Librerie in R per le Support Vector Machines e loro utilizzo

Corso di Modellizzazione Statistica, prof. M. Bilancia

Michele Di Nanni, mat. 729187

1.1 Il package *e1071* - caso lineare

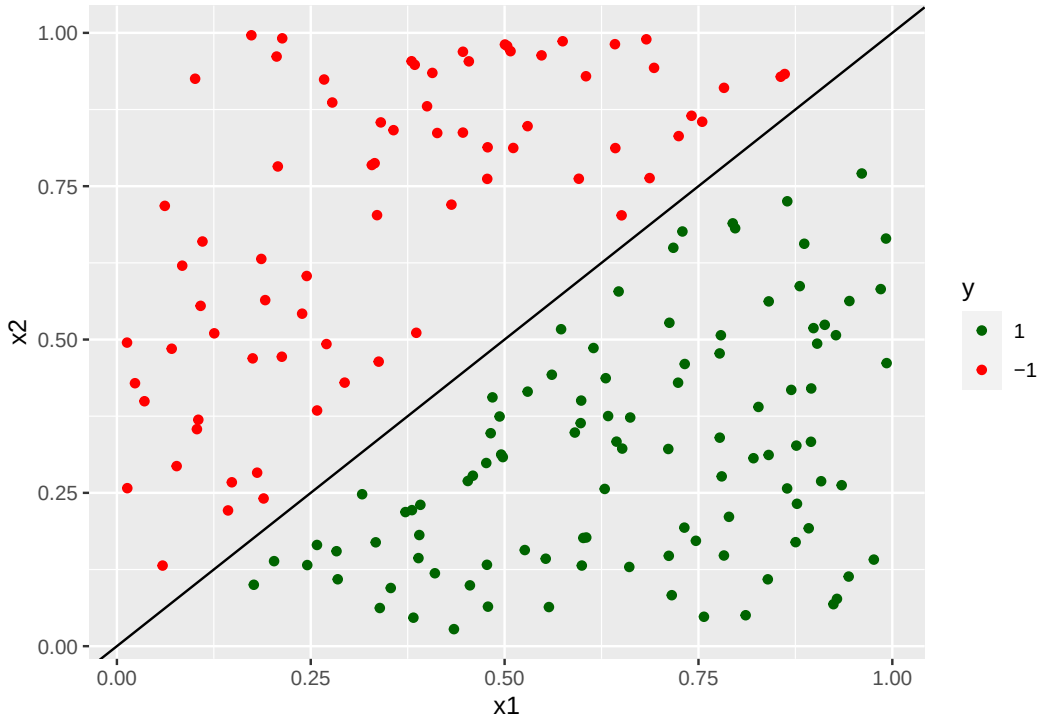
Per poter utilizzare le *support vector machines* in R, possiamo utilizzare il package **e1071**. Se quest'ultimo non è installato sulla propria macchina, procediamo ad installarlo in R tramite il comando:

```
install.packages("e1071")
```

Dopo aver eseguito tale comando, andiamo ad importare il package per poterci lavorare:

```
library(e1071)
library(ggplot2) #libreria che include ggplot2 per la visualizzazione dei dati
```

Procediamo a mostrare un primo utilizzo della libreria, utilizzando un dataset creato in maniera artificiosa ai fini illustrativi. Il task che andremo a considerare adesso sarà quello di separabilità lineare a due classi:



Ora procediamo ad effettuare uno splitting train/test con percentuale 75% / 25%:

```
library(caret)
set.seed(1)
inTrain <- createDataPartition(y = df_thresholded$y, p = .75, list = FALSE)
training <- df_thresholded[ inTrain, ]
testing <- df_thresholded[ -inTrain, ]
```

A questo punto procediamo ad utilizzare il metodo **svm** della libreria *e1071*, con kernel lineare. Il metodo **svm** contiene differenti parametri, di seguito sono spiegati uno ad uno quelli utilizzati:

1. **formula**: ci permette di specificare variabili indipendenti e dipendenti
2. **data**: si riferisce ai dati passati in input per la classificazione
3. **type**: attraverso questo parametro possiamo settare il tipo di classificazione/regressione desiderata; difatti il parametro può assumere valore *C-classification* (come nel seguente esempio), *nu-classification* per indicare la classificazione basata sul parametro $\nu \in [0, 1]$, *one-classification* per effettuare rilevamento di *outlier*, *eps-regression* e *nu-regression* per quanto riguarda tasks di regressione
4. **kernel**: ci permette di impostare quale tipo di kernel vogliamo utilizzare(in questo caso stiamo utilizzando un kernel lineare); altri valori del parametro che denotano un kernel valido sono *poly* (kernel polinomiale), *radial*, *sigmoid*
5. **cost**: attraverso tale parametro possiamo impostare il livello di complessità del modello(infatti, se quest'ultimo coincide con un valore molto basso potremmo ricadere in *underfitting*, se invece abbiamo una valore molto alto potremmo ricadere in *overfitting*)
6. **scale**: indica se occorre riscalare o meno le variabili

```
svm_model <- e1071::svm(formula = y~., data = training, type = "C-classification",
              kernel = "linear", cost = 1, scale = F)

summary(svm_model)
```

```
Call:
svm(formula = y ~ ., data = training, type = "C-classification",
    kernel = "linear", cost = 1, scale = F)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  51

 ( 25 26 )


Number of Classes:  2

Levels:
 1 -1
```

Da questo output possiamo osservare di quanto, con un valore di C pari a 1 abbiamo un numero pari a 51 vettori di supporto.

Visualizziamo adesso i primi sei vettori di supporto:

```r
head(svm_model$SV)
```

```
          x1        x2
2  0.3721239 0.2186453
3  0.5728534 0.5167968
31 0.4820801 0.3472307
33 0.4935413 0.3744869
37 0.7942399 0.6894134
42 0.6470602 0.5783539
```

A questo punto andiamo a valutare la accuratezza del modello sul test set:

```r
#accuratezza sul test set valutata
pred = predict(svm_model, testing)
table(predicted = pred, true = testing$y)
```

```
         true
predicted  1 -1
       1  25  0
       -1  0 18
```

Possiamo notare di come la accuratezza sia pari a 1. Non abbiamo errori di predizione in questo caso molto idealizzato.

