Corso di Statistica Numerica 2020-21.
Traccia per il progetto di Machine Learning

Davide Evangelista e Elena Loli Piccolomini

davide.evangelista5@unibo.it, elena.loli@unibo.it

Contents

1 Selezionare il Dataset **1**

2 Caricare il Dataset **2**

3 Pre-Processing **2**

4 Splitting **2**

5 Visualizzazione **3**

6 Addestriamo il Modello **4**

7 Valutare la performance di un Modello **4**

8 Hyperparameter Tuning **4**

9 Valutazione della Performance **5**

10 Interpretazione Probabilistica **5**

11 Studio statistico sui risultati della valutazione **6**

12 Feature selection **7**

13 Codice Completo **7**

1 Selezionare il Dataset

Su `https://www.kaggle.com` o `http://archive.ics.uci.edu/ml` scegliere il dataset di riferimento e scaricarlo.

Ricordarsi di utilizzare sempre, in maniera approfondita, la descrizione del dataset, che ci darà le informazioni necessarie per comprendere il significato delle variabili presenti, che spesso sono codificate da nomi non troppo intuitivi.

In questo esempio, utilizzerò come dataset `https://www.kaggle.com/ronitf/heart-disease-uci`.

2 Caricare il Dataset

Il primo passaggio in un qualunque progetto di Machine Learning è quello di caricare il Dataset. Ricordando di impostare come working directory quella in cui è presente il dataset, in `.csv` che si vuole importare (comando `setwd()` per impostare la working directory):

```
# Carichiamo il dataset
df <- read.csv("heart.csv")
num.df <- df
```

In questo caso mi sono salvato una copia del dataset originale `df` in una variabile di backup `num.df`. Questo perché nella seconda fase andremo a modificare `df`, ed è buona norma tenersi in memoria anche il dataset non modificato, se dovesse servire.

3 Pre-Processing

In questa seconda fase, bisogna:

1. Ripulire il dataset da eventuali NaN (comando `na.omit(df)`).
2. Controllare che le variabili di tipo categorico siano dei `factor` e, in caso contrario, renderle dei `factor`.
3. Controllare che le variabili di tipo numerico non presentino dei valori fuori soglia (numeri troppo bassi da essere realistici, o troppo alti).
4. Controllare, in generale, che gli elementi del dataset siano corretti ed eliminare eventuali dati corrotti.

```
# Ripuliamo (eventualmente) il dataset
df <- na.omit(df)
names(df)[names(df) == "age"] <- "age"

# PROBLEMA: le variabili categoriche sono salvate come numeriche
levels(df$sex)

df$sex <- as.factor(df$sex)
df$target <- as.factor(df$target)
levels(df$sex)
```

4 Splitting

Nella fase di Splitting, il dataset deve essere diviso in training set, validation set e test set, seguendo le indicazioni presenti nelle ultime pagine delle slides del corso. Le dimensioni dei tre sottoinsiemi così ottenute sono arbitrarie. E' consigliato far sì che validation set e test set siano all'incirca grandi uguali, mentre il training set sia più grande degli altri due. Fare alcune prove fino a trovare una buona dimensione.

```r
# Split
N.train <- 200
N.test <- 50
N.val <- 53

train.sample <- sample(N, N.train)
df.train <- df[train.sample, ] # [righe, colonne]
df.test <- df[-train.sample, ]

val.sample <- sample(N.test + N.val, N.val)
df.val <- df.test[val.sample, ]
df.test <- df.test[-val.sample, ]
```

Una volta splittato il dataset, si consiglia di rinominare con un nome semplice il training set (magari sostituendolo al nome che aveva il dataset iniziale). Questo perché, per rendere valida l'analisi, da questo momento in poi dovremo far finta di non aver a disposizione il test set (che rappresenta il dataset contenente i dati futuri, che devono ancora essere collezionati), e quindi useremo solo training set e validation set.

```r
# Dimenticare dell'esistenza del test
df <- df.train
N <- nrow(df)
```

