Analisi dei dati sul set Red and White Wine Quality

A cura di:

- Pievaioli Stefano, matricola 816592
- Presot Federico Davide, matricola 817290
- Zhigui Guerrero Bryan Ivan, matricola 816335

Sommario

Dataset

Abbiamo scelto di utilizzare questo dataset per il nostro progetto:
https://archive.ics.uci.edu/ml/datasets/wine+quality

Esso riguarda le caratteristiche chimiche e sensoriali di diverse bottiglie di “Vinho Verde”, un tipo di vino portoghese collegato alla regione del Minho; il nome è dato non dalla colorazione (sono presenti infatti bottiglie di Vinho verde di vino bianco, rosso e rosè) ma dal fatto che essi sono dei vini “giovani”, la cui consumazione è consigliata solitamente entro un anno dall’imbottigliamento.

Il Dataset è diviso in due sotto-dataset, uno riguardante i vini rossi e uno per i vini bianchi, che noi abbiamo scelto di raggruppare. Dopo un raggruppamento iniziale, sono presenti 6497 istanze, caratterizzate da 13 attributi che indicano caratteristiche chimiche o sensitive dei vini. Di queste, abbiamo scelto di considerare la variabile “quality”, che esprime un voto dato da 1 a 10 al vino, come variabile target; il motivo di questa decisione è stato dato dall’idea che un possibile committente, come un proprietario di una cantina, sia maggiormente interessato a capire quali siano i vini da scartare prima che vengano immessi sul mercato con una valutazione negativa che possa intaccare il prestigio della cantina stessa.

Attributi

Presentazione degli attributi

Gli attributi del dataset sono:

1. **Fixed acidity**, l'acidità fissa è costituita dalle sostanze acide presenti in un vino, che non sono portate a volatilizzare, ma al contrario restano all'interno del vino per tutta la sua vita.
2. **Volatile acidity**, acidità volatile è la quantità di acido acetico presente nel vino.
3. **Citric acid**, L'acido citrico apporta una sensazione di freschezza, contribuendo all'equilibrio gustativo del vino.
4. **Residual sugar**, residuo zuccherino, è lo zucchero che proviene dagli zuccheri naturali dell'uva rimasti in un vino dopo la fine della fermentazione alcolica.
5. **Chlorides**, il cloruro viene utilizzato nel processo di produzione del vino in specifici casi.
6. **Free sulfur dioxide**, anidride solforosa in forma gassosa.
7. **Total sulfur dioxide**, l'anidride solforosa viene aggiunta durante i processi di vinificazione in quanto abbia proprietà antiossidanti, antisettiche e solubilizzanti. Total Sulfur dioxide in questo caso indica l'anidride solforosa totale, somma tra l'anidride solforosa libera e quella collegata alle altre sostanze chimiche.
8. **Density**, densità del vino.
9. **pH**, acidità del vino.
10. **Sulphates**, solfati, un additivo del vino che può contribuire ai livelli di anidride solforosa (S02), che agisce come antimicrobico e antiossidante.
11. **Alcohol**, percentuale di alcol presente nel vino.
12. **Quality**, attributo che ne definisce la qualità; è un voto numerico che va da 1 a 10 e definisce le caratteristiche sensoriali del vino. Nel nostro Dataset, abbiamo scelto di raggruppare i voti insufficienti (da 1 a 5) nella categoria "notSuff" e i voti sufficienti (dal 6 in su) nella categoria "Suff"

