Data Mining and Machine Learning Lab. Lezione 5 Master in Data Science for Economics, Business and Finance 2018

27.04.18

Marco Frasca

Università degli Studi di Milano

Classificazione bianaria con dati sbilanciati

Richiamiamo la definizione di problema di classificazione binaria

- 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 $y_i \in \{-1, 1\}$, per ogni $i \in \{1, 2, \ldots, n\}$
- Le istanze sono vettori reali d-dimensionali con etichetta binaria (-1, +1)
- 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 binario $f : \mathbb{R}^d \longrightarrow \{-1, 1\}$ che permetta di classificare le istanze in T (e in generale in $\mathbb{R}^d$)
- Il problema di classificazione è *sbilanciato* se una classe (supponiamo la classe positiva) è fortemente sotto-rappresentata, cioè $|S^+|/|S^-| \ll 1$, dove

$$S^+ = \{(\boldsymbol{x}, y) \in S | y = 1\}$$

$$S^- = \{(\boldsymbol{x}, y) \in S | y = -1\}.$$

Classificazione bianaria con dati sbilanciati

- Lo sbilanciamento delle etichette è un problema serio quando si vogliono apprendere dei classificatori
- In genere si ha un forte decadimento delle prestazioni
- L'estrema rarità dei positivi induce i classificatori ad apprendere sostanzialmente una sola classe (quella maggioritaria/negativa)
- I positivi in tali casi rappresentano paradossalmente gran parte dell'informazione
- **Esempio**: pensiamo ai pazienti affetti da una patologia rara. Tutto il resto della popolazione risulterà negativo (sano), mente un piccolissimo gruppo di pazienti costituirà la classe dei positivi (malati). I casi da studiare sono senz'altro i positivi, perché presentano le caratteristiche fenotipiche peculiari della patologia. Un classificatore che non tenga conto di questo otterrà necessariamente 'prestazioni basse'.
- Cosa si intende per prestazioni basse?

Misura delle prestazioni in casi sbilanciati

- Dati due vettori $\boldsymbol{y}, \bar{\boldsymbol{y}} \in \{-1, 1\}^m$, l'accuratezza (A) della predizione $\bar{\boldsymbol{y}}$ rispetto alle etichette reali $\boldsymbol{y}$ è:

$$A(\boldsymbol{y}, \bar{\boldsymbol{y}}) := |\{i \in m | y_i = \bar{y}_i\}|/m \ = \ 1 - err$$

  dove *err* è l'errore assoluto. Quindi A è la proporzione di predizioni corrette, per cui $0 \leq A \leq 1$.

- **Esempio**. Supponiamo $m = 100$ e che $\boldsymbol{y}$ contenga 98 etichette negative. Sia $\bar{\boldsymbol{y}}' = \{-1\}^m$ una possibile predizione per le stesse istanze (il classificatore associato predice tutto negativo). In questo caso $A(\boldsymbol{y}, \bar{\boldsymbol{y}}') = 0.98$, che è un valore relativamente alto. Quindi apparentemente $\bar{\boldsymbol{y}}'$ sarebbe una buona predizione. Sia ora $\boldsymbol{y}'' \in \{-1, 1\}^m$ un'altra predizione, che predice correttamente le due istanze positive e 96 delle 98 istanze negative. In questo caso otteniamo la stessa accuretezza $A(\boldsymbol{y}, \bar{\boldsymbol{y}}'') = 0.98$, ma il classificatore che produce $\bar{\boldsymbol{y}}''$ si fa nettamente preferire, avendo classificato correttamente le uniche due istanze positive.

- Pertanto l'accuratezza (e quindi l'errore assoluto) **non** sono adatti a misurare le prestazioni dei classificatori in presenza di dati sbilanciati

Misura delle prestazioni in casi sbilanciati

- Dati due vettori $\boldsymbol{y}, \bar{\boldsymbol{y}} \in \{-1, 1\}^m$, definiamo:

$$
\begin{aligned}
TP(\boldsymbol{y}, \bar{\boldsymbol{y}}) &:= |\{i \in m | y_i = 1 \wedge \bar{y}_i = 1\}| \\
FP(\boldsymbol{y}, \bar{\boldsymbol{y}}) &:= |\{i \in m | y_i = -1 \wedge \bar{y}_i = 1\}| \\
FN(\boldsymbol{y}, \bar{\boldsymbol{y}}) &:= |\{i \in m | y_i = 1 \wedge \bar{y}_i = -1\}|
\end{aligned}
$$

dove TP è il numero di veri positivi (*true positive* - istanze positive classificate correttamente), FP è il numero di falsi positivi (*false positive* - istanze negative classificate come positive) e FN è il numero di falsi negativi (*false negative* - istanze positive classificate come negative)

