



Resoconto Tecnico

Riepilogo delle Teorie e delle Leggi empiriche emerse dall'analisi dei dati fatta su algoritmi di Machine Learning, per la realizzazione un'interfaccia comune.

I temi affrontati riguardano metodi di:

Regressione
Classificazione
Random Forest
Clustering



Periodo: 08/06/2020 – 30/07/2020

Indice

1. Premessa.....	3
2. Algoritmi supervisionati.....	3
2.1 Regressione Lineare.....	3
2.1.1 Analisi dei dati con R: Dataset Iris.....	5
2.1.2 Algoritmi di regressione a confronto.....	17
2.2 Support Vector Machines.....	20
2.2.1 Teorema del limite centrale.....	21
2.2.2 Legge dei grandi numeri.....	22
2.2.3 Criterio di minimizzazione del rischio.....	22
2.2.4 Analisi dei dati con R: Dataset Iris.....	26
2.2.5 Algoritmi di classificazione a confronto.....	27
2.2.6 Trasformazioni Box Cox: Dataset Cars.....	33
2.3 Regressione Logistica.....	38
2.4 Linear Discriminant Analysis (LDA).....	41
2.5 Random Forest.....	43
2.5.1 Funzionamento dell'algoritmo.....	45
3. Algoritmi non supervisionati.....	47
3.1 Clustering: il cluster.....	47
3.2 Metodi di clustering.....	47
3.3 Classificazione degli algoritmi di clustering.....	47
3.4 Hierarchical clustering.....	48
3.5 Partitional clustering: analisi degli algoritmi.....	49
3.5.1 K-Means.....	49
3.5.2 Algoritmi a confronto: K-Medoids vrs K-Means.....	51
3.5.3 Algoritmi basati sulla densità: DBSCAN E OPTICS.....	52
DBSCAN.....	52
OPTICS.....	54
I due algoritmi a confronto.....	54
3.5.4 EM: la potenza delle gaussiane.....	55
3.6 Algoritmi Natural Network: SOM.....	56
4. La giusta ricetta per il Machine Learning esiste?.....	57
5. Appendice.....	61
5.1 Analisi con Orange Canvas: Dataset Wine.....	61

1. Premessa

Grazie all'ausilio di software per l'analisi dei dati, come *Orange Canvas* e R, o ancora, grazie alla graduale realizzazione di un'interfaccia comune contenente molteplici algoritmi di *Machine Learning* (supervisionati e non), si è provveduto ad analizzare alcuni set di dati per comprenderne le tendenze e i comportamenti.

Questa attività si è svolta al fine di formulare tesi ed ipotesi che fungessero da supporto ai risultati ottenuti sperimentalmente dall'applicazione degli algoritmi stessi.

Particolare attenzione è stata applicata all'impiego di R nella Regressione Lineare e nelle SVM. Essendo infatti i primi algoritmi affrontati durante il lavoro, e racchiudendo al contempo le caratteristiche di **semplicità** ed efficienza ricercate, le molte informazioni offerte dal software hanno permesso di inquadrare bene molteplici tipologie di problemi, dando importanza al processo:

$$\text{evento reale} \rightarrow \text{ipotesi} \rightarrow \text{legge empirica} \rightarrow \text{teoria/legge}$$

2. Algoritmi supervisionati

2.1 Regressione Lineare

Quando si osserva un fenomeno reale tre sono i passi da svolgere per cercare di costruire un modello che lo descriva a pieno senza errori o fraintendimenti:

- *semplificazione della realtà*: riproduzione degli aspetti essenziali ed eliminazione di quelli ritenuti superflui;
- *analogia con la realtà*: il modello deve essere una riproduzione della realtà;
- *rappresentazione necessaria della realtà*: anche se semplificato il modello risulta necessario per capire la realtà grazie proprio alle relazioni semplici che lo compongono.

Il modello da cui iniziamo è quello della *Regressione lineare*; esso ci fornisce una legge che ci permette di capire se i dati che stiamo osservando si adattano o meno a distribuirsi lungo una retta. Appare chiaro come quindi non sempre tale modello potrà risultare applicabile: ci saranno casi in cui le osservazioni che avremo a disposizione, seguendo un comportamento lineare, vi si adatteranno bene, altri invece meno.

La struttura che presenta la regressione lineare semplice è la seguente:

$$y = ax + b$$

dove:

- y = variabile dipendente (l'output: quello che vogliamo saper predire);
- x = variabili indipendenti (chiamate anche predittori o input);
- a = coefficiente angolare (l'inclinazione della retta);
- b = costante.

Tale algoritmo ha quindi lo scopo di valutare, entro i limiti dei dati osservati, come variabile dipendente e indipendente dipendano o si influenzino fra loro: *quale possa essere il valore della prima al variare della seconda*.

A tale equazione va inoltre aggiunta una certa percentuale di errore (è impensabile non farne) che punta ad essere la minore possibile grazie alla **regola dei minimi quadrati**.

La storia della regressione lineare vede le sue origini tra la fine del '700 e gli inizi del '800 ad opera di Adrien-Marie Legendre e Carl Friedrich Gauss. Sebbene la paternità di tale scoperta venga normalmente attribuita al secondo in realtà essa venne concepita in modo disgiunto da entrambi basandosi, per l'appunto sulla sopra citata regola dei minimi quadrati.

Successivamente l'impiego in tale contesto del termine “*Regressione*”, col quale ancora oggi è conosciuta, si deve grazie al lavoro svolto al biologo Francis Galton che nel 1886 esaminando le altezze dei figli (Y) in funzione di quelle dei genitori (X) vi notò la presenza di una relazione funzionale: più alti erano i genitori e più alti si presentavano i figli, e viceversa. Tuttavia per i genitori che si collocavano agli estremi (molto bassi o molto alti) non corrispondevano figli altrettanto estremi. Da tale osservazione se ne concluse che l'altezza dei figli si spostava verso un valore medio costituendo quindi una *Regression Towards Mediocrity*. Ecco che tale relazione prese il nome di “modello di regressione”.

Nello specifico, la regola dei minimi quadrati sul quale si fonda l'algoritmo, si basa sulla sommatoria delle differenze (chiamate anche scarti o errori), che vi sono fra i vari punti osservati (x_i, y_i) e i loro reali corrispettivi presenti nella retta di regressione $(x_i, \alpha x_i + b)$; tale somma di differenze viene poi elevata al quadrato in modo da enfatizzarne l'effetto su ciascun punto (quelli più vicini alla retta avranno un peso minore, mentre quelli lontani maggiore), per poi individuarne il minimo essendo interessati a trovare la funzione ottima.

La funzione da minimizzare è dunque:

$$\phi(a, b) = \sum_{i=1}^n (y_i - a - bx_i)^2$$

Il grafico a seguire mostra bene quanto appena enunciato.

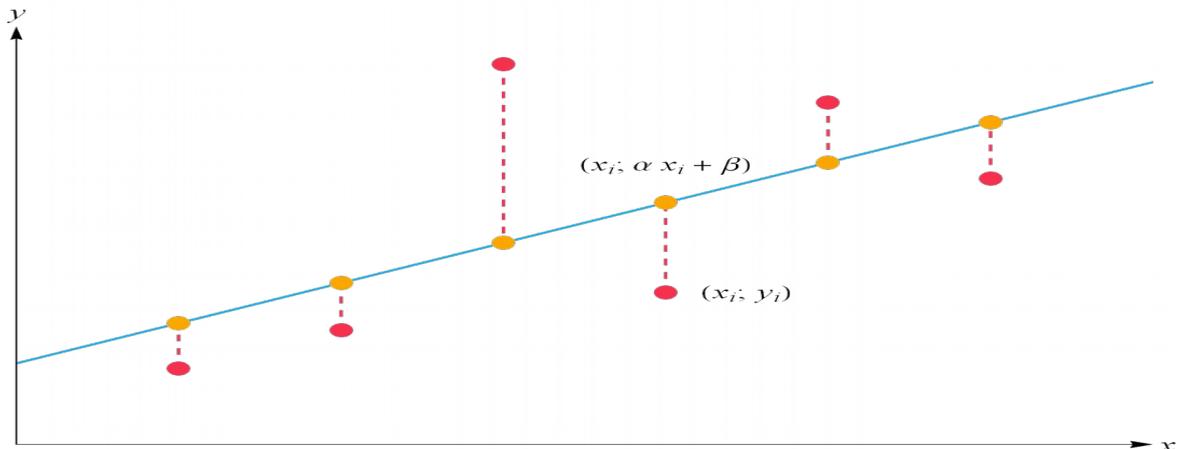


Figura 1: Esempio di retta di Regressione lineare

A questo punto fondamentale risulta porsi una domanda:

Come si valuta la bontà di un modello di regressione lineare?

Molto utile a tal proposito è la misura del **coefficiente di determinazione (R^2)**; esso infatti ci permette proprio di capire quanto buono è il nostro modello, affermando se esso dia informazioni in più o in meno rispetto ad un modello di riferimento, individuato facendo la media dei valori di y . Questo infatti risulta essere il modello di riferimento più adeguato presentando degli errori molto elevati a causa della non conoscenza delle variabili indipendenti.

Nello specifico:

$$R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$$

dove:

- TSS: *devianza totale*, indica quanto spiega il modello di riferimento (calcolato sulla media), rispetto ai valori osservati;
- ESS: *devianza spiegata*, ossia quanto bene spiega il modello ottenuto dalla regressione rispetto al modello di riferimento;
- RSS: *devianza residua*, ciò che non viene spiegato dal modello di regressione realizzato (l'errore).

In particolare si vuole che la devianza totale e quella spiegata siano il più simili possibile, in modo che la quantità di non spiegato sia minima.

Ecco quindi che più tale coefficiente risulta prossimo ad 1 maggiore è la precisione e la bontà con cui spiega i dati, fino ad arrivare ad 1 dove li spiega perfettamente; al contrario un valore di R^2 prossimo allo 0 sta ad indicare una mancanza di adattamento del modello a quanto osservato con conseguente qualità della previsione pessima (non va a spiegare nulla di più di quanto predetto dal modello di riferimento).

Particolare attenzione va comunque posta ai valori troppo elevati di R^2 : essi potrebbero voler significare che diamo troppa fiducia ai dati, con conseguente rischio di overfitting. Ecco che un valore di circa l'85%, con conseguente 0,15 di non spiegato, è già da considerarsi un risultato soddisfacente.

Un altro discorso da affrontare è il **modo** in cui i dati, a cui si applica la regressione, si distribuiscono. Nel caso infatti essi manifestino una forma non propriamente idonea per l'algoritmo (e.g: seguano una distribuzione a parabola invece che a retta), risulta essere molto utile eseguirvi delle trasformazioni, differenti da caso a caso, e solo dopo, applicarvi l'algoritmo di regressione. Nel caso di una forma originaria a parabola si potrebbe, ad esempio, eseguirvi la radice, in modo da annullare l'azione dell'elevazione a potenza; od ancora, per dati eteroschedastici (distribuzione a triangolo), dove la varianza cresce col progredire dell'asse delle ordinate (asse x), caratterizzandosi per la prevalenza delle operazioni di % e * (un animale mangia in percentuale al suo peso, e più è grande maggiore sarà la quantità ingerita), può essere risolta trasformando tali operazioni in somme.

2.1.1 Analisi dei dati con R: Dataset Iris

Domanda:

Osservando la realtà vi si possono individuare alla base regole specifiche che ne spieghino in modo rigoroso il comportamento?

Al fine di comprendere ciò, nel presente documento verranno mostrate molteplici attività di analisi: esse differiranno fra loro per algoritmo trattato, software impiegato e dati esaminati.

In particolare, nel suddetto paragrafo viene presentata l'analisi del *dataset Iris* per la Regressione Lineare; se ne include il codice relativo al linguaggio R, accompagnandolo con le immagini e i commenti dei risultati ottenuti.

```
library(datasets)
data(iris)

help(iris) ##dà informazioni sul dataset
```

```

summary(iris) ## permette di vedere la "tabella" di iris

dim(iris) ## dice che si hanno entry con 5 colonne in totale
n <- nrow(iris) ## assegnazione n=150
n #stampo n
summary(iris$Petal.Width) ## dà informazioni utili sulla variabile (mediana, quartili ecc.)

plot(iris$Petal.Length,iris$Sepal.Width) ## mostra il grafico (x, y): si nota che vi è una buona separazione dei dati

```

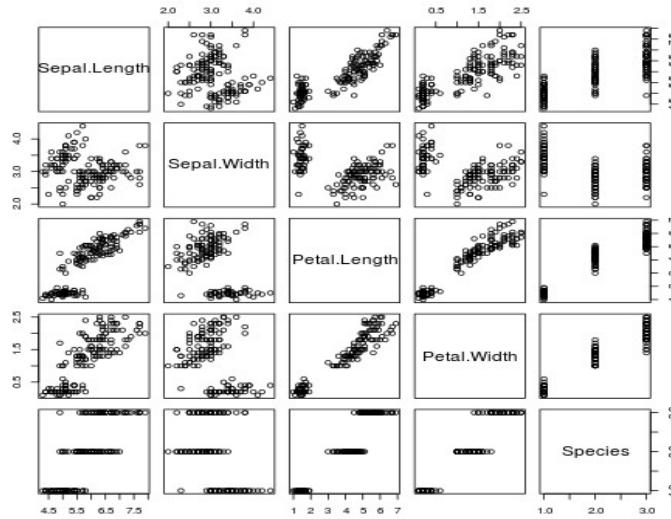


Figura 2: Vari grafici di Iris

`plot(iris)` # mostra i vari grafici di iris; si possono capire quali sono gli assi che di volta in volta mostrano una distribuzione migliore dei punti, es. y=Petal.Length e x=Petal.Width

`lm(Sepal.Length ~ Petal.Width, data=iris)` ## creazione del modello di regressione (y, predittore), inoltre dà informazioni utili su intercetta e coefficiente angolare

```

> summary(modello)

Call:
lm(formula = Sepal.Length ~ Petal.Width, data = iris)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.38822 -0.29358 -0.04393  0.26429  1.34521 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 4.77763   0.07293  65.51   <2e-16 ***
Petal.Width 0.88858   0.05137  17.30   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.478 on 148 degrees of freedom
Multiple R-squared:  0.669,    Adjusted R-squared:  0.6668 
F-statistic: 299.2 on 1 and 148 DF,  p-value: < 2.2e-16

```

Figura 3: Informazioni modello

```

modello <- lm(Sepal.Length ~ Petal.Width, data=iris)
##assegnazione del modello ad una variabile così da poterlo usare con più semplicità
summary(modello)
##dà informazioni utili sulla regressione fatta

```

In particolare:

- residui: presentano una buona distribuzione in quanto la mediana è prossima allo zero (circa metà e 1 e 3 quartile sono rispettivamente -0.29... e 0.26...., inoltre anche minimo e massimo sono simmetrici);
- abbiamo:

$$\begin{aligned} h0: x \text{ e } y \text{ non sono dipendenti} \\ h1: x \text{ e } y \text{ sono dipendenti} \end{aligned}$$

- **valori della retta:** intercetta e coefficiente angolare;
- **standard error:** indica la distanza delle stime dal valor vero, valuta la precisione. Qui la distanza è molto piccola, ciò indica una buona stima;
- **t-value:** valore generato dal test t; presenta un valore pari a 0 quando h0 vale, che cresce man mano l'ipotesi nulla non è più verificabile. Qui i valori sono elevati, caratteristica che porta a rifiutare l'ipotesi nulla;
- **livello di significatività del test:** rappresentabile come il p-value, ossia il minor valore per cui rifiuto h0 (è improbabile che la relazione fra x e y sia dovuta al caso, ecco quindi che sono sicuramente dipendenti fra loro). Più basso è il suo valore più il risultato è significativo. Qui è molto basso, inoltre ci sono i 3 *** che evidenziano che tale predittore è molto importante.
- RSE: misura la distanza media tra i valori stimati e quelli osservati; più piccolo è il suo valore migliore è l'adattamento del modello ai dati; qui risulta piccolo;
- R^2 : valore abbastanza buono che spiega lo 0.67 della devianza (suggerisce che probabilmente ci sarà qualche modello migliore);
- F-statistica: corrisponde alla statistica-test; fornisce un giudizio complessivo sulla bontà esplicativa del modello: probabilità che il modello non sia significativo. Se $F = 1$ allora non vi è alcuna relazione tra y e x, d'altra parte come in questo caso, con $F > 1$ accetto h1.

Conclusione: t-value e statistica-F presentano valori ampiamente superiori ai valori tabulati (difatti i relativi p-value sono di molto inferiori a 0.05). Si rifiuta pertanto l'ipotesi nulla:

La regressione è significativa (il valore del coefficiente angolare così calcolato è statisticamente differente da zero).

```
plot(modello) ## mostra vari grafici del modello di regressione
```

Residui vs Leverage: evidenzia se ci sono valori anomali influenti, ovvero la cui eliminazione porterebbe una sostanziale variazione al modello di regressione. Essendo che tutti i punti si trovano entro le linee di distanza di Cook non è questo il caso;

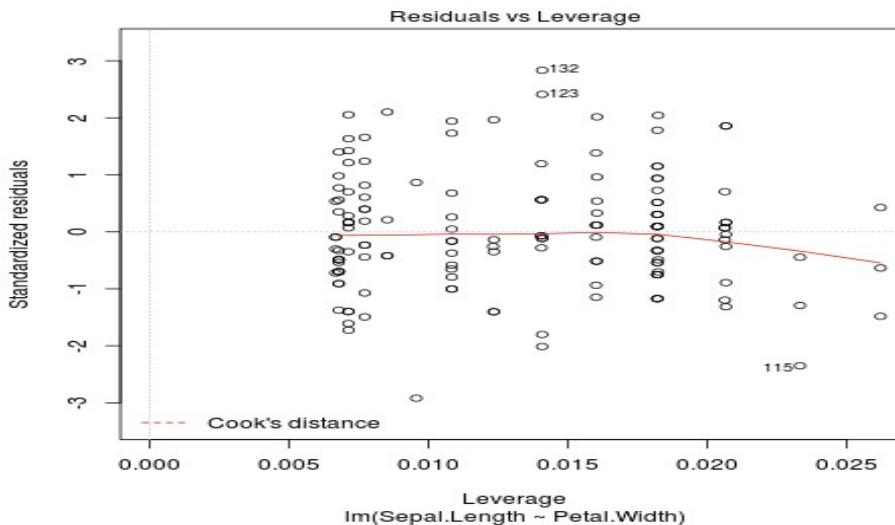


Figura 4: Residui vs Leverage

Scaled-Location: mostra se i residui sono distribuiti equamente lungo gli intervalli dei predittori; rappresenta inoltre il modo in cui è possibile verificare l'ipotesi di uguale varianza (omoscedasticità). Ecco che quando la distribuzione si presenta, come mostrato sotto, casuale, omogenea e attraversata da linea orizzontale, significa che il principio di omoscedasticità viene rispettato;

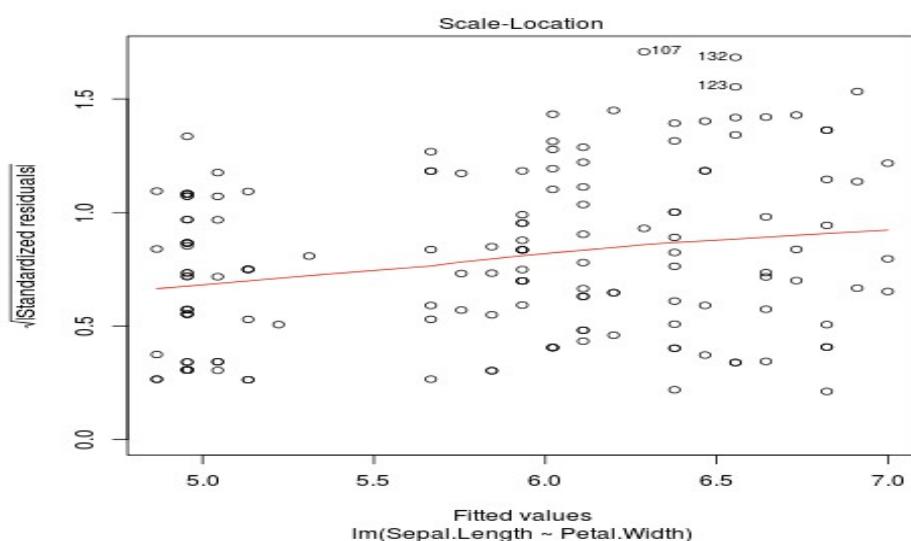


Figura 5: Scale-Location

Q-Q Normal: rappresentazione grafica dei quantili di una distribuzione; confronta la distribuzione cumulata della variabile osservata (residui) con la distribuzione cumulata della normale (distribuzione teorica). Il fatto che questi valori si mostrino tutti abbastanza vicini alla diagonale, tranne per alcuni agli estremi, è una buona cosa, rappresentando una distribuzione dei dati molto vicino alla normale. Si possono inoltre notare alcuni valori particolari: 107, 123 e 132; questi sono punti il cui peso influisce parecchio sul calcolo della regressione lineare. Se tali punti fossero eccessivamente lontani andrebbero eliminati (non in questo caso);

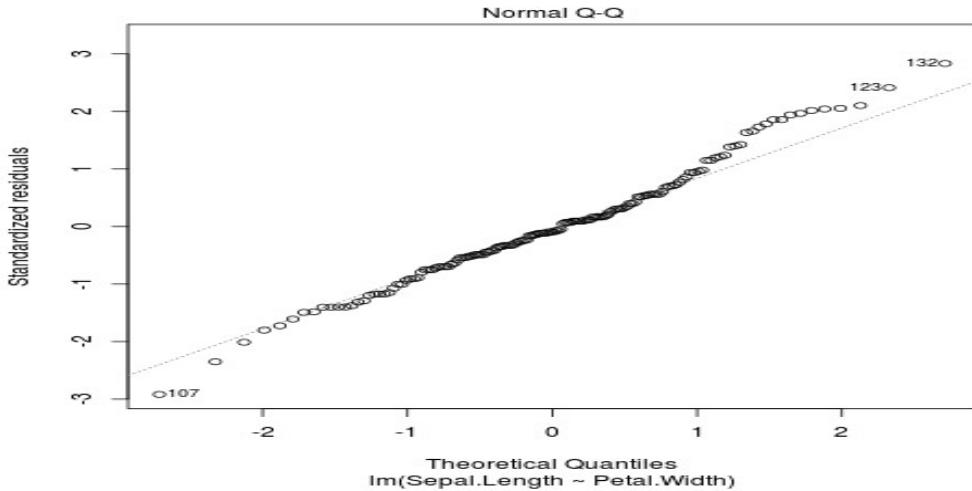


Figura 6: Q-Q Normal

Residual vs Fitted: mostra la presenza o assenza di relazioni lineari fra i predittori ed il predetto. Quando come nel caso in esame, si evidenziano residui sparsi su una linea orizzontale, senza schemi distinti, questa sta ad indicare che le relazioni sono tutte lineari;

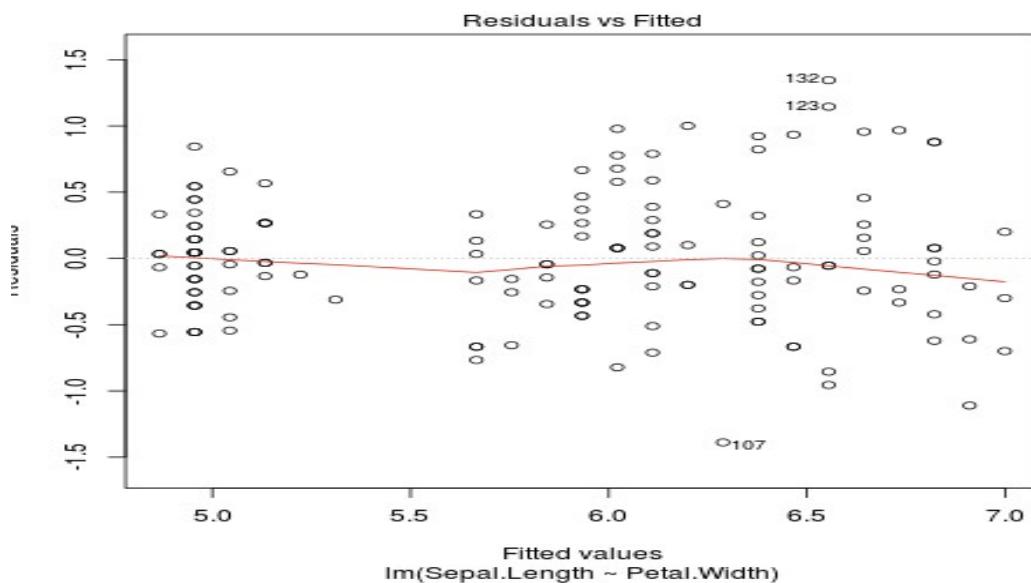


Figura 7: Residual-Fitted

Si presentano ora ulteriori modelli in modo da comprendere l'importanza dei vari predittori caso per caso.

```
modello1 <- lm(Sepal.Length ~ Petal.Length+Petal.Width, data=iris)
summary(modello1)

> summary(modello1)

Call:
lm(formula = Sepal.Length ~ Petal.Length + Petal.Width, data = iris)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.18534 -0.29838 -0.02763  0.28925  1.02320 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 4.19058   0.09705  43.181 < 2e-16 ***
Petal.Length 0.54178   0.06928   7.820 9.41e-13 ***
Petal.Width -0.31955   0.16045  -1.992   0.0483 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4031 on 147 degrees of freedom
Multiple R-squared:  0.7663,    Adjusted R-squared:  0.7631 
F-statistic:  241 on 2 and 147 DF,  p-value: < 2.2e-16
```

Figura 8: Informazioni modello1

L'aggiunta di *Petal.Length* rispetto al modello precedente non ha portato cambiamenti importanti, ecco che esso non è un predittore molto significativo.

```
modello2<- lm(Sepal.Length ~ Sepal.Width+Petal.Width, data=iris)
summary(modello2)
```

```
> summary(modello2)

Call:
lm(formula = Sepal.Length ~ Sepal.Width + Petal.Width, data = iris)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.2076 -0.2288 -0.0450  0.2266  1.1810 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 3.45733   0.30919  11.18 < 2e-16 ***
Sepal.Width 0.39907   0.09111   4.38 2.24e-05 ***
Petal.Width 0.97213   0.05210   18.66 < 2e-16 *** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4511 on 147 degrees of freedom
Multiple R-squared:  0.7072,    Adjusted R-squared:  0.7033 
F-statistic: 177.6 on 2 and 147 DF,  p-value: < 2.2e-16
```

Figura 9: Informazioni modello2

In realtà per individuare il modello maggiormente informativo la tecnica sopra mostrata del **best subsets selection** (ricerca esaustiva fra tutti i modelli che si distinguono fra loro per la combinazione di esplicative che presentano), in quanto sarebbe un metodo troppo dispendioso.

