— Parte 2. —

Regressione lineare e lineare logistica

Paolo Bosetti (paolo.bosetti@unitn.it)

Indice

1 Regressione lineare

I modelli lineari in R sono costruiti mediante la funzione `lm()`, già vista nella Parte 1.

Si ricorda che `lm()` costruisce modelli lineari *nei coefficienti* mediante una *formula*. Tali modelli possono correlare un numero arbitrario di *predittori* (cioè variabili indipendenti) ad una variabile dipendente. È inoltre possibile applicare *trasformazioni* alla variabile dipendente ma anche ai predittori: ad esempio, un modello $\log(y) = Ax_1 + Bx_2^2$ rimane lineare nei coefficienti. Al contrario, un modello $y = (Ax_1 + Bx_2)/(Cx_3)$ **non è** lineare nei coefficienti.

1.1 Esempio: modello quadratico, univariato

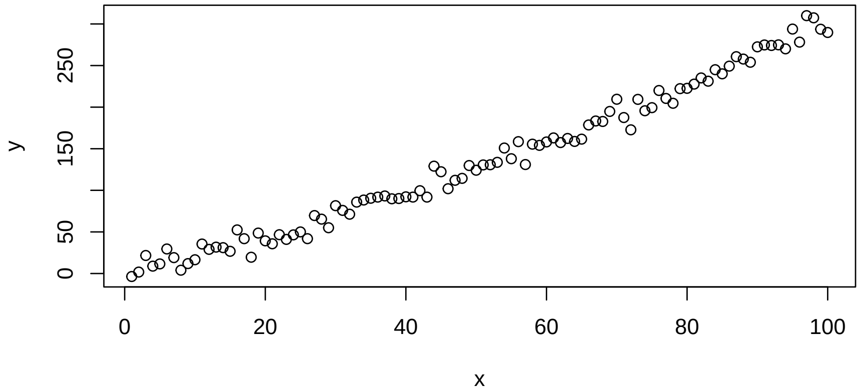
1.1.1 Costruzione del modello

Costruiamo ad arte un esperimento ad un solo fattore per dimostrare l'uso di `lm()` per costruire modelli di regressione.

Assumiamo che la variabile dipendente y sia correlata ad un unico regressore x in modo quadratico, ma con un contributo relativamente modesto del termine di secondo grado (piccola curvatura). Costruiamo un data frame che raccoglie i valori di y per una sequenza di valori del regressore secondo la relazione $y(x) = 2x + 0.01x^2$, ed aggiungiamo un disturbo normale a media nulla:

```
set.seed(123)
N <- 100
```

```
df <- data.frame(x=1:N)
df$yn <- 2*df$x + 0.01*df$x^2   # nominale
df$y <- df$yn + rnorm(N, 0, 10) # nominale + disturbo
# randomizziamo l'ordine di raccolta dei dati!
df$run <- sample(N)
df <- df[order(df$run),]
plot(y~x, data=df, typ="p")
```



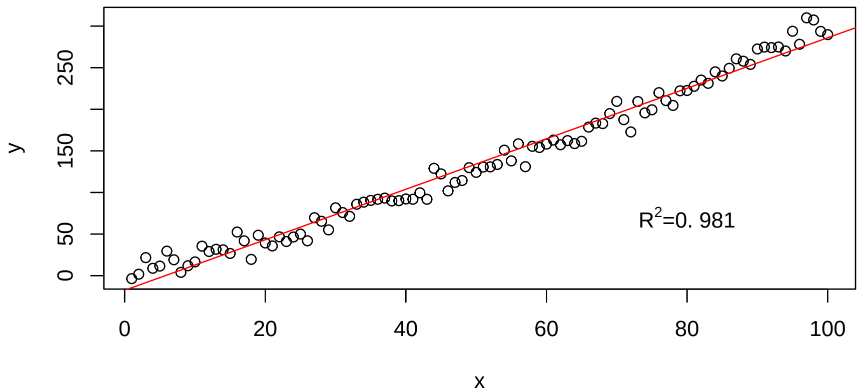
A questo punto, supponiamo di *non conoscere* la relazione nominale con la quale abbiamo generato i dati: nel nostro esperimento concettuale i dati derivano direttamente da delle misurazioni di un fenomeno la cui fisica ci è ignota.