5 Visualizzazione

In questa fase, ci si può sbizzarrire. L'idea è quella di sfruttare gli strumenti grafici messi a disposizione da `R` per indagare alcune proprietà statistiche del dataset. Il numero e la tipologia di grafici dipende dal dataset a disposizione, l'importante è concludere questa fase avendo coscienza di come interagiscono tra loro (a livello statistico) le variabili di input del dataset.

```r
# Visualizzazione
library(ggplot2)
ggplot(data=df, aes(sex, age)) +
  geom_boxplot()

df.disease <- subset(df, subset=(target==1))
ggplot(data=df.disease, aes(sex, age)) +
  geom_boxplot()

 # Correlation Matrix
install.packages("corrplot")
library(corrplot)

cor.matrix <- cor(num.df)
corrplot(cor.matrix, method="circle")
```

Ho aggiunto anche il codice per stampare la matrice di correlazione, strumento che può essere particolarmente utile nell'indagine della maggior parte dei dataset. Notare come, per la matrice di correlazione, ho utilizzato il dataset di backup `num.df`. Questo perché, per stamparla, è necessario utilizzare solo colonne di tipo numerico.

6 Addestriamo il Modello

Siamo quindi pronti ad addestrare il nostro modello (in questo caso una Support Vector Machine). Per farlo, una volta scelto il tipo di `kernel`, e gli altri parametri che dipendono dal kernel (fare riferimento alle slides), lanciamo il comando

```
# Machine learning
install.packages("e1071")
library(e1071)

# Addestriamo il modello
model.SVM <- svm(target ~ ., df, kernel="polynomial", cost=10, degree=10)
summary(model.SVM)
```

Come ci addestra un modello `model.SVM` sfruttando le informazioni contenute nel training set `df`.

7 Valutare la performance di un Modello

Una volta definito un modello, bisogna valutarne la performance. Per farlo, definiamo prima di tutto le funzioni che serviranno allo scopo:

```
# Misurare la qualita del modello
MR <- function(y.pred, y.true) {
  res <- mean(y.pred != y.true)
  return(res)
}

Acc <- function(y.pred, y.true) {
  res <- 1 - mean(y.pred != y.true)
  return(res)
}
```

Fatto questo, possiamo provare ad utilizzare `SVM.model` per predire il test set, e valutare la qualità della predizione.

```
# Usiamo il modello per fare delle previsioni
y.pred <- predict(model.SVM, df.test)

# Valutiamo la performance del modello
MR.test <- MR(y.pred, df.test$target)
MR.test

Acc.test <- Acc(y.pred, df.test$target)
Acc.test
```

8 Hyperparameter Tuning

Abbiamo visto come le performance del modello dipendono drasticamente dalla scelta degli iperparametri (ovvero tutti quei parametri che vanno passati in input alla funzione `svm`, come

il `kernel`, il `cost` e il `degree` / `gamma`.

Dobbiamo quindi identificare la combinazione ottimale per questi parametri, utilizzando il validation set (e sperando che questa combinazione sia ottima anche sul validation set).

```
# Visualizziamo la performance al variare del grado
MR.total <- 1:10
for (d in 1:10) {
  model.SVM <- svm(target ~ ., df, kernel="polynomial", cost=10, degree=d)
  y.pred <- predict(model.SVM, df.val)
  MR.poly <- MR(y.pred, df.val$target)
  MR.total[d] <- MR.poly
}

plot(MR.total, type='p', xlab="Degree", ylim=c(0, 1))
```

Osservando il grafico così ottenuto, è possibile trovare il parametro `degree` ottimale per la nostra situazione. Fissato quello, bisognerà poi ripetere per il parametro di costo e cercare una combinazione ottima di questi parametri. E' buona norma, seguendo quello fatto a lezione, visualizzare su uno stesso grafico anche la curva dell'errore valutata sul training set, per rendersi conto di come differiscono le due curve.