Le tre strategie preferite sono le seguenti:

1. **Forward selection**: si parte inserendo nel modello la covariata con la maggiore correlazione significativa (test t) e stabilendo un livello di significatività. A questo punto si inseriscono man mano i predittori successivi selezionandoli fra quelli con un coefficiente di correlazione parziale che sia il più elevato e significativo. Il procedimento termina quando si individua un coefficiente che non rientra nel livello di significatività precedentemente stabilito; il modello definitivo è quello ottenuto al penultimo passo.
2. **Backward selection**: si parte dal modello che include tutte le variabili, e come sopra si fissa poi un livello di significatività. Ad ogni passo vanno tolte le variabili col coefficiente di regressione meno significativo in base al test t; inoltre le stime dei coefficienti delle variabili rimaste dovranno essere ricalcolate di volta in volta. Si ripeterà tale procedimento sino a quando le covariate non risultino tutte significative rispetto al livello prefissato.
3. **Stepwise regression**: mix dei due criteri precedenti. Prima di tutto aggiungo le variabili seguendo il metodo forward selection. A un certo punto aggiungendo una nuova variabile, i coefficienti di regressione delle variabili già incluse potrebbero risultare singolarmente non significativi a causa della forte correlazione con essa. Pertanto dopo l'inserimento di ciascuna variabile il modello viene riconsiderato attraverso il backword selection dove si controlla se vi è qualche variabile da eliminare.

Se proviamo ad usare il secondo metodo (partendo quindi da un modello contenente tutte le esplicative, ne risulta il seguente output:

```
> bk<-lm(Sepal.Length ~ Petal.Length+Petal.Width+Sepal.Width, data=iris) ##backword selection
> summary(bk)

Call:
lm(formula = Sepal.Length ~ Petal.Length + Petal.Width + Sepal.Width,
    data = iris)

Residuals:
    Min      1Q  Median      3Q     Max 
-0.82816 -0.21989  0.01875  0.19709  0.84570 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 1.85600   0.25078   7.401 9.85e-12 ***
Petal.Length 0.70913   0.05672  12.502 < 2e-16 ***
Petal.Width -0.55648   0.12755  -4.363 2.41e-05 ***
Sepal.Width   0.65084   0.06665   9.765 < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3145 on 146 degrees of freedom
Multiple R-squared:  0.8586,    Adjusted R-squared:  0.8557 
F-statistic: 295.5 on 3 and 146 DF,  p-value: < 2.2e-16
```

Figura 10: Modello con tutte le covariate

Si può notare come non sia necessario fare altri step in quanto non vi sono variabili inutili a fini informativi. Facendo inoltre un controllo incrociato anche con le altre combinazioni di modelli questo risulta infatti quello con la maggiore quantità di varianza spiegata.

Nel dettaglio si ha che:

PREDITTORI	OSSERVAZIONI
Petal.Length+Petal.Width+Sepal.Length	<ul style="list-style-type: none"> Tutti i predittori sono significativi <ul style="list-style-type: none"> $R^2=0,859$
Sepal.Width+Petal.Width	<ul style="list-style-type: none"> Entrambi i predittori sono significativi <ul style="list-style-type: none"> $R^2=0,707$
Petal.Length+Petal.Width	<ul style="list-style-type: none"> Petal.Length risulta non significativo <ul style="list-style-type: none"> $R^2=0,767$
Petal.Length+Sepal.Width	<ul style="list-style-type: none"> Entrambi i predittori sono significativi <ul style="list-style-type: none"> $R^2=0,840$

Possiamo quindi concludere che:

$$\text{Sepal.Length} \sim \text{Petal.Length} + \text{Petal.Width} + \text{Sepal.Width}$$

Questo sebbene potesse essere un risultato anche abbastanza prevedibile non essendoci un numero eccessivo di variabili dipendenti, mostra l'utilità di saper individuare il contributo informativo che ciascuna variabile porta al modello, soprattutto se ci si trova a trattare con un numero di dimensioni superiori.

A seguire se ne include la rappresentazione grafica. Essendo un modello di *regressione lineare multivariato* (k predittori > 1) abbiamo che la relazione tra un dato regressore e la variabile che si vuole prevedere (*Sepal.Width*), può venire influenzata dai restanti regressori; ecco che le varie relazioni vengono presentate singolarmente al netto dell'influenza degli altri regressori del modello.

```
bk<-lm(Sepal.Length ~ Petal.Length+Petal.Width+Sepal.Width, data=iris) ##modello
library(car) ##package necessario
ceresPlots(bk) ##grafico multivariato
```

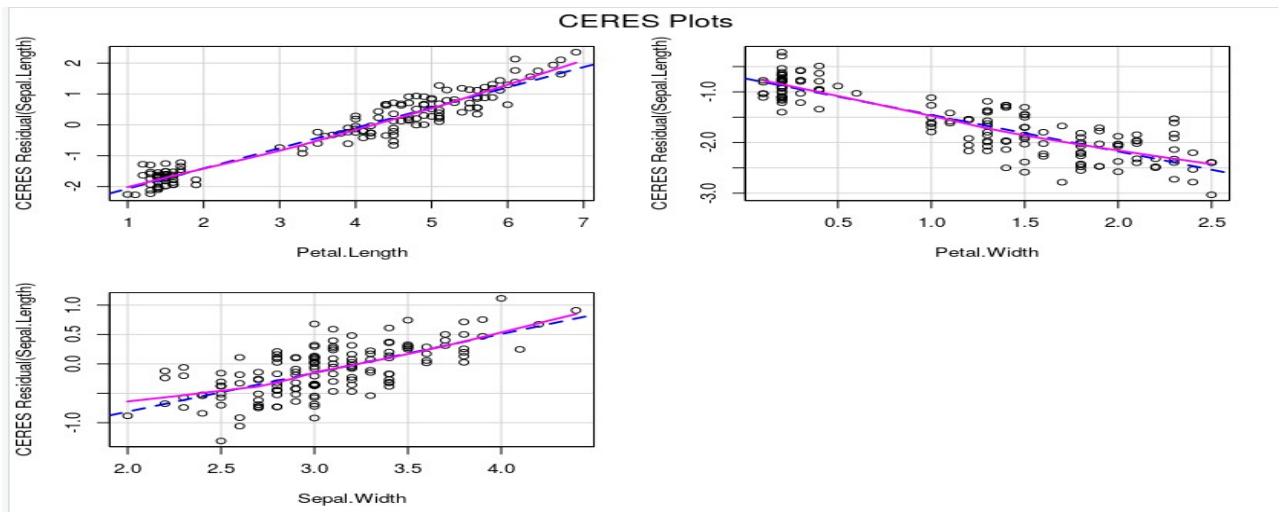


Figura 11: Grafici Regressione lineare multipla dei singoli regressori

Come si può notare la qualità della previsione (linea tratteggiata) è molto buona rispetto alla retta di regressione dei dati reali (linea continua), cosa conforme al valore della devianza spiegata coperta per più di un 80%.

Un altro elemento di cui è importante comprendere il funzionamento è il metodo ANOVA; esso consiste nell'*analisi della varianza*, e permette di individuare le differenze presenti tra più gruppi di dati confrontando la loro variabilità interna con la variabilità tra i gruppi. L'applicazione di tale metodologia richiede preventivamente il soddisfacimento di due proprietà:

- normalità (distribuzione gaussiana dei dati);
- omoschedasticità.

Il set di dati in esame, rispettando entrambe queste caratteristiche, risulta idoneo per la sua applicazione.

Nel dettaglio, R presenta la funzionalità `anova()`: confrontando fra loro vari modelli permette di capire se le variabili presenti in più o in meno in un modello rispetto che in un altro, apportano effettivamente un contributo significativo nello spiegare la variabile risposta (il tutto viene verificato tramite i valori della probabilità che H_0 sia vera e del test F).

```
modello7 <- lm(Sepal.Length ~ Petal.Length, data=iris)
modello1 <- lm(Sepal.Length ~ Petal.Length + Petal.Width , data=iris)
anova(modello7, modello1)
```

L'output che ci viene fornito è il seguente:

```
> anova(modello7, modello1)
Analysis of Variance Table

Model 1: Sepal.Length ~ Petal.Length
Model 2: Sepal.Length ~ Petal.Length + Petal.Width
  Res.Df   RSS Df Sum of Sq    F Pr(>F)
1     148 24.525
2     147 23.881  1  0.64434 3.9663 0.04827 *
...
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figura 12: Risultati anova - confronto fra modelli

In sostanza significa che *Petal.Width* rispetto al modello che non la include, non è una variabile che porta moltissima informazione.

Attenzione: comunque un po' di informazione la variabile *Petal.Width* la porta (altrimenti non sarebbe indicato neppure lo *).

A riprova di ciò si noti che facendosi aiutare anche dal valore dell'*Akaike Information Criterion* esso non dice che il `modello1` è inutile anzi, essendo quello con l'AIC minore viene selezionato come migliore rispetto al `modello7`; semplicemente pone l'accento sulla necessità di attribuire ai numeri la giusta interpretazione in base al contesto in cui sono calati.

Ne emerge quindi che:

Andando a guardare con attenzione il peso che ciascuna variabile possiede nel modello, si conclude che *Petal.Length* e *Petal.Width* non rappresentano probabilmente la combinazione migliore da attuare.

```
x <- c(AIC(modello7), AIC(modello1))
delta <- x - min(x) ##il modello migliore risulta quello con AIC più piccolo
```

```
> x <- c(AIC(modello7), AIC(modello1))
> delta <- x - min(x)
> delta
[1] 1.993607 0.000000
```

Figura 13: Valore AIC

Confutiamo quanto esposto andando ad osservare i dati per ciascun singolo modello.

```
> summary(modello7)                                > summary(modello1)

Call:                                              Call:
lm(formula = Sepal.Length ~ Petal.Length, data = iris) lm(formula = Sepal.Length ~ Petal.Length + Petal.Width, data = iris)

Residuals:                                         Residuals:
Min      1Q   Median      3Q      Max 
-1.24675 -0.29657 -0.01515  0.27676  1.00269 
-1.18534 -0.29838 -0.02763  0.28925  1.02320 

Coefficients:                                     Coefficients:
Estimate Std. Error t value Pr(>|t|)          Estimate Std. Error t value Pr(>|t|) 
(Intercept) 4.30660   0.07839  54.94 <2e-16 *** (Intercept) 4.19058   0.09705  43.181 < 2e-16 ***
Petal.Length 0.40892   0.01889  21.65 <2e-16 *** Petal.Length 0.54178   0.06928  7.820 9.41e-13 ***
...                                                 Petal.Width -0.31955   0.16045 -1.992  0.0483 * 
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4071 on 148 degrees of freedom
Multiple R-squared:  0.76,    Adjusted R-squared:  0.7583
F-statistic: 468.6 on 1 and 148 DF, p-value: < 2.2e-16
Residual standard error: 0.4031 on 147 degrees of freedom
Multiple R-squared:  0.7663,   Adjusted R-squared:  0.7631
F-statistic: 241 on 2 and 147 DF, p-value: < 2.2e-16
```

Figura 14: Confronto fra i due modelli

Che non fa altro che confutare quanto già affermato da anova.

Al netto di ciò e riprendendo il modello di regressione ottimo individuato in precedenza, si evince che:

Conclusione: La variabile *Petal.Width* acquisisce la sua importanza massima quando si trova in combinazione con *Sepal.Width*, perdendone in assenza.

```

> anova(modello1, modello7, bk)
Analysis of Variance Table

Model 1: Sepal.Length ~ Petal.Length + Petal.Width
Model 2: Sepal.Length ~ Petal.Length
Model 3: Sepal.Length ~ Petal.Length + Petal.Width + Sepal.Width
  Res.Df   RSS Df Sum of Sq    F Pr(>F)
  1     147 23.881
  2     148 24.525 -1   -0.6443  6.5124 0.01174 *
  3     146 14.445  2   10.0796 50.9375 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Figura 15: I tre modelli a confronto

Domanda:

Cosa accade se inseriamo nel modello anche valori discreti?

Nota: *Orange Canvas* infatti non lo lascia fare.

Aggiungendoli al modello, come mostrato a seguire, si nota che sorprendentemente portano comunque un loro contributo, andando anche ad influire sulla variabile *Petal.Width* che perde di significatività (passa da *** a *), e facendo alzare R^2 anche se non di moltissimo.

```

Call:
lm(formula = Sepal.Length ~ Petal.Length + Petal.Width + Sepal.Width +
Species, data = iris)

Residuals:
    Min      1Q  Median      3Q      Max 
-0.79424 -0.21874  0.00899  0.20255  0.73103 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.17127   0.27979   7.760 1.43e-12 ***
Petal.Length  0.82924   0.06853  12.101 < 2e-16 ***
Petal.Width   -0.31516   0.15120  -2.084  0.03889 *  
Sepal.Width    0.49589   0.08607   5.761 4.87e-08 ***
Speciesversicolor -0.72356   0.24017  -3.013  0.00306 ** 
Speciesvirginica -1.02350   0.33373  -3.067  0.00258 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3068 on 144 degrees of freedom
Multiple R-squared:  0.8673,    Adjusted R-squared:  0.8627 
F-statistic: 188.3 on 5 and 144 DF,  p-value: < 2.2e-16

```

Figura 16: Modello inclusivo delle specie

È chiaro quindi come anche tali variabili abbiano il loro peso all'interno del modello, e come la loro conseguente inclusione o meno possa andare ad influire sui legami con le altre covariate.

Viene ora mostrato come si distribuiscono i valori osservati distinti per specie di appartenenza. Se ne include il codice impiegato e il plot generato da R.

Nell'asse y viene indicata la variabile dipendente, mentre la x cambia in base alla variabile selezionata:

```
library(RGraphics)
library(grid)
library(gridExtra)
library(ggplot2)
##grafici carini in base alle specie
g1<-ggplot(iris,aes(x=Sepal.Width,y=Sepal.Length, shape=Species, color=Species))+geom_point(size=2.5)
g2<-ggplot(iris,aes(x=Petal.Width,y=Sepal.Length, shape=Species, color=Species))+geom_point(size=2.5)
g3<-ggplot(iris,aes(x=Petal.Length,y=Sepal.Length, shape=Species, color=Species))+geom_point(size=2.5)

grid.arrange(g1,g2,g3, nrow=2, ncol=2, top = "Species Distributions")
```

I risultati non stupiscono in quanto corrispondono a quelli ottenuti con *Orange Canvas* ad una prima analisi (vedi Appendice).

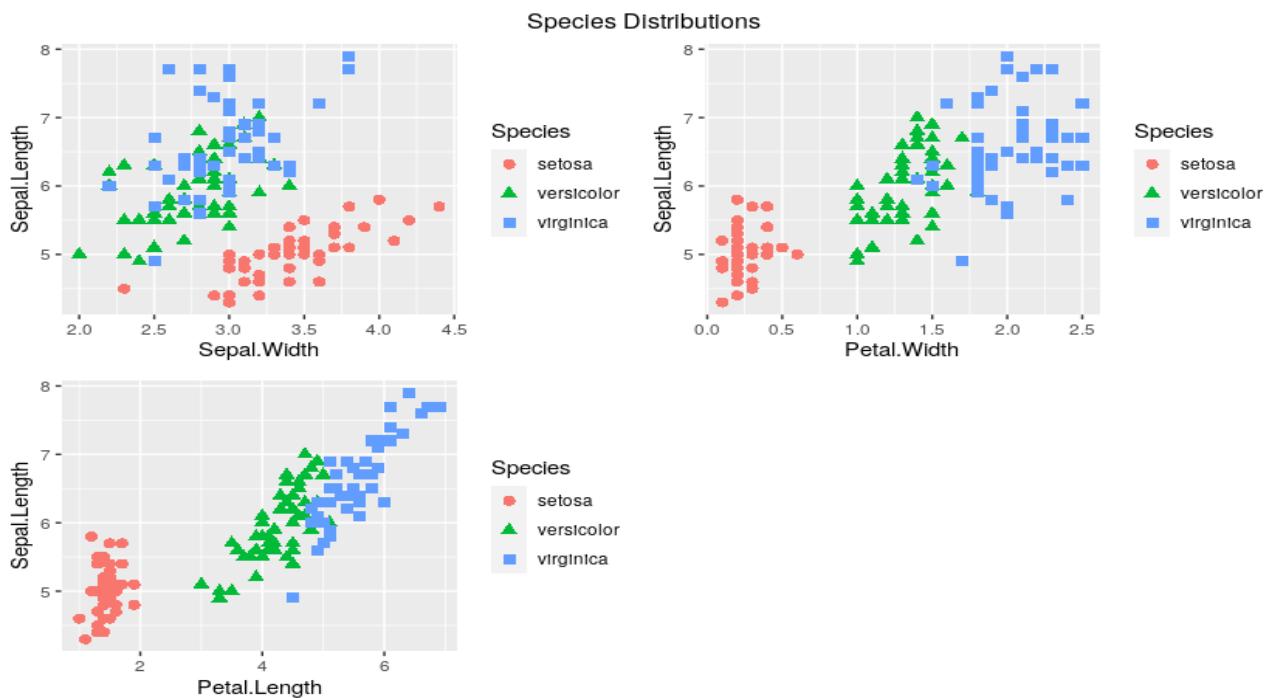


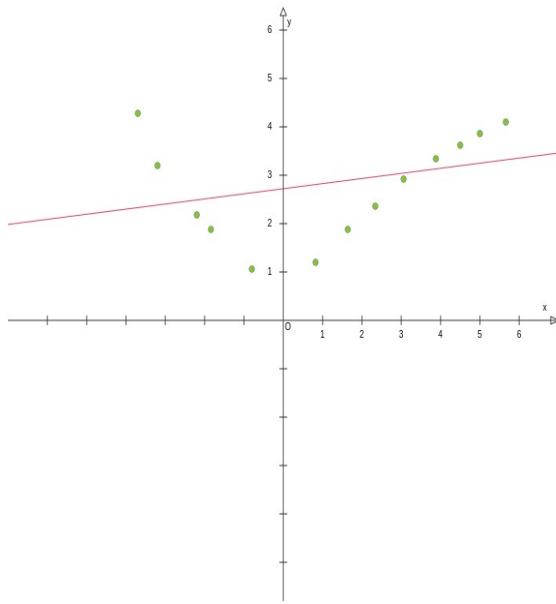
Figura 17: Distribuzione delle specie

2.1.2 Algoritmi di regressione a confronto

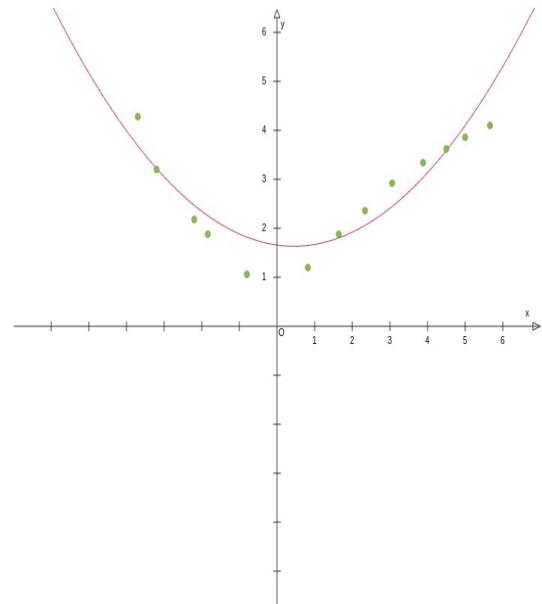
Fino ad ora si sono confrontati fra loro modelli differenti limitandosi tuttavia ad agire esclusivamente sulla tipologia e sulla quantità dei predittori selezionati (e.g: sempre all'interno della regressione lineare). Operiamo ora un passo successivo:

Cosa accade se si confrontano fra loro differenti tipologie di modelli di regressione?