Misura delle prestazioni in casi sbilanciati

- Misure che tengono conto della qualità delle predizioni delle istanze positive sono le seguenti:

$$\begin{aligned} \mathit{Precision} &:= \frac{TP}{TP+FP} \\ \mathit{Recall} &:= \frac{TP}{TP+FN} \\ F_\beta &:= \frac{(1+\beta^2)\mathit{Precision}\cdot\mathit{Recall}}{\beta^2\mathit{Precision}+\mathit{Recall}} \end{aligned}$$

- La Precision è quindi la frazione di predizioni positive corrette (TP + FP rappresenta il numero totale di predizioni positive)
- La Recall è la frazione di istanze positive predette correttamente (TP + FN è il numero di positivi nel data set)
- F_β è detta *F measure* (o *F score*) e rappresenta la media armonica di Precision e Recall quando $\beta = 1$. In tal caso la F_1 si indica con F
- Tutte queste misure assumono valori nell'intervallo $[0, 1]$.

Misura delle prestazioni in casi sbilanciati

- Una Precision elevata è indice di pochi falsi positivi, ma non garantisce che il classificatore riesca a individuare correttamente i positivi nel data set
- Una Recall elevata indica che il classificatore predice correttamente buona parte delle istanze positive nel data set, ma non dà garanzie sul numero di falsi positivi (un classificatore che predice sempre positivo ha Recall 1)
- La F invece è una misura che raggiunge valori elevati soltanto se entrambe Precision e Recall sono elevate. Inoltre, per definizione, la F è spostata verso il valore più basso tra Precision e Recall
- Tornando all'esempio precedente otteniamo
  $F(\boldsymbol{y}, \bar{\boldsymbol{y}}') = 0$
  $F(\boldsymbol{y}, \bar{\boldsymbol{y}}'') = 2\frac{0.5 \cdot 1}{0.5+1} = 0.\bar{6}$
- Quindi il secondo classificatore è nettamente migliore del primo secondo la F, che quindi è molto più informativa dell'accuratezza in questo contesto

Gestire lo sbilanciamento

- Una volta stabiliti gli strumenti adatti a misurare le prestazioni dei classificatori in casi di etichettature sbilanciate, dobbiamo capire cosa è possibile fare per evitare classificazioni di bassa qualità
- Esistono 4 approcci principali:
  1. Undersampling
  2. Oversampling
  3. Generazione di dati sintetici
  4. Cost-Sensitive (CS) Learning

Gestire lo sbilanciamento - Undersampling

- Questo metodo ha come obiettivo quello di ridurre il numero di istanze nella classe maggioritaria (supponiamo quella dei negativi) per bilanciare i dati
- La strategia casuale (*random*) estrae casualmente senza rimpiazzo un numero fissato (di solito pari al numero di positivi) di istanze negative
- Un'altra possibile strategia è quella di usare un certo criterio (*informative*) per scegliere il sottoinsieme delle istanze più significative
  - Due note strategie sono la *EasyEnsemble* e la *BalanceCascade* [**?**].
    EasyEnsemble estrae casualmente diversi sottoinsiemi dai negativi e usa ciascun sottoinsieme (e le istanze positive) per addestrare un classificatore. La predizione delle istanze non etichettate è la combinazione (*ensemble*) delle predizioni dei singoli classificatori (es. tramite voto di maggioranza)
    BalanceCascade dopo aver addestrato un classificatore, rimuove dall'insieme dei negativi quelli che il classificatore già predice come negativi
- La strategia *random* è utile con dati di grandi dimensioni, perché riduce il tempo di esecuzione. Tuttavia, potrebbe scartare istanze molto informative
- Le strategie *informative* invece tendono ad usare tutta l'informazione e a scartare solo le istanze ridondanti, ma sono computazionalmente costose