Osservando i dati, ipotizziamo una relazione lineare del tipo $y(x) = a + bx$ e costruiamo un modello di conseguenza, usando la formula `y~x`

```
df.lm <- lm(y~x, data=df)
(df.lm.s <- summary(df.lm)) # per estrare R^2
```

```
##
## Call:
## lm(formula = y ~ x, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.246  -9.763  -0.661   7.859  33.092
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.53404    2.50538  -6.999 3.24e-10 ***
## x             3.03511    0.04307  70.467  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.43 on 98 degrees of freedom
## Multiple R-squared:  0.9806, Adjusted R-squared:  0.9804
## F-statistic:  4966 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
plot(y~x, data=df, typ="p")
abline(df.lm, col="red")    # abline accetta oggetti lm!
text(80, 70, lab=TeX(paste0("$R^2$=", round(df.lm.s$r.squared, 3))))
```



Come si vede, il `summary()` del modello lineare mi fornisce diverse informazioni interessanti, tra le quali i coefficienti del modello di regressione: $y = -17.5340385 + 3.0351108x$, e il coefficiente di regressione $R^2 = 0.981$.

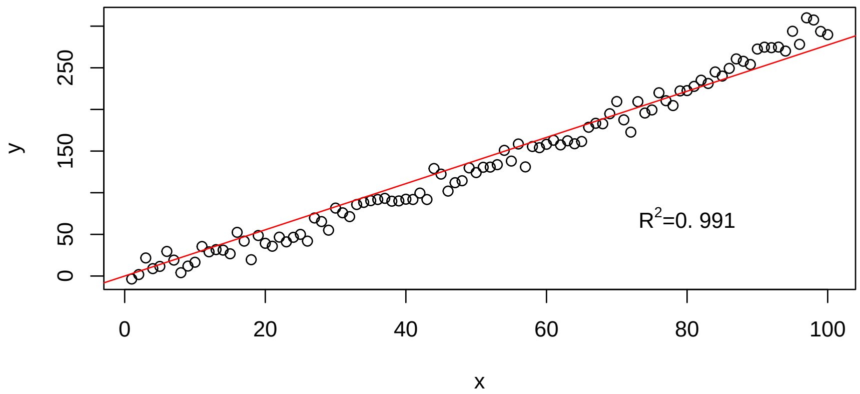
È anche evidente che la formula `y~x` contiene anche un termine di intercetta, che è quindi implicito e corrisponde al termine a nel modello $y(x) = a + bx$.

Dato che il *p-value* dell'intercetta è molto più grande di quello del termine di primo grado, possiamo anche provare a rimuovere l'intercetta e provare con il modello $y(x) = bx$. In termini di formula, si scrive `y~x-1`:

```
df.lm <- lm(y~x-1, data=df)
(df.lm.s <- summary(df.lm))
```

```
##
## Call:
## lm(formula = y ~ x - 1, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -30.348 -14.537  -5.888   4.465  40.943
##
## Coefficients:
##   Estimate Std. Error t value Pr(>|t|)
## x  2.77341    0.02604   106.5   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.15 on 99 degrees of freedom
## Multiple R-squared:  0.9913, Adjusted R-squared:  0.9913
## F-statistic: 1.134e+04 on 1 and 99 DF,  p-value: < 2.2e-16
```

```
plot(y~x, data=df, typ="p")
abline(df.lm, col="red")
text(80, 70, lab=TeX(paste0("$R^2$=", round(df.lm.s$r.squared, 3))))
```



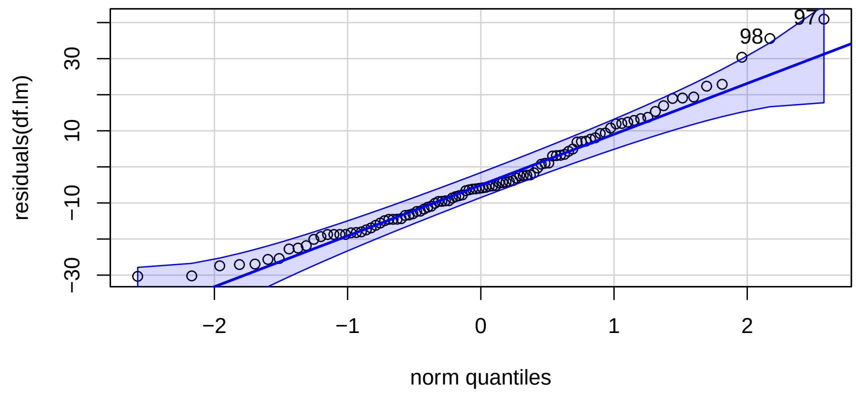
Come si vede, il parametro di regressione R^2 è ulteriormente aumentato, quindi questo secondo modello rappresenta i dati meglio del primo.