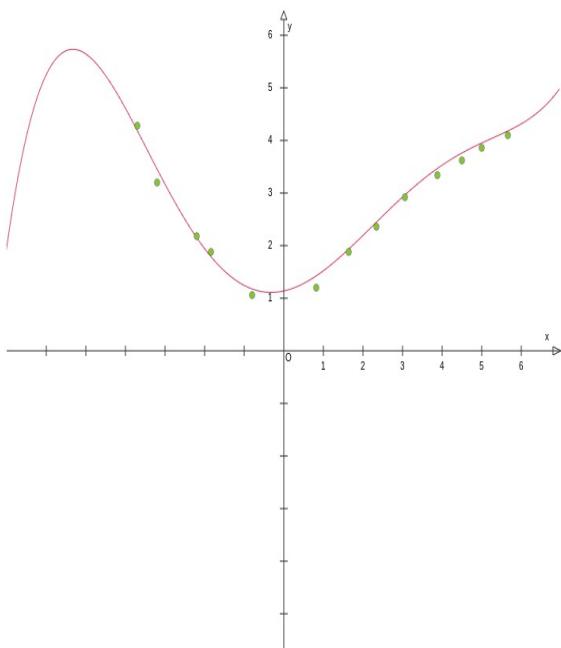
Regressione lineare classica (retta)



Regressione polinomiale di grado 2 (parabola)



Regressione polinomiale di grado 5



K-nearest neighbors (K-nn - 8 predittori)

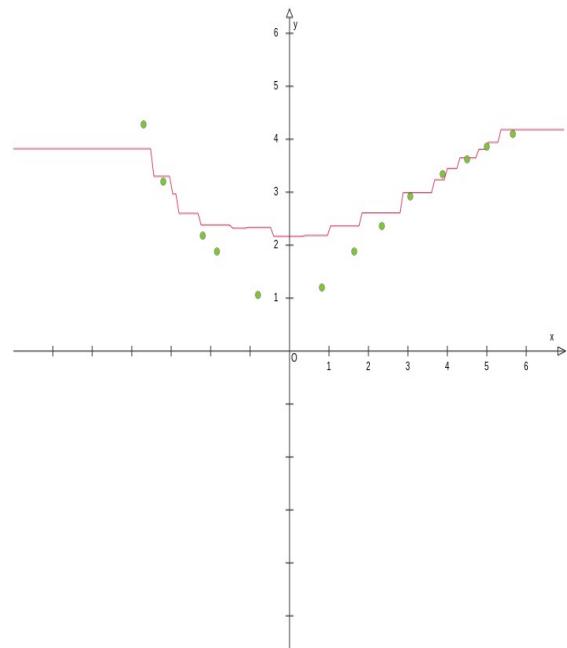


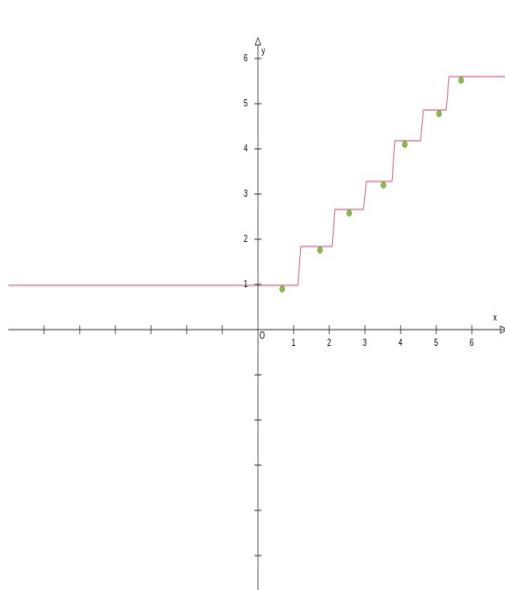
Figura 18: Metodi di Regressione differenti a confronto

Al di là di quella che può essere l'effettiva accuratezza dei vari modelli di regressione (già da un primo sguardo sulla distribuzione dei dati il bravo Data Scientist dovrebbe essere infatti in grado di intuire che una regressione polinomiale si adatterà meglio rispetto ad una lineare), risulta importante evidenziare che:

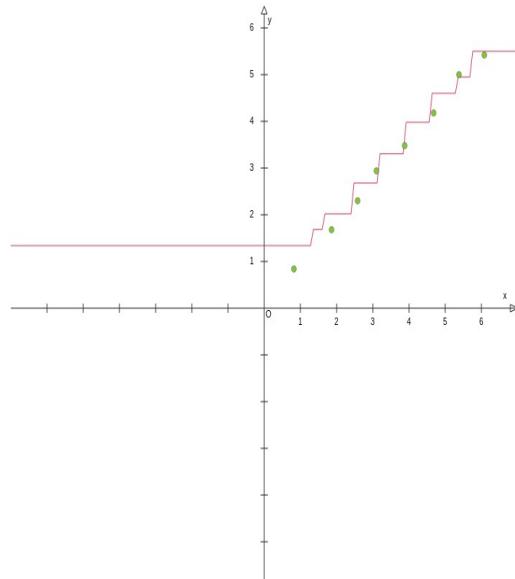
Vi sono algoritmi che fondano il loro operato su regole certe, integrando il pensiero intelligente con l'osservazione delle Leggi Naturali; altri invece che seguono l'intuizione.

La regressione lineare classica e il K-nn sono un esempio lampante di quanto appena affermato. La prima infatti deve il suo comportamento alle rigide regole che le vengono imposte dal seguire una *Legge di Natura* ben specifica: la retta (opera controlli sui residui); il secondo preferisce invece affidarsi all' "intuizione" fornendo la risposta migliore esclusivamente in base ai dati (esamina i suoi k vicini – rischio di overfitting), trascurandone le possibili regole che vi possono essere celate al loro interno.

K-nn con 1 preditore



K-nn con 4 preditori



K-nn con 8 preditori

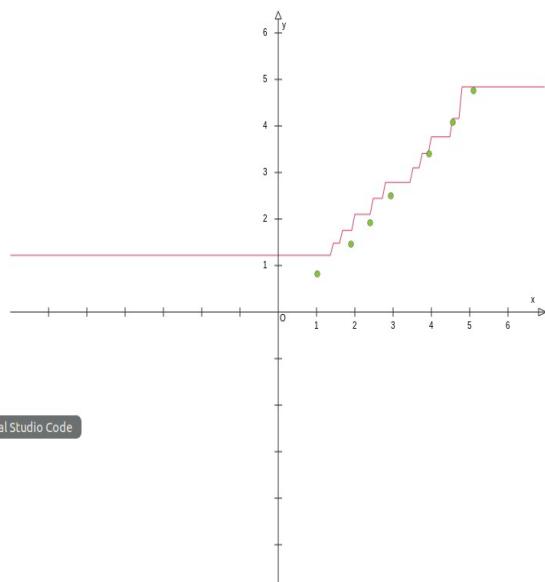


Figura 19: K-nn a confronto

Si noti come l'incremento del numero di vicini esaminati dall'algoritmo ($k = n$. predittori), comporti un aumento dei "balzi" eseguiti dalla linea; tale fenomeno si genera in quanto all'aumentare del parametro k aumenteranno conseguentemente il numero di punti da prendere in considerazione (con $k = 1$ si esamina solo il punto più vicino).

Generalmente valori elevati di k portano una diminuzione delle interferenze (rumore), rendendo però più difficile l'individuazione dei confini fra i punti.

Alla luce di quanto visto

È meglio affidarsi alle regole o all'intuito?

Fra i due principi non esiste un comportamento migliore e uno peggiore, bisogna vedere caso per caso quale fra le due strade è più saggio intraprendere.

2.2 Support Vector Machines

Per comprendere a pieno le *Support Vector Machine (SVM)* è necessario fare un passo indietro e trattare le idee che vi stanno alla base, oltre che le origini.

Tutto ebbe inizio attorno agli anni '30 grazie alla grande diffusione che ebbe la statistica.

Essa si può considerare molto legata alla teoria della probabilità, ma mentre in quest'ultima conoscendo il **processo di generazione dei dati sperimentali** (modello probabilistico), si è in grado di valutare la probabilità dei diversi possibili risultati di un esperimento, nella statistica tale processo non è noto in modo completo, anzi diventa esso stesso l'oggetto dell'indagine. Ecco che le tecniche statistiche si prefiggono di indurre le caratteristiche di tale processo sulla base dell'osservazione dei dati sperimentali da esso generati.

Come non citare in questo contesto **Andrej Nikolaevič Kolmogorov**, matematico sovietico, definito il padre della probabilità di cui ne ha stabilito gli assiomi. Ha inoltre formulato l'omonimo test non parametrico che verifica la forma delle distribuzioni campionarie. Adattandosi ad ogni distribuzione permette infatti di capire se i dati che si hanno a disposizione rispettano o meno una specifica distribuzione a cui si pensa si debbano adeguare (*Test di Kolmogorov Smirnov*)

O personaggi come **Karl Pearson**, **Ronald Aylmer Fisher**, o ancora **Vladimir Naumovič Vapnik**: proprio quest'ultimo rappresenta una figura d'importanza strategica proprio nell'ambito degli algoritmi SVM.

Prima di tutto risulta fondamentale distinguere fra test parametrici e test non parametrici:

- **test parametrici:** si sa già che le cose si comporteranno secondo quella determinata regola o processo; quello che non si conosce invece e che si è quindi interessati a trovare, riguarda le caratteristiche, in altri termini i parametri, che contraddistinguono la regola stessa;

Basti pensare ad esempio al decadimento dell'uranio: si è già a conoscenza del fatto che tale processo presenta un' andamento di decrescita esponenziale (in termini matematici corrispondente al %), dimezzando la sua radioattività di volta in volta al trascorrere di un tempo fissato.

- **test non parametrici:** caratterizzati dalla non conoscenza del tipo di distribuzione, a differenza di quelli sopra, questi ignorano il comportamento (il processo) caratterizzante il fenomeno in esame (non implicano la stima di parametri statistici come media, deviazione standard, varianza, ecc.).

I primi tendono ad essere preferiti rispetto ai secondi in quanto a parità di potenza quelli non parametrici richiedono un numero nettamente superiore di dati; tuttavia fu proprio con Vapnik nell'ambito del SVM che vennero riscoperti.

Con l'arrivo degli anni '60 gli statisti si dovettero infatti confrontare con una nuova sfida: **la maledizione della dimensionalità** (termine coniato da R. Bellman). Con l'avvento del computer era idea diffusa che si sarebbe stati in grado di processare quantità di dati maggiori rispetto ai periodi precedenti, aumentando al contempo la precisione dei risultati ottenuti. Tuttavia le cose si mostrarono molto diverse dall'auspicato: vi era certamente una capacità di elaborazione superiore ma questo non portò a guadagnarne in precisione bensì a perderne a causa dell'introduzione di un numero di errori superiore.

A cosa si dovette tale particolare fenomeno?

La risposta è alquanto semplice: l'aumento dei punti su cui si lavora (i predittori) porta lo spazio di cui si dispone a diventare più rarefatto. Appare evidente come di conseguenza i punti saranno più distanziati fra loro, rendendone più complessa la loro corretta individuazione.

È proprio tale fenomeno che ha cercato di risolvere Vapnik introducendo le *Support Vector Machine*, che mostrano un aumento della loro precisione man mano che la dimensione dello spazio cresce.

Scendiamo ora più nei dettagli di ciò.

Vapnik decise di recuperare la statistica non parametrica e di concentrarsi sul **principio di minimizzazione del rischio**. Tale comportamento evidenzia una netta differenza rispetto alla visione impiegata dai suoi contemporanei che si articolava invece sul concetto di massima probabilità e su teoremi come il **Teorema del limite centrale** e la **Legge dei grandi numeri**.

2.2.1 Teorema del limite centrale

In una popolazione che non segue il modello gaussiano, le medie campionarie, se calcolate su campioni abbastanza grandi, tendono a distribuirsi secondo una legge gaussiana. In altre parole, sommando eventi che presentano una distribuzione casuale, si otterrà sempre come risultato finale una normale.

Basti pensare ad esempio al lancio di due dati e alla probabilità che ne esca il valore 7: esisteranno vari modi con cui si potrà ottenere tale risultato (5+2, 4+3 ...); ecco che sommando ad ogni passo proprio questi vari risultati la distribuzione che si osserverà risulterà sempre più una normale, fino alla fine ad adattarvisi completamente.

L'immagine sotto spiega molto bene il fenomeno.

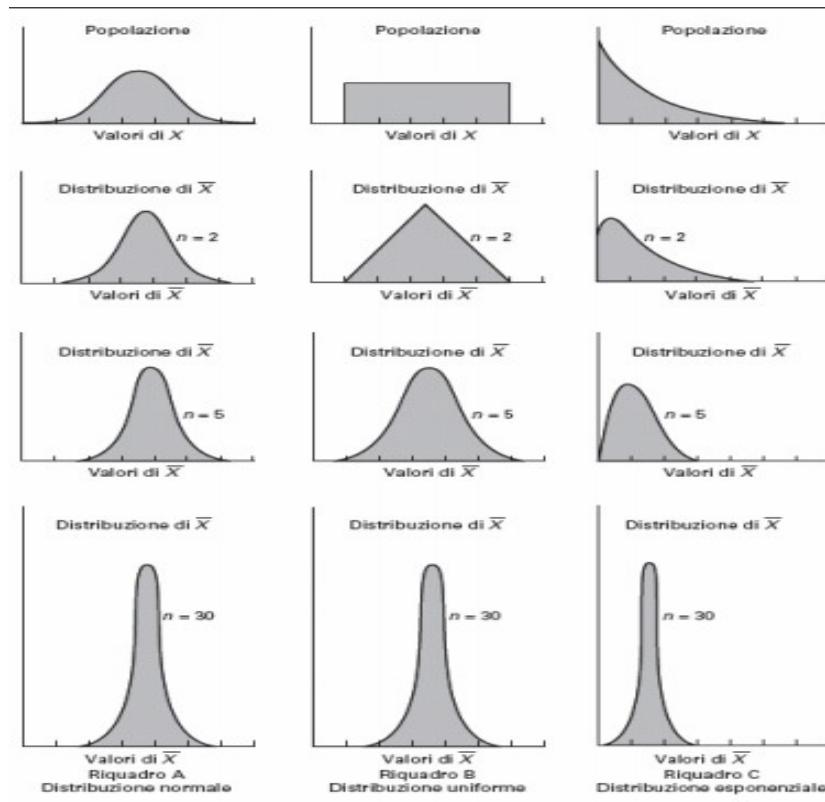


Figura 20: Distribuzione normale

2.2.2 Legge dei grandi numeri

Noto anche come **teorema di Bernoulli**, afferma che all'aumentare del numero delle prove fatte il valore della frequenza tende al valore teorico della probabilità.

Dunque:

- la media calcolata teoricamente è un'approssimazione di quelle sperimentali, ed aumenta la sua precisione al crescere di n ;
- viceversa, si può prevedere che i risultati sperimentali mostreranno una media tanto più prossima alla media teorica, quanto più grande sarà n .

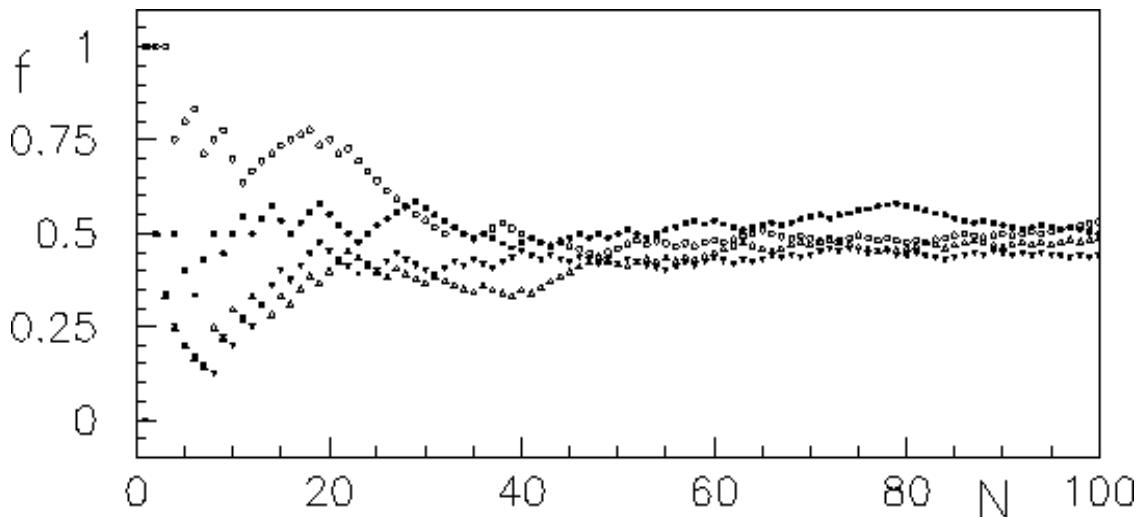


Figura 21: Legge empirica del caso

Questo teorema fornisce una possibile giustificazione alla **Legge empirica del caso** secondo la quale la frequenza relativa di un evento tende ad stabilizzarsi all'aumentare del numero delle prove.

È evidente come il Teorema del limite centrale e la Legge dei grandi numeri siano fortemente connesse fra loro sancendo le basi della statistica classica.

2.2.3 Criterio di minimizzazione del rischio

Passaggio dal pensiero probabilistico che regola il verificarsi di un evento, al concetto di rischio:

“Cosa rischio facendo questa previsione?”

Vapnik infatti introduce l'idea che si hanno a disposizione dei predittori, ciascuno con le proprie specifiche caratteristiche, ed i dati di test. Quest'ultimi grazie ai risultati che mi forniscono (R^2 , varianza ecc.) rappresentano l'errore noto, permettendo poi di risalire all'errore che non si conosce.

L'obiettivo in questo contesto è quello di individuare il modello ottimo; esso è tale proprio perché minimizza il rischio di commettere errori, gli fornisce quindi una buona capacità predittiva non andando però a sovraccaricare eccessivamente di predittori il modello (compromesso fra bias e varianza). Questo è proprio ciò che sta alla base del **Teorema di Vapnik Chervonenkis** e delle SVM.

Per capirlo meglio a seguire vengono spiegate nel dettaglio cosa sono le *Support Vector Machine*.

Idealmente è necessario figurarsi in uno spazio popolato da delle osservazioni. L'obiettivo in tutto ciò consiste nell'individuare l'iperpiano che taglia meglio questo spazio permettendo di classificare i vari punti in due classi distinte. Uno spazio qualsiasi può venire tuttavia attraversato da un infinito numero di iperpiani, si sarà quindi interessati esclusivamente all'individuazione di quello ottimo.

La potenza delle SVM sta proprio nel fatto che non temono la dimensionalità anzi, più il grado del polinomio è elevato più lavorano meglio, in quanto puntano ad individuare solo i punti più importanti: **i vettori di supporto**. Sono proprio quest'ultimi che permettono di trovare l'iperpiano in questione.

A seguire l'immagine mostra meglio quanto detto.

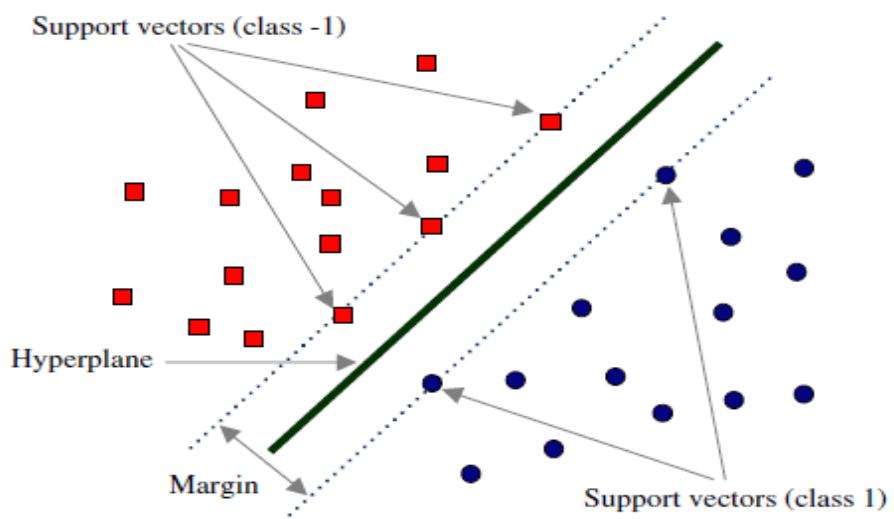


Figura 22: Support Vector Machine

Si può notare come il *margine*, non sia altro che la distanza tra i vettori di supporto delle due classi, alla cui metà si posiziona l'iperpiano (o retta nel caso si stia lavorando a due dimensioni).

Come si seleziona l'iperpiano migliore?

Come mostra l'immagine a fianco, si seleziona l'iperpiano che si trova alla **massima distanza** dai vettori di supporto delle varie classi; maggiore è infatti la distanza fra di lui e i punti, e più cresce la fiducia che la classificazione sia stata svolta correttamente.

È proprio in questo ragionamento che si posiziona il *Teorema di Vapnik Chervonenkis*:

"La macchina con la capacità più piccola è la migliore."

Principio che si muove in simbiosi con il rasoio di Occam:

"A parità di fattori la spiegazione più semplice è da preferire"

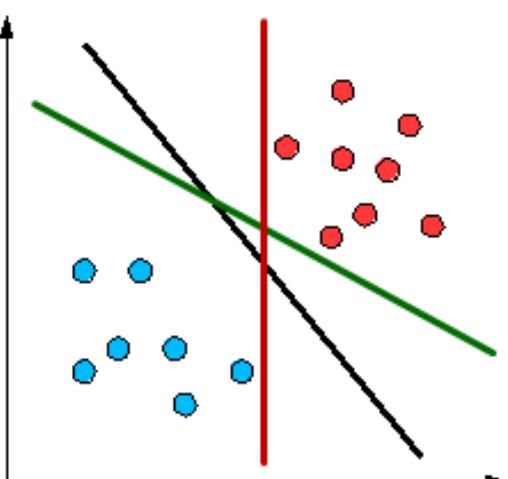


Figura 23: Iperpiani SVM

Si noti che non va confuso il concetto di "capacità minima" con quello di "semplicità" (nel senso di un ridotto numero di parametri), ma va intesa come **dimensione Vapnik Chervonenkis** che deve quindi puntare ad essere la più piccola possibile; vi possono essere infatti modelli con tanti parametri che presentano una bassa VC e viceversa.

A questo punto appare spontanea una domanda:

Cos'è la dimensione Vapnik Chervonenkis?

Essa rappresenta la cardinalità dell'insieme più grande frantumabile.

In generale per uno spazio a k dimensioni corrisponde a:

$$VC(H) = k+1$$

La dimensione VC per un classificatore lineare è almeno 3 perché non si riescono a frammentare 4 punti. Vengono mostrati alcuni esempi che spiegano tale cosa.

Nel caso di 2 punti si avrà una $VC(H) \geq 1$

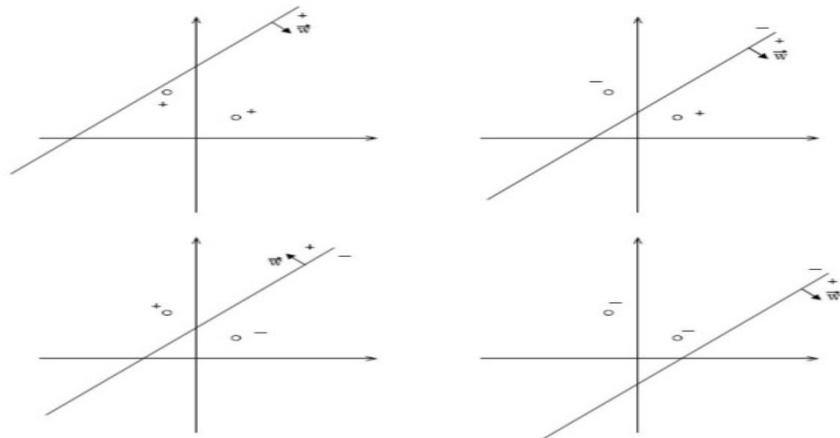


Figura 24: VC-dimension 2 punti

Con 3 punti sarà $VC(H) \geq 2$

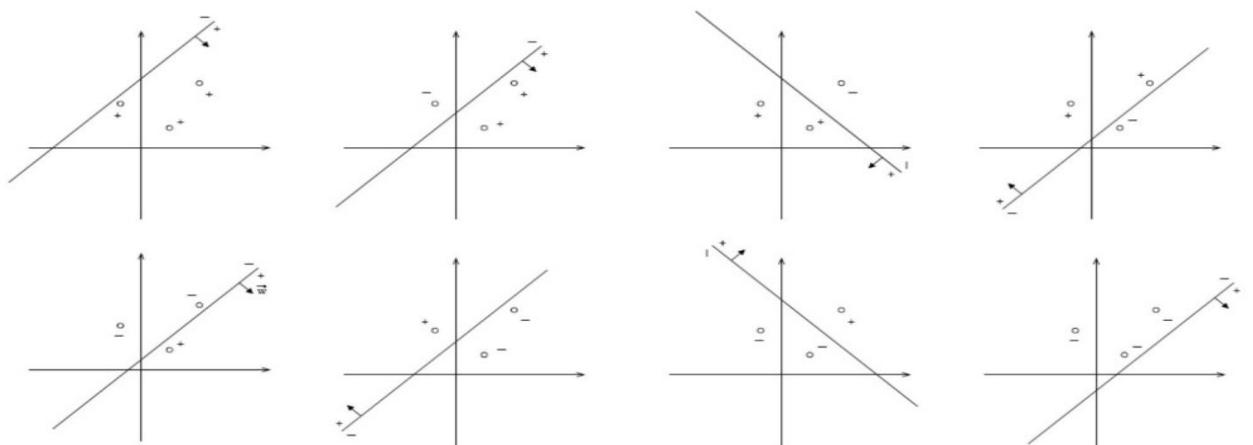


Figura 25: VC-dimension 3 punti

Con 4 punti invece

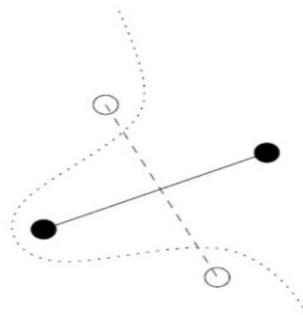


Figura 26: VC-dimension 4 punti

Ecco che come preannunciato poco sopra, l'immagine mostra chiaramente come la dimensione VC per un classificatore lineare è almeno 3 proprio per l'incapacità di frammentare 4 punti. Esisteranno sempre due coppie di punti che se unite con un segmento daranno luogo ad un'intersezione fra i due segmenti; ponendo ogni coppia di punti in classi differenti, per separarli non sarebbe sufficiente una retta bensì una curva.

$$VC(H) = 3$$

Ecco che la dimensione VC viene usata per classificare i diversi tipi di algoritmi in base alla loro complessità.