9 Valutazione della Performance

Una volta identificata la combinazione ottimale di iperparametri per un determinato `kernel`, si può passare alla fase di valutazione, in cui il modello viene valutato (possibilmente utilizzando un numero variabile di metriche) sul test set, e se ne delinano punti di forza e di debolezza.

```
# Abbiamo visto che il punto di minimo si ha in cost=1 e deg=3
model.SVM <- svm(target ~ ., df, kernel="polynomial", cost=1, degree=3)

# Valutiamo la performance sul Test Set
y.pred <- predict(model.SVM, df.test)
MR.test <- MR(y.pred, df.test$target)
MR.test

Acc.test <- Acc(y.pred, df.test$target)
Acc.test
```

10 Interpretazione Probabilistica

Nel caso di Classificazione Binaria (in cui, cioè, la variabile y di output abbia due classi possibili, è possibile utilizzare l'interpretazione probabilistica della classificazione per avere una stima della distanza di ciascun punto dalla linea di separazione tra le classi.

Per addestrare una SVM probabilistica, il codice è il seguente:

```
# Approccio Probabilistico
SVM.probs <- svm(target ~ ., data=df, kernel="linear", cost=1, probability=TRUE)
```

```
y.pred <- predict(SVM.probs, df.test, probability=TRUE)
y.probs <- attr(y.pred, "probabilities")

y.probs <- y.probs[, 2]
```

Dove, chiaramente, i parametri di `kernel`, `cost` ecc, sono quelli ottimali misurati nei punti precedenti (io ho messo `kernel="linear"` e `cost=1` solo a scopo di esempio. In questo modo, il vettore `y.probs` è un vettore di probabilità, in cui l'elemento i-esimo, rappresenta la probabilità che l'i-esimo dato di `df.test` sia assegnato alla classe 1. E' quindi necessario convertirlo in un vettore di predizioni (ovvero, di elementi 0 o 1).

```
# Convertire y.probs in un vettore di predizioni
y.total <- rep(0, 50)
y.total[y.probs > 0.5] <- 1
```

dove il parametro oltre il quale assegnamo alla classe 1, in questo caso 0.5, è detto *treshold*. Selezionare il giusto treshold permette di migliorare di molto la performance del modello.

Uno dei grandi vantaggi dell'intepretazione probabilistica è quello di potersi stampare la matrice di confusione, che ci permette di visualizzare il numero di Falsi Negativi e Falsi Positivi del nostro modello.

```
# Confusion Matrix
table(y.pred, df.test$target)
```

E' necessario quindi ottimizzare il treshold tale da minimizzare i Falsi Negativi o i Falsi Positivi, dipendentemente dal senso insito nel dataset.

Una volta trovato il treshold ottimale, valutare nuovamente il modello e stamparsi curva di ROC e AUC, per concludere la valutazione.

```
# Curva di ROC
install.packages("ROCR")
library(ROCR)
pred <- prediction(y.probs, df.test$target)
perf <- performance(pred, "tpr", "fpr")

# AUC
auc <- performance(pred, "auc")
auc <- auc@y.values[[1]]

plot(perf, colorize=TRUE, main=auc)
```

11 Studio statistico sui risultati della valutazione

L'esecuzione del modello una sola volta non è sufficiente per dare una valutazione corretta del modello, data la aleatorietà dei dati utilizzati. Per questo si suggerisce di ripetere le fasi di addestramento e testing un numero k di volte con $k \geq 10$. Di ogni metrica di errore abbiamo quindi un SRS(k).

- Usare strumenti di statistica descrittiva (calcolo centro dei dati, diffusione...) e grafici (istogramma, boxplot) per descrivere statisticamente il campione.
- Usare strumenti di statistica inferenziale per fare inferenza riguardo alla distribuzione cui appartiene il campione. In particolare, stimare la media e calcolare l' intervallo di confidenza con livello di confidenza $\alpha = 0.05$ (quindi con probabilità del 95%).

12 Feature selection

Un ulteriore studio sul modello viene effettuato realizzando la cosiddetta operazione di *feature selection*. Consiste nell'analizzare l' influenza che hanno alcune delle features o caratteristiche dei dati sulla performance del modello. In modo molto euristico, si possono selezionare alcune delle caratteristiche ed eseguire il modello SOLO sulle features selzionate. Questa operazione, ripetuta piu' volte scegliendo caratteristiche differenti, permette appunto di individuare le features piu importanti ai fini della qualità del modello.

```
# Model Selection
model.SVM <- svm(target ~ age + sex, df, kernel="polynomial", cost=10, degree=5)
y.pred <- predict(model.SVM, df.test)
MR(y.pred, df.test$target)
Acc(y.pred, df.test$target)
```

Ripetere tutti i passaggi di valutazione, con i nuovi input, e giustificare il perché avete scelto proprio quegli input rispetto agli altri.