Tuttavia non possiamo accettare questo modello senza prima analizzare i residui:

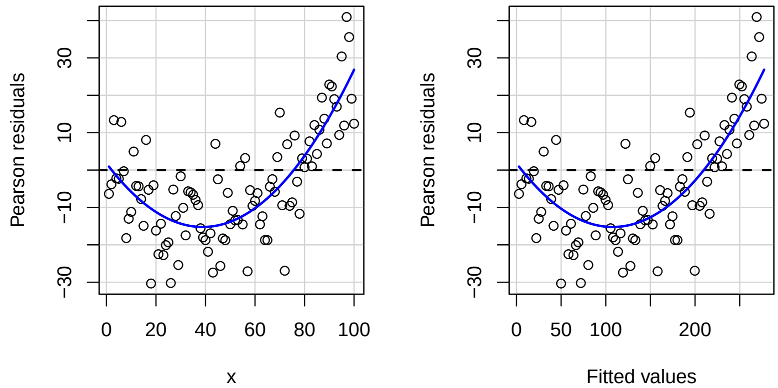
```
shapiro.test(residuals(df.lm))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(df.lm)
## W = 0.97073, p-value = 0.02517
```

```
invisible(qqPlot(residuals(df.lm)))
```




```
residualPlots(df.lm, test=F)
```



Si osserva come, nonostante i residui siano distribuiti normalmente, è evidente un pattern a U degli stessi sia verso il regressore x sia verso i valori fittati.

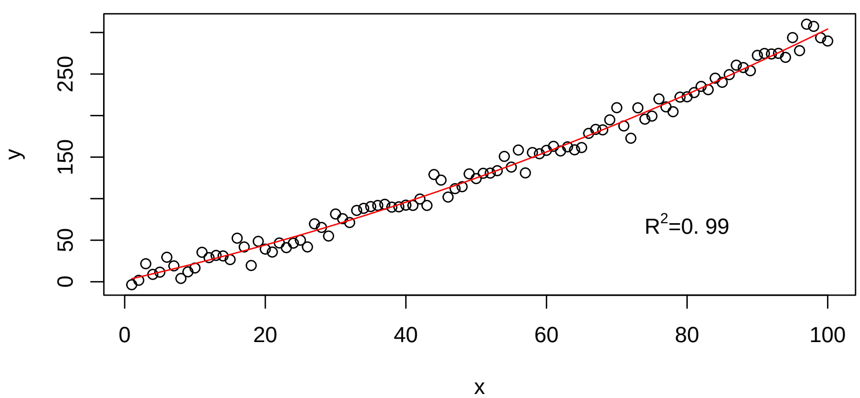
Tipicamente, questo significa che il grado del modello non è sufficiene a descrivere la complessità dei dati: in altre parole, è necessario aumentare il grado del polinomio di regressione.

Dato che la linea blu nell'ultimo grafico attraversa l'origine solo due volte, un modello quadratico del tipo $y(x) = a + bx + cx^2$ dovrebbe essere sufficiente. Costruiamo quindi un modello quadratico, ricordando che secondo l'algebra delle formule di R `A*B = A+B+A:B`, quindi `A^2 = A*A = A+A+A:A = A`. Per esprimere il quadrato si usa quindi la funzione *identità* `I()`:

```
df.lm <- lm(y~x+I(x^2), data=df)
(df.lm.s <- summary(df.lm))
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2), data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24.0680  -5.9884  -0.2665   6.7743  21.9457
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.800042   2.798617   0.643    0.522
## x           1.897812   0.127904  14.838   <2e-16 ***
## I(x^2)      0.011260   0.001227   9.178    8e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.143 on 97 degrees of freedom
## Multiple R-squared:  0.9896, Adjusted R-squared:  0.9894
## F-statistic:  4633 on 2 and 97 DF,  p-value: < 2.2e-16
```

```
plot(y~x, data=df, typ="p")
text(80, 70, lab=TeX(paste0("$R^2$=", round(df.lm.s$r.squared, 3))))
curve(df.lm$coefficients[1]+x*df.lm$coefficients[2] + x^2*df.lm$coefficients[3],
      col="red",
      add=T)
```