In particolare con l'aumentare della complessità di un modello, si passa dal sottoadattamento a quello eccessivo (da underfitting a overfitting); ciò rappresenta il passaggio chiave che permette di comprendere il sopra citato Teorema di Vapnik Chervonenkis. L'aggiungere complessità infatti non fornisce una certezza sicura che il modello sia migliore di quello meno complesso. Nello specifico tale aggiunta si mostra valida fino a un certo punto, dopo di che si avrà una discesa causata da un sovraccarico dei dati di addestramento.

Un altro modo di pensare la cosa, ma che racchiude il medesimo significato, è attraverso i concetti di *Bias* e *Varianza* e di come sia necessario trovare il giusto trade-off fra questi due indici. Un modello a bassa complessità infatti se da un lato, a causa del suo ridotto potere espressivo non permette di fidarsi completamente dei valori ottenuti, dall'altro recupera in semplicità, cosa che porta prestazioni molto prevedibili e bassa varianza. Al contrario, un modello più complesso avrà un bias inferiore proprio per la sua maggiore espressività, ma presenterà una varianza più elevata a causa della presenza di più parametri.

Appare quindi evidente come ad un certo livello di complessità del modello esisterà un equilibrio ideale tra distorsione e varianza, in corrispondenza della quale non si è né insufficienti né adatti ai propri dati.

Questo riporta esattamente al Teorema di Vapnik Chervonenkis:

“Si dovrebbe mirare a scegliere un modello con un livello di complessità (una bassa VC) appena sufficiente per svolgere la classificazione in questione.”

2.2.4 Analisi dei dati con R: Dataset Iris

Come fatto anche per la Regressione anche per le SVM si è analizzato attraverso l'impiego di R il *dataset Iris*; in tal modo non si sono comprese solo le eventuali differenze rispetto ai modelli di regressione, ma si sono aumentate le informazioni possedute rispetto al caso in esame.

Vengono qui inclusi alcuni grafici inizialmente impiegati per comprendere su quali covariate fosse meglio concentrarsi per la costruzione del modello:

Petal.Width e *Petal.Length* appaiono come parametri promettenti al fine di ottenere un buon modello, in quanto permettono di separare con la massima fiducia la specie Setosa dalle altre due (che risultano invece sovrapposte).

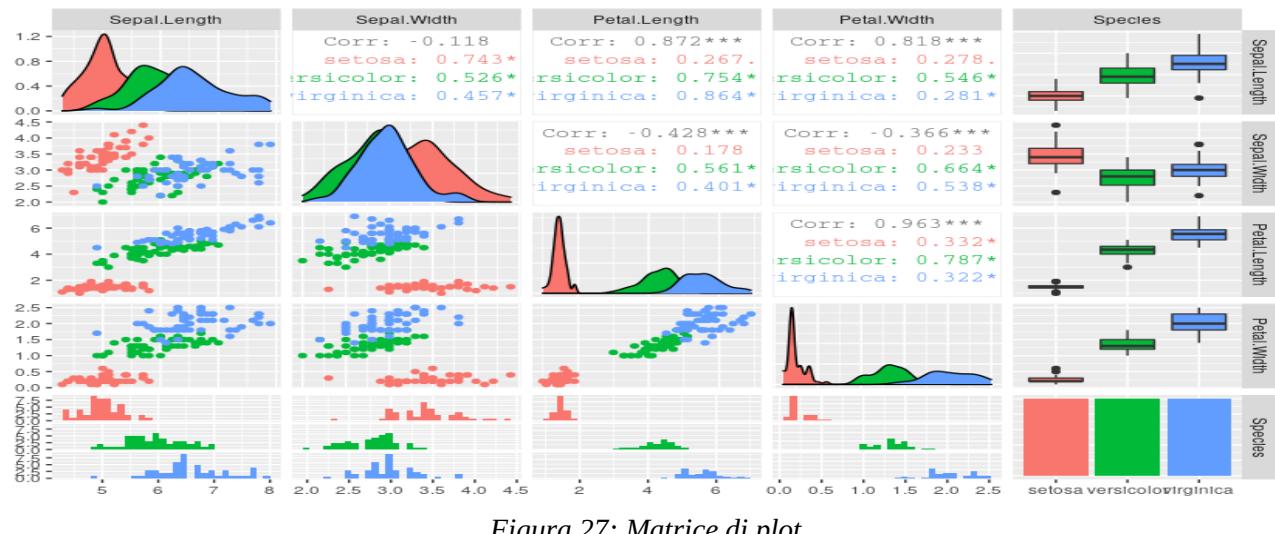


Figura 27: Matrice di plot

Il modello **SVM a kernel lineare** realizzato si è concentrato nel predire le specie usando proprio tali variabili dipendenti; l'ampiezza del campione di train è stato inoltre di 100 unità, contro le 50 impiegate per il test.

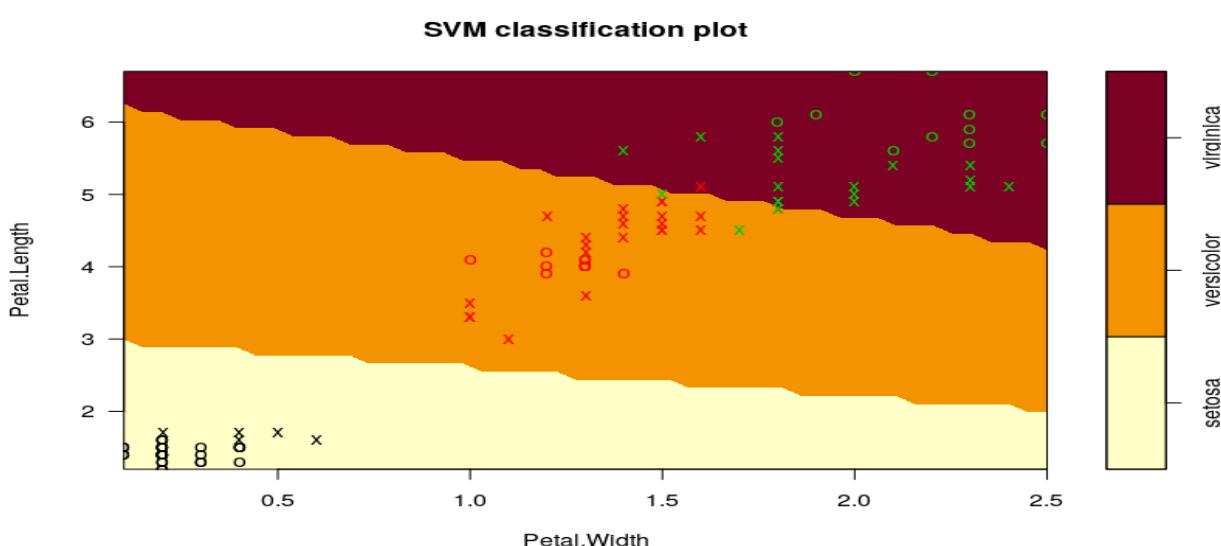


Figura 28: SVM a kernel lineare

Il plot e l'output generato mostrano molto bene i vettori di supporto (le X) che influenzano la determinazione degli iperpiani separatori delle tre classi.

Si evidenzia come essendo il campione di addestramento casuale, al ripetersi di tale operazione il numero di vettori di supporto per la parte in esame e per il generale, potrebbe variare sensibilmente; anche il costo qui fissato a 0.1 potrebbe causare delle modifiche di tale quantità: un costo maggiore determina una loro diminuzione.

```
> plot(svm.model.linear, train)
> tab.l <- table(Prediction=predict(svm.model.linear, train), Real=train$Species) ##confusion matrix
> tab.l
      Real
Prediction setosa versicolor virginica
setosa      37       0       0
versicolor   0      31       2
virginica    0       2      28
> 1-sum(diag(tab.l))/sum(tab.l)) ##tasso di classificazione errata 0.04
[1] 0.04
```

Figura 29: Confusion Matrix

Appare evidente come tale modello nel complesso si adatti bene alla realtà che cerca di modellare con un tasso di classificazione errata molto basso.

2.2.5 Algoritmi di classificazione a confronto

Si confrontino ora differenti tipologie di modelli; i modelli *mod1* e *mod2* qui riportati ne mostrano un pratico esempio, tuttavia essendo entrambi con un kernel di tipo lineare non ne rappresentano l'unico modo.

```
65  ##confronto fra piu' modelli lineari
66  sp = sample(150, 100) ##in 150 vado a prendere un sottinsieme de 100
67
68  train = iris[sp, c("Petal.Length", "Petal.Width", "Sepal.Length", "Sepal.Width", "Species")]
69  test = iris[-sp, c("Petal.Length", "Petal.Width", "Sepal.Length", "Sepal.Width", "Species")]
70
71  model1 = svm(formula = Species~., data = train, kernel = 'linear')
72  model2 = svm(formula = Species~ Petal.Width + Petal.Length, data = train, kernel = 'linear')
73
74  summary(model1) ##22 vettori di supporto
75  summary(model2) ##26 vettori di supporto
76
77  tab.mod1 <- table(Prediction=predict(model1, train), Real=train$Species) ##confusion matrix
78  tab.mod1
79  1-sum(diag(tab.mod1))/sum(tab.mod1)) ##tasso di classificazione errata 0.04
80
81  tab.mod2 <- table(Prediction=predict(model2, train), Real=train$Species) ##confusion matrix
82  tab.mod2
83  1-sum(diag(tab.mod2))/sum(tab.mod2)) ##tasso di classificazione errata 0.06
84
85  ##entrambi sono modelli solidi, comunque il primo e' meglio
86
75:41 (Top Level) <
Console Terminal ×
~/Scrivania/stageZucchetti/stage/SVM/ ↗
setosa versicolor virginica
```



```
> tab.mod1 <- table(Prediction=predict(model1, train), Real=train$Species) ##confusion matrix
> tab.mod1
      Real
Prediction setosa versicolor virginica
setosa      33       0       0
versicolor   0      29       1
virginica    0       3      34
> 1-sum(diag(tab.mod1))/sum(tab.mod1)) ##tasso di classificazione errata 0.04
[1] 0.04
> tab.mod2 <- table(Prediction=predict(model2, train), Real=train$Species) ##confusion matrix
> tab.mod2
      Real
Prediction setosa versicolor virginica
setosa      33       0       0
versicolor   0      30       4
virginica    0       2      31
> 1-sum(diag(tab.mod2))/sum(tab.mod2)) ##tasso di classificazione errata 0.06
[1] 0.06
>
> ##entrambi sono modelli solidi, comunque il primo e' meglio
```

Figura 30: Confronto modelli SVM a kernel lineari

Esistono infatti altri tipi di kernel per le SVM. Tuttavia prima di affrontare tale tematica è necessario fare una breve regressione verso le **SVM lineari**. Formulate da Vapnik nel 1963 (vedi sezione precedente), esse si mostrano infatti praticabili solo quando il set di dati di cui si dispone risulta essere linearmente separabile.

Domanda:

Se questa caratteristica non fosse rispettata?

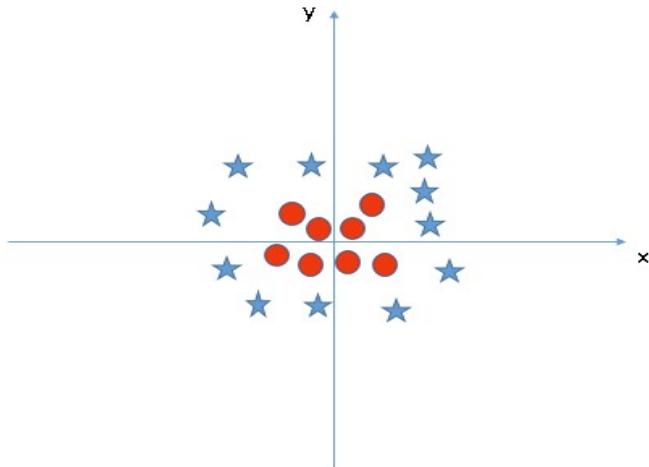


Figura 31: Dati non linearmente separabili

Ecco entrare in gioco il **Kernel Trick** (trucco del Kernel) grazie a cui si possono modellare modelli non lineari.

Nello specifico le **SVM non lineari** utili proprio per risolvere problemi di tale natura, permettono trasformando lo spazio, passo da R^2 a R^3 , di definire un prodotto interno grazie a cui non è più necessario mappare l'intero dominio $\Phi(x)$. Tutti i vettori diventano dunque di supporto e grazie ad una funzione kernel che fa il prodotto scalare fra il vettore d'esempio (di supporto) e quello di test, si riesce a capire quanto simile sia quest'ultimo al primo. La terza dimensione introdotta avrà inoltre la seguente forma:

$$z = x^2 + y^2 \text{ (equazione del cerchio)}$$

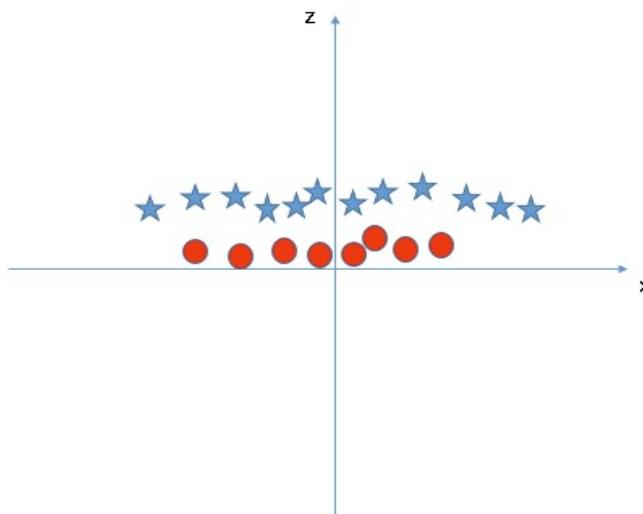


Figura 32: Visione di una parte del piano trasformato

Quindi applicare la funzione Kernel significa calcolare il prodotto scalare per la terza dimensione z.

Dovendo fare comunque tale operazione con ciascun vettore di supporto tale calcolo in alcuni casi risulta comunque eccessivamente troppo complesso (basti pensare alle predizioni attraverso frasi SQL), tuttavia rappresenta una buona strategia per risolvere problemi scomodi che richiederebbero certamente più tempo se affrontati in modo diverso.

Le SVM possono essere a kernel: lineare (esempio fatto con R), polinomiale, gaussiano, RBF, sigmoide.

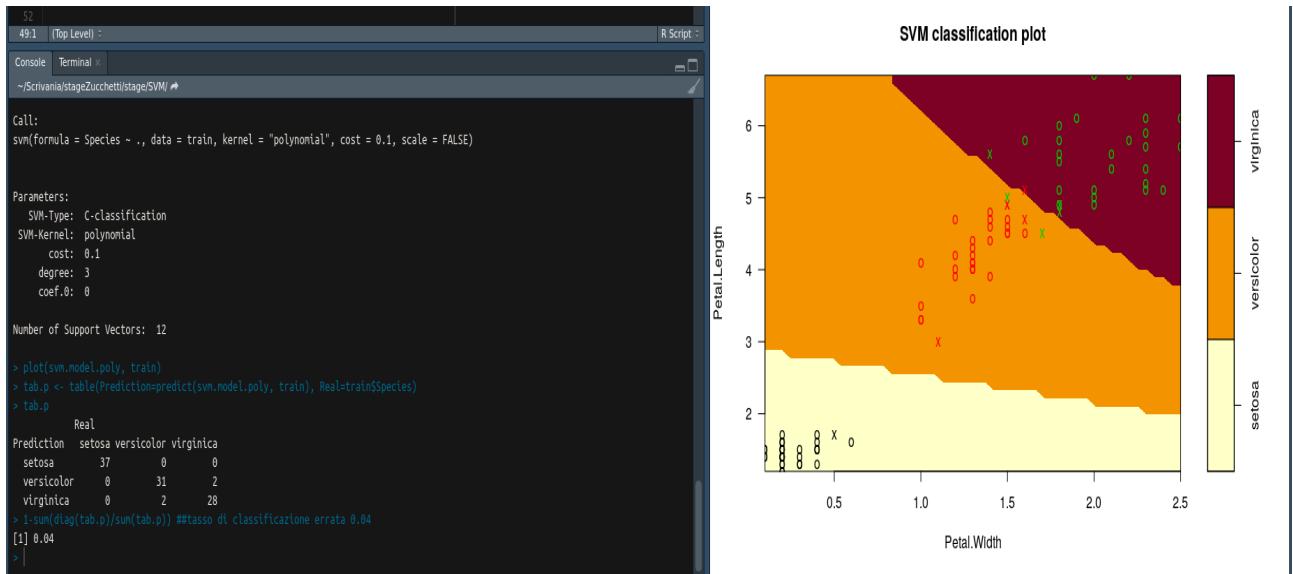


Figura 33: SVM a kernel polinomiale

Domanda:

I metodi che impiegano il kernel risultano essere sempre la scelta migliore?

Dipende da qual'è il problema che si sta cercando di risolvere; ci saranno casi in cui le SVM lineari saranno la scelta più adatta, altri dove si opterà per SVM non lineari, e altri ancora dove il confine risulterà più labile, lasciando quindi al Data Scientist la scelta.

A tal proposito una giusta interpretazione degli indici di valutazione del modello come la *Confusion Matrix* o la *Probabilità di classificazione errata* sono di certo da non trascurare; tuttavia può non essere sufficiente una mera valutazione dei dati oggettivi:

È necessario sapersi calare nella realtà che si vuole saper prevedere in modo da saperne valutare costi e benefici attesi.

A tal proposito si presenta a seguire un esempio; tale discorso viene qui fatto per i metodi di classificazione, nulla vieta di trattarlo comunque più trasversalmente per ogni algoritmo di *Machine Learning*.

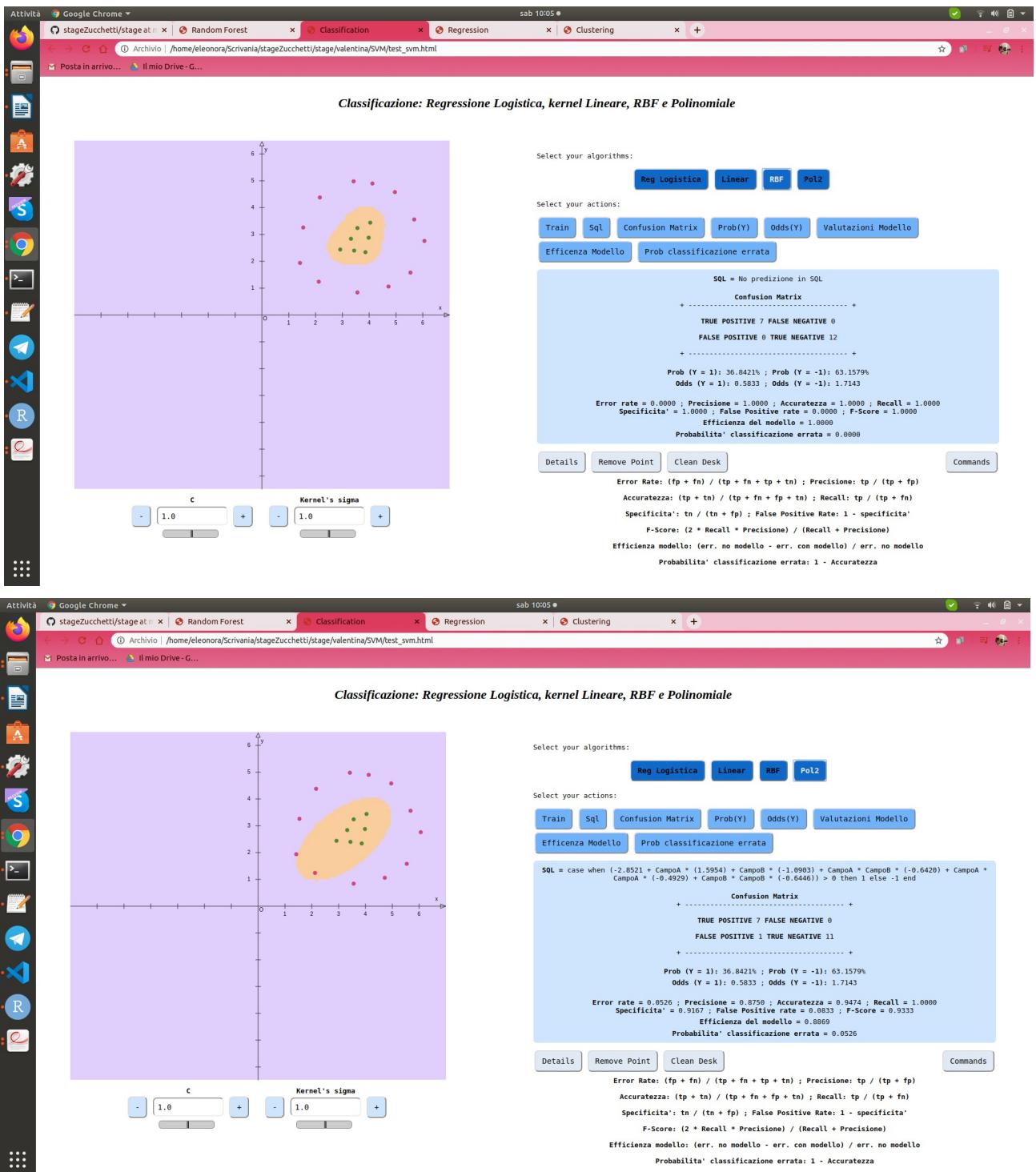


Figura 34: Confronto modelli SVM – kernel RBL e polinomio 2 lineare

Nelle immagini sopra mostrate si hanno:

- SVM a kernel RBF (non lineare);
- SVM lineare di grado due.

Cosa succede? Quali sono le considerazioni che si possono fare osservando i due grafici?

Innegabile è la capacità degli algoritmi a kernel RBF di adattarsi alla realtà che si sta cercando di modellare. I puntini verdi vengono infatti tutti classificati correttamente, ma non solo: se fossero stati inseriti ulteriori punti verdi in qualche altra zona del grafico essi sarebbero stati anch'essi considerati correttamente.

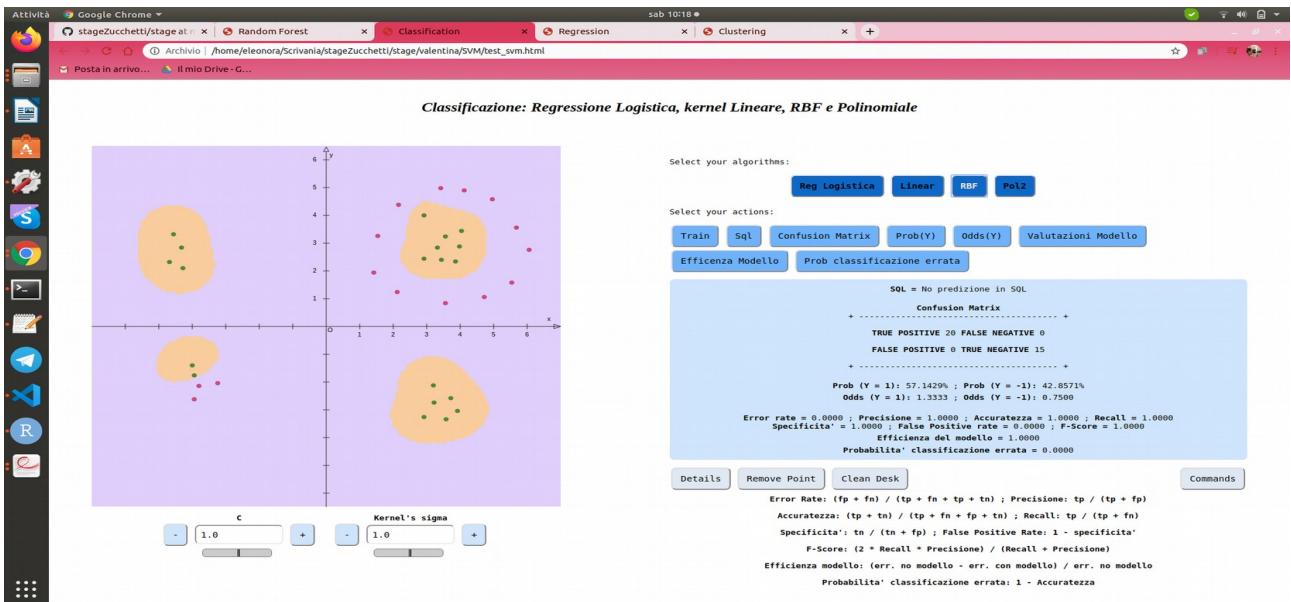


Figura 35: SVM a Kernel RBF

Questo è dovuto alla sua capacità di creare una vera e propria nuvola attorno ai punti prevedendoli perfettamente.

