print(round(DT.perf/niter, 3))
    A     P     R     F
0.746 0.656 0.583 0.614
```

Errore

- Questo è la stima dell'errore di test di una delle random forest apprese
- I dati *PimaIndiansDiabetes* sono molto 'rumorosi', ci sono delle osservazioni con valori clinici non coerenti. Per questo le prestazioni sono basse



Alcuni argomenti `randomForest`

- `sampsize`: vettore di lunghezza pari al numero di classi (possibili valori della variabile di outcome) che indica quante istanze estrarre da ciascuna classe per costruire l'albero
- `nodesize`: controlla la complessità degli alberi. Indica il minimo numero di istanze in un nodo foglia, quindi è il numero minimo di istanze in un nodo affinché se ne possa effettuare lo split
- `importance`: argomento booleano che indica se l'importanza delle variabili (predittori) deve essere calcolata. Restituita in una matrice contenuta nella componente `importance` della RF. Due misure calcolate:
  a. *Mean decrease in accuracy*. Viene calcolata l'accuratezza della random forest sui dati out-of-bag, quindi, per ogni albero per ogni variabile i, si permuta la colonna i-ma dei dati OOB, in modo da perdere la correlazione (eventuale) con la variabile di outcome. Si predicono le istanze così ottenute e si ricalcola l'accuratezza, valutando la differenza con il precedente valore. Quindi si fa la media sugli alberi
  b. *Mean decrease in the Gini index*. Misura analoga a quella usata negli alberi di decisione: per ogni albero e per ogni variabile i, si valuta ogni nodo in cui lo split coinvolge tale variabile, calcolando la differenza dell'indice di Gini (impurità) prima e dopo la suddivisione, pesandola per il numero di istanze nel nodo. Quindi si fa la media sugli alberi

Variable importance

```
RF <- randomForest(diabetes ~ . , data = data, importance=TRUE)
round(RF$importance,5)
             neg      pos MeanDecreaseAccuracy MeanDecreaseGini
pregnant 0.02195  0.00013                0.01432         28.06205