Un metodo molto utile che ci permette di comprendere quali siano i parametri migliori da settare per il modello è il metodo *tune()* che effettua di default una *10-fold cross validation*. Tale metodo ci permette di visualizzare anche quale sia il modello migliore ottenibile, attraverso:

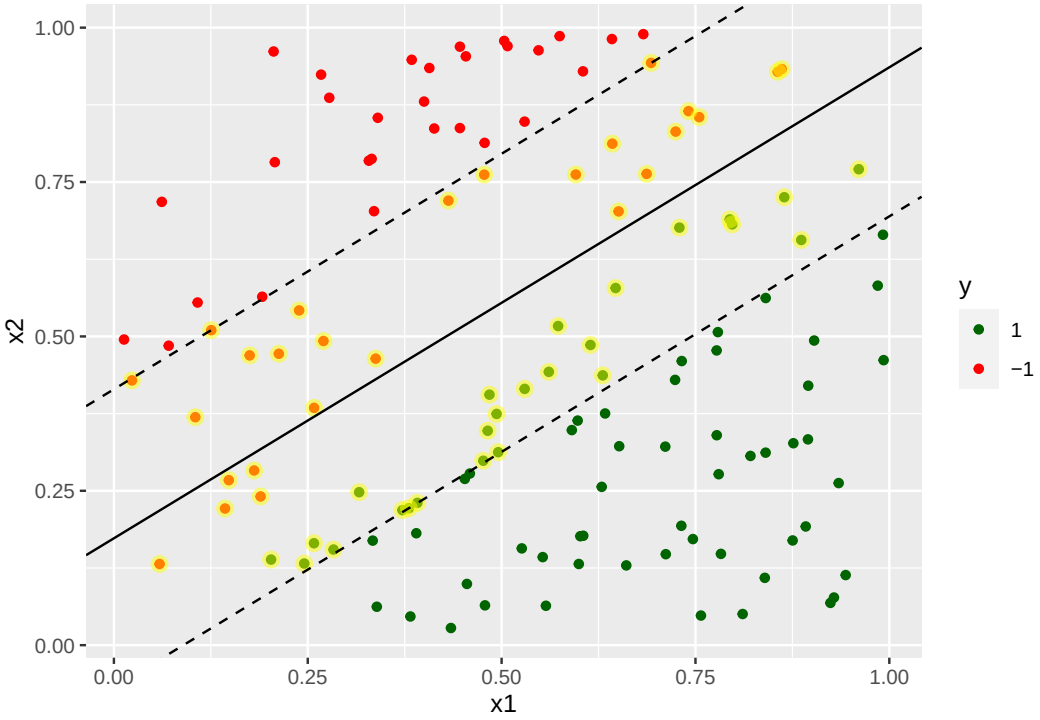
```r
set.seed(1)
# Imposto C = 10 e ottengo che il valore migliore di C = 1
tune_out_train = tune.svm(
                y~.,
                data = training,
                kernel = "linear",
                scale = F,
                C = 10)
summary(tune_out_train$best.model)
```

```
Call:
best.svm(x = y ~ ., data = training, kernel = "linear", scale = F,
    C = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  51

 ( 25 26 )
```

```
Number of Classes:  2

Levels:
 1 -1
```

Come possiamo evincere da quanto ottenuto, il miglior modello prevede il parametro di costo C pari ad 1.

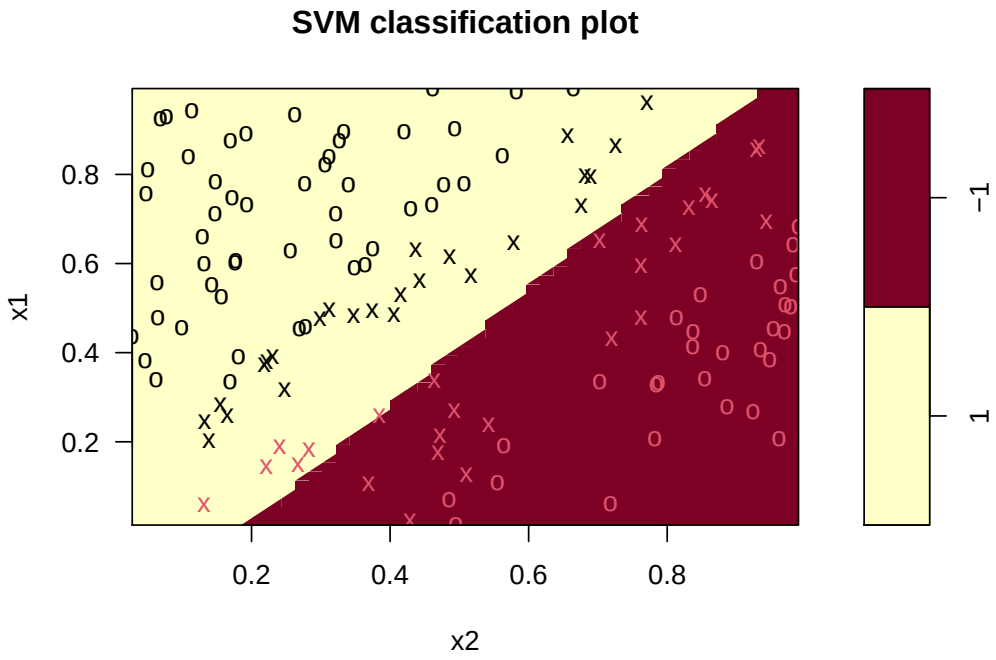
Per poter rappresentare la retta di separazione ottimale, occorre calcolare i parametri $\mathbf{w}$ e w_0, rispettivamente la pendenza(slope) e l' intercetta, in quanto la retta sarà definita come: $y = \mathbf{w}^T x + w_0$.