Nel caso del SVM lineare di grado 2, anch'essa presenta una buona capacità di previsione con però una piccola variazione: il cerchio che individua converge sempre verso l'origine. Tale problema si deve al fatto che i dati in questo caso non sono stati normalizzati (impiegare metodo del baricentro); ecco che se fatto al centro tale tiratura scompare.

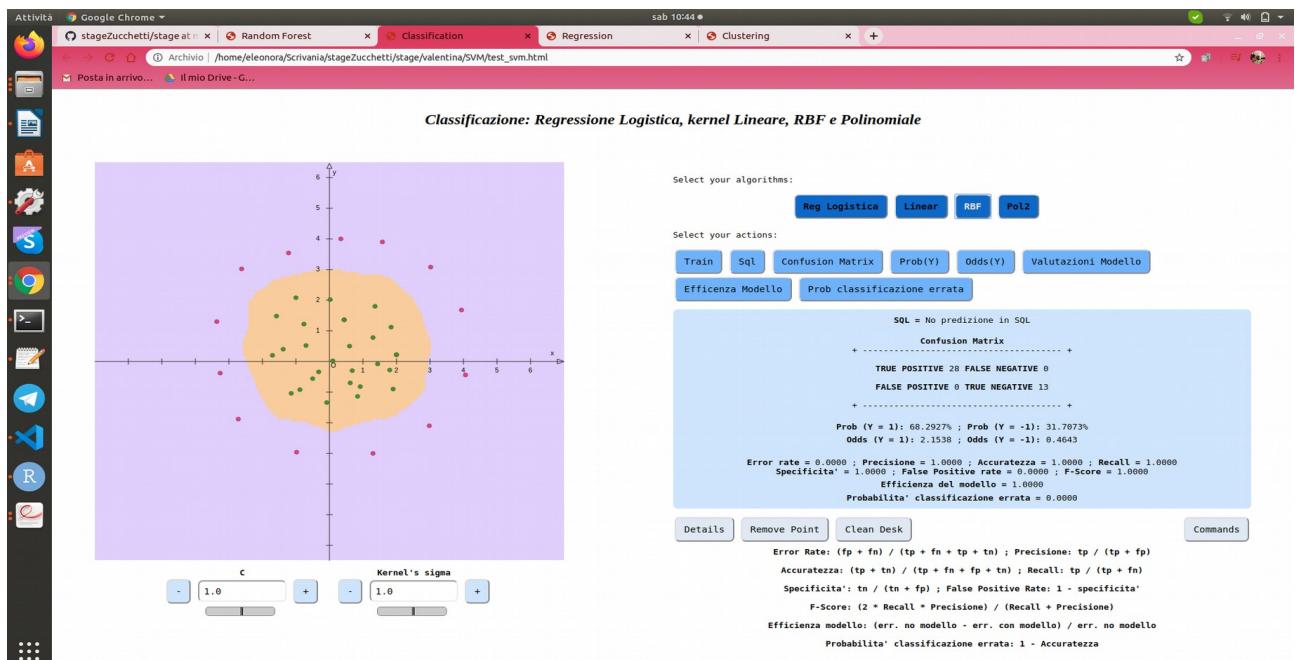


Figura 36: SVM di grado 2 lineare

Questa problematica si sarebbe potuta ovviare se si fosse impiegato un modello a kernel polinomiale, scelta che non è stata fatta perché risulta essere un tipo di modello, per il caso in questione, troppo complesso che non avrebbe permesso la predizione attraverso una semplice frase SQL (come si vede a destra per l'algoritmo RBF non è stata implementata).

Molto importante inoltre per quanto riguarda i metodi kernel, potrebbe risultare il settaggio di particolari parametri come la C ed il Σ (vedi foto sopra); nello specifico:

- **C:** misurazione delle interferenze. È quindi un parametro attraverso cui l'SVM pesa le classificazioni errate (peso dei punti che sfuggono alla classificazione). Il suo valore può presentarsi:
 - *piccolo*, dove il margine si allarga in quanto vengono accettati gli errori;
 - *grande*, dove il margine si restringe in quanto si cerca di evitare gli errori di classificazione.
- **Sigma:** quanto pesa la distanza gaussiana; anche qui se:
 - *piccola*, significa che l'influenza degli elementi diminuisce e che tutti gli elementi sono nel vettore di supporto;
 - *grande*, significa che l'influenza dei punti arriva più lontano (vi sono meno punti nel vettore di supporto).

Conclusione: È molto importante esaminare in modo critico i dati su cui si lavora; comprenderne le regole di natura sottostanti che ne determinano il comportamento, e fare le giuste valutazioni anche in base ai risultati che si vogliono ottenere.

2.2.6 Trasformazioni Box Cox: Dataset Cars

Si passi ora dataset: *Cars*.

Non essendo mai stato esaminato prima (neppure con *Orange Canvas*) se ne include una breve introduzione iniziale così da avere maggiormente chiara la situazione che si va ad esaminare. I dati *Cars* riguardano la distanza percorsa da un auto, che viaggia ad una certa velocità prima di fermarsi. La distanza è espressa in piedi, la velocità in miglia orarie. Sono 50 osservazioni che risalgono agli anni venti.

```
> summary(cars)
   speed          dist
  Min.   : 4.0   Min.   :  2.00
  1st Qu.:12.0  1st Qu.: 26.00
  Median :15.0  Median : 36.00
  Mean   :15.4  Mean   : 42.98
  3rd Qu.:19.0  3rd Qu.: 56.00
  Max.   :25.0  Max.   :120.00
> dim(cars)
[1] 50  2
```

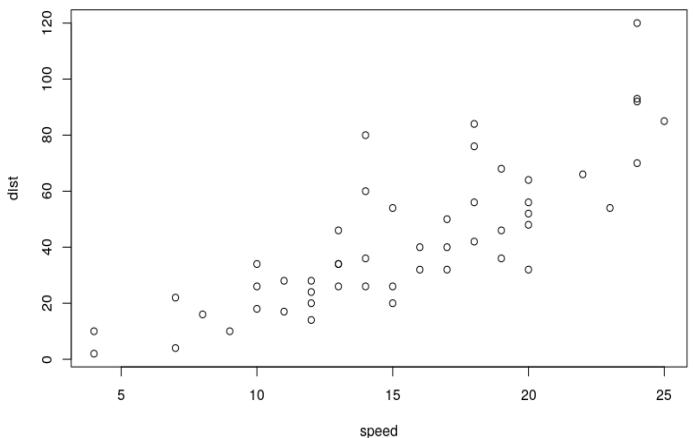


Figura 37: Informazioni generali Cars

Ciò che si nota subito è la presenza di un tipico caso di eteroschedasticità (forma a triangolo dei dati sul grafico di destra); risulta quindi questo un buon caso studio per provare ad applicare le trasformazioni degli algoritmo Box- Cox.

Equazione del modello:

$$dist = speed + \xi$$

Ciò che si vuole è predire la *distanza* avendo come unico regressore: la variabile *velocità* (modello di regressione lineare semplice).

```
modello <- lm(dist ~ speed, data=cars)
summary(modello)
plot(modello)
```

Si allegano alcuni dati forniti da R sul modello prescelto, e alcuni plot sulla distribuzione delle osservazioni.

Si evince che:

- R^2 risulta un valore abbastanza buono collocandosi su un $\simeq 65\%$;
- *speed* si dimostra essere un predittore molto significativo (essendo anche l'unico).

Questo permette di rifiutare l'ipotesi che non vi sia nessuna dipendenza fra il predittore ed il predetto (ossia l'ipotesi nulla).

```

> modello <- lm(dist ~ speed, data=cars)
> summary(modello)

Call:
lm(formula = dist ~ speed, data = cars)

Residuals:
    Min      1Q  Median      3Q     Max 
-29.069 -9.525 -2.272  9.215 43.201 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -17.5791    6.7584  -2.601   0.0123 *  
speed        3.9324    0.4155   9.464 1.49e-12 *** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438 
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12

```

Figura 38: Dati del modello di Regressione lineare semplice

Tuttavia, visto anche il grafico in *Figura 37*, è alquanto improbabile che tale relazione così com'è sia lineare, rendendo in tal modo non abbastanza qualitativo il modello di regressione applicato.

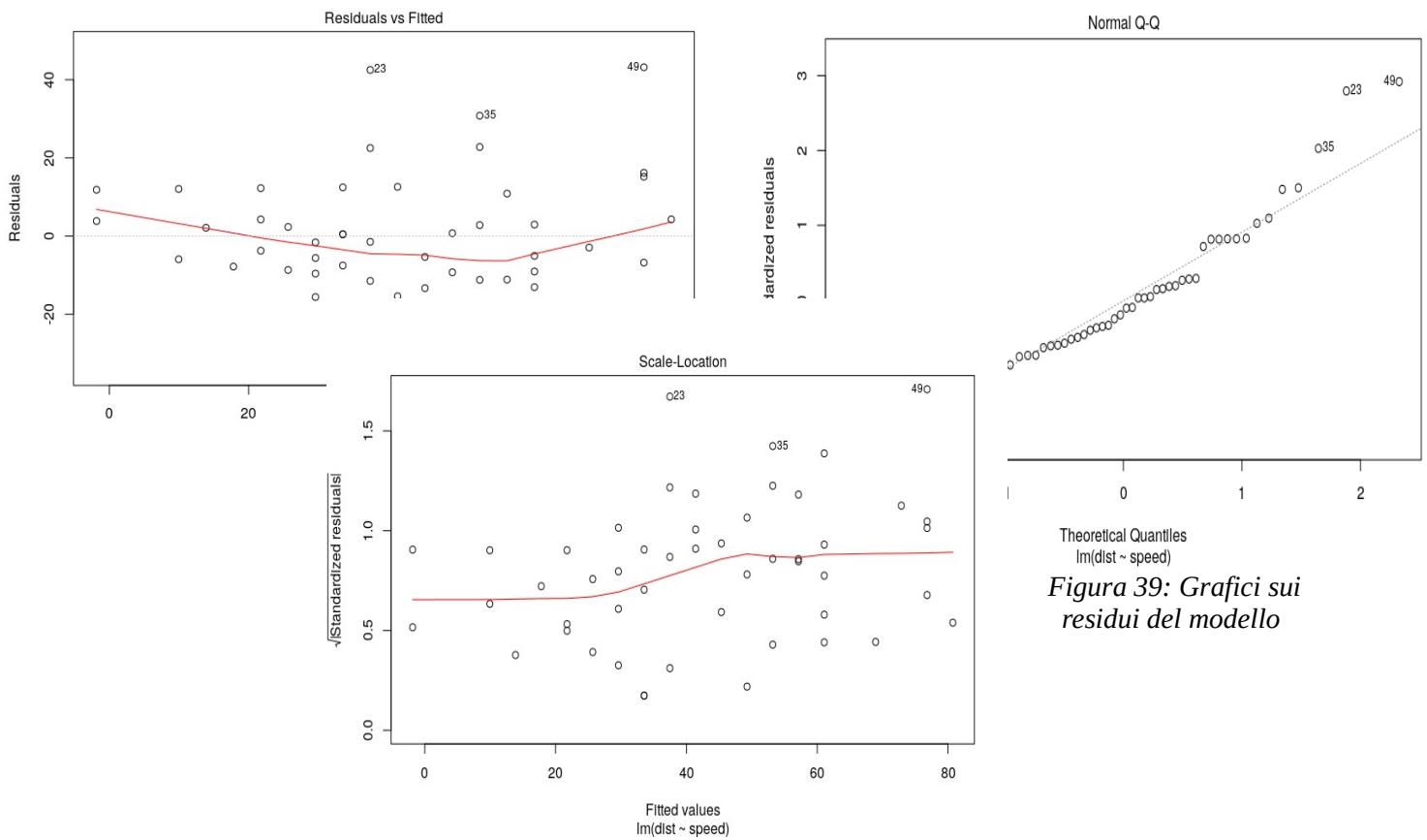


Figura 39: Grafici sui residui del modello

Quanto affermato dai grafici appena sopra non stupisce dunque:

- i dati non rispettano il principio di normalità (Q-Q plot);
- la distribuzione non risulta uniforme, segno evidente che i legami non sono pienamente lineari (Residual Fitted);
- viene meno il principio di omoschedasticità (Scale Location);
- vi sono alcuni valori anomali molto distanti che si ripetono: come il 23, 35 e 49. Tuttavia facendo un confronto sempre con l'ausilio di R, con le distanze di Cook essi sembrano rientrarvi. Probabilmente quindi sono valori anomali ma la loro considerazione o meno non dovrebbe portare a modifiche sostanziali del modello in questione.

Dopo tali preamboli applichiamo quindi il metodo *Box Cox*, trasformiamo dunque la variabile dipendente rendendola più idonea ad un modello lineare.

Esso si basa sull'individuazione di un valore λ grazie a cui il modello diventa nella forma:

$$\frac{\text{dist}^{\lambda} - 1}{\lambda} = \text{speed} + \xi$$

I dati vengono quindi trasformati impiegando le proprietà dell'elevazione a potenza, potendo così lavorare su dati più normali e con una distribuzione più stabile.

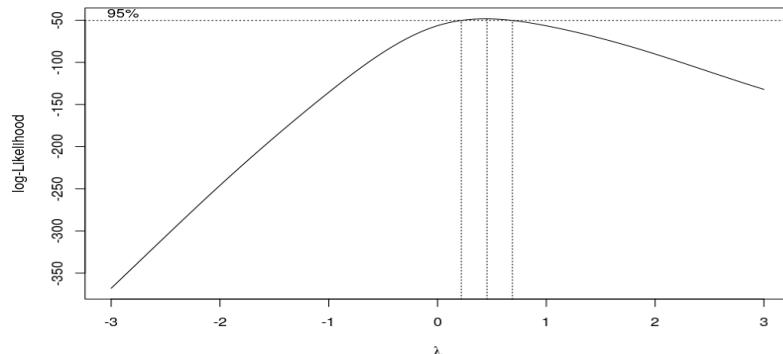
La formula generale è

$$y_{tras} = \frac{y^\lambda - 1}{\lambda} \text{ per } \lambda \neq 0$$

$$y_{tras} = \log(y) \text{ per } \lambda = 0$$

Il parametro λ può assumere qualunque valore da -3 a +3. È dunque la semplice elevazione a potenza del predetto a permettere la trasformazione del modello; si evidenzia inoltre come per valori < -2 e > 2 ha poco senso applicare Box Cox.

```
bc <- boxcox(modello, lambda = seq(-3, 3)) ##trasformata, fornisce lambda
lambda <- bc$x[which(bc$y==max(bc$y))] ##0.4545
summary(lambda)
```



```
> lambda <- bc$x[which(bc$y==max(bc$y))] ##0.4545
> summary(lambda)
   Min. 1st Qu. Median Mean 3rd Qu. Max.
0.4545 0.4545 0.4545 0.4545 0.4545 0.4545
```

Figura 40: Funzione λ

```
trasf <- (((cars$dist)^lambda)-1)/lambda ##trasformazione della variabile dipendente
model.inv <- lm(trasf ~ speed, data=cars) ##nuovo modello
```

```
> trasf <- (((cars$dist)^lambda)-1)/lambda
>
> model.inv <- lm(trasf ~ speed, data=cars)
> summary(model.inv)

Call:
lm(formula = trasf ~ speed, data = cars)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.4737 -1.1661 -0.3283  0.9386  5.3205 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 0.89905   0.82243   1.093    0.28    
speed       0.55029   0.05056  10.883 1.48e-14 ***  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 1.872 on 48 degrees of freedom
Multiple R-squared:  0.7116,    Adjusted R-squared:  0.7056 
F-statistic: 118.4 on 1 and 48 DF,  p-value: 1.475e-14
```

Figura 41: Valutazione del modello trasformato

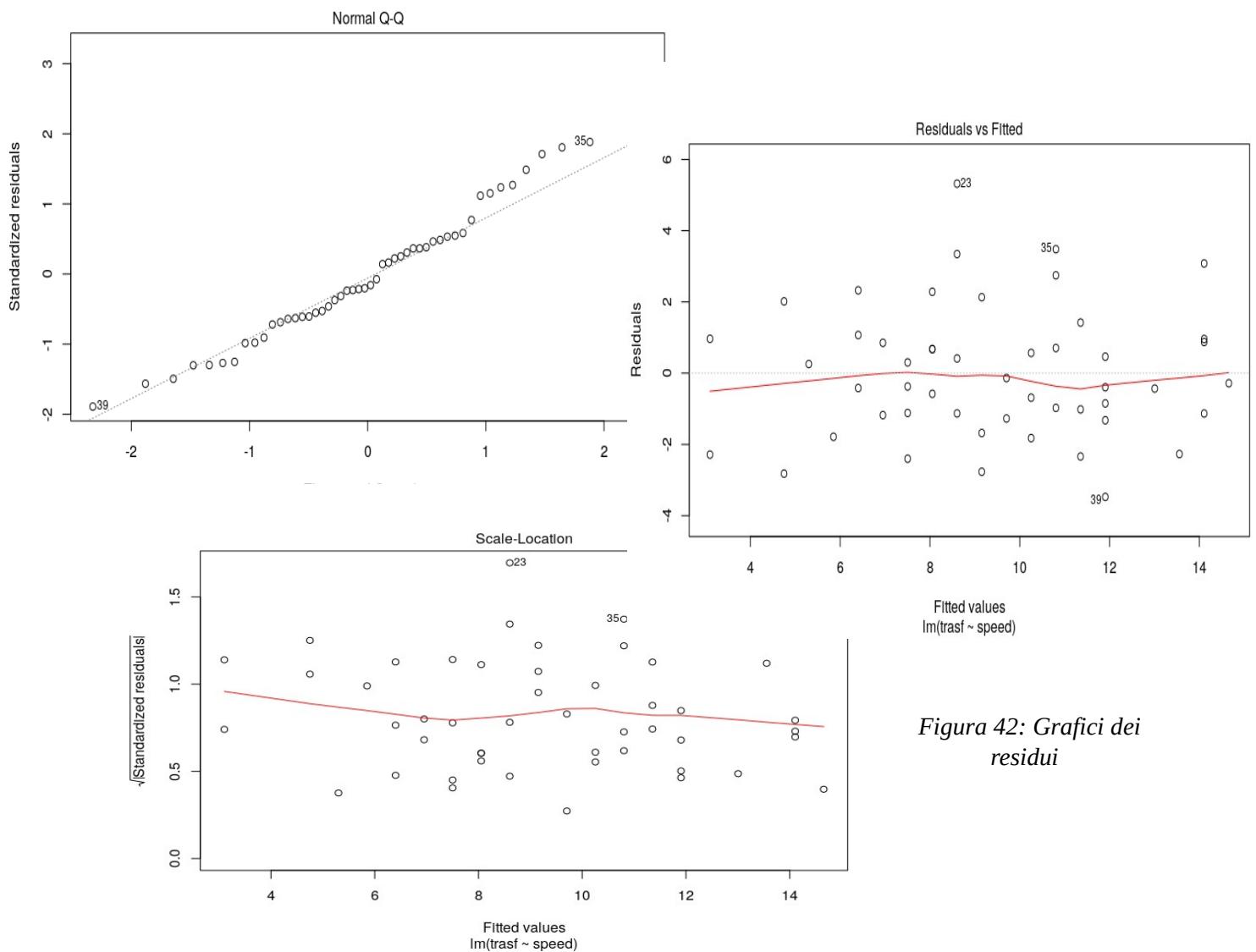


Figura 42: Grafici dei residui

Conclusione:

Applicando una semplice trasformazione di potenza a ciò che si vuole predire (trasformazione sui dati), si ottengono dunque miglioramenti non indifferenti:

- la devianza spiegata dal modello è cresciuta del 6%;
- le osservazioni presentano una maggiore linearità, omoschedasticità e normalità.

2.3 Regressione Logistica

Analogamente alle Support Vector Machine anche la *Regressione Logistica* rappresenta un metodo di classificazione supervisionata.

Tale algoritmo permette di generare un risultato che, di fatto, rappresenta la *probabilità* che un dato valore, fornito in ingresso, appartenga ad una determinata classe.

Esempio: Nei problemi di **regressione logistica binomiale**, la probabilità che l'output appartenga alla classe 0 sarà P , mentre che appartenga alla classe 1 sarà $1-P$ (con $0 \leq P \leq 1$).

Tale algoritmo, rimanendo pur sempre un metodo di regressione, consiste quindi in un'analisi predittiva al fine di misurare la relazione esistente fra la variabile dipendente (il predetto) e le variabili indipendenti (i predittori), stimandone la probabilità mediante una funzione logistica. Successivamente, per poter operare effettivamente una previsione queste probabilità sono trasformate in operatori binari (0 o 1).

A questo punto il risultato della previsione viene quindi assegnato alla classe di appartenenza, in base alla classe più prossima (arrotondamento).

Esempio: Con un risultato ottenuto dalla funzione logistica pari a 0.8 si conclude che si è generata una classe positiva; il punto verrà quindi assegnato alla classe 1, rappresentando così l'esito della classificazione (classe 0 invece, nel caso di un valore < 0.5).

L'immagine ne illustra meglio i passaggi.



Figura 43: step della Regressione Logistica

La regressione logistica si avvale quindi dell'omonima funzione logistica per operare la classificazione dei valori che si vogliono predire.

Chiamata anche funzione sigmoidea, la funzione logistica è una curva a forma di S che può prendere qualsiasi numero reale e mapparlo in un valore compreso tra 0 e 1, estremi esclusi.

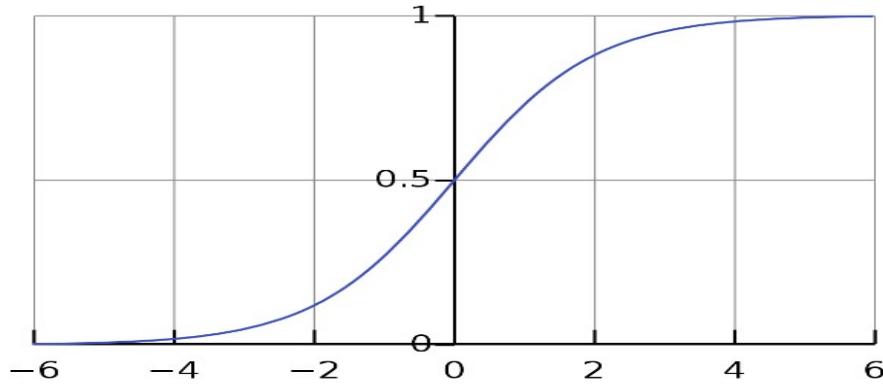


Figura 44: Funzione Logistica

L'equazione della regressione logistica sarà la seguente:

$$y = \frac{e^{(b_0+b_1*x)}}{1+e^{(b_0+b_1*x)}}$$

dove:

- X = valore di input;
- b_0 e b_1 = coefficienti dei valori di input (numeri reali costanti);
- Y = valore output da predire.

In generale, i valori di input vengono combinati linearmente usando pesi o coefficienti (chiamati coefficienti Beta) per prevedere un valore di output. Tali coefficienti vengono stimati secondo il [metodo della verosimiglianza](#). Essa permette di ottenere coefficienti di regressione la dove la probabilità di ottenere i dati che abbiamo osservato è massima, ecco che si possono individuare, per i coefficienti, quei valori che minimizzano l'errore di previsione del modello.