glucose  0.05406  0.08848                0.06595         89.33329
pressure 0.00506 -0.00165                0.00270         31.81083
triceps  0.00366  0.00419                0.00375         23.43905
insulin  0.00846  0.00609                0.00757         24.89327
mass     0.01956  0.03909                0.02621         57.82641
pedigree 0.00675  0.00682                0.00682         43.45703
age      0.02606  0.02101                0.02424         47.92347
```

- Le prime due colonne rappresentano la *mean decrease in accuracy* per ciascuna classe
- Le ultime due colonne invece rappresentano la *mean decrease in accuracy* e la *mean decrease in the Gini index* su tutte le istanze
- Più alti i valori, maggiore è la rilevanza della variabile

Variable importance

La funzione `varImpPlot` permette di visualizzare graficamente l'importanza delle variabili.

```
varImpPlot(RF)
```

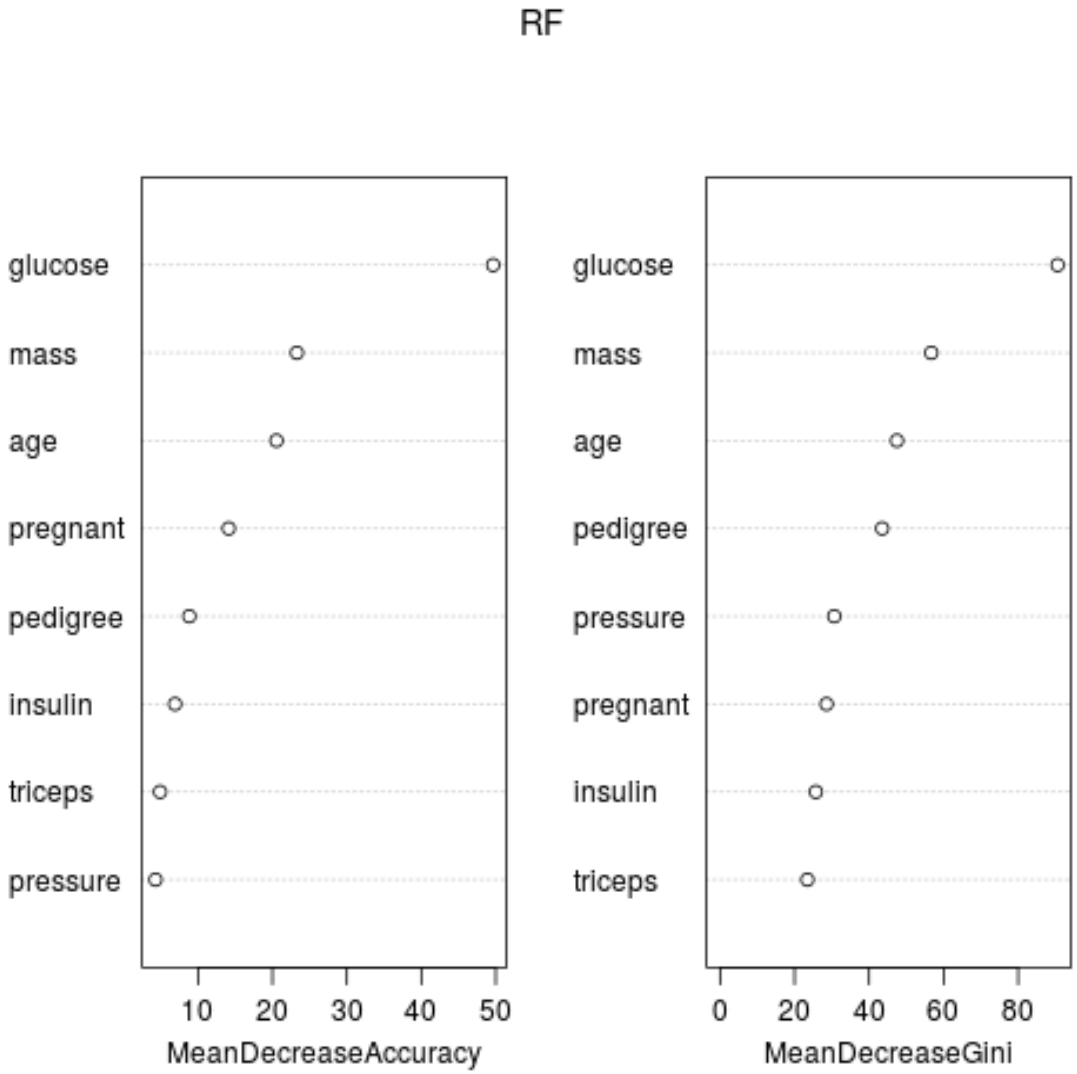
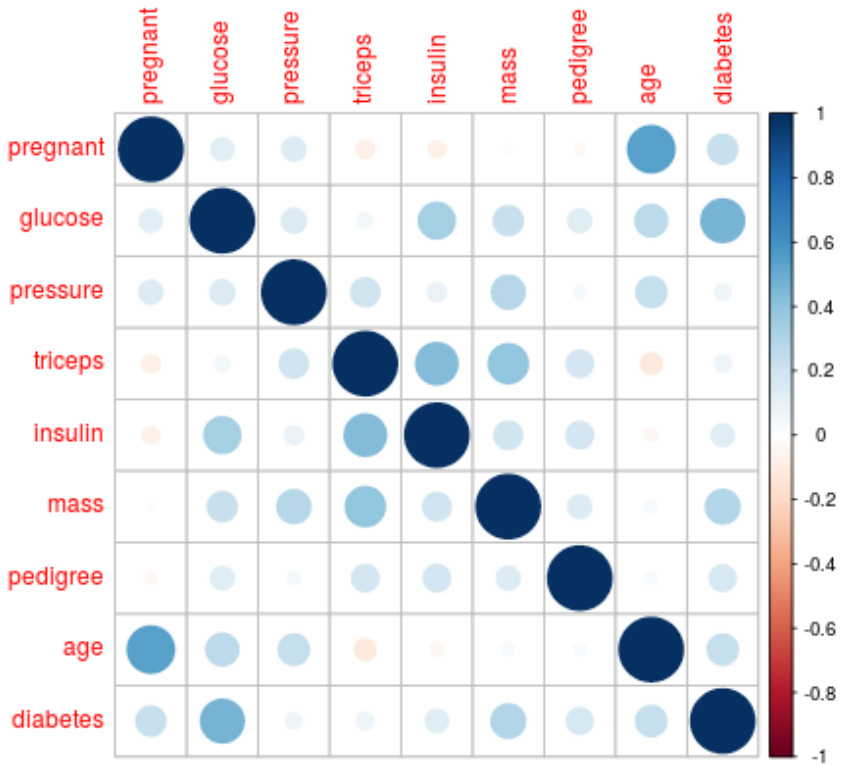


Grafico della variable importance

- Come era ovvio, l'indice di glucosio nel sangue è altamente informativo per predire la classe 'è diabetico'. La seconda variabile più informativa è l'indice di massa corporea (peso diviso per il quadrato dell'altezza espressa in metri)
- Anche l'indice di correlazione viene usato come modo per valutare quanto due variabili siano dipendenti
- In R la funzione `cor(X)` permette di calcolare l'indice di correlazione (di default quella di Pearson) tra ogni coppia di variabili contenute nella matrice (numerica) X
- Usando poi la funzione `corrplot` della libreria omonima si può tracciare il grafico della risultante matrice delle correlazioni

```
library(corrplot)
# assigning to the factor outcome to the corresponding numeric outcome
data[,"diabetes"] <- num.lab.diab
png("corrplot_pima.png")
corrplot(cor(as.matrix(data)))
dev.off()
```

Grafico della variable importance



- Anche secondo l'indice di correlazione, le variabili più *correlate* con *diabetes* sono *glucose*, *mass* e *age*.