Gestire lo sbilanciamento - Oversampling

- Questo metodo ha come obiettivo quello di aumentare il numero di istanze della classe minoritaria per bilanciare i dati
- L'idea è di replicare alcune (scelte opportunamente) istanze positive
- Non vi è perdita di informazione
- Tuttavia replicare istanze può indurre overfitting
- Ovviamente è possibile utilizzare oversampling insieme a strategie di undersampling, per ridurre il numero di repliche

Gestire lo sbilanciamento - Generazione di dati sintetici

- Idea simile all'oversampling, ma piuttosto che replicare le istanze positive, si generano nuove istanze positive
- Un esempio è l'algoritmo SMOTE (synthetic minority oversampling technique) [**?**]
- Fissata una istanza positiva $\boldsymbol{x}$, sceglie casualmente uno dei k nearest neighbor di $\boldsymbol{x}$, diciamo $\boldsymbol{z}$. Sia $\boldsymbol{w}$ il nuovo punto
- Per ogni dimensione i dello spazio delle feature, calcola la differenza su quella dimensione $x_i - z_i$
- Genera un valore casuale uniforme tra $r \in [0, 1]$ e pone $w_i = x_i + r(x_i - z_i)$
- In nuovo punto ha ciascuna componente che si trova sul segmento tra le corrispondenti feature di $\boldsymbol{x}$ e $\boldsymbol{z}$

Gestire lo sbilanciamento - Cost-Sensitive Learning

- L'idea è di penalizzare maggiormante la classificazione errata dei positivi rispetto a quella dei negativi
- Non crea dati bilanciati
- Come abbiamo visto, i due tipi di errore che il classificatore può commettere sono il falso positivo e il falso negativo. Il secondo deve essere penalizzato maggiormente
- Definiamo matrice dei costi (*cost o loss matrix*) la seguente matrice:

Etichette	**Predizione**		
	1	−1
1	0	C_{FN}
−1	C_{FP}	0

- Di solito, con dati sbilanciati, si pone $C_{FP} = 1$ e $C_{FN} = \frac{|S^-|}{|S^+|}$
- Lo vedremo in un esempio con gli alberi di decisione

Data set sbilanciato - ROSE

- Il package R `ROSE` contiene un data set sbilanciato chiamato `hacide`
- Contiene due data.frame `hacide.train` e `hacide.test`, rispettivamente da usare come dati di train e di test
- 1000 istanze di training e 250 di test, tre attributi, x_1, x_2, `cls`, di cui `cls` è la variabile da predire, fattore con livelli 0 e 1 con la classe 1 sotto-rappresentata

```
#install.packages("ROSE")
library(ROSE)
data(hacide)
dim(hacide.train)
[1] 1000    3
dim(hacide.test)
[1] 250   3
colnames(hacide.train)
[1] "cls" "x1"  "x2"
table(hacide.train$cls)
0   1
980 20
table(hacide.test$cls)
  0   1
245   5
```

Albero di decisione per `hacide`

Apprendiamo un albero di decisione sul data set `hacide` con i parametri di default e classifichiamo le istanze di test

```
library(rpart)

tree <- rpart(cls ~ x1 + x2, method="class", data=hacide.train)

pred <- predict(tree, newdata=hacide.test, type="class")

perf <- do.perf(hacide.test$cls, pred)

print(round(perf, 3))
    A  Prec   Rec     F
0.984 1.000 0.200 0.333
```

Funzione `do.perf`

```
# y and pred are factors with levels "1" and "0".
# y corresponds to the true labels.
# Computes the performance of pred with regard to y

do.perf <- function (y, pred){
    n <- length(y)
    results <- rep(0, 4)
    names(results) <- c("A", "Prec", "Rec", "F");
    TP <- sum(y=="1" & pred=="1")
    FP <- sum(y=="0" & pred=="1")
    FN <- sum(y=="1" & pred=="0")
    if(TP + FP > 0) results["Prec"] <- TP/(TP+FP)
    if(TP + FN > 0) results["Rec"] <- TP/(TP+FN)
    if(TP + FP + FN >0)
        results["F"] <- 2*TP/(2*TP + FP + FN)
    results["A"] <- 1-(FP+FN)/n
    return(results)
}
```