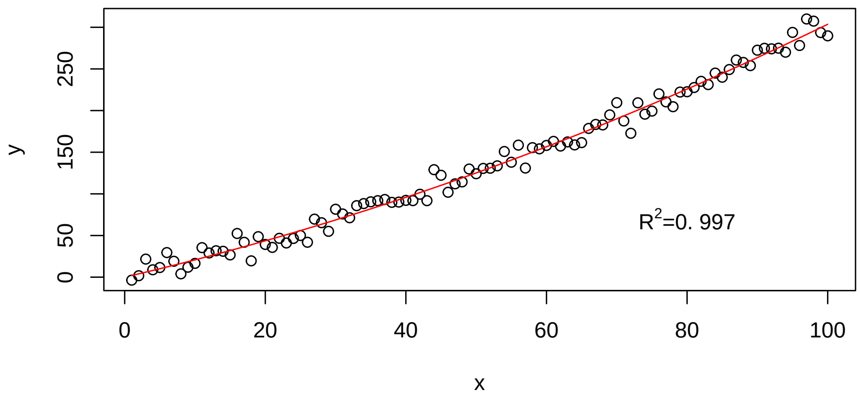
Dato che il *p-value* per l'intercetta è molto alto, possiamo certamente rimuoverla e riformulare il modello come $y(x) = bx + cx^2$:

```
df.lm <- lm(y~x+I(x^2)-1, data=df)
(df.lm.s <- summary(df.lm))
```

```
##
## Call:
```

```
## lm(formula = y ~ x + I(x^2) - 1, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24.3466  -5.7759  -0.4472   7.1411  21.7436
##
## Coefficients:
##         Estimate Std. Error t value Pr(>|t|)
## x      1.9694525  0.0626872   31.42   <2e-16 ***
## I(x^2) 0.0106664  0.0008053   13.24   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.115 on 98 degrees of freedom
## Multiple R-squared:  0.9969, Adjusted R-squared:  0.9968
## F-statistic: 1.575e+04 on 2 and 98 DF,  p-value: < 2.2e-16
```

```
plot(y~x, data=df, typ="p")
text(80, 70, lab=TeX(paste0("$R^2$=", round(df.lm.s$r.squared, 3))))
curve(x*df.lm$coefficients[1] + x^2*df.lm$coefficients[2], col="red", add=T)
```

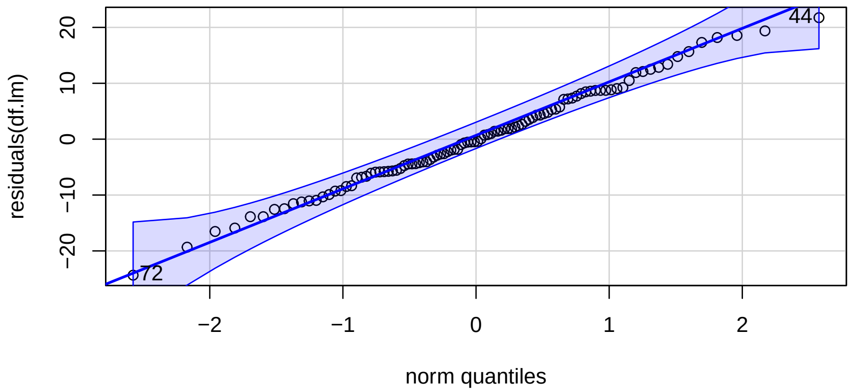


Come vediamo, R^2 è ulteriormente aumentato. Non resta che verificare di nuovo i residui, non riscontrando nessun problema né nella normalità né nella sequenza. **Accettiamo quindi il modello** $y(x) = 1.9694525x + 0.0106664x^2$.

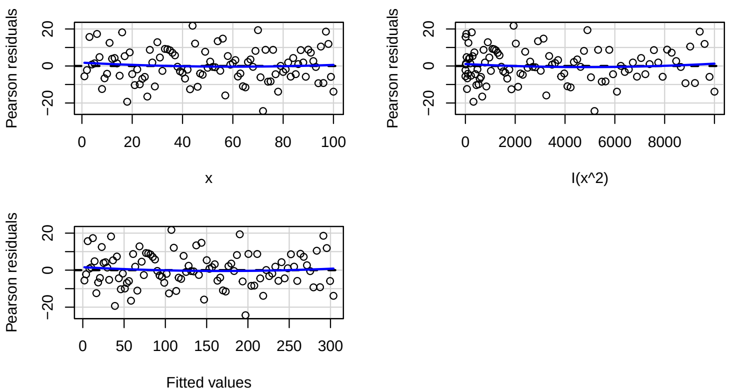
```
shapiro.test(residuals(df.lm))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(df.lm)
## W = 0.99473, p-value = 0.9674
```

```
invisible(qqPlot(residuals(df.lm)))
```




```
residualPlots(df.lm, test=F)
```



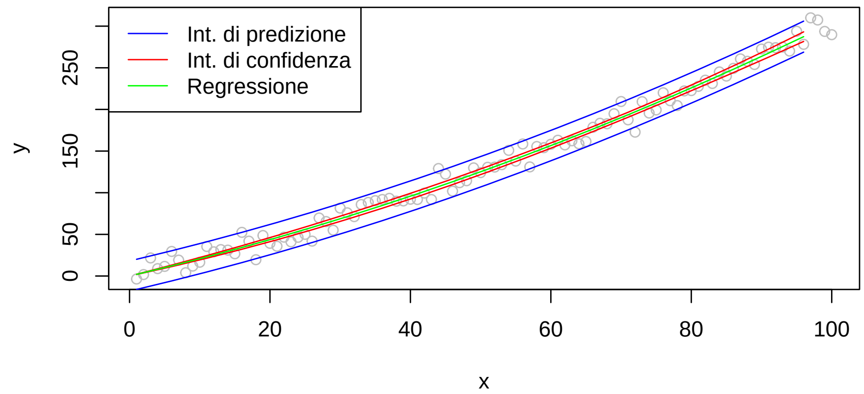
1.1.2 Regressione e predizione

Il modello lineare costruito e verificato al punto precedente può essere utilizzato per effettuare la *regressione* e la *predizione*, termini alternativi a *interpolazione* e *estrapolazione*.