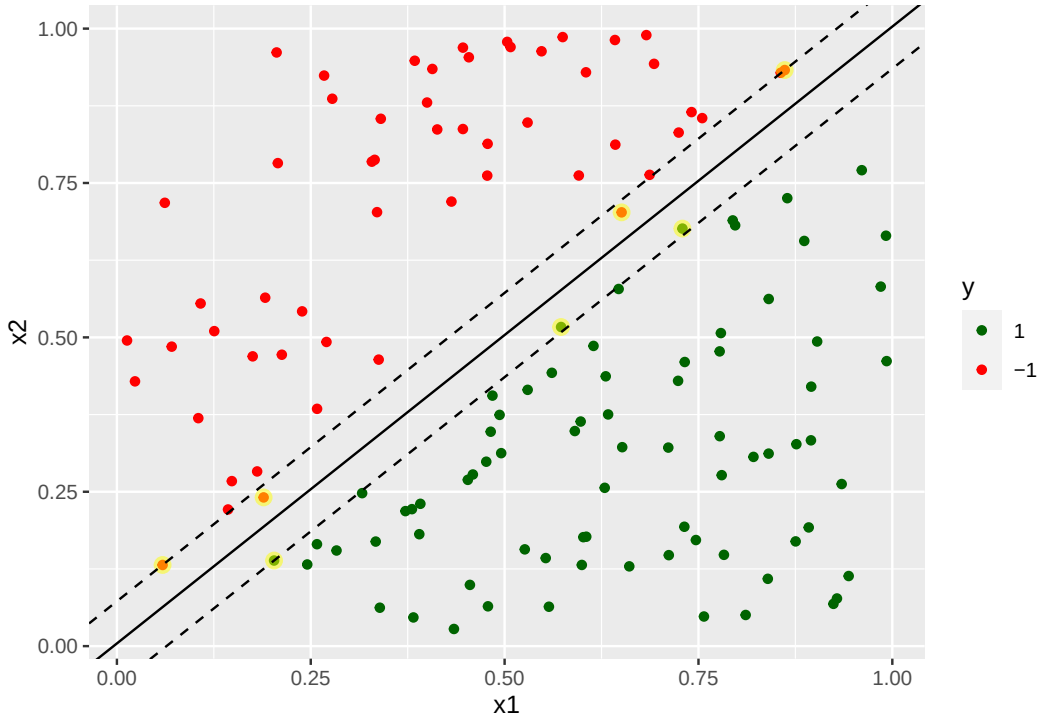
I punti cerchiati in giallo corrispondono ai vettori di supporto.

Un' altra modalità che ci permette di visualizzare i dati è tramite la funzione *plot()*

```
plot(svm_model, training)
```



Aumentando il grado di complessità $C = 100$ possiamo notare di come otterremo meno vettori di supporto, ma tale parametro non dev'essere settato a valori troppo alti, in quanto si potrebbe ricadere in *overfitting*:



1.2 Il package *e1071* - caso non lineare

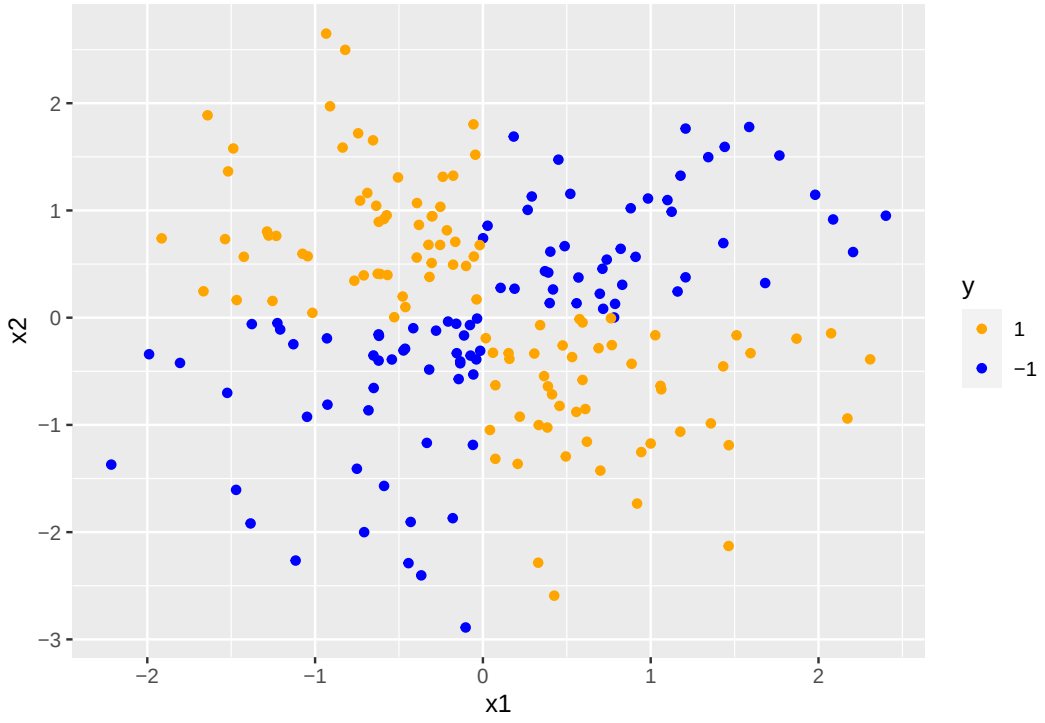
In questa sezione vedremo come utilizzare il metodo *svm()* della libreria *e1071* nel caso in cui non abbiamo separabilità lineare.

Generiamo 200 punti da una normale con media nulla e varianza unitaria e utilizziamo la funzione *xor logico*, ovvero per ogni coppia di punti generata in maniera aleatoria calcolo il risultato in questo modo:

$$y = \begin{cases} +1, & \text{se } (x_1 > 0) \oplus (x_2 > 0) \\ -1, & \textit{altrimenti} \end{cases}$$

In seguito visualizziamo i dati attraverso *ggplot2*.

```
n <- 200 #numero di punti nel dataset
set.seed(1) #set del seme di partenza della generazione casuale

# Genero il dataframe con le due variabili x1 e x2, uniformemente distribuite ()
df_ns <- data.frame(x1 = rnorm(n), x2 = rnorm(n))

# Creo la variabile y in maniera non lineare, usando lo xor logico
df_ns$y <- factor(ifelse(xor(df_ns$x1>0, df_ns$x2>0), 1, -1), levels = c(1, -1))

plotting_notseparable <- ggplot(data = df_ns, aes(x = x1, y = x2, color = y)) +
        geom_point() +
        scale_color_manual(values = c("1" = "orange", "-1" = "blue"))
plotting_notseparable
```