Funzione do.perf.numeric

```
# y and pred are binary numeric vectors having value 1
# for positive labels.
# Computes the performance of pred with regard to y

do.perf.numeric <- function (y, pred){
    n <- length(y)
    results <- rep(0, 4)
    names(results) <- c("A", "Prec", "Rec", "F");
    TP <- sum(y>0 & pred>0)
    FP <- sum(y<=0 & pred>0)
    FN <- sum(y>0 & pred<=0)
    if(TP + FP > 0) results["Prec"] <- TP/(TP+FP)
    if(TP + FN > 0) results["Rec"] <- TP/(TP+FN)
    if(TP + FP + FN >0)
        results["F"] <- 2*TP/(2*TP + FP + FN)
    results["A"] <- 1-(FP+FN)/n
    return(results)
}
```

Albero di decisione Cost-Sensitive

- La funzione `rpart` ha il parametro `loss` della lista `parms` che permette di specificare la matrice dei costi

```
# cost/loss matrix
cost_matrix <- matrix(0, nrow=2, ncol=2);
rownames(cost_matrix) <- colnames(cost_matrix) <- c("0", "1")

# false positive misclassification cost
cost_matrix["0","1"] <- 1
n_neg <- sum(hacide.train$cls=="0")
n_pos <- sum(hacide.train$cls=="1")

# false negative misclassification cost
cost_matrix["1","0"]<-floor(n_neg/n_pos)

treeCS <- rpart(cls ~ x1 + x2, method="class", data=hacide.train,
                parms = list(loss = cost_matrix))
predCS <- predict(treeCS, newdata=hacide.test, type="class")
perfCS <- do.perf(hacide.test$cls, predCS)
print(round(perfCS, 3))
    A  Prec   Rec     F
0.980 0.500 0.600 0.545
```

Valutiamo i due alberi

```
 print(tree)
n= 1000
node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 1000 20 0 (0.98000000 0.02000000)
  2) x1>=-1.495967 989 10 0 (0.98988878 0.01011122) *
  3) x1< -1.495967 11  1 1 (0.09090909 0.90909091) *

print(treeCS)
n= 1000
node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 1000 980 0 (0.980000000 0.020000000)
   2) x2>=-1.030385 811  98 0 (0.997533909 0.002466091)
     4) x1>=-1.310813 804   0 0 (1.000000000 0.000000000) *
     5) x1< -1.310813 7   5 1 (0.714285714 0.285714286) *
   3) x2< -1.030385 189 171 1 (0.904761905 0.095238095)
     6) x1>=-0.7313171 158 153 1 (0.968354430 0.031645570)
      12) x1< 1.01075 151   0 0 (1.000000000 0.000000000) *
      13) x1>=1.01075 7   2 1 (0.285714286 0.714285714) *
     7) x1< -0.7313171 31  18 1 (0.580645161 0.419354839) *
```

Esercizi

1. Implementare la funzione **R** `kNN.predict(M,y,newdata,K)` che restituisca le predizioni del classificatore kNN con $k = K$, dove `M` è una matrice $n \times d$ che descrive il data set di input, `y` è un vettore a valori -1, 0, 1 di lunghezza n, e `newdata` è un vettore di interi tra 1 e n contenente gli indici delle istanze di `M` da predire. $y[i] = 0$ se i è un elemento di `newdata`, -1 o 1 altrimenti. L'output deve essere un vettore `pred` che ha la stessa lunghezza di `newdata` e tale che `pred[i]` è la predizione per l'istanza `newdata[i]`. Suggerimenti: estrapolare le istruzioni dalla funzione `train.kNN.test` vista nelle lezioni precedenti.
2. Apprendere il classificatore kNN mediante cross validation interna sui dati `hacide.train` e poi usare la funzione scritta nell'esercizio precedente con il k appreso per predire le istanze `hacide.test`. Calcolare le prestazioni corrispondenti (mediante la funzione `do.perf.numeric`).
3. Modificare la funzione `train.minslipt.DT.CV` per stimare il parametro ottimale `minslipt` di un albero di decisione cost-sensitive dove i costi degli errori sono calcolati con il metodo visto a lezione. Applicare la funzione al data set `hacide.train` e valutarne le prestazioni su `hacide.test`.

References I