In R, la regressione (inter- e estra-polazione) si effettua con la funzione `predict()`, che vuole come argomenti principali un modello `lm` e un nuovo data frame di regressori su cui valutare il modello, in altre parole una griglia di *nuovi* punti x.

Si noti che `predict()` può restituire anche gli intervalli di predizione e di confidenza, specificando il parametro `interval`:

```
plot(y~x, data=df, typ="p", col="gray")
new <- data.frame(x=seq(1,100,5)) # nuovi predittori
new <- data.frame(new,            # aggiungo a new anche la predizione con c.i.
      conf=predict(df.lm, newdata=new, interval="conf", level=0.99),
      pred=predict(df.lm, newdata=new, interval="pred", level=0.95))
lines(new$x, new$conf.lwr, col="red")
lines(new$x, new$conf.upr, col="red")
lines(new$x, new$pred.lwr, col="blue")
lines(new$x, new$pred.upr, col="blue")
lines(conf.fit~x, data=new, col="green")
legend("topleft",
       legend=c("Int. di predizione", "Int. di confidenza", "Regressione"),
       lty=1,
       col=c("blue", "red", "green"))
```



L'**intervallo di predizione** raccoglie il 95% dei valori osservati. In altre parole, se ripeto l'esperimento con valori del regressore nell'intervallo 0–100 ho il 95% di probabilità che essi cadano in questo intervallo.

L'**intervallo di confidenza**, invece, è l'intervallo all'interno del quale ho il 95% di probabilità che rientri il modello nominale.

2 Regressione logistica

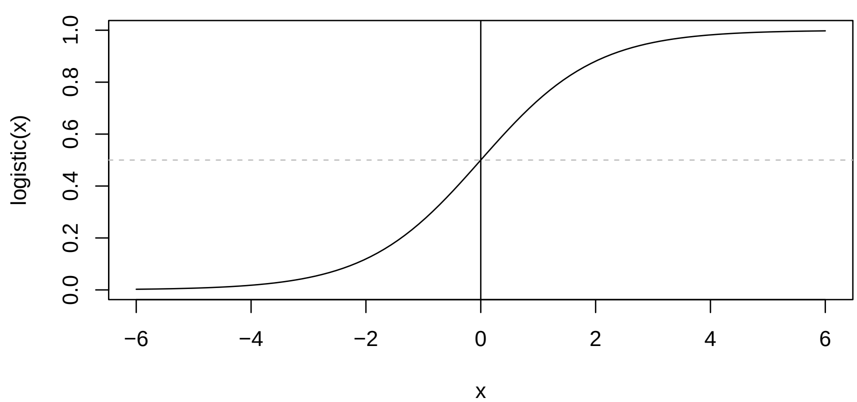
2.1 La funzione logistica

La regressione logistica è utile nei casi in cui si voglia classificare un evento in due categorie (vero/falso, alto/basso, vibo/morto, OK/NO, ecc.) in funzione dei valori assunti da uno o più regressori. Si tratta di uno dei metodi più semplici per realizzare dei *classificatori* e fa parte, di conseguenza, delle tecniche di *machine learning* (ML).

L'idea di base è effettuare la regressione dei dati mediante un modello logistico, cioè basato sulla *funzione logistica*:

$$f(x) = \frac{1}{1 + e^{-p(x-x_0)}}$$

La funzione logistica è una funzione sigmoidale che assume valori nell'intervallo $(0, 1)$; il parametro x_0 è il regressore per cui la funzione assume il valore di 0.5, e il parametro p è la pendenza del tratto di transizione. Il grafico della funzione è:



L'idea è che i regressori per cui la funzione logistica vale meno di 0.5 appartengano alla prima classe (bassa) e quelli per cui vale più di 0.5 appartengano alla seconda classe (alta).