A seguire vengono mostrati alcuni modelli di questo tipo.

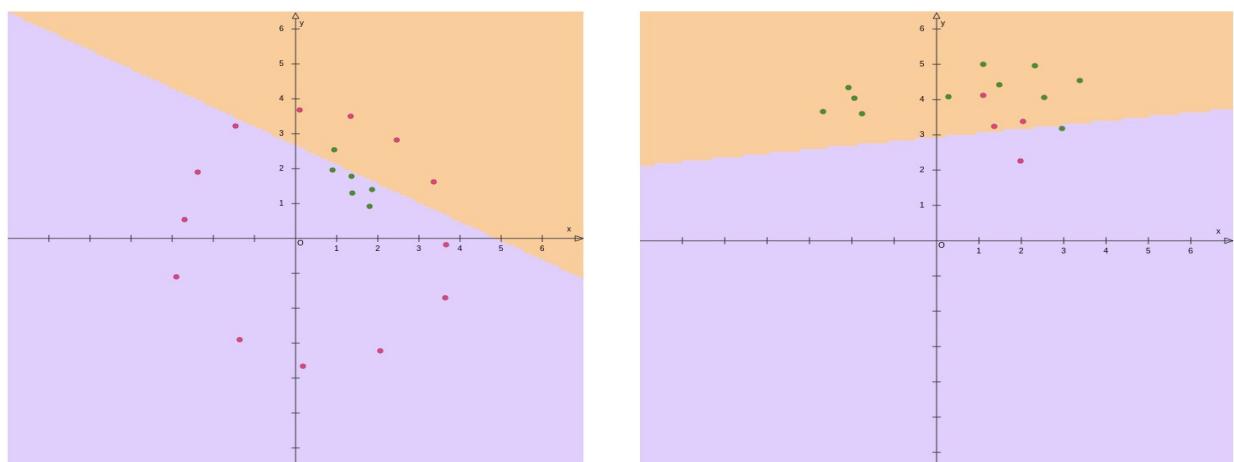


Figura 45: Classificazione logistica errata

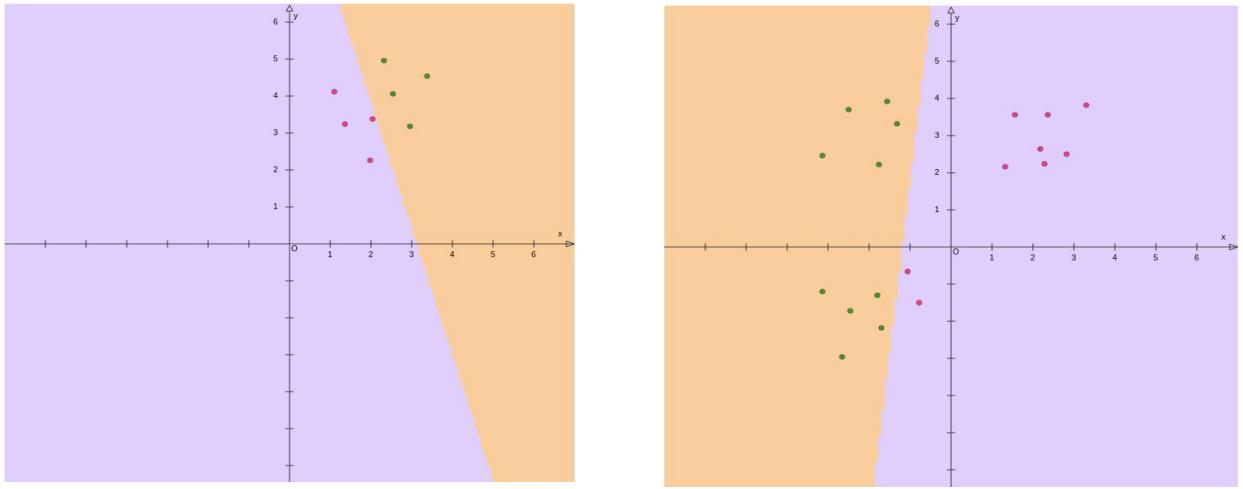


Figura 46: Classificazione logistica buona

Senza dubbio ciò che si comprende dalle immagini, è che anche se si tratta di un modello di classificazione, lavora pur sempre con un modello di regressione; si adatterà quindi poco a lavorare su set di dati non linearmente separabili, per poi mostrare tutte le sue capacità in caso contrario.

Non sono infatti affatto pochi i vantaggi offerti dall'algoritmo:

- modella la probabilità che un evento si verifichi a seconda dei valori delle variabili indipendenti (categoriali o numeriche);
- stima la probabilità che si verifichi un evento per un'osservazione scelta casualmente, rispetto alla probabilità che l'evento non si verifichi;
- prevede l'effetto di una serie di variabili su una variabile di risposta binaria;
- classifica le osservazioni stimando la probabilità che un'osservazione si trovi in una particolare categoria.

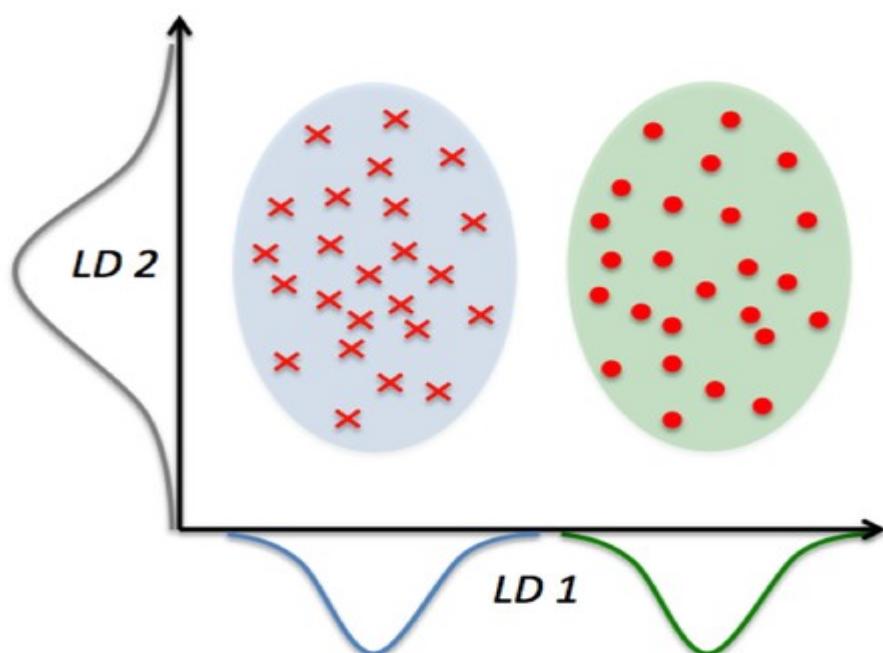
Conclusione: Analogamente a quanto già detto negli algoritmi precedenti, anche in questo caso risulta poco realistico pensare di applicare tale modello senza una corretta visione d'insieme. Le performance che si otterranno dipenderanno sempre e comunque in primo luogo dalla natura stessa dei dati sui cui si sta lavorando.

2.4 Linear Discriminant Analysis (LDA)

L'algoritmo LDA, acronimo di *Linear Discriminant Analysis*, viene impiegato per risolvere problemi di classificazione predittiva; cerca inoltre di risolvere le debolezze della regressione logistica.

Nel dettaglio le carenze che cerca di sanare sono:

- *problemi con più di due classi*; la regressione logistica è intesa per problemi di classificazione binaria o di due classi. Può essere estesa alla classificazione multi-classe, ma è raramente utilizzata a questo scopo;
- *instabilità nel caso di classi ben separate*; la regressione logistica può diventare instabile quando le classi sono ben separate;
- *instabilità con pochi esempi*; la regressione logistica può diventare instabile quando ci sono pochi esempi da cui stimare i parametri.



Disegno 47: Linear Discriminant Analysis

La *Linear Discriminant Analysis* affronta ciascuno di questi punti e rappresenta il metodo lineare per problemi di classificazione multi-classe (nulla vieta tuttavia di provarla anche nel caso di classificazioni binarie).

Generalizzazione del **discriminante lineare di Fisher**, lo utilizza al fine di individuare una combinazione lineare di caratteristiche, in grado di caratterizzare o separare due o più classi di oggetti o eventi.

Questo metodo attua una **riduzione dimensionale** in modo da rendere le classi facilmente separabili evitando in tal modo un eccesso di *overfitting* (adattamento) e assicurando una riduzione dei costi di calcolo. Ecco che in combinazione col metodo **PCA** (vedi *Appendice*), aiuta a ridurre set di dati ad alta dimensione trasportandoli su uno spazio di dimensione inferiore.

L'analisi discriminante viene inoltre utilizzata quando i gruppi sono noti a priori.

L'obiettivo dell'algoritmo LDA è proiettare uno spazio di caratteristiche su un sottospazio minore, mantenendo le informazioni discriminatorie di classe. In altre parole, **trovare quel vettore che al meglio permette di dividere la rappresentazione delle classi.**

L'implementazione standard del modello presuppone inoltre una **distribuzione gaussiana** delle variabili di input. A livello intuitivo esso infatti risulta molto simile all'*Expectation Maximization* (vedi paragrafo 3.5.4), attendendosi delle gaussiane quando vi sono tante cause differenti che si sommano fra loro. Immagina dunque che i dati seguano delle distribuzioni normali caratterizzate da una concentrazione dei punti al centro.

Dire tuttavia che le classi sono delle gaussiane; è riduttivo in quanto:

A differenza del Expectation Maximization che individua solo i vari raggruppamenti gaussiani, il Linear Discriminant Analysis è in grado di predire anche la colorazione delle classi che individua.

Conclusione: LDA è molto simile alla regressione logistica: entrambe possono essere utilizzate per le stesse domande di ricerca. La regressione logistica fra le due risulta essere quella che presenta un numero inferiore di presupposti e restrizioni, d'altro canto LDA si dimostra invece più potente. Ovviamente questo non accade come regola a priori: **dipenderà dalla natura stessa del caso in esame.** Inoltre, a differenza della regressione logistica, la Linear Discriminant Analysis può essere utilizzata con campioni di piccole dimensioni.

Nota: È stato dimostrato che quando le dimensioni del campione sono uguali e varianza e covarianza si mostrano omogenee, il metodo LDA è più accurato. Nonostante tutti questi vantaggi tuttavia la regressione logistica tende a restare la scelta più comune, in quanto i presupposti dell'analisi discriminante vengono raramente soddisfatti.

2.5 Random Forest

Classe di algoritmi sempre di tipo supervisionato, denominate anche *foreste casuali*, devono la loro efficacia non tanto per la singola capacità predittiva di ciascun albero decisionale, ma grazie alla combinazione di molti alberi in un unico modello.

Alla base le Random Forest impiegano il metodo Bootstrap Aggregation (o Bagging in breve): metodo ensemble semplice e molto potente.

Nel dettaglio:

Un metodo ensemble è una tecnica che combina le previsioni di più algoritmi di apprendimento automatico per fare previsioni più accurate rispetto a qualsiasi singolo modello.

È proprio tale filosofia, come detto sopra, che sancisce la potenza di questo metodo: punta alla quantità. Semplice ma efficace rappresenta un insieme di cose diverse (e.g: orchestra).

Il bagging consiste in una procedura generale che può essere utilizzata per ridurre la varianza di quegli algoritmi che per loro natura ne presentano una elevata.

Gli alberi decisionali inoltre risultano essere **molto sensibili ai dati specifici su cui vengono formati**, ecco che se i dati di train vengono modificati l'albero decisionale risultante può essere abbastanza diverso e, a loro volta, anche le previsioni possono essere piuttosto differenti.

Il bagging dunque è una procedura **Bootstrap** da applicarsi ad un algoritmo ad alta varianza (l'albero decisionale appunto); nello specifico esso consiste in una tecnica statistica di ricampionamento con reimmissione , che permette, non essendo a conoscenza della distribuzione statistica, comunque di:

- approssimare media e varianza di uno stimatore;
- costruire gli intervalli di confidenza;
- calcolare il p-value.

Esempio: Nel caso di 6 alberi decisionali con i seguenti esiti di previsione: rosa, azzurro, rosa, verde, rosa e rosa, quale sarebbe l'esito della predizione? Il risultato sarebbe **rosa**, essendo stata la classe selezionata con maggiore frequenza.

L'esito della predizione è dunque dato dal valore che maggiormente è stato selezionato dai vari modelli (l'unione fa la forza).

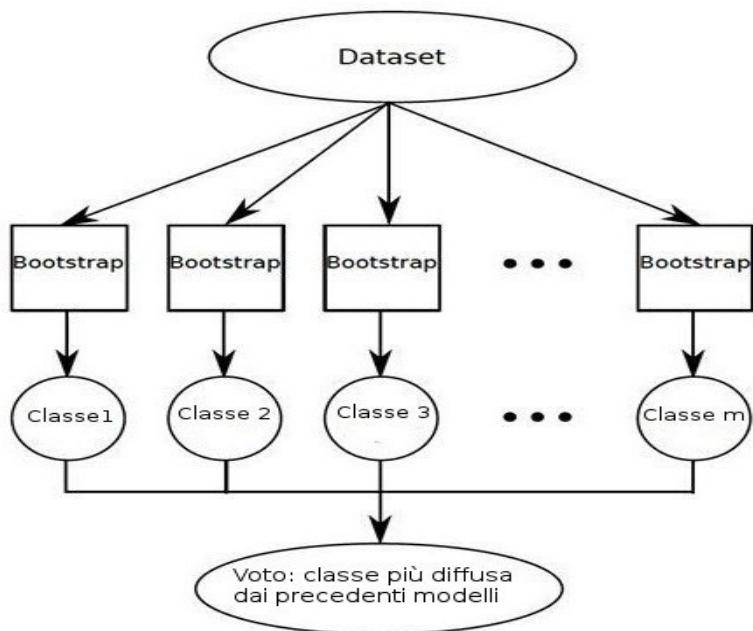


Figura 48: Esempio albero decisionale

Le Random Forest quindi combinano molti alberi decisionali in un unico modello: individualmente, le previsioni fatte dagli alberi decisionali potrebbero non essere accurate, ma combinate insieme, esse saranno in media più vicine al risultato reale.

Conclusione: Questo non significa che l'operato di un'insieme di individui sia da ritenersi sempre una scelta migliore rispetto al singolo. Semplicemente nel caso delle Random Forest si è preferito, invece di costruire un albero “superpotente”, puntare sull'operato di tanti piccoli elementi. Ecco che il risultato finale restituito da tale algoritmo, altro non è che il valor medio dei risultati ottenuti dai diversi alberi nel caso di un problema di **regressione**; o la classe restituita con maggiore frequenza nel caso la Random Forest sia stata utilizzata per risolvere un problema di **classificazione**.

2.5.1 Funzionamento dell'algoritmo

Tale modello deve parte del suo nome (*random = casuale*) alle seguenti sue caratteristiche:

- durante la fase di train vi è un campionamento casuale dei punti, grazie a cui si costruisco gli alberi: durante l'allenamento, ogni albero impara da un campione casuale di punti dati;
- durante la divisione dei nodi vengono considerati sottoinsiemi casuali di funzionalità: viene considerato solo un sottoinsieme di tutte le caratteristiche per suddividere ciascun nodo in ciascun albero decisionale (e.g: se ci sono 16 caratteristiche per ciascun nodo di ogni albero, saranno prese in considerazione solo 4 caratteristiche casuali per dividere il nodo).

Le Random Forest combinano centinaia o migliaia di alberi decisionali. Ciascuno di essi si allena su un insieme leggermente diverso di osservazioni e suddivide i nodi fra i vari alberi, considerando un numero limitato di caratteristiche.

Ecco alcune immagini che mostrano più chiaramente come questo “agglomerato” di alberi decisionali lavora.

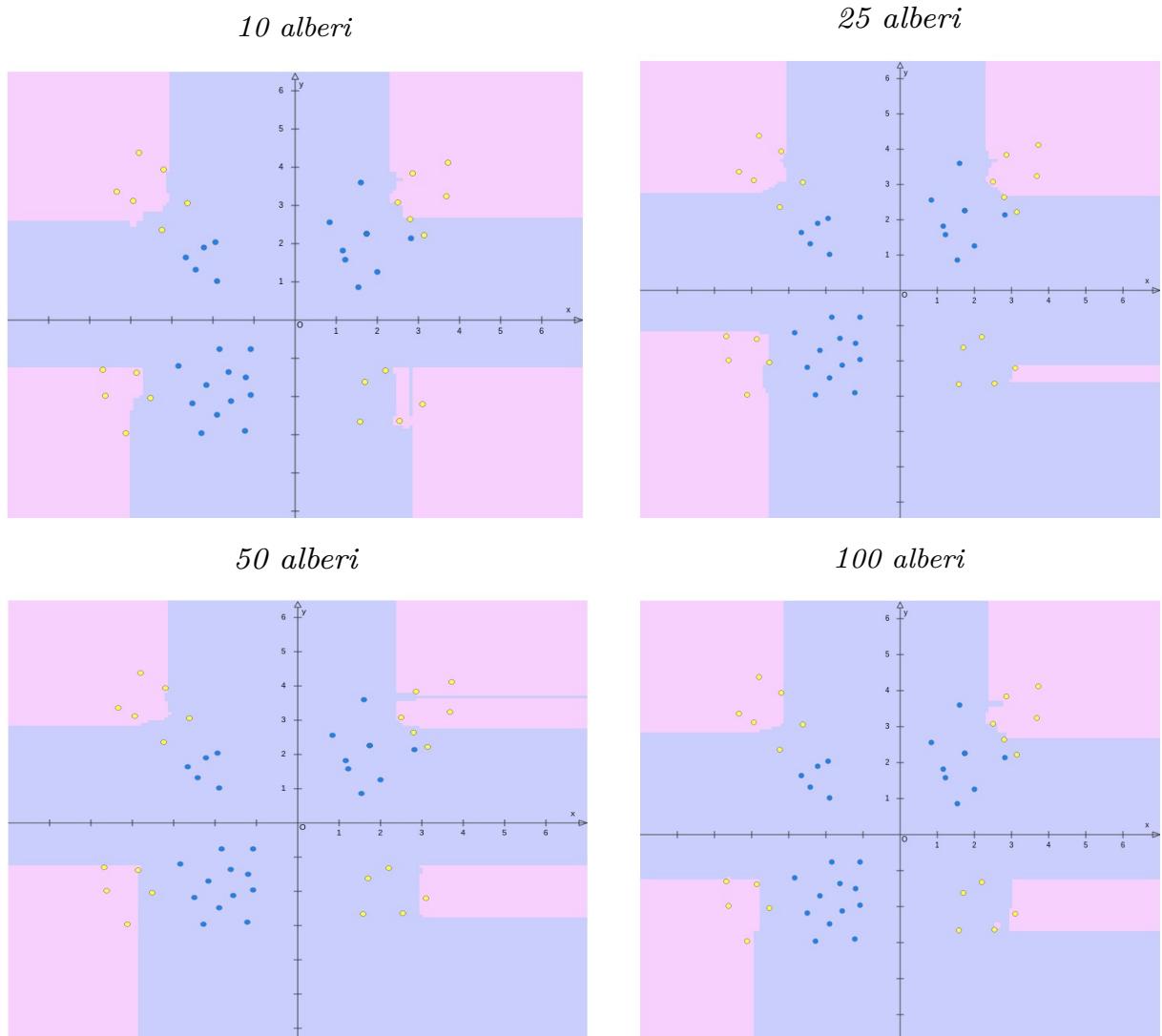


Figura 49: Random Forest a diversa numerosità

Come illustrato l'efficacia predittiva dell'algoritmo risulta essere molto buona, inoltre a livello puramente pratico il risultato con 100 alberi decisionali (numero di per sé molto elevato), non si discosta molto da quelli a numerosità inferiore.

Questo ci suggerisce che molto probabilmente non sta tanto nella quantità numerica il punto chiave delle Random Forest, piuttosto nel passaggio da:

singolo albero decisionale → “orchestra” di alberi decisionali

3. Algoritmi non supervisionati

3.1 Clustering: il cluster

Il *clustering* consiste in un insieme di tecniche per la selezione e il raggruppamento di oggetti in classi omogenee. Nello specifico esse si basano su delle misure di similarità, da intendersi come la distanza fra i punti sparsi in uno spazio multidimensionale.

Ecco che:

Un **cluster** non è null'altro che un semplice insieme di oggetti; esso si contraddistingue per la presenza di somiglianze fra gli elementi interni ad esso, e di diversità con gli elementi degli altri cluster.

3.2 Metodi di clustering

Prima di tutto, le tecniche di clustering presentano due visioni distinte:

- **metodi *bottom-up o agglomerativi* (*basso → alto*):** inizialmente ogni singolo elemento viene considerato come un cluster a sé; successivamente l'algoritmo provvederà ad unire fra loro i cluster più prossimi. Tale processo di unione continuerà fino al raggiungimento di una fra le seguenti soglie:
 - l'ottenimento di un numero prefissato di cluster;
 - il superamento da parte della distanza minima tra i cluster, di un certo valore;
 - l'ottenimento di uno specifico criterio statistico prefissato.
- **metodi *top-down o distintivi* (*alto → basso*):** all'inizio gli elementi vengono tutti racchiusi in un unico cluster; poi l'algoritmo inizierà a partizionare questo unico grande cluster in tanti cluster di dimensioni inferiori, man mano ottenendo gruppi sempre più omogenei. Anche qui il processo prosegue fino a che non viene soddisfatta una specifica regola di arresto, generalmente legata al raggiungimento di un numero prefissato di cluster.

3.3 Classificazione degli algoritmi di clustering

Esistono vari modi attraverso cui classificare le molteplici tipologie di algoritmi di clustering.

Una prima categorizzazione dipende dalla possibilità o meno da parte di un elemento, di venire assegnato a più cluster. In questo caso si distingue fra:

- *clustering esclusivo*: ogni elemento può essere assegnato ad uno e ad un solo gruppo (approccio *hard clustering*);
- *clustering non-esclusivo*: un elemento può appartenere a più cluster con gradi di appartenenza diversi (approccio *soft clustering* o *fuzzy clustering*).

Un'altra suddivisione delle tecniche di clustering tiene invece conto del tipo di algoritmo utilizzato per dividere lo spazio:

- **Hierarchical clustering:** i dati vengono organizzati attraverso una gerarchia di partizioni; essi si trovano a livelli di partizionamento differenti e sono rappresentabili mediante una struttura ad albero (dendrogramma), che mostra chiaramente i vari passi di accorpamento/divisione dei gruppi;
- **Partitional clustering:** la suddivisione dei dati fra i vari cluster cerca di ridurre il più possibile la dispersione all'interno del singolo cluster e, viceversa, incrementare quella fra cluster diversi. A differenza del punto sopra, qui i cluster si presentano tutti allo stesso livello di partizionamento. Dal punto di vista pratico, per stabilire l'appartenenza o meno di un punto ad uno specifico cluster, viene misurata la distanza fra il dato e il *centroide/medioide* del cluster (punto rappresentativo dell'insieme); il numero di gruppi della partizione è inoltre prestabilito, si è quindi già a conoscenza del numero di cluster che conterrà il risultato.

3.4 Hierarchical clustering

Nel dettaglio tale metodologia presenta la seguente prassi per eseguire l'identificazione dei cluster (visione agglomerativa):

- inizialmente ciascun elemento del data set forma un cluster separato;
- ad ogni iterazione, grazie alla soglia di similarità vengono accorpati man mano i cluster più simili;
- l'iterazione termina quando tutti gli oggetti considerati simili sono accorpati in un unico cluster.

Di seguito se ne mostra l'albero rappresentato sotto forma di *dendrogramma*.