13 Codice Completo

```
# Carichiamo il dataset
df <- read.csv("heart.csv")
num.df <- df

# Ripuliamo (eventualmente) il dataset
df <- na.omit(df)
names(df)[names(df) == "age"] <- "age"

# PROBLEMA: le variabili categoriche sono salvate come numeriche
levels(df$sex)

df$sex <- as.factor(df$sex)
df$target <- as.factor(df$target)
levels(df$sex)

# Split
N.train <- 200
N.test <- 50
N.val <- 53

train.sample <- sample(N, N.train)
df.train <- df[train.sample, ] # [righe, colonne]
df.test <- df[-train.sample, ]
```

```
val.sample <- sample(N.test + N.val, N.val)
df.val <- df.test[val.sample, ]
df.test <- df.test[-val.sample, ]

# Dimenticare dell'esistenza del test
df <- df.train
N <- nrow(df)

# Visualizzazione
library(ggplot2)
ggplot(data=df, aes(sex, age)) +
  geom_boxplot()

df.disease <- subset(df, subset=(target==1))
ggplot(data=df.disease, aes(sex, age)) +
  geom_boxplot()

 # Correlation Matrix
install.packages("corrplot")
library(corrplot)

cor.matrix <- cor(num.df)
corrplot(cor.matrix, method="circle")

# Machine learning
install.packages("e1071")
library(e1071)

# Addestriamo il modello
model.SVM <- svm(target ~ ., df, kernel="polynomial", cost=10, degree=10)
summary(model.SVM)

# Misurare la qualita del modello
MR <- function(y.pred, y.true) {
  res <- mean(y.pred != y.true)
  return(res)
}

Acc <- function(y.pred, y.true) {
  res <- 1 - mean(y.pred != y.true)
  return(res)
}

# Usiamo il modello per fare delle previsioni
y.pred <- predict(model.SVM, df.test)

# Valutiamo la performance del modello
MR.test <- MR(y.pred, df.test$target)
MR.test

Acc.test <- Acc(y.pred, df.test$target)
Acc.test

# Visualizziamo la performance al variare del grado
MR.total <- 1:10
for (d in 1:10) {
  model.SVM <- svm(target ~ ., df, kernel="polynomial", cost=10, degree=d)
  y.pred <- predict(model.SVM, df.val)
```

```
  MR.poly <- MR(y.pred, df.val$target)
  MR.total[d] <- MR.poly
}

plot(MR.total, type='p', xlab="Degree", ylim=c(0, 1))

# Abbiamo visto che il punto di minimo si ha in cost=1 e deg=3
model.SVM <- svm(target ~ ., df, kernel="polynomial", cost=1, degree=3)

# Valutiamo la performance sul Test Set
y.pred <- predict(model.SVM, df.test)
MR.test <- MR(y.pred, df.test$target)
MR.test

Acc.test <- Acc(y.pred, df.test$target)
Acc.test

# Approccio Probabilistico
SVM.probs <- svm(target ~ ., data=df, kernel="linear", cost=1, probability=TRUE)
y.pred <- predict(SVM.probs, df.test, probability=TRUE)
y.probs <- attr(y.pred, "probabilities")

y.probs <- y.probs[, 2]

# Convertire y.probs in un vettore di predizioni
y.total <- rep(0, 50)
y.total[y.probs > 0.5] <- 1

# Confusion Matrix
table(y.pred, df.test$target)

# Curva di ROC
install.packages("ROCR")
library(ROCR)
pred <- prediction(y.probs, df.test$target)
perf <- performance(pred, "tpr", "fpr")

# AUC
auc <- performance(pred, "auc")
auc <- auc@y.values[[1]]

plot(perf, colorize=TRUE, main=auc)

# Model Selection
model.SVM <- svm(target ~ age + sex, df, kernel="polynomial", cost=10, degree=5)
y.pred <- predict(model.SVM, df.test)
MR(y.pred, df.test$target)
Acc(y.pred, df.test$target)
```