2.2 Esempio

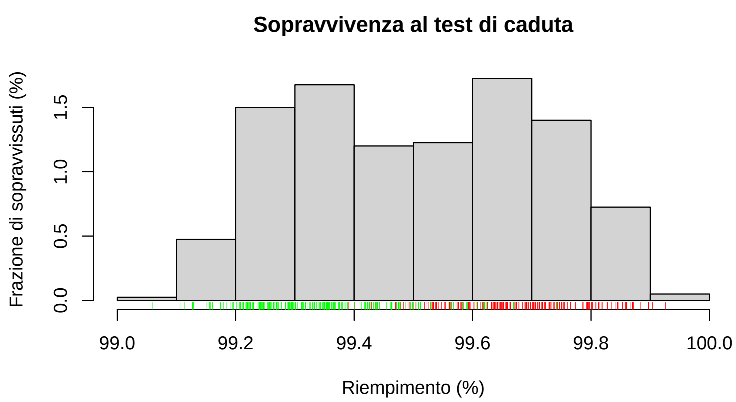
Vogliamo prevedere la sopravvivenza di una bottiglia di sapone liquido al test di caduta, in funzione del livello di riempimento. Più la bottiglia è piena, infatti, più piccolo è il polmone di aria che contiene e, quindi, maggiore la sobrapressione in caso di caduta. Ovviamente il problema è stocastico, dato che le bottiglie non sono tutte uguali e il riempimento stesso ha una certa variabilità.

Abbiamo un data frame che contiene i risultati di 400 diversi test di caduta per bottiglie riempite a differenti livelli rispetto alla capacità nominale. Cominciamo caricando e visualizzando i dati con un istogramma del livello di riempimento:

```
data <- read.table("http://repos.dii.unitn.it:8080/data/soap_bottles.txt", h=T)
knitr::kable(head(data))
```

run	p	OK
1	99.314	TRUE
2	99.357	TRUE
3	99.624	TRUE
4	99.429	FALSE
5	99.860	FALSE
6	99.212	TRUE

```
hist(data$p, probability=T,
     xlab="Riempimento (%)",
     ylab="Frazione di sopravvissuti (%)",
     main="Sopravvivenza al test di caduta")
rug(data[!data$OK,]$p, col="red")
rug(data[data$OK,]$p, col="green")
```



Si noti la cosiddetta “stuoia” (*rug*) subito sopra le ascisse: essa riporta le reali osservazioni, in verde i sopravvissuti e in rosso i morti. È evidente che l’istogramma rappresenta due popolazioni, e che c’è una sovrapposizione tra la popolazione dei sopravvissuti e quella dei morti. In altre parole, non c’è un limite netto al livello di riempimento che discrimina nettamente le due classi; piuttosto, siamo interessati a identificare un limite che minimizzi il numero di errori di classificaizone.

Questa è la tipica situazione adatta ad una regressione logistica.

2.2.1 Preparazione dei dati

Come in tutte le applicazioni di ML, è opportuno dividere i dati in due sottoinsiemi: uno (tipicamente più grande) da utilizzare per l’addestramento, o la messa a punto del modello di regressione; l’altro, complementare, verrà poi utilizzato per *validare* il modello, cioè per verificarne l’affidabilità.

La libreria `caTools` mette a disposizione la funzione `sample.split()`, che semplifica la suddivisione:

```
library(caTools)
set.seed(1) # la selezione è casuale!
data$split <- sample.split(data$OK, SplitRatio=0.8) # 80% training set
train <- data[data$split,]
test <- data[!data$split,]
str(data)
```

```
## 'data.frame':    400 obs. of  4 variables:
##  $ run  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ p    : num  99.3 99.4 99.6 99.4 99.9 ...
##  $ OK   : logi  TRUE TRUE TRUE FALSE FALSE TRUE ...
##  $ split: logi  TRUE TRUE TRUE TRUE TRUE FALSE ...
```

2.2.2 Regressione

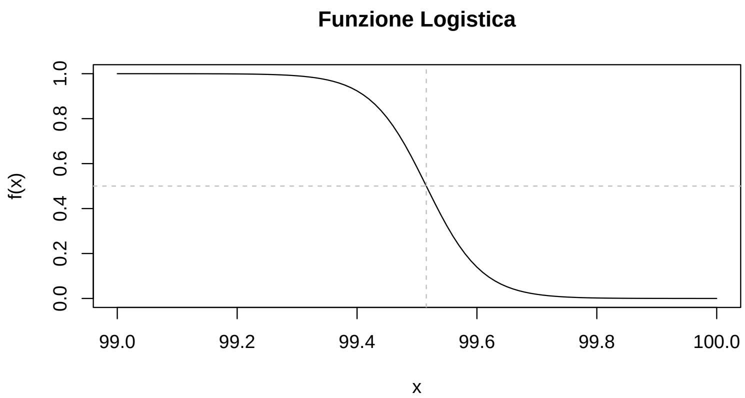
Ora costruiamo il modello logistico, utilizzando la funzione `glm()` (Generalized Linear Model) e specificando l'opzione `family="binomial"`, che corrisponde alla funzione logistica:

```
model <- glm(OK~p, data=train, family="binomial")
summary(model)
```

```
##
## Call:
## glm(formula = OK ~ p, family = "binomial", data = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.35386  -0.21796  -0.00332   0.22669   3.12695
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 2144.954    251.113   8.542   <2e-16 ***
## p            -21.554      2.523  -8.542   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 443.61  on 319  degrees of freedom
## Residual deviance: 133.24  on 318  degrees of freedom
## AIC: 137.24
##
## Number of Fisher Scoring iterations: 7
```