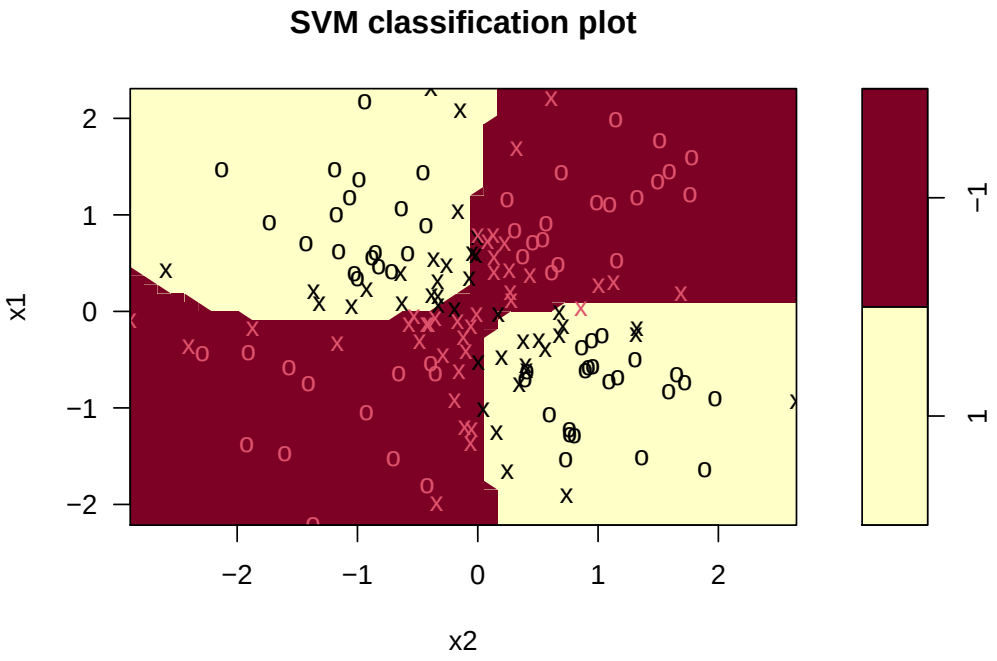
Osserviamo di come non sia possibile dividere le istanze del dataset in maniera lineare, in quanto le non abbiamo separabilità lineare. Procediamo con il partizionare il nostro dataset in training e test sempre con la stessa percentuale (75%/25%) e ad applicare un modello lineare con $C = 100$:

```
     1 -1
  1  26 23
  -1  0  0
```

Come possiamo evincere, l'accuratezza non è molto alta nel test set(circa il 53% (26/49)): notiamo la presenza di 23 istanze che non sono state classificate correttamente.

Procediamo ad utilizzare un kernel *radiale* su questo dataset per verificare di come l'accuratezza predittiva abbia un valore maggiore rispetto al caso precedente.

```
# Il parametro gamma ci permette di settare la forma dei "picchi"
# relativi alle protuberanze nel nuovo spazio di dimensione
# superiore: difatti se gamma ha valore grande avremo delle
# protuberanze più ampie e sinuose, con un gamma basso più
# appuntite
svm_model <- svm(formula = y~., data = training,
                 type = "C-classification", kernel = "radial")
plot(svm_model, training)
```



Andiamo a calcolare l'accuratezza predittiva e la matrice di confusione:

```
pred = predict(svm_model, testing)
table(predicted = pred, true = testing$y)
```

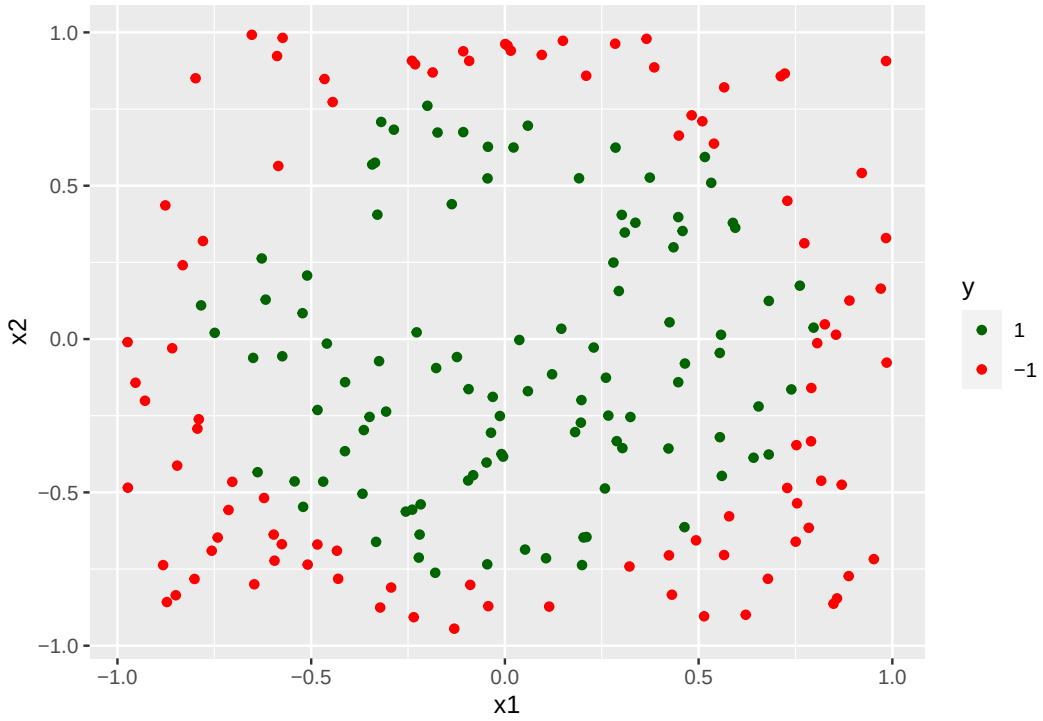
```
         true
predicted  1 -1
       1  26  2
       -1  0 21
```

Possiamo notare di come l'accuratezza predittiva sul test set sia notevolmente maggiore rispetto al caso precedente(lineare). In questo caso è circa il'96% : $[(26 + 21)/49] \cdot 100$.