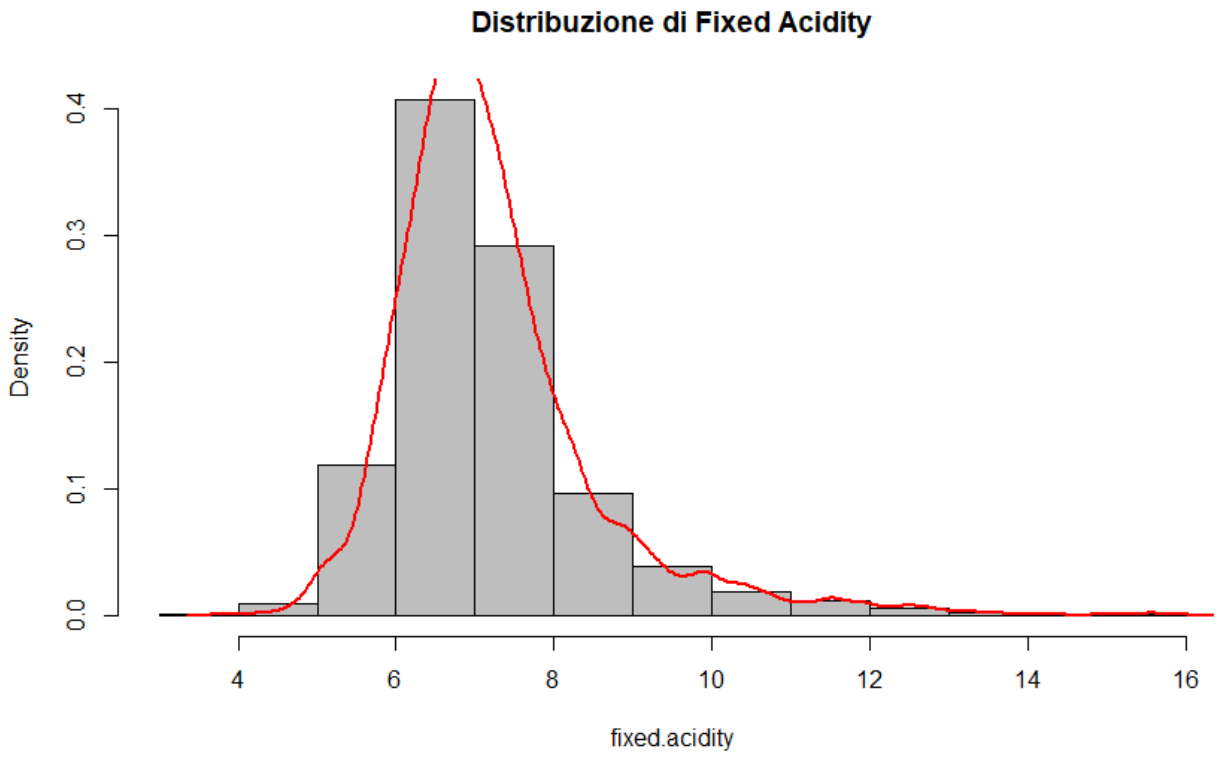
Abbiamo aggiunto il tredicesimo attributo per indicare il tipo di viso, rosso o bianco.

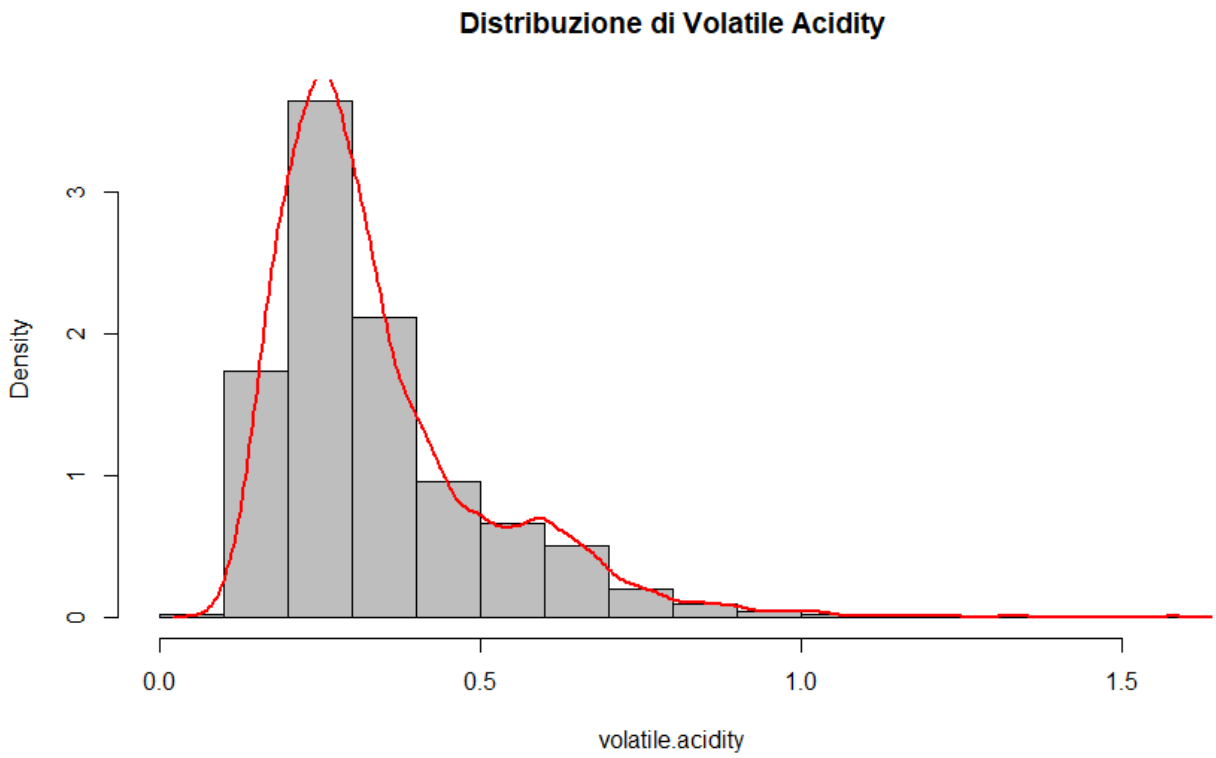
13. **Color**, Colore del vino, rosso o bianco.