- C'è da osservare che la correlazione è una misura che non tiene conto dell'interazione tra variabili (metodi detti univariati)
- La mean decrease in accuracy invece valuta sì l'impatto sulla predizione di una singola variabile, ma al contempo la variazione in accuratezza dipende anche della interazione con le altre variabili (metodi detti multivariati)
- Esistono in letteratura diversi metodi per il calcolo delle feature/variabili più rilevanti (noti come metodi di feature selection). In questo corso ci fermiamo a questi accenni

RF cost sensitive

- Il package `randomForest` non prevede la specifica di costi di classificazione, come visto per gli alberi del package `rpart`
- Tuttavia esistono due argomenti per gestire in qualche modo lo sbilanciamento: `case.weights` e `cutoff`
- `case.weights` è un vettore di lunghezza pari al numero di istanze, contenente una distribuzione di probabilità sulle stesse: l'*i*-ma posizione del vettore è la probabilità che l'istanza *i*-ma sia selezionata per il traning durante il boostrap
- `cutoff` è un vettore di lunghezza pari al numero di classi, contenente per ogni classe la proporzione di voti dei singoli alberi da superare affinché un'istanza possa essere classificata come membro di quella classe. La somma degli elementi del vettore deve dare 1

Esercizi

1. Ripetere l'esercizio alla slide 11 usando una 10-fold cross validation invece di un holdout iterato, riportando la media e la varianza dei risultati ottenuti su ciascun fold
2. Scrivere la funzione **R** `RF.eval` che ha i seguenti input: un dataframe `data`, di cui una colonna (di outcome) è un *factor*; un argomento di tipo formula `fmla` che descrive le dipendenze tra le variabili di `data`; un intero `nfolds` che indica il numero di fold in cui partizionare i dati; `ntree`, numero di alberi della foresta; `nodesize`, intero da assegnare all'argomento omonimo della funzione `randomForest`. La funzione deve stimare l'errore di generalizzazione di una random forest con i parametri forniti, partizionando i dati in `nfolds` sottoinsiemi e applicando una cross validation stratificata e restituendo una lista con due componenti: il vettore delle prestazioni (accuratezza, precisione, recall, F) medie sui fold e il vettore delle deviazioni standard per ciascuna metrica.

Classificazione multiclasse

Un problema di classificazione multiclasse è definito come segue:

- Dati di input: coppie $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$, con $\boldsymbol{x}_i \in \mathbb{R}^d$ e $y_i \in \{1, 2, \ldots, m\}$, per ogni $i \in \{1, 2, \ldots, n\}$
- Le istanze sono vettori reali d-dimensionali con una tra le etichette $1, 2, \ldots, m$
- Sottoinsieme $S \subset D$ delle istanze di cui è nota l'etichetta
- Sottoinsieme $T := D \setminus S$ delle istanze non etichettate
- OBIETTIVO: Apprendere una classificatore $f : \mathbb{R}^d \longrightarrow \{1, 2, \ldots, m\}$ che permetta di classificare le istanze in T (e in generale in $\mathbb{R}^d$)

Classificazione multilabel

Un problema di classificazione multilabel è definito come segue:

- Dati di input: coppie $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$, con $\boldsymbol{x}_i \in \mathbb{R}^d$ e $y_i \subset \{1, 2, \ldots, m\}$, per ogni $i \in \{1, 2, \ldots, n\}$
- Le istanze sono vettori reali d-dimensionali con etichette (potenzialmente) multiple: e.g. $y_i = \{1, 3, 5\}$
- Sottoinsieme $S \subset D$ delle istanze di cui è nota l'etichetta
- Sottoinsieme $T := D \setminus S$ delle istanze non etichettate
- OBIETTIVO: Apprendere una classificatore $f : \mathbb{R}^d \longrightarrow \mathcal{P}(\{1, 2, \ldots, m\})$ che permetta di classificare le istanze in T (e in generale in $\mathbb{R}^d$)
- $\mathcal{P}$ è l'insieme delle parti (serve convenzione per $\emptyset$)
- Possono esserci dipendenze tra le classi (correlazione, relazioni gerarchiche, etc.)