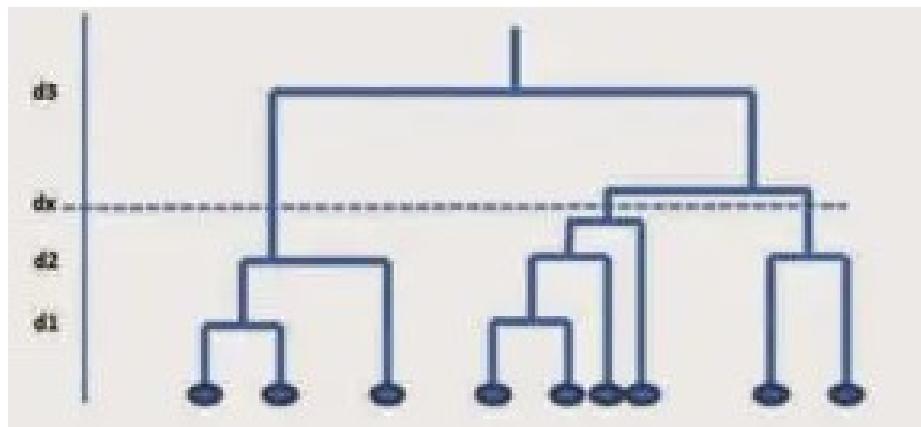
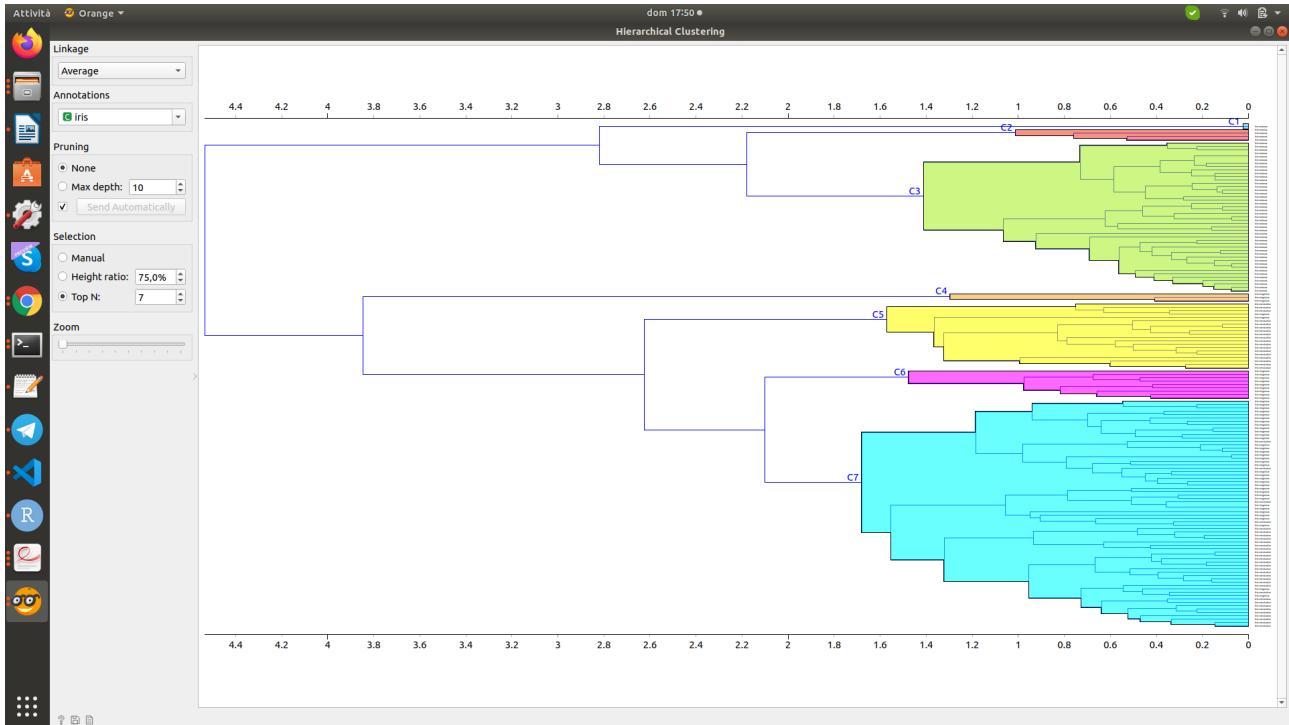


Figura 50: Dendrogramma Hierarchical clustering

La linea tratteggiata che taglia orizzontalmente l'albero va a stabilire il numero di cluster su cui andare a suddividere i dati.

Ecco un' ulteriore esempio.



L'immagine, prodotta grazie al software *Orange Canvas* sul dataset *Iris*, mostra come appaiano ben visibili le sette partizioni stabilite (fissate sulla barra di sinistra, ciascuna con una propria colorazione), e le conseguenti ramificazioni per ciascun cluster, esplicative del raggrupparsi dei vari dati ad altezze differenti.

L'algoritmo in questione è accurato, tuttavia presenta una scarsa efficienza dal punto di vista del calcolo. Non prevede inoltre una riassegnazione degli elementi ai cluster durante il passo iterativo (cosa che invece fa il K-Means).

3.5 Partitional clustering: analisi degli algoritmi

Nel presente paragrafo vengono trattati alcuni algoritmi *Unsupervised* di tipo partitional clustering.

Per ciascuno ne viene esaminato il comportamento, con i conseguenti pregi e difetti.

3.5.1 K-Means

L'algoritmo K-Means è da considerarsi quello maggiormente rappresentativo di tale categoria (si veda anche in *Appendice*).

In esso il numero di cluster tra cui suddividere i dati seguendo il criterio della similarità, è noto a priori prima dell'esecuzione dell'algoritmo stesso. Una volta attribuiti al cluster i dati prendono il nome di **data points**, che una volta analizzati costituiranno il set di dati analizzato dal K-Means.

La caratteristica fondamentale di tale strategia non supervisionata è costituita dal fatto che per ogni gruppo viene posizionato in modo casuale un **centroide** (dato fittizio) che funge da rappresentante del cluster.

Essendo un algoritmo di tipo iterativo esegue ripetutamente una serie di passi:

- **inizializzazione:** definizione dei parametri di input (numero di centroidi) che eseguono l'algoritmo. In tal modo si stabiliscono i raggruppamenti a cui il dataset sarà sottoposto;
- **assegnazione al cluster:** assegnazione di ogni data points al cluster (o centroide) più vicino. Tale operazione viene fatta prima calcolando la distanza euclidea tra il data points e il centroide, e successivamente attribuendo il dato al centroide che minimizzi il valore:

$$\arg \min_{c_i \in C} dist(c_i, x)^2$$

Dove:

- c_i rappresenta un centroide;
- C è l'insieme contenente tutti i centroidi;
- x sono i data points;
- $dist()$ è la distanza euclidea standard.

- **aggiornamento della posizione del centroide:** ricalcolo del punto esatto del centroide con conseguente modifica della sua posizione. Dopo lo step precedente è infatti possibile che si siano formati nuovi cluster (a quelli precedenti possono essere stati assegnati o tolti nuovi data points). Di conseguenza, si ricalcola la posizione media dei centroidi. Il nuovo valore del centroide sarà dato dalla media di tutti i data points che gli sono stati assegnati.

I passaggi 2 e 3 continueranno a essere ripetuti fin quando i centroidi non smetteranno di spostarsi: in tal caso significherà che è stato raggiunto un punto di convergenza tale per cui non si hanno più modifiche dei cluster.

Appare evidente come proprio per questo suo comportamento l'algoritmo presenti una serie di svantaggi:

- *vanno selezionati quanti gruppi si intende visualizzare.* Questo non è sempre un procedimento banale in quanto non sempre risulta possibile farlo (soprattutto man mano che il problema cresce di complessità);
- risulta essere abbastanza *instabile*, presentando nello specifico:
 - una variazione costante dei centroidi;
 - a volte può accadere che la soluzione a cui si converge sia nettamente differente rispetto alla precedente;
 - cerca di bilanciare il numero di elementi presenti nei gruppi, a discapito anche della bontà dell'algoritmo stesso.

A seguire si mostrano alcuni esempi di tale instabilità algoritmica dei risultati.

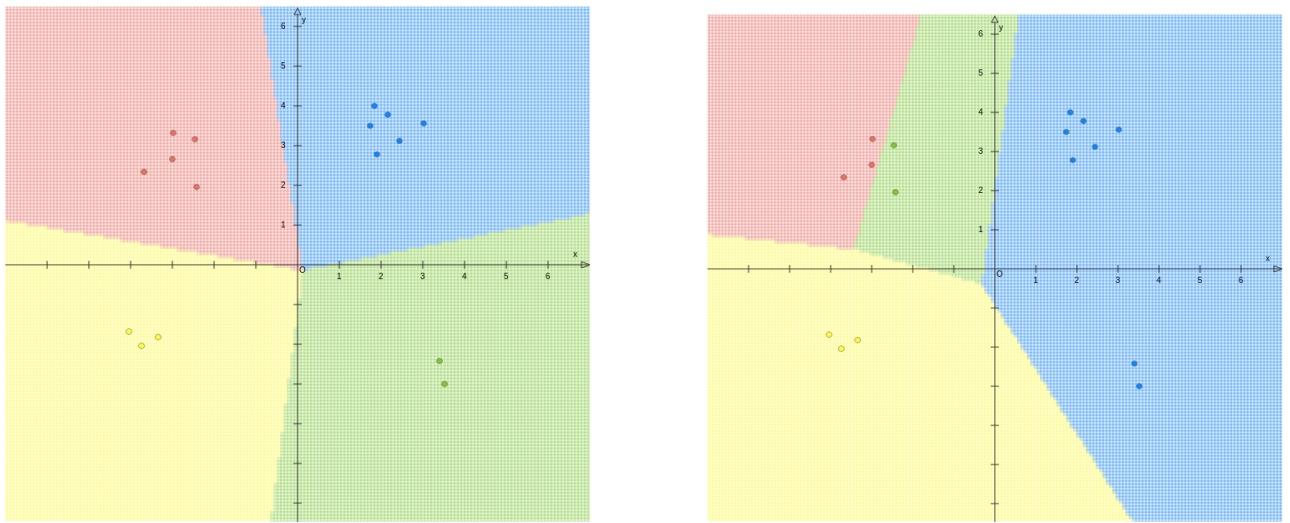


Figura 52: Instabilità algoritmo k-Means ($k = 4$)

Dalle immagini appare evidente come non solo abbiamo una rotazione dei colori dei centroidi ma anche i risultati stessi variano la loro configurazione.

Tuttavia tali debolezze vengono bilanciate dalla sua semplicità implementativa e per la sua capacità di creare gruppi di oggetti simili da una collezione di oggetti distribuiti casualmente.

Conclusione: Al netto di una certo grado di instabilità da tenere ben presente, l'algoritmo K-Means risulta adatto nei casi in cui si sia già a conoscenza del numero di cluster, allo scopo di ottenere gruppi distinti (i cluster stessi) dal set di dati.

3.5.2 Algoritmi a confronto: K-Medois vrs K-Means

Variazione del classico K-Means, il K-Medois presenta la sua caratteristica distintiva nel modo in cui individua i centroidi: qui chiamati **mediodi**.

A differenza dei centroidi, individuati dall'algoritmo ma non esistenti nella realtà, i mediodi (punti centrali del cluster), sono invece punti reali del set di dati.

Proprio per questo motivo il K-Means risente maggiormente della presenza di valori anomali, rispetto al fratello, in quanto impiega un *processo di minimizzazione del valore d'errore*. Ecco che, K-Medois che usa al posto della media proprio l'oggetto più centrale, ne risente in modo minore.

3.5.3 Algoritmi basati sulla densità: DBSCAN E OPTICS

Diversamente dai metodi che operano la suddivisione in cluster sulla base della distanza fra i pattern, gli algoritmi basati sulla densità identificano gli insiemi come regioni dello spazio dense, distinguendole da altre, a più scarsa densità, rappresentanti il **rumore**.

Questi algoritmi sono in grado di rilevare cluster di forma arbitraria e di filtrare il rumore identificando gli outlier.

Domanda:

Come si misura la densità di una regione?

A questo scopo si può disporre di molteplici elementi:

- *il numero minimo di punti* necessari a formare una regione densa;
- *il raggio* (*eps*) del cerchio centrato nel punto in esame P;
- *i vicini* nell'insieme di dati D del punto P entro la distanza *eps*:

$$N(p) = \{q \in D \mid \text{dist}(q, p) \leq \text{eps}\}$$

Grazie proprio a queste loro caratteristiche questi algoritmi risultano generalmente ben performanti.

A seguire sono affrontati le seguenti strategie di modellazione:

- DBSCAN, il primo algoritmo basato sulla densità;
- OPTICS, suo diretto discendente (i due infatti tendono a dare risultati molto simili).

DBSCAN

DBSCAN, *Density Based Spatial Clustering of Application with Noise*, rappresenta il primo algoritmo a fornire una nozione di **densità**.

Idea di base: Ogni pattern di un cluster deve presentare, nelle vicinanze di un certo raggio, almeno un certo numero di altri pattern; in altre parole, **la densità nelle vicinanze del pattern considerato deve superare una certa soglia**.

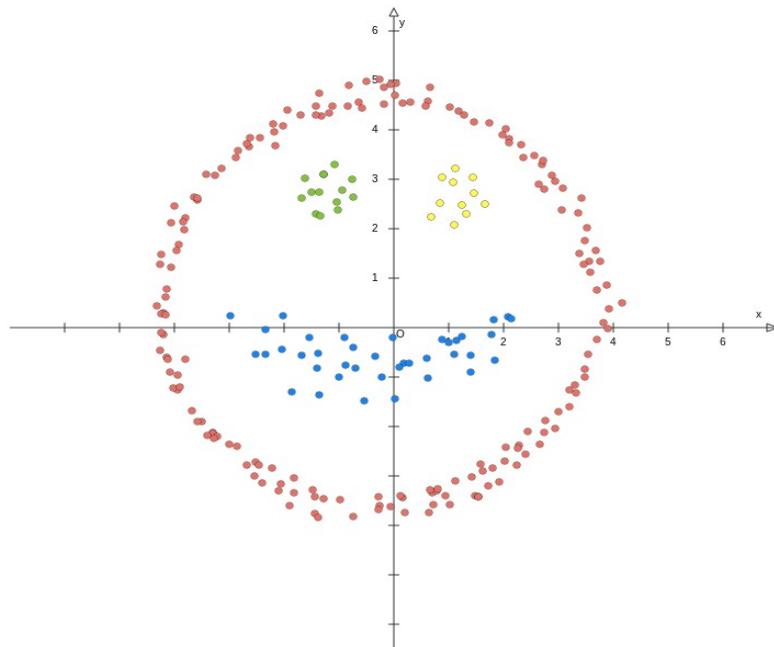


Figura 53: Problema tipico che riesce a risolvere DBSCAN

Proprio grazie a questa sua capacità di esaminare l'intorno di un punto, attuando una sorta di operazione di “legaggio a cascata” dei punti che vi rientrano, DBSCAN si mostra in grado di risolvere correttamente problemi per nulla banali.

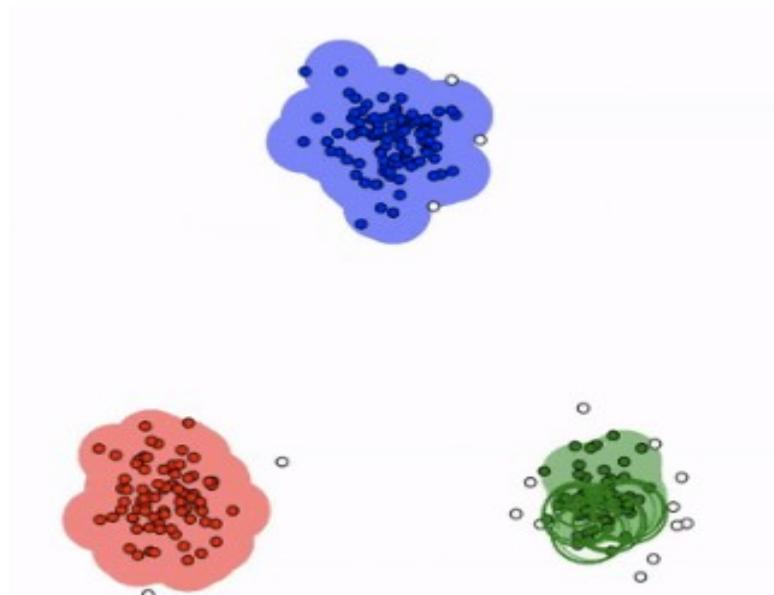


Figura 54: Operato di DBSCAN

Sopra si mostra bene l’azione di tale algoritmo: man mano esamina i punti del set di dati, attribuendo al corrispettivo cluster di riferimento quelli rientranti all’interno di un certo raggio, e tralasciando invece gli autalieri (pallini bianchi).

Conclusione: Molti sono i vantaggi di tale algoritmo tuttavia presentano anche degli svantaggi: nel caso di dati caratterizzati da distribuzioni lasche funziona male; problematica invece che non presenta il K-Means. Anche qui quindi la scelta di quale strategia impiegare deve essere ponderata e preceduta da un'attenta analisi dei dati:

Dipende la Legge di Natura sottostante.

OPTICS

Per esteso *Ordering Points To Identify the Clustering*, tale algoritmo unsupervised presenta le stesse idee del DBSCAN, cercando tuttavia di risolverne uno dei suoi principali punti deboli:

La rilevazione dei corretti cluster significativi anche in dati a diversa densità.

Per fare ciò viene generato un nuovo database dove i punti vengono ordinati linearmente. Questo database contiene le stesse informazioni ricavabili da un clustering di tipo DBSCAN, eseguito però per un più vasto intervallo di valori di densità.

I punti spazialmente più vicini diventano più vicini anche nell'ordinamento; inoltre viene memorizzata per ciascun punto una distanza speciale (fra due punti), rappresentante la densità, che deve essere accettata dal cluster, perché entrambi i punti possano appartenere allo stesso cluster.

I due algoritmi a confronto

Di seguito si mostra l'operato di entrambi gli algoritmi sopra trattati. Si noti come il risultato in questo caso risulti il medesimo.

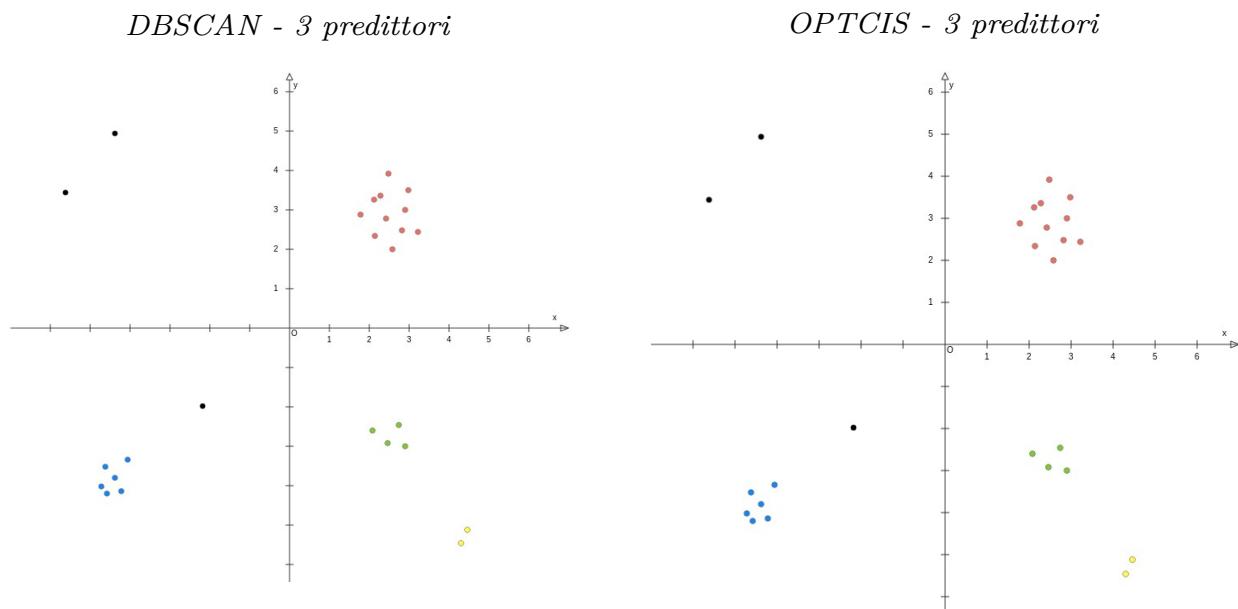


Figura 55: OPTCIS e DBSCAN a confronto

I cluster individuati sono 4: rosso, blue, verde e giallo; inoltre come giustamente ci si aspettava i punti più isolati vengono etichettati come rumore.

3.5.4 EM: la potenza delle gaussiane

L'algoritmo EM, Expectation Maximization, sfrutta il fatto che un cluster può venire rappresentato attraverso una distribuzione di probabilità parametrica. I dati possono quindi essere visti come una composizione di tali distribuzioni in cui ciascuna distribuzione viene tipicamente riferita come una gaussiana.

L'EM fa proprio questo:

Interpreta i dati secondo distribuzioni gaussiane (con propria media e deviazione standard), attribuendone a ciascuna una propria colorazione.

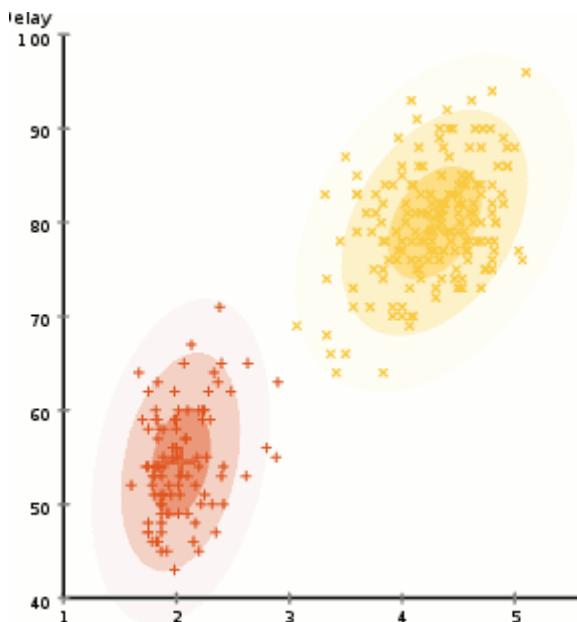
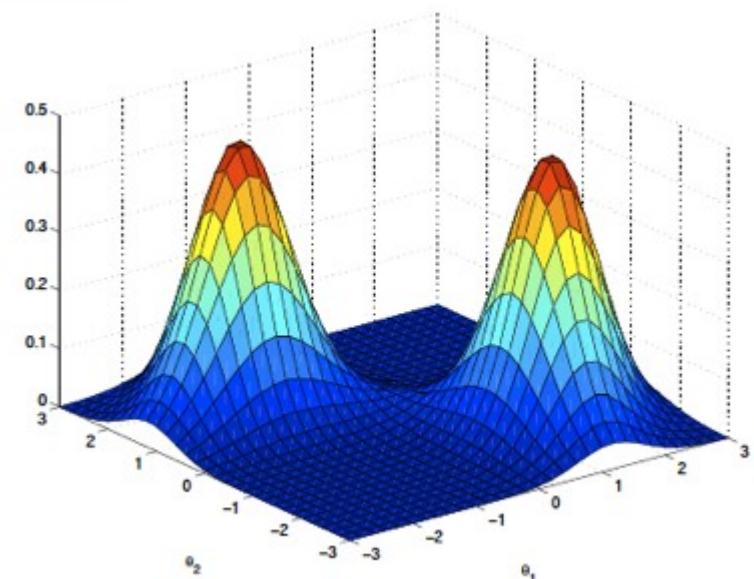


Figura 56: Distribuzioni gaussiane individuate dal Expectation Maximization



Ecco quindi che in base a tale procedimento vengono individuati due cluster differenti: ciascuno con la propria distribuzione gaussiana. Sopra se ne fornisce, a sinistra, una visione in R^2 , mentre a destra in R^3 .

Uno dei problemi più conosciuti che tale strategia riesce a risolvere consiste nella “faccia di topolino”. Grazie alla corretta individuazione rispettivamente di orecchie e viso, per questo specifico caso, EM dimostra una capacità predittiva certamente superiore al K-Means, non in grado invece di distinguere nettamente i confini fra le tre zone.