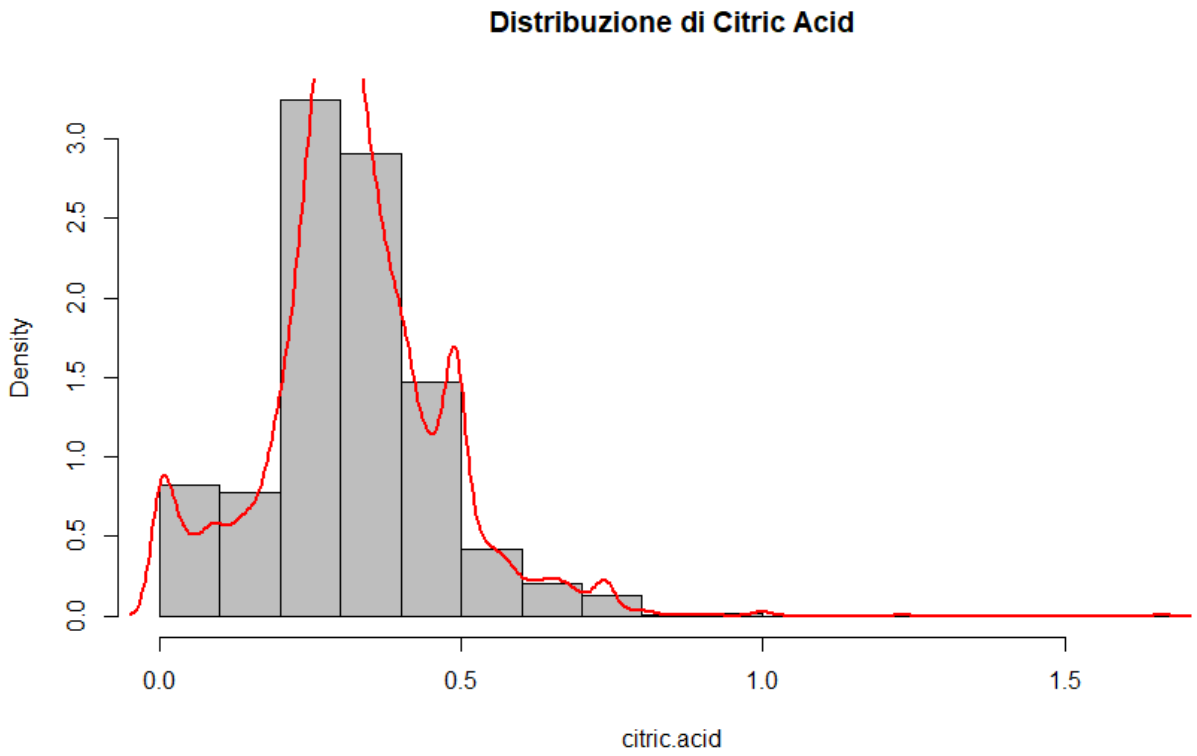
Distribuzione degli attributi

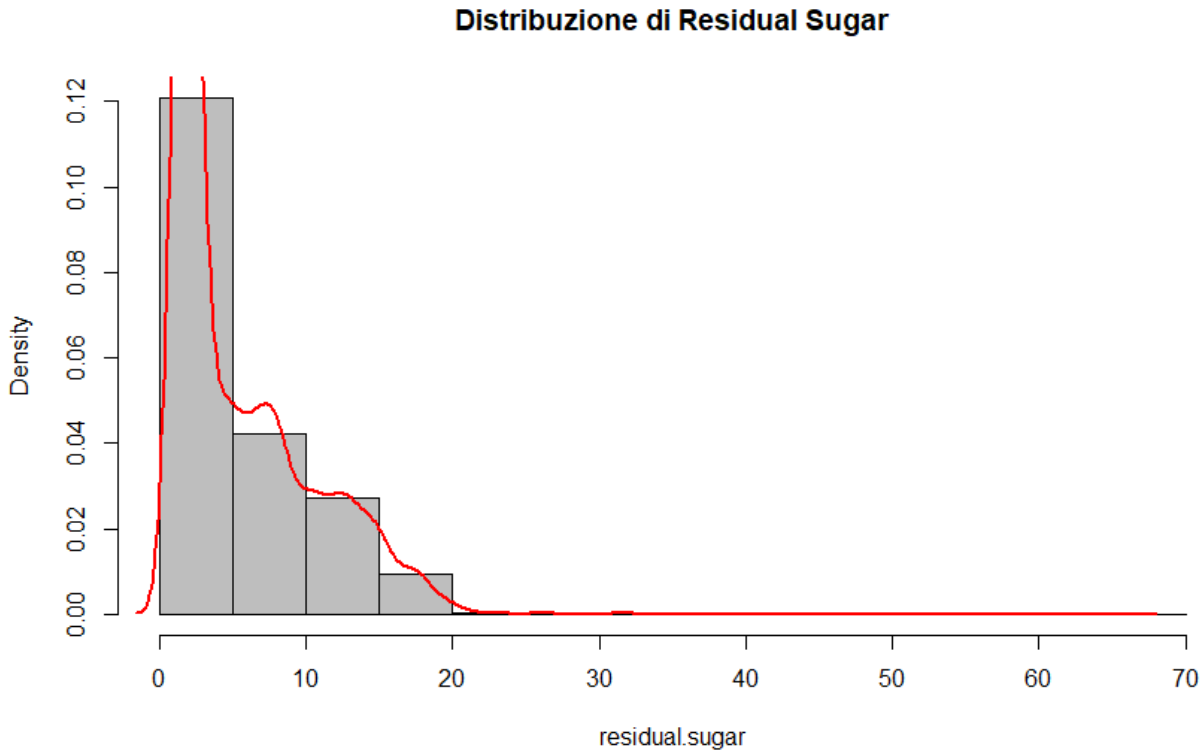
Codice che abbiamo utilizzato per disegnare i grafici. Tutti i valori qui presentati sono per le variabili numeriche del nostro dataset, che hanno valori numerici continui.

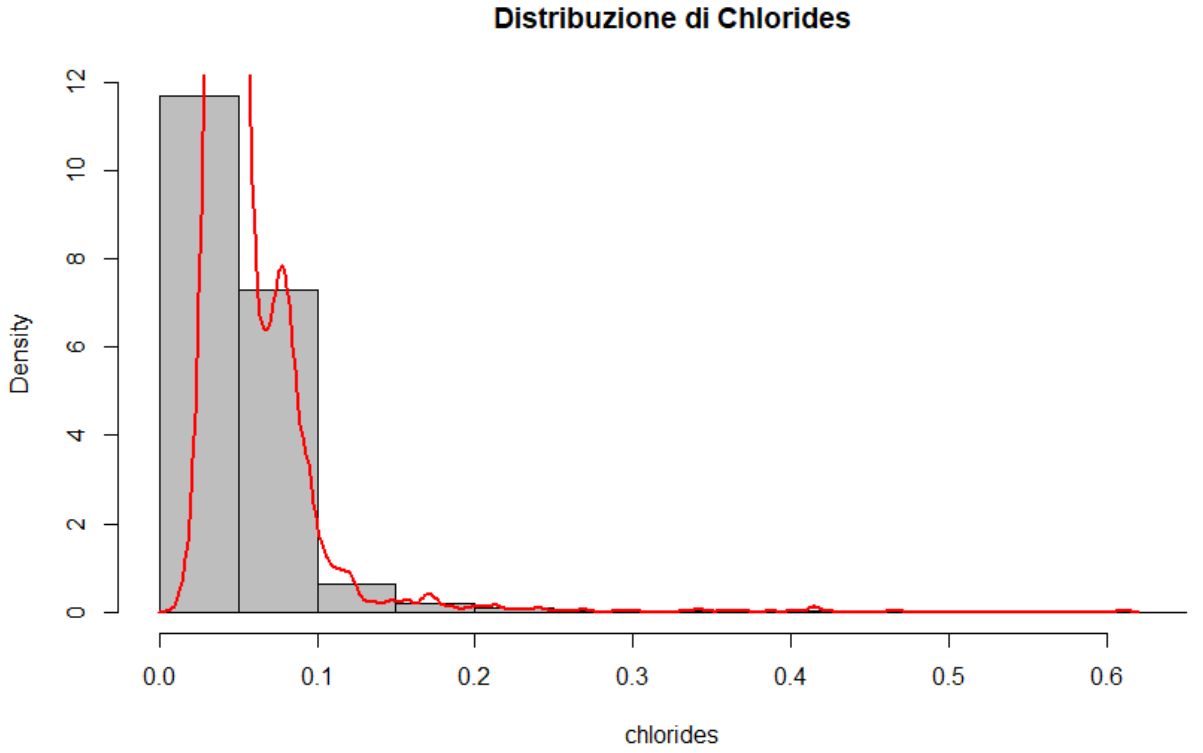
```
hist(dataset$alcohol, # histogram
     col="grey", # Colore delle colonne
     border="black",
     prob = TRUE, # visualizza la densità al posto della frequenza di distribuzione
     xlab = "alcohol",
     main = "Distribuzione di Alcohol")
lines(density(dataset$alcohol), # density plot
      lwd = 2, # spessore linea
      col = "red")
```