Accenni alla soluzione di problemi multiclasse/multilabel

- Ci sono due strade principali per la soluzione di problemi multiclasse/multilabel: trasformare il problema o adattare/sviluppare l'algoritmo di classificazione
- La prima consiste nel trasformare il problema in problemi multipli di classificazione binaria, da risolvere indipendentemente
- La seconda invece adatta i classificatori esistenti oppure ne sviluppa di nuovi *ad hoc*

Trasformare il problema: OVA - One Versus All

- Una semplice idea è quella di apprendere m distinti classificatori binari $f_i : \mathbb{R}^d \longrightarrow \{0, 1\}$, $i \in \{1, \ldots, m\}$, dove etichetta 1 indica appartenenza alla classe i, 0 appartenenza ad una qualsiasi altra classe.
- La multietichetta di una istanza $\boldsymbol{x} \in T$ si ottiene concatenando le singole predizioni: $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))$
- Nel caso di classificazione multiclasse dovremmo invece utilizzare classificatori $f_i : \mathbb{R}^d \longrightarrow \mathbb{R}$ che anziché fornire una etichettatura binaria, assegnino alle istanze un valore reale $f_i(\boldsymbol{x})$, tale che maggiore $f_i(\boldsymbol{x})$ più elevata è 'l'evidenza' che quella istanza appartenga alla classe i
- Per esempio, nel caso del classificatore kNN, $f_i(\boldsymbol{x})$ potrebbe essere il numero di vicini (tra i k più vicini) ad appartenere alla classe i oppure nelle random forest la proporzione di alberi che hanno classificato l'istanza nella classe i
- Ovviamente bisogna utilizzare lo stesso classificatore per ogni classe
- La classe $f(\boldsymbol{x}) \in \{1, \ldots, m\}$ assegnata all'istanza $\boldsymbol{x}$ è quella che corrisponde alla predizione $f_i(\boldsymbol{x})$ maggiore, cioè $f(\boldsymbol{x}) = \underset{i}{\text{argmax}}\ f_i(\boldsymbol{x})$.

Trasformare il problema: AVA - All Versus All

- Un'altra semplice idea per affrontare un problema multiclasse è quella di apprendere $m(m-1)$ classificatori binari distinti $f_{ij} : \mathbb{R}^d \longrightarrow \mathbb{R}$, $i, j \in \{1, \ldots, m\}$, $i \neq j$, per distinguere le classi i e j
- Ciascun classificatore è appreso soltanto sulle istanze positive per queste due classi, quelle positive per la classe i avranno etichetta 1, -1 le rimanenti
- Si noti quindi che $f_{ij} = -f_{ji}$
- La classe $f(\boldsymbol{x}) \in \{1, \ldots, m\}$ assegnata all'istanza $\boldsymbol{x}$ è quella che corrisponde alla predizione $f(\boldsymbol{x}) = \underset{i}{\text{argmax}} \left(\sum_j f_{ij}(\boldsymbol{x}) \right)$

Adattare/sviluppare un classificatore specifico (accenno)

- Alcuni classificatori binari si possono naturalmente estendere a più classi
- kNN per esempio funziona bene anche con etichette multiclasse, predicendo per una data l'istanza l'etichetta più presente tra i suoi k vicini più prossimi
- Ci sono anche varianti multilabel di kNN (e.g. MLkNN [1]) che predicono una multi-etichetta per la nuova istanza in base alle multi-etichette dei k punti più vicini (informalmente, per ciascuna classe, si predice 1 se essa è in maggioranza nel k-vicinato)
- Gli alberi di decisione (e quindi le RF) sono inerentemente multiclasse
- Ogni nodo è etichettato con una sola classe e lo split è eseguito considerando le istanze nelle altre classi come negative
- Esistono varianti degli alberi di decisione (e.g. Multi-label C4.5 [2]) che invece consentono di predire multilabel
- I nodi foglia sono etichettati con etichette multiple. Gli split utilizzano una entropia multilabel
- Esistono molti altri classificatori multiclasse e multilabel specifici

References I

[1] Min-Ling Zhang and Zhi-Hua Zhou.
Ml-knn: A lazy learning approach to multi-label learning.
Pattern Recognition, 40(7):2038 – 2048, 2007.

[2] Amanda Clare and Ross D. King.
Knowledge discovery in multi-label phenotype data.
In Luc De Raedt and Arno Siebes, editors, *Principles of Data Mining and Knowledge Discovery*, pages 42–53, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.