1.3 Il package *e1071* - dati distribuiti in maniera circolare

Considero un dataset creato allo stesso modo in maniera artificiosa, in cui i dati sono disposti in maniera circolare. La variabile di output sarà definita come segue:

$$y = \begin{cases} +1, & \text{se } \mathbf{x}_1^2 + x_2^2 < raggio^2 \\ -1, & altrimenti \end{cases}$$



Procediamo a questo punto ad applicare 3 differenti tipologie di kernels:

1. Kernel polinomiale
2. Kernel sigmoidale
3. Kernel radiale

e ne valutiamo l'accuratezza predittiva.

Prima di tutto, come al solito, occorre partizionare in training e test set(75%/25%).

Ora possiamo andare ad applicare i tre modelli su dati di training, scegliendo il grado 2 per il polinomio e i valori di default dei parametri *cost, gamma, coef0*:

```
svm_polynomial <- svm(formula = y~., data = training, type = "C-classification",
              kernel = "poly", degree = 2) #grado due del polinomio
svm_sigmoidal <- svm(formula = y~., data = training, type = "C-classification",
              kernel = "sigmoid")
svm_radial <- svm(formula = y~., data = training, type = "C-classification",
              kernel = "radial")
```

Nota:

I valori di default sono: cost = 1, gamma = 0.5, coef0 = 0

Adesso possiamo passare a valutare l'accuratezza predittiva su dati di training e di testing. Visualizziamo inoltre anche la matrice di confusione.

Predizione con il modello basato su kernel polinomiale

```
test_results <- predict(svm_polynomial, newdata = training)
confusionMatrix(test_results, training$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  1 -1
        1  75  5
        -1  3 67

               Accuracy : 0.9467
                 95% CI : (0.8976, 0.9767)
    No Information Rate : 0.52
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.893

 Mcnemar's Test P-Value : 0.7237

            Sensitivity : 0.9615
            Specificity : 0.9306
         Pos Pred Value : 0.9375
         Neg Pred Value : 0.9571
             Prevalence : 0.5200
         Detection Rate : 0.5000
   Detection Prevalence : 0.5333
      Balanced Accuracy : 0.9460

       'Positive' Class : 1
```

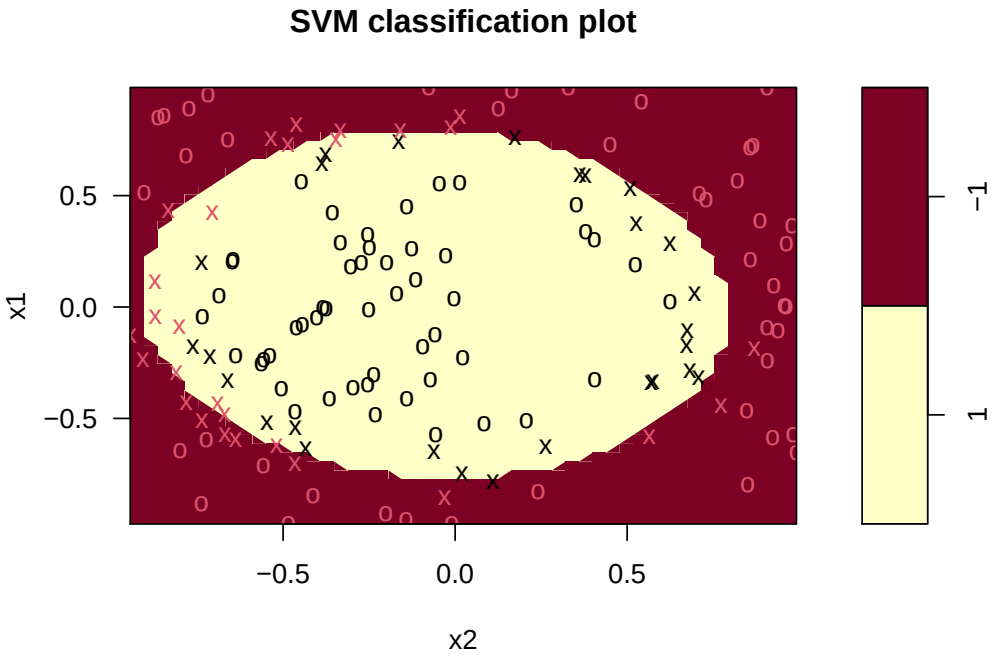
```
test_results <- predict(svm_polynomial, newdata = testing)
confusionMatrix(test_results, testing$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  1 -1
        1  23  3
        -1  3 21

               Accuracy : 0.88
                 95% CI : (0.7569, 0.9547)
    No Information Rate : 0.52
    P-Value [Acc > NIR] : 7.217e-08

                  Kappa : 0.7596

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8846
```

```
            Specificity : 0.8750
         Pos Pred Value : 0.8846
         Neg Pred Value : 0.8750
             Prevalence : 0.5200
         Detection Rate : 0.4600
   Detection Prevalence : 0.5200
      Balanced Accuracy : 0.8798

       'Positive' Class : 1
```

Possiamo visualizzare di come abbiamo ottenuto un accuratezza di circa 95% su dati di train, mentre di 88% su dati di test. Abbiamo solo 6 errori di classificazione errata nel test. Il valore dell'accuratezza è abbastanza buono, usando un polinomio di grado 2. Visualizziamo graficamente i risultati:

```
plot(svm_polynomial, training)
```



Vediamo ora quali sono i parametri migliori che possono essere usati in questo caso, nel caso in cui si voglia usare un kernel *polinomiale*

```
set.seed(1)
tune_out<- tune.svm(x = training[,-3],
                    y = training[,3],
                    type = "C-classification",
                    kernel = "polynomial",
                    degree = c(1,2,3),
                    cost = 10^(-1:2),
                    gamma = c(0.1,1,10),
                    coef0 = c(0.1, 1, 10))

tune_out$best.parameters$cost
```

```
[1] 1
```

```
tune_out$best.parameters$gamma
```

```
[1] 1
```

```
tune_out$best.parameters$coef0
```

```
[1] 1
```

```
tune_out$best.parameters$degree
```

```
[1] 2
```

I parametri migliori sono *cost = 1*, *gamma = 1*, *coef0 = 1*, *degree = 2*.