Si noti che `glm()` usa una formulazione della funzione logistica leggermente differente, con $f(x) = 1/(1 + \exp(-px - m))$, ossia dove $m = -px_0$, quindi i valori dell'intercetta m e del coefficiente p sono quelli riportati dall'output di `glm()`. Possiamo quindi visualizzare la funzione:

```
k <- model$coefficients[2]
x0 <- -model$coefficients[1]/model$coefficients[2]
curve(1/(1+exp(-k*(x-x0))),
      xlim=c(99,100),
      ylab="f(x)",
      main="Funzione Logistica")
abline(v=x0, col="gray", lty=2)
abline(h=0.5, col="gray", lty=2)
```

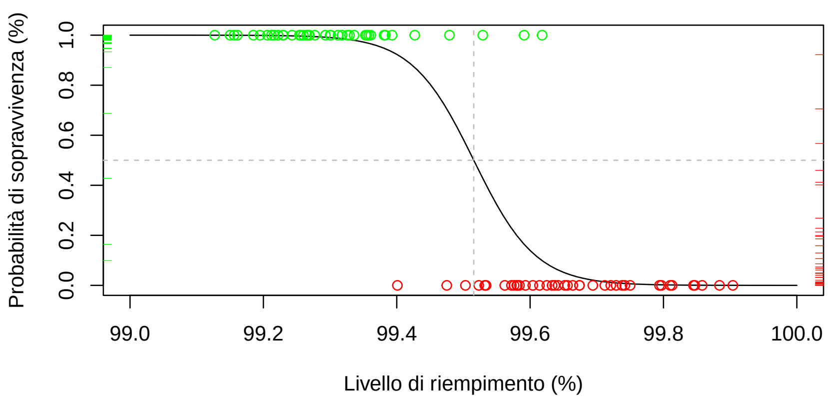


2.2.3 Predizione e validazione

Il modello fittato può essere utilizzato per predire la sopravvivenza al test di caduta: riempimenti inferiori a $x_0 = 99.5$ appartengono alla classe `OK`, gli altri alla classe `FAIL`.

Questa volta, anziché plottare la *funzione* logistica, usiamo la funzione `predict()` per valutarla su un nuovo vettore di regressori, e lo confrontiamo con i dati del set di validazione `test`:

```
lvl <- seq(99,100, 0.01)
plot(lvl, predict(model, newdata=data.frame(p=lvl), type="response"),
     typ="l",
     xlab="Livello di riempimento (%)",
     ylab="Probabilità di sopravvivenza (%)")
points(OK~p, data=test[test$OK,], col="green")
points(OK~p, data=test[!test$OK,], col="red")
# Aggiungiamo anche dei rug verticali per mostrare i valori regrediti
# per le bottiglie sopravvissute (verde) e non (rosso)
test$fit <- predict(model, newdata=test, type="response")
rug(test[test$OK,]$fit, col="green", side=2)
rug(test[!test$OK,]$fit, col="red", side=4)
abline(h=0.5, col="gray", lty=2)
abline(v=x0, col="gray", lty=2)
```



È evidente che si può spostare la soglia di classificazione più in alto o più in basso di 0.5: se la spostiamo più in alto, ad esempio a 0.7, riusciamo a evitare tutte le rotture, ma classificheremo come probabili rotture anche bottiglie che potrebbero sopravvivere. Viceversa, se spostiamo la soglia più in basso eviteremo di scartare bottiglie che potrebbero sopravvivere, ma al prezzo di accettare più bottiglie “deboli”.