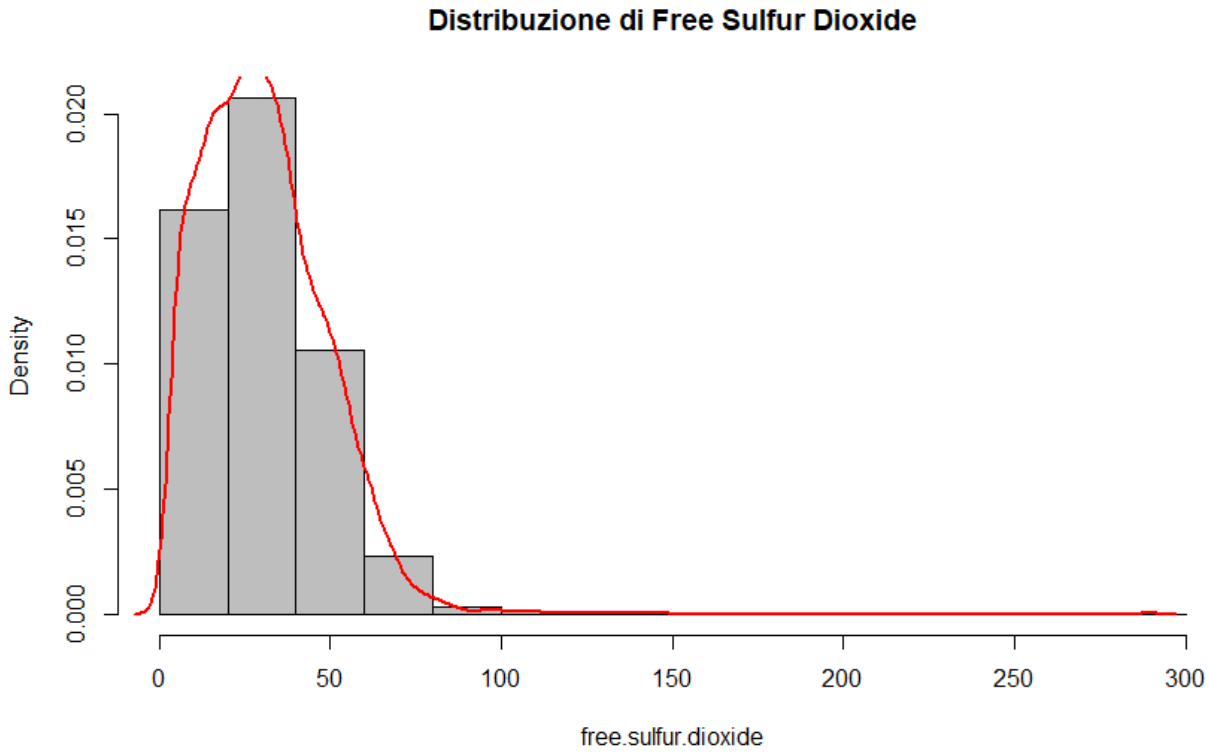


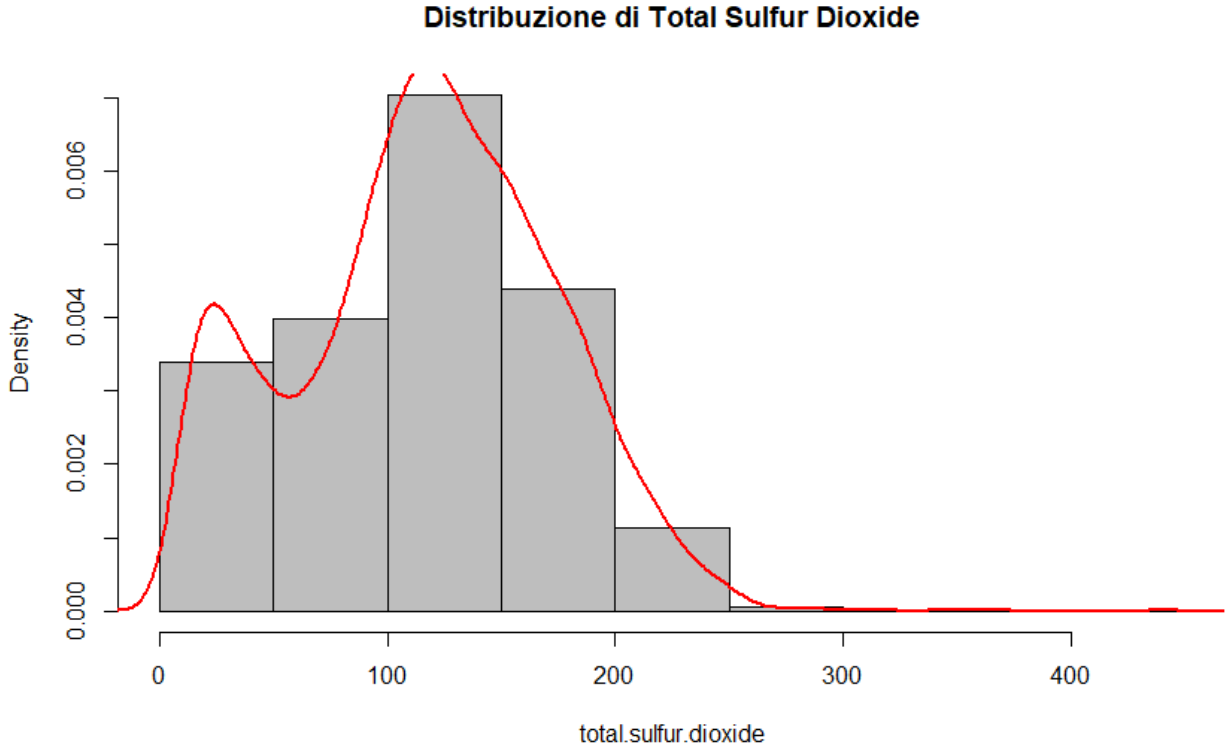

Distribuzione di Volatile Acidity
Density
0
1
2
3
0.0
0.5
1.0
1.5