Predizione con il modello basato su kernel sigmoidale

Passiamo ad effettuare la predizione su dati di training e su dati di test per quanto riguarda una svm basata su kernel sigmoidale

```
test_results <- predict(svm_sigmoidal, newdata = training)
confusionMatrix(test_results, training$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  1 -1
        1  59 49
        -1 19 23

               Accuracy : 0.5467
                 95% CI : (0.4634, 0.628)
    No Information Rate : 0.52
    P-Value [Acc > NIR] : 0.2839823

                  Kappa : 0.0771

 Mcnemar's Test P-Value : 0.0004368

            Sensitivity : 0.7564
            Specificity : 0.3194
         Pos Pred Value : 0.5463
         Neg Pred Value : 0.5476
             Prevalence : 0.5200
         Detection Rate : 0.3933
   Detection Prevalence : 0.7200
      Balanced Accuracy : 0.5379

       'Positive' Class : 1
```

Accuratezza su dati di testing:

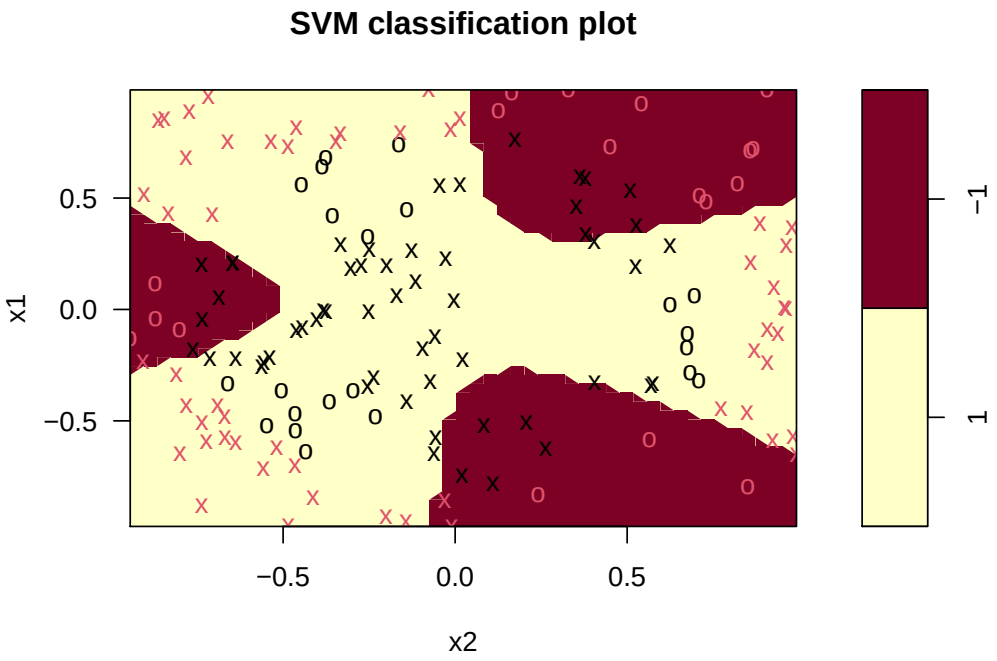
```
test_results <- predict(svm_sigmoidal, newdata = testing)
confusionMatrix(test_results, testing$y)
```

```
Confusion Matrix and Statistics

          Reference
```

```
Prediction  1 -1
        1  18 17
        -1  8  7

               Accuracy : 0.5
                 95% CI : (0.3553, 0.6447)
    No Information Rate : 0.52
    P-Value [Acc > NIR] : 0.6648

                  Kappa : -0.0163

 Mcnemar's Test P-Value : 0.1096

            Sensitivity : 0.6923
            Specificity : 0.2917
         Pos Pred Value : 0.5143
         Neg Pred Value : 0.4667
             Prevalence : 0.5200
         Detection Rate : 0.3600
   Detection Prevalence : 0.7000
      Balanced Accuracy : 0.4920

       'Positive' Class : 1
```

L'accuratezza è circa 50% sui dati di test, pertanto possiamo concludere che il kernel sigmoidale non ci permette di separare al meglio il nostro dataset. Visualizziamo il risultato:

```
plot(svm_sigmoidal, training)
```



Predizione con il modello basato su kernel di base radiale

Passiamo ad effettuare la predizione su dati di training e su dati di test per quanto riguarda il vsm basato su kernel radiale

```
train_results <- predict(svm_radial, newdata = training)
confusionMatrix(train_results, training$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  1 -1
        1  72  2
        -1  6 70

               Accuracy : 0.9467
                 95% CI : (0.8976, 0.9767)
    No Information Rate : 0.52
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8934

 Mcnemar's Test P-Value : 0.2888

            Sensitivity : 0.9231
            Specificity : 0.9722
         Pos Pred Value : 0.9730
         Neg Pred Value : 0.9211
             Prevalence : 0.5200
         Detection Rate : 0.4800
   Detection Prevalence : 0.4933
      Balanced Accuracy : 0.9476

       'Positive' Class : 1
```

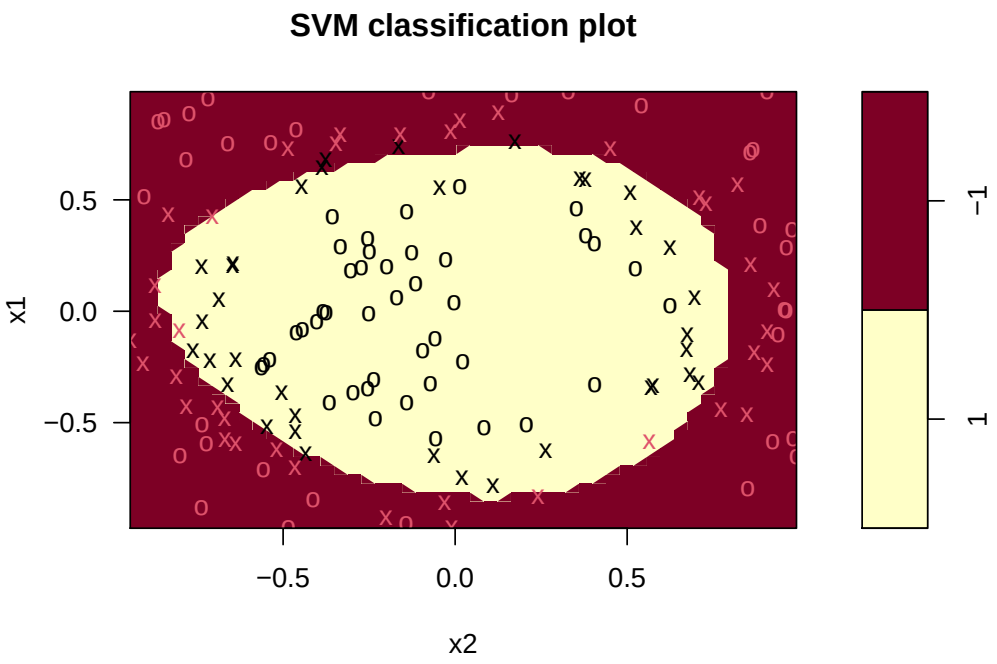
```
test_results <- predict(svm_radial, newdata = testing)
confusionMatrix(test_results, testing$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  1 -1
        1  25  2
        -1  1 22

               Accuracy : 0.94
                 95% CI : (0.8345, 0.9875)
    No Information Rate : 0.52
    P-Value [Acc > NIR] : 1.042e-10

                  Kappa : 0.8796

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9615
            Specificity : 0.9167
```

```
         Pos Pred Value : 0.9259
         Neg Pred Value : 0.9565
             Prevalence : 0.5200
         Detection Rate : 0.5000
   Detection Prevalence : 0.5400
      Balanced Accuracy : 0.9391

       'Positive' Class : 1
```