In altre parole è un problema di bilanciamento di falsi positivi e falsi negativi. Per capire l'affidabilità del modello è opportuno costruire quindi una **matrice di confusione**, in grado di riassumere falsi positivi, falsi negativi e predizioni corrette (si noti l'uso della funzione `table()`):

```
# in conteggio:
table(Actual=test$OK, Predicted=test$fit>0.5)
```

```
##        Predicted
## Actual  FALSE TRUE
##   FALSE    37    3
##   TRUE      3   37
```

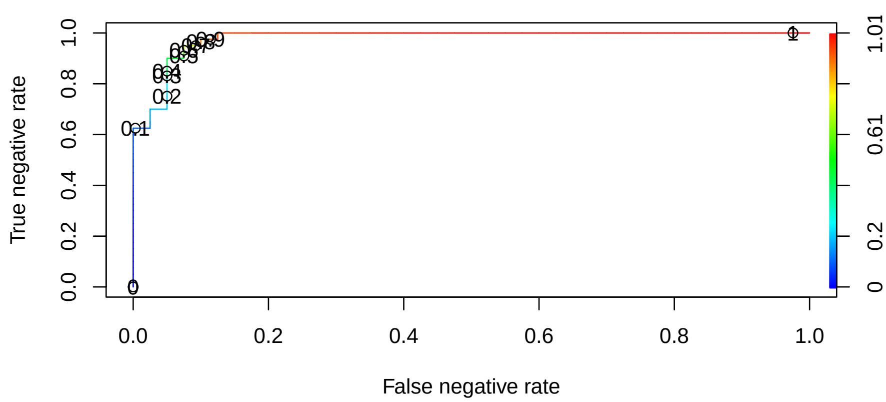
```
# in frazione percentuale:
(ct <- round(table(Actual=test$OK,
                  Predicted=test$fit>0.5)/length(test$OK)*100, 1))
```

```
##        Predicted
## Actual  FALSE TRUE
##   FALSE  46.2  3.8
##   TRUE    3.8 46.2
```

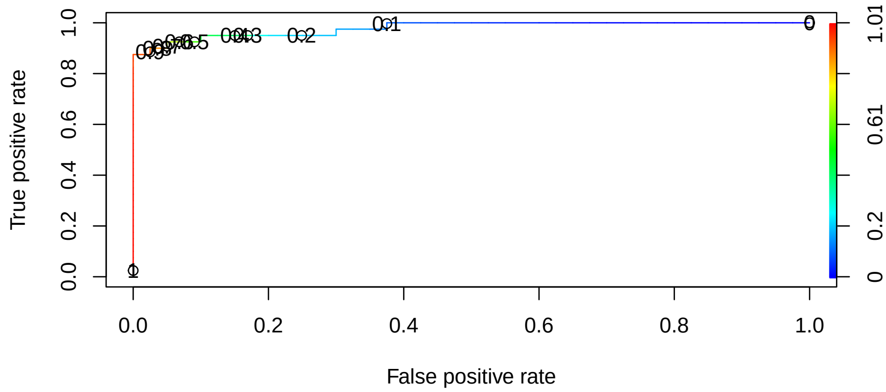
In particolare, osserviamo che con una soglia di classificazione a 0.5 il modello ha una probabilità di falsi positivi (FPR, *false positive rate*) pari a 3.8%, e una probabilità di falsi negativi (FNR) pari a 3.8%. Tipicamente, nei classificatori dove la cifra di merito è la *sopravvivenza* si preferisce abbassare il più possibile FNR a scapito del FPR.

Le funzioni `prediction()` e `performance()` della libreria `ROCR` consentono di calibrare con precisione la soglia di classificazione.

```
library(ROCR)
pred <- prediction(test$fit, test$OK)
perfn <- performance(pred, "tnr", "fnr")
plot(perfn, colorize=T, print.cutoffs.at=seq(0,1,0.1))
```




```
perfp <- performance(pred, "tpr", "fpr")
plot(perfp, colorize=T, print.cutoffs.at=seq(0,1,0.1))
```



Su questi grafici il parametro color rappresenta la soglia di classificazione. Il **primo grafico** mostra che abbassare la soglia è efficace nel ridurre FNR fino a 0.4, poi però non cambia nulla tra 0.4 e 0.2.

Parimenti, dal secondo grafico si osserva che ridurre la soglia sotto 0.4 aumenta sensibilmente FPR. Di conseguenza scegliamo la soglia a 0.4 e ricalcoliamo la matrice di confusione:

```
(ct <- round(table(Actual=test$OK,
                              Predicted=test$fit>0.4)/length(test$OK)*100, 1))
```

```
##        Predicted
## Actual  FALSE TRUE
##   FALSE  42.5  7.5
##   TRUE    2.5 47.5
```

Questo ci consente di ridurre FNR a 2.5%, alle spese di un aumento del FPR a 7.5%.

2.3 Nota finale

L'esempio di regressione logistica sopra riportato può sembrare banale, ma in realtà la potenza del metodo si apprezza meglio quando il modello di regressione è *multivariato*, cioè quando prende in considerazione due o più regressori. In questi casi il modello è del tipo `glm(y~a+b, data=df, family="binomial")`. In questo documento si è scelto il caso univariato perché la visualizzazione è più semplice, ma la stessa tecnica può essere facilmente estesa al caso multivariato.