volatile.acidity

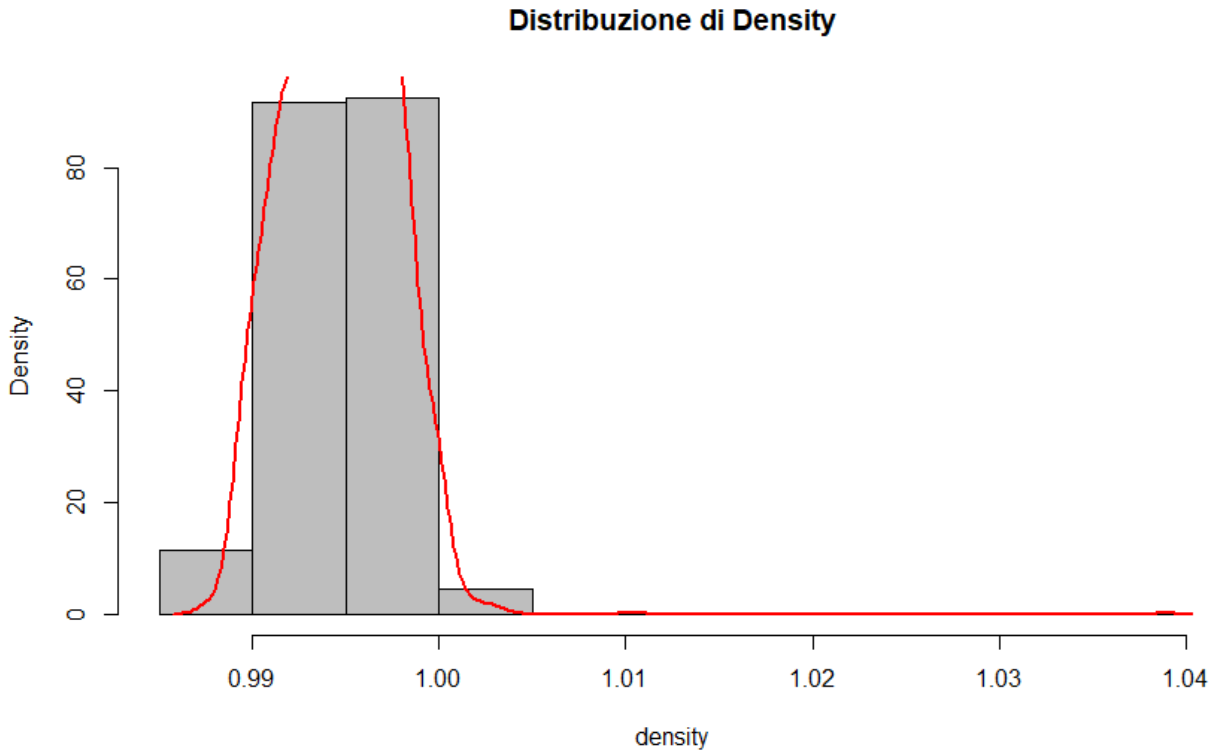

Distribuzione di Citric Acid
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.5
1.0
1.5
citric.acid