Attraverso un kernel radiale riusciamo ad avere una accuratezza di circa il 95%, il che indica che con il kernel gaussiano riusciamo ad avere una quasi ottima separazione. Passiamo alla visualizzazione:

```
plot(svm_radial, training)
```



1.4 Il package *Kernlab*

Un'altra modalità con cui poter lavorare con le *svm* in R è utilizzare il package *Kernlab*, al cui interno troviamo il metodo **ksvm**, il quale supporta la classificazione basata sul parametro di costo C, quella basata sul parametro $\nu \in [0, 1]$, la regressione basata su ϵ e su ν ed infine la classificazione multiclasse.

Utilizziamo il package e lavoriamo sul dataset *diabetes indiani Pima* :

```
library(kernlab)
#importo il dataset indiani Pima
library(MASS)
data(Pima.te)
str(Pima.te)
```

```
'data.frame':   332 obs. of  8 variables:
 $ npreg: int  6 1 1 3 2 5 0 1 3 9 ...
 $ glu  : int  148 85 89 78 197 166 118 103 126 119 ...
```

```
 $ bp   : int  72 66 66 50 70 72 84 30 88 80 ...
 $ skin : int  35 29 23 32 45 19 47 38 41 35 ...
 $ bmi  : num  33.6 26.6 28.1 31 30.5 25.8 45.8 43.3 39.3 29 ...
 $ ped  : num  0.627 0.351 0.167 0.248 0.158 0.587 0.551 0.183 0.704 0.263 ...
 $ age  : int  50 31 21 26 53 51 31 33 27 29 ...
 $ type : Factor w/ 2 levels "No","Yes": 2 1 1 2 2 2 2 1 1 2 ...
```

Il dataset è costituito da 7 variabili di input e da 1 variabile di output indicante la presenza o meno del diabete.

Procediamo col partizionare in training e test set, utilizzando questa volta una percentuale 80%/20%:

```
set.seed(1)
inTrain <- createDataPartition(y = Pima.te$type, p = .80, list = FALSE)
training_pima <- Pima.te[ inTrain, ]
testing_pima <- Pima.te[ -inTrain, ]
```

Procediamo con applicare il modello *svm* con un kernel lineare, lavorando con *ksvm*:

```
set.seed(1)
train_ksvm_pima <- ksvm(data = training_pima,
     type~.,
     C = 1,
     type = "C-svc",
     scaled = TRUE,
     kernel = "vanilladot")
```

```
 Setting default kernel parameters
```

```
predicted <- kernlab::predict(train_ksvm_pima, testing_pima)
confusionMatrix(predicted, testing_pima$type)
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  39  10
       Yes  5  11

               Accuracy : 0.7692
                 95% CI : (0.6481, 0.8647)
    No Information Rate : 0.6769
    P-Value [Acc > NIR] : 0.06937

                  Kappa : 0.4374

 Mcnemar's Test P-Value : 0.30170

            Sensitivity : 0.8864
            Specificity : 0.5238
         Pos Pred Value : 0.7959
         Neg Pred Value : 0.6875
             Prevalence : 0.6769
         Detection Rate : 0.6000
   Detection Prevalence : 0.7538
      Balanced Accuracy : 0.7051
```

```
       'Positive' Class : No
```

Notiamo la presenza di accuratezza di circa il 77%. Procediamo con applicare il modello *svm* con un kernel radiale(*gaussiano*), lavorando sempre con *ksvm*:

```
set.seed(1)
train_ksvm_pima <- ksvm(data = training_pima,
     type~.,
     scaled = TRUE,
     type = "C-svc",
     kernel = "rbfdot"
     )
predicted <- kernlab::predict(train_ksvm_pima, testing_pima)
confusionMatrix(predicted, testing_pima$type)
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  34   9
       Yes 10  12

               Accuracy : 0.7077
                 95% CI : (0.5817, 0.814)
    No Information Rate : 0.6769
    P-Value [Acc > NIR] : 0.3505

                  Kappa : 0.3399

 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 0.7727
            Specificity : 0.5714
         Pos Pred Value : 0.7907
         Neg Pred Value : 0.5455
             Prevalence : 0.6769
         Detection Rate : 0.5231
   Detection Prevalence : 0.6615
      Balanced Accuracy : 0.6721

       'Positive' Class : No
```

Notiamo la presenza di una accuratezza di circa il 71% con tale kernel gaussiano.