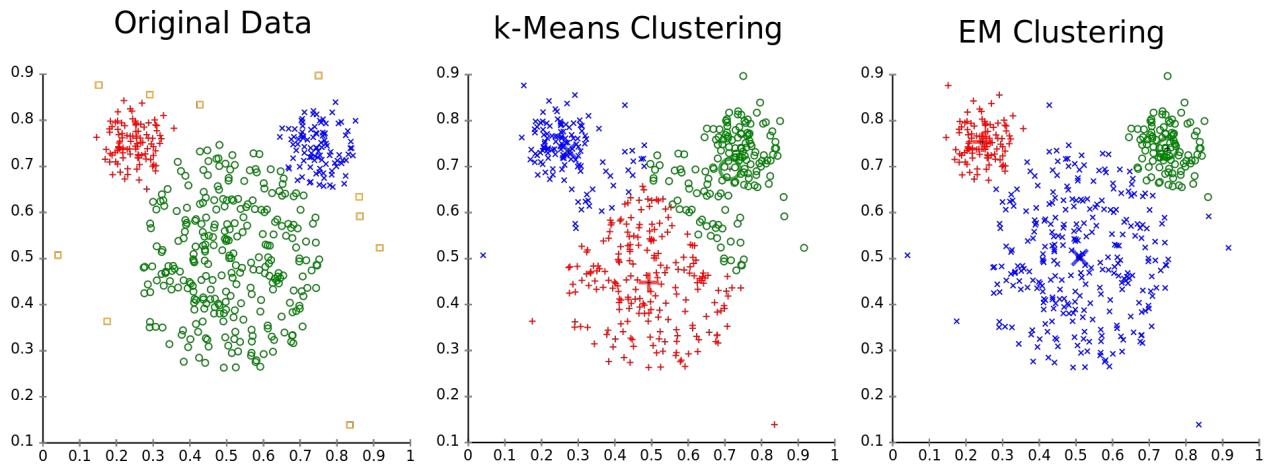


Figura 57: Mickie Mouse Problem

3.6 Algoritmi Natural Network: SOM

Acronimo di *Self Organizing Map*, è un tipo di rete neurale artificiale (ANN) che viene addestrata utilizzando l’apprendimento non supervisionato. A differenza del K-Means che separa i punti in gruppi, tale algoritmo unisce anche se in modo diverso dal DBSCAN che lega al punto successivo.

Attua una riduzione dimensionale cercando di coprire la zona d’interesse.

4. La giusta ricetta per il Machine Learning esiste?

Nel presente documento sono stati esposti tutta una serie di algoritmi di Machine Learning sia di tipo Supervised che Unsupervised.

Anche se a primo impatto le tecniche trattate possono sembrare relativamente più semplici rispetto ad altre (Regressione Lineare vrs Reti neurali), non sono assolutamente da sottovalutare in quanto:

La potenza di un metodo non sta tanto nella complessità che racchiude bensì nella sua efficacia: non sempre questi due principi coincidono.

Per ogni algoritmo si è cercato di fornire una visione che fosse il più dettagliata possibile. Si è deciso di non focalizzarsi tanto sul processo algoritmico sottostante ma, piuttosto, sui comportamenti che i vari modelli di predizione, una volta addestrati, mostravano.

Nel dettaglio per ogni algoritmo si presenta:

- enunciazione dei concetti matematici, fisici e statistici su cui vengono costruiti (e.g: regressione lineare → retta, SVM → iperpiano e così via);
- analisi e manipolazione attraverso software specifici, di dataset reali. Tale attività è stata svolta al fine di riuscire a creare, ancora prima di applicarvi qualunque tipo di modello, un'idea precisa di ciò su cui si sarebbe lavoro;
- conclusioni per ciascun algoritmo trattato di quanto osservato;
- immagini esplicative dei risultati ottenuti e dei processi regolatori degli algoritmi.

Due sono state le principali domande che hanno governato tale lavoro:

- **Come cambia la bontà predittiva di un modello, al cambiare dei dati su cui esso viene applicato?**
- **Come varia la qualità della predizione da un modello all'altro?**

È evidente come in realtà entrambi questi quesiti siano fortemente connessi fra loro e si possano ricondurre ad un'unica grande questione:

Esistono algoritmi o modelli predittivi che sempre si dimostrano sicuramente migliori di altri?

Come già enunciato al termine di ogni sezione dedicata, è probabile che la risposta a questa domanda sia no. Spieghiamo quanto appena affermato con un esempio.

La regressione lineare rappresenta l'algoritmo “intuitivamente” più semplice (anche se in realtà è più complesso e ampio di quello che appare) e rigido facendosi governare da una **precisa legge presente in natura**: la retta.

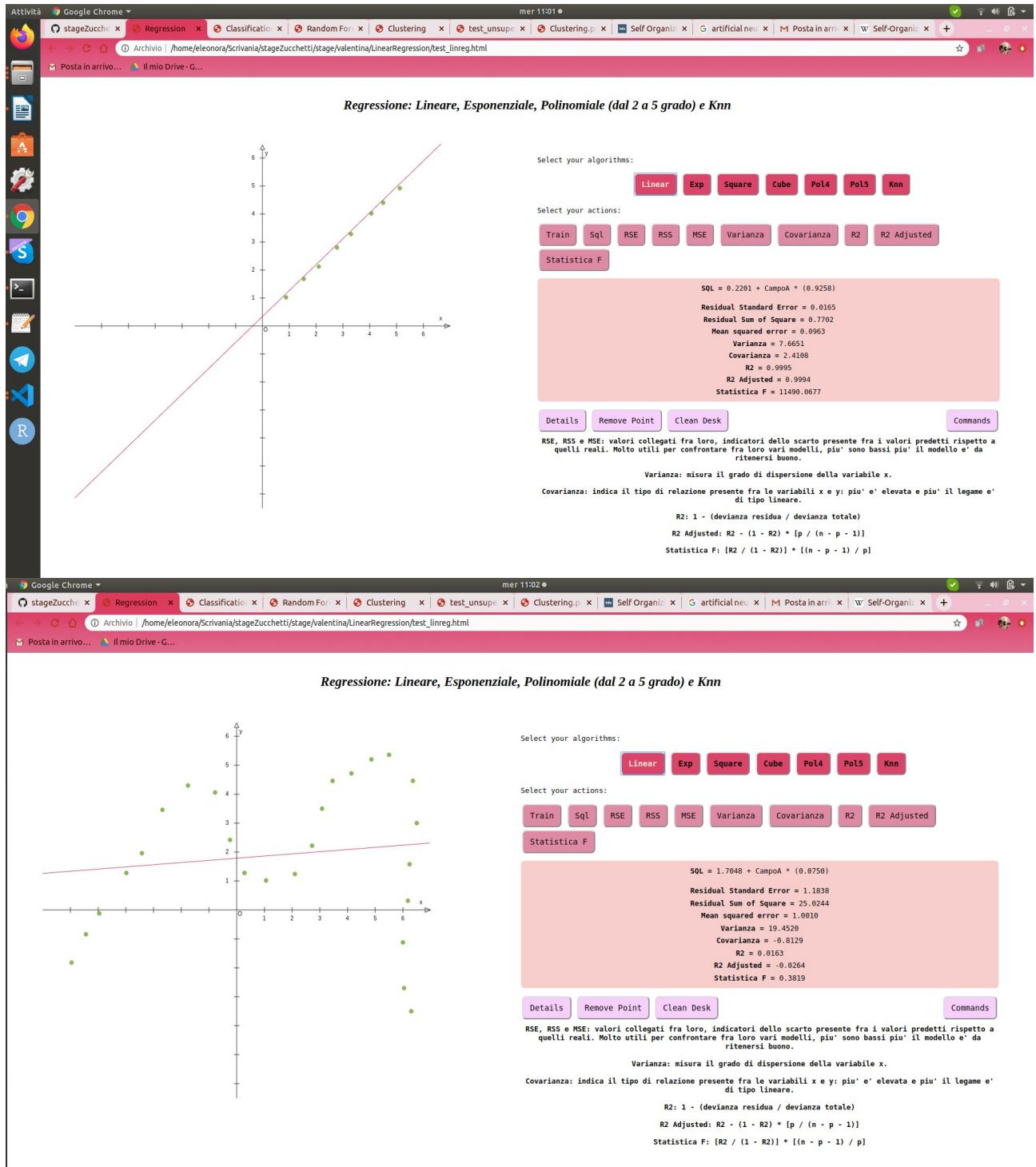


Figura 58: Esempi di Regressione Lineare su distribuzioni di dati differenti

È ovvio che tale caratteristica non cambierà mai: la regressione lineare si adatterà bene su dati a distribuzione lineare, e male su altri. Ciò fornisce anche maggiore chiarezza sul perché esistano delle trasformazioni che possono essere applicate ai dati (e.g: trasformazioni *Box Cox* – paragrafo 2.2.6) in modo da renderli maggiormente lineari, o ancora, perché esistano modelli di tipo non lineare (chiaramente la gamma di algoritmi fra cui scegliere è vastissima).

Ecco che quando si lavora coi dati cercando di attuare previsioni sui loro comportamenti, il comportamento, al quale a sua volta bisognerebbe attenersi, dovrebbe sempre essere il seguente:

- **analisi critica dei dati di cui si dispone:** ciascun dataset contiene al suo interno informazioni preziose e comportamenti precisi che è importante conoscere per evitare di incorrere in errori iniziali difficilmente corregibili poi;
- **selezione di un modello:** se occorre se ne potrebbero selezionare anche di più, per poi confrontarne almeno superficialmente i risultati;
- **riflessione sul modello prescelto:** riflessione circa la correttezza della scelta attuata;
- **osservazione in modo critico dei risultati ottenuti:** rilevazione dei punti in cui la predizione ha dato esito positivo e di quelli dove invece ha fallito. In entrambi i casi è fondamentale comprendere bene per quale motivo questo si verifica.

Per ciascuno dei passi sopra risulta di fondamentale importanza chiedersi *se quanto si sta osservando (dati, modello o risultati) sia conforme alle assunzioni iniziali e alla natura stessa dei dati: alle attese*. Se così non è, risulta chiaro che non si sta procedendo nel modo migliore.

In genere infatti:

Le assunzioni di base non si possono verificate a priori, in quanto per quasi nessuno dei fenomeni naturali sono note le vere relazioni causa-effetto.

Per questo motivo si procede, in primo luogo al *fitting* (adattamento) del modello e, successivamente, si verifica il rispetto delle assunzioni di base.

Conclusione:

Ogni evento solo per il semplice fatto di essere un qualcosa di reale, presenta specifici comportamenti e caratteristiche: se osservati attentamente permettono di individuare precise Leggi Naturali.

L'equazione della retta sarà sempre $y = mx + q$ e verrà sempre rappresentata come una linea dritta: nel momento in cui inizia a fare curve non si parla più di retta.

Tale affermazione può sembrare banale ma è proprio questa che sta alla base di una buona applicazione degli algoritmi di Machine Learning:

Non esistono modelli “buoni” o modelli “cattivi” nei modelli di predizione. Semplicemente ve ne sono alcuni che si adattano meglio, per loro natura, a interpretare alcune situazioni, e altri altre.

Sta all'intelligenza e all'esperienza dell'osservatore comprendere di volta in volta quali tecniche è più saggio usare.

5. Appendice

5.1 Analisi con Orange Canvas: Dataset Wine

Di seguito si riportano le osservazioni fatte durante l'analisi del dataset Wine, grazie all'impiego del software Orange Canvas.

Si è iniziato dando uno sguardo ai vari dati presenti nel database: 3 tipologie differenti di vini (la nostra variabile dipendente), e 13 attributi (le variabili indipendenti) rappresentanti alcune caratteristiche chimiche del vino (intensità del colore, flavonoidi, alcol, acido malitico...).

L'obiettivo è quello di, adottando l'algoritmo non supervisionato k-Means, essere in grado di predire in base alle variabili prese in esame di volta in volta, se si sta parlando del vino di tipo 1, 2 oppure 3.

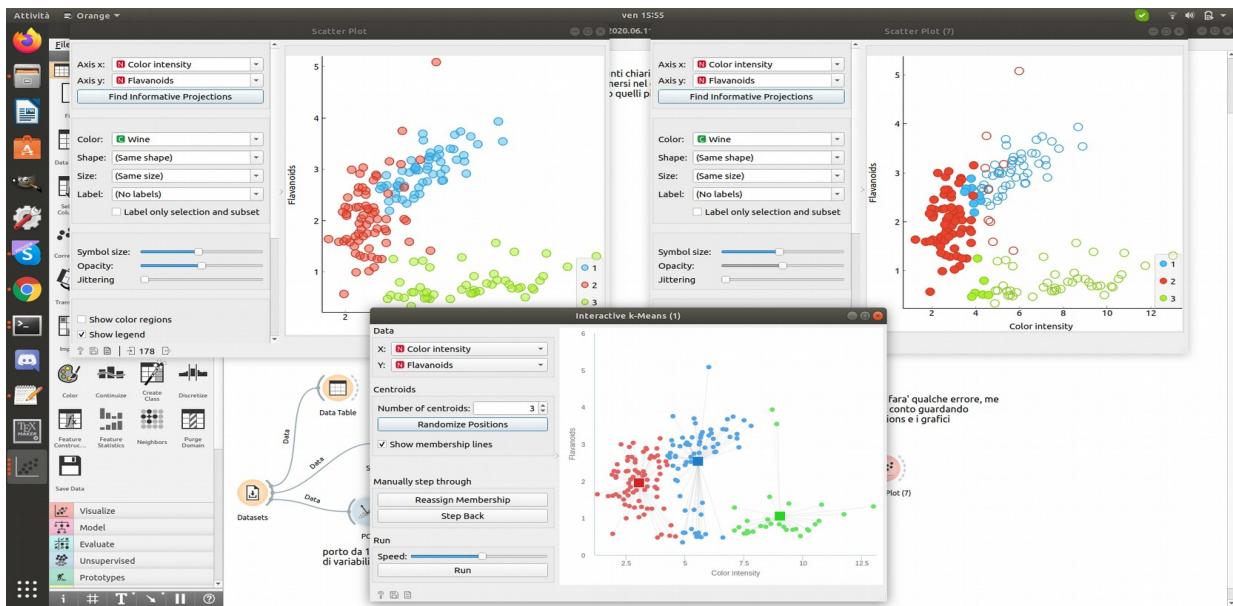
Innanzitutto avendo a disposizione una cospicua quantità di covariate, si è cercato di individuare quelle che effettivamente fossero portatrici di informazione utile alla trattazione del nostro problema: non è infatti poco frequente che più dati vogliano dire in realtà la medesima cosa, risultando quindi inutili a livello pratico. Si è quindi deciso di impiegare il metodo delle *Principal Component Analysis (PCA)*. Tale strategia sfrutta il fatto che i punti possono essere trasformati in vettori tramite una trasformazione lineare delle variabili; in tal modo si ottengono delle nuove variabili che vengono proiettate nei vari assi del piano cartesiano in base al risultato della loro varianza: la nuova variabile con la maggiore varianza viene proiettata sul primo asse, la seconda per dimensione della varianza, sul secondo e così via. La riduzione della complessità avviene a questo punto limitandosi a trattenere le variabili che presentano le varianze più significative.

Utilizzando il widget PCA si è provveduto a fare tale riduzione di dimensionalità: l'80% di varianza spiegata è stata ritenuta un buon valore soglia da impiegare, permettendo di ridurre il numero di variabili da 13 a 5. Successivamente si è applicato l'algoritmo non supervisionato k-Means. A tal proposito precedentemente, da un primo sguardo sulla distribuzione dei dati, si era già osservato che essi molto probabilmente risultavano essere un po' troppo "sparpagliati" per poter adattarsi completamente al suddetto algoritmo, che presenta invece l'apice della sua performance su distribuzioni a grappolo e ben concentrare. Si è comunque ritenuta questa una buona opportunità per osservare come in situazioni non pienamente idonee esso si sarebbe comportato, esaminando inoltre quali siano gli strumenti e le strategie che si possono impiegare per individuare i punti dove l'algoritmo di clustering fallisce.

Per avere una visione più precisa della situazione si è applicato il widget Interactive k-Means che permette di mostrare il funzionamento dell'algoritmo di clustering. È stato quindi possibile vedere come stabilendo un numero di cluster pari a 3 (equivalente al numero di tipologie di vini), questo vada a posizionare i centroidi e successivamente a distribuirvi i vari punti attorno, assegnandovi l'appartenenza ad un cluster oppure ad un altro in base alla loro distanza. Confrontando tale distribuzione con quella di partenza già a questo punto si può notare come in realtà l'algoritmo attribuisca con un certo margine di errore i punti ai vari cluster.

Successivamente si è andati ad applicare l'algoritmo non supervisionato. Gli esiti sono stati esaminati singolarmente, grazie al widget SelectRow che isolando un cluster alla volta ha permesso di vedere quanti match o missmatch si fossero verificati in rapporto col grafico originario. Ci si è inoltre fatti aiutare al widget Distributions per avere una visione ancora più chiara del tutto.

A seguire si includono alcune immagini esplicative di quanto appena enunciato.



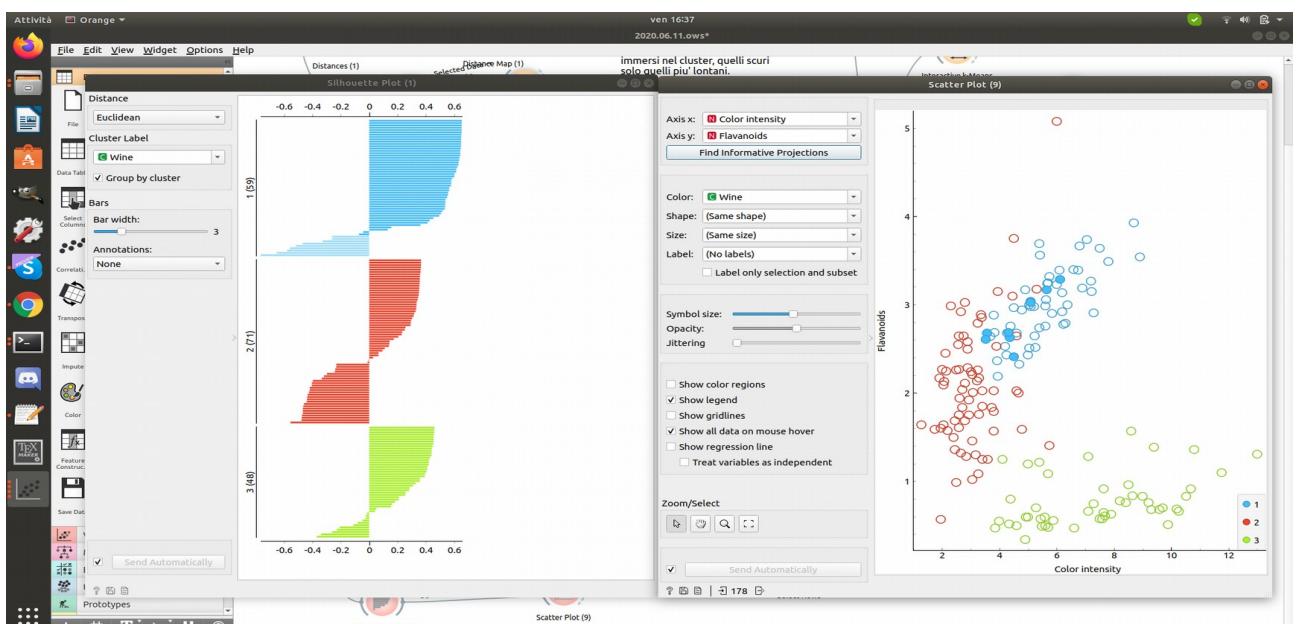
App 1: Grafici dei dati (Wine) – Orange Canvas

Si può notare che a ciascuna famiglia di vino corrisponde un differente colore: azzurro, rosso o verde; appare chiaro come sia complicato in realtà individuare una precisa separazione fra le classi, molti sono infatti i punti border-line o che entrano dentro l'area della classe sorella.

Nel dettaglio si presenta:

- in alto: trasformazione dei dati una volta applicata la tecnica PCA;
- sotto: redistribuzione a seguito dell'applicazione dell'algoritmo k-Means con evidenziazione dei centroidi;
- a destra: in relazione alla classe 2 (rossa), come in realtà l'algoritmo vi consideri facenti parte anche alcuni punti verdi e altri blu, escludendone altri di rossi.

Un altro argomento trattato che ha permesso di comprendere meglio il problema di attribuire ai vari punti il cluster corretto, è stato quello di *Silhouette*.



App 2: Silhouette Plot e relativo grafico dei dati (Wine) - Orange Canvas

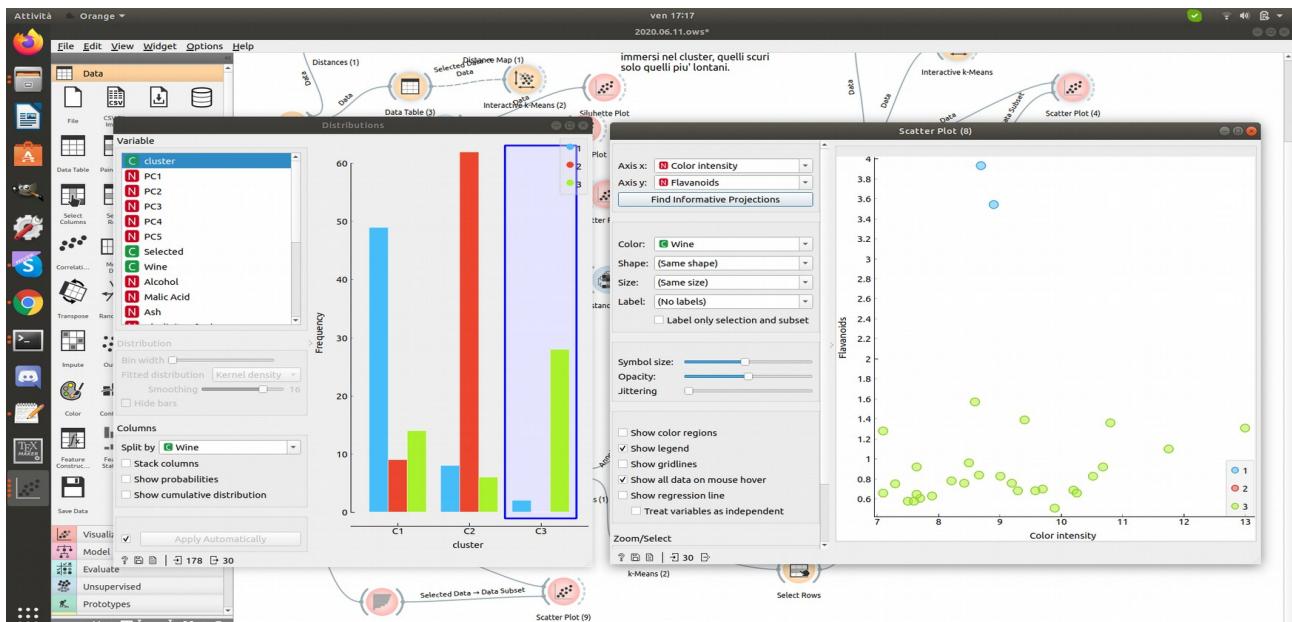
Indicatore con un valore compreso fra 0 ed 1, di quanto un punto sia coeso rispetto al cluster attribuitogli, questo indice presenta il seguente comportamento:

- più il suo valore tende a crescere e più significa che l'istanza si trova circondata da elementi dello stesso cluster;
- più invece diminuisce più sarà evidente che si sta parlando di un valore lontano dal cluster in esame (o considerato anomalo nel caso di valori negativi).

L'immagine sopra presenta il grafico inerente alle osservazioni, prive della trasformazione PCA, e a sinistra quello di Silhouette. In particolare si sono evidenziati i valori negativi del primo tipo (azzurro), che nel grafico a destra si presentano molto vicini a quelli rossi.

Conclusione: appare chiaro come l'algoritmo non supervisionato k-Means in tale circostanza faccia fatica ad individuare in modo corretto la classe di appartenenza di un dato: molte sono infatti le zone confuse che vedono la presenza di punti di colore diverso, ecco che un algoritmo basato sulle distanze non è probabilmente la scelta migliore.

A seguire vengono presentati grazie al widget Distributions, le distribuzioni di ciascun cluster, dove ad esempio il cluster C3, che dovrebbe corrispondere alla classe verde, comprende anche dei punti che dovrebbero essere blu, escludendone, tuttavia, degli altri che dovrebbero invece farne parte.



App 3: Distribuzione e grafico dei dati (Wine) – Orange Canvas

Nota: Precedentemente si era lavorato sul dataset *Iris*, simile per difficoltà di comprensione dei dati ma con una distribuzione più “a insiemi” (anche se non completamente). In tale circostanza l'algoritmo K-means aveva presentato delle prestazioni migliori.