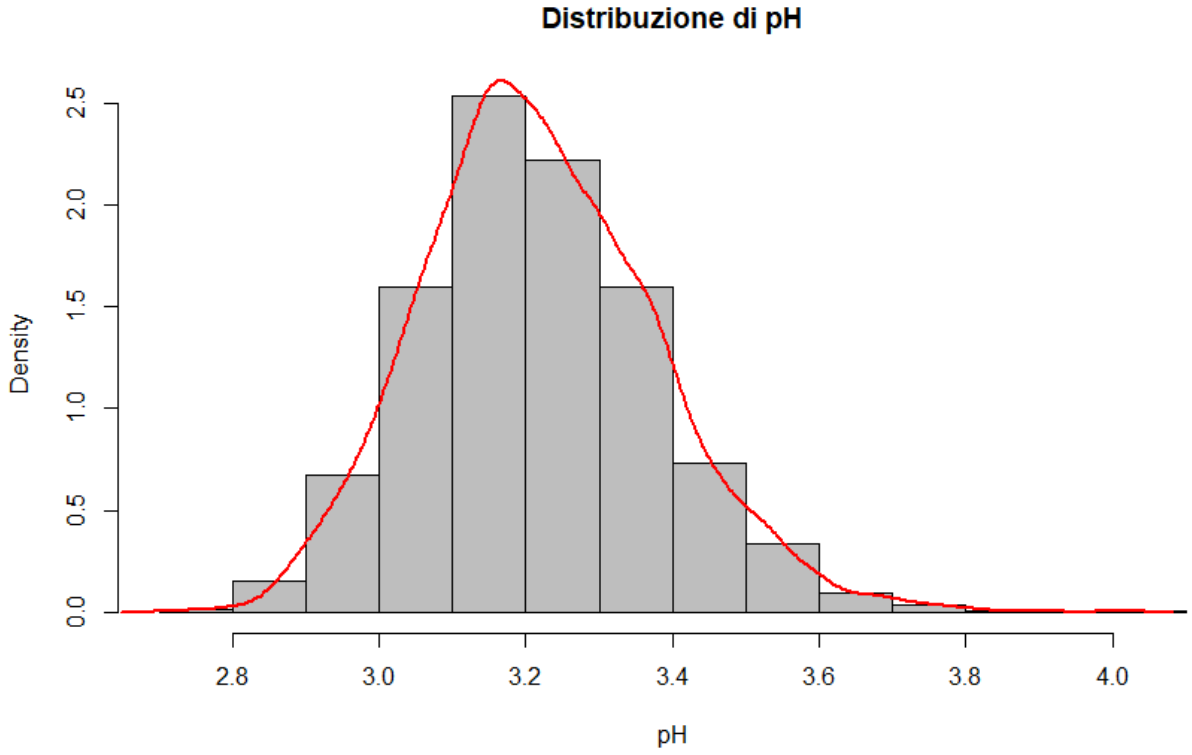

Distribuzione di Residual Sugar
Density
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
10
20
30
40
50
60
70
residual.sugar

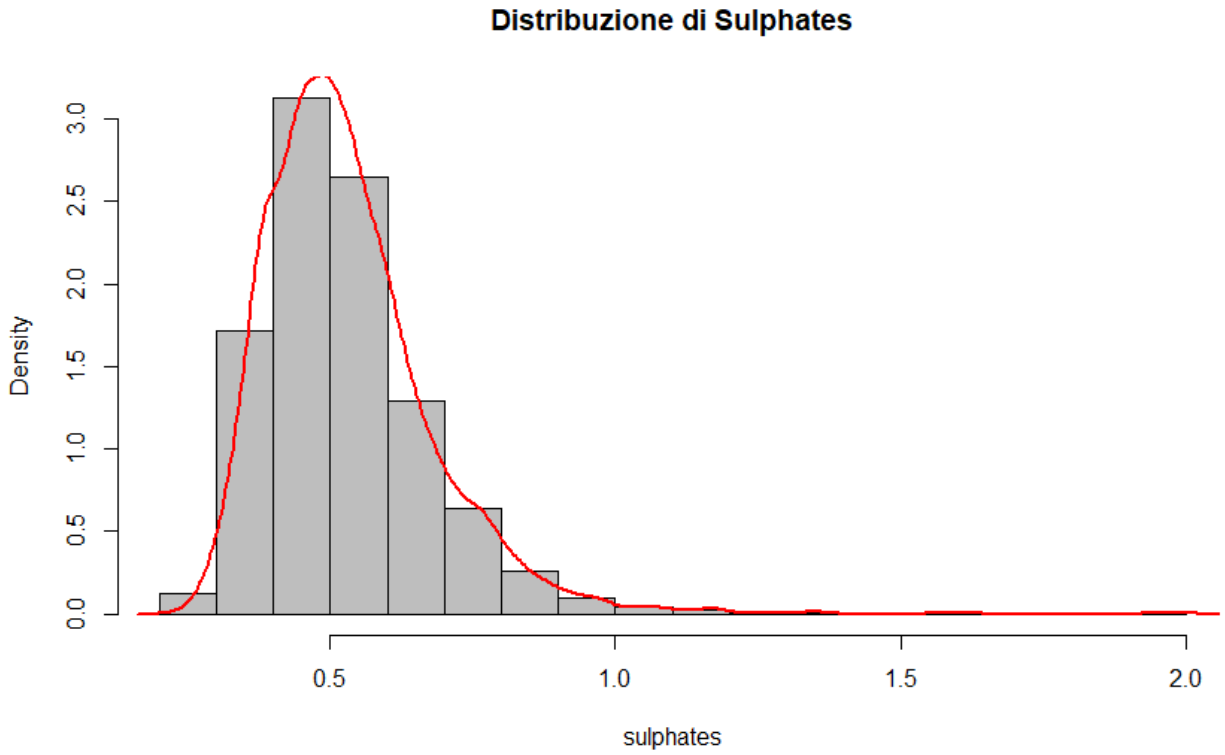

Distribuzione di Chlorides
Density
0
2
4
6
8
10
12
0.0
0.1
0.2
0.3
0.4
0.5
0.6
chlorides

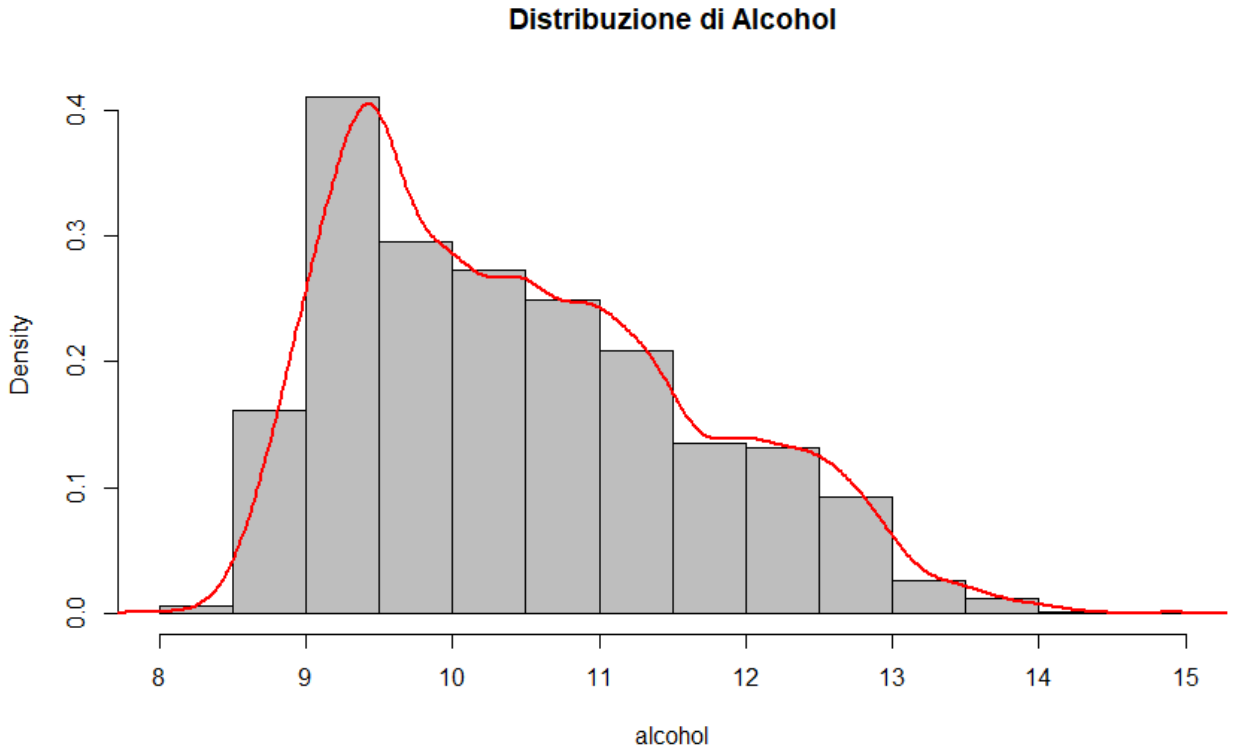

Distribuzione di Free Sulfur Dioxide
Density
0.000
0.005
0.010
0.015
0.020
0
50
100
150
200
250
300
free.sulfur.dioxide


Distribuzione di