Procediamo ad effettuare la classificazione basata su kernel polinomiale, usando tuttavia un modello che è il *ν-classification*, che utilizza il valore di $\nu \in [0, 1]$ piuttosto che C,il quale può assumere un qualunque valore positivo.

```
set.seed(1)
train_ksvm_pima <- ksvm(data = training_pima,
     type~.,
     scaled = TRUE,
     type = "nu-svc",
     kernel = "vanilladot",
     nu = 0.5,
```

```
    kpar=list()
    )
predicted <- kernlab::predict(train_ksvm_pima, testing_pima)
confusionMatrix(predicted, testing_pima$type)
```

```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  39   9
       Yes  5  12

               Accuracy : 0.7846
                 95% CI : (0.6651, 0.8769)
    No Information Rate : 0.6769
    P-Value [Acc > NIR] : 0.03894

                  Kappa : 0.4818

 Mcnemar's Test P-Value : 0.42268

            Sensitivity : 0.8864
            Specificity : 0.5714
         Pos Pred Value : 0.8125
         Neg Pred Value : 0.7059
             Prevalence : 0.6769
         Detection Rate : 0.6000
   Detection Prevalence : 0.7385
      Balanced Accuracy : 0.7289

       'Positive' Class : No
```

Notiamo di come l'accuratezza predittiva sia di circa il 78%.

1.5 Il package *gensvm*

Questo package ci permette di lavorare al meglio nel caso in cui ci troviamo in situazioni *multiclasse*. Procediamo con l'installazione e l'import del package:

```
install.packages("gensvm")

library(gensvm)
```

Utilizzeremo il dataset iris, già disponibile in R. Possiamo già passare al partizionamento in train/test: in questo caso, poichè la libreria mette già a disposizione un metodo per poterlo fare, utilizzeremo tale approccio.

```
set.seed(1)
x <- iris[, -5]
y <- iris[, 5]
split_iris <- gensvm::gensvm.train.test.split(x, y, train.size = .75)
```

Procediamo con l'applicazione del modello SVM, utilizzando un kernel lineare, tramite il metodo *gensvm*:

```
set.seed(1)
fit <- gensvm::gensvm(x=split_iris$x.train,
```

```
               y=split_iris$y.train,
               kernel = "linear",
              )
fit
```

```
Data:
    n.objects: 112
    n.features: 4
    n.classes: 3
    classes: setosa versicolor virginica
Parameters:
    p: 1
    lambda: 1e-08
    kappa: 0
    epsilon: 1e-06
    weights: unit
    max.iter: 1e+08
    random.seed: 570175512
    kernel: linear
Results:
    time: 0.1739224
    n.iter: 4457
    n.support: 13
```

Procediamo con la fase predittiva:

```
pred<- predict(fit, split_iris$x.test)
gensvm::gensvm.accuracy(split_iris$y.test, pred)
```

```
[1] 0.9736842
```

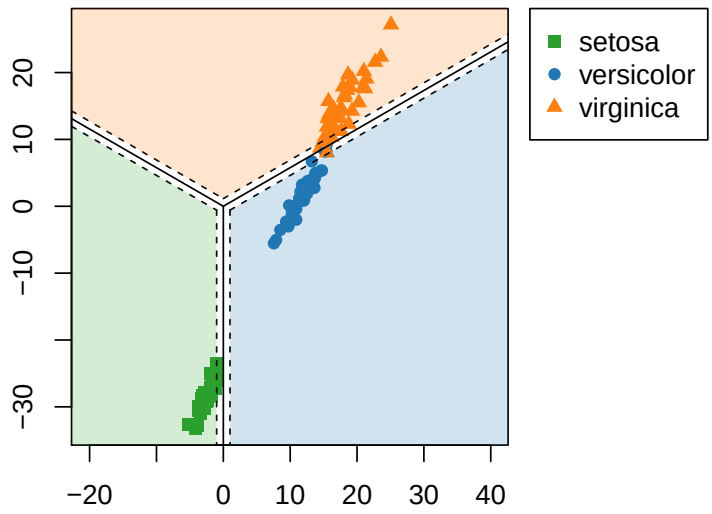
```
table(pred, split_iris$y.test)
```

```
pred         setosa versicolor virginica
  setosa         13          0         0
  versicolor      0         14         1
  virginica       0          0        10
```

Possiamo notare di come la predizione sia molto alta, usando un kernel lineare (97%) sui dati di test.

Passiamo alla visualizzazione dei dati tramite il metodo *plot()*:

```
plot(fit, split_iris$y.train)
```



1.6 Il package *liquidSVM*

Il package *liquidSVM* è estremamente rapido in datasets di grandi dimensioni.

```
library(liquidSVM)
```

Proprio perchè si mostra essere molto veloce in dataset grandi, partiamo con l'utilizzo di un dataset contenente 4000 istanze e 3 attributi(2 indipendenti, 1 dipendente, ovvero di output).

```
gi <- liquidData("banana-mc")
```

Confrontiamo *liquidSVM* con *e1071*, usando un kernel gaussiano.

```
system.time(liquid_train <- liquidSVM::svm(Y ~.,
                                    gi$train,
                                    l=c(2^(-2:2)),
                                    g=c(2^(-2:2)),
                                    threads = 4,
                                    folds = 5))
```

```
   user  system elapsed
 117.84    1.16   37.85
```

```
system.time(a <- e1071::tune.svm(Y~.,
                                 data = gi$train,
                                 scale = F,
                                 gamma = c(2^(-2:2)),
                                 cost = c(2^(-2:2)),
                                 folds = 5,
                                 ))
```

```
   user  system elapsed
 238.57    0.53  264.69
```

Come possiamo evincere dai risultati precedenti, una *5-fold* cross-validation risulta essere molto più rapida con l'utilizzo di *liquidSVM*, piuttosto che con *e1071*.

1.7 Ulteriori packages

Vi sono molte altre librerie in R per le *support vector machines*. Enunciamo le più rilevanti ed utilizzate:

1. ***gkmsvm***: utilizzata in ambito biomedico, in particolare in biologia, per gestire al meglio il parametro k delle cosiddette *k-mer*, (cioè le sottosequenze di lunghezza k contenute in una sequenza genomica). Inoltre, la libreria è stata ideata per questo specifico task, ma comunque può essere estesa a qualunque altro problema di classificazione di sequenze;
2. ***parallelSVM***: utilizzata per ottenere una predizione maggiormente accurata in quanto sfrutta il calcolo in parallelo, indispensabile nel caso in cui ci troviamo a lavorare con *big data*;
3. ***sparseSVM***: utilizzata anche questa in ambito *big data* ed in particolare quando sono presenti matrici di dati sparse;
4. ***svmplus***: implementazione più efficiente delle *svm* per problemi di classificazione
5. ***WeightSVM***: di recente creazione, migliora l'efficienza, poichè sfrutta l'approccio di assegnare pesi diversi a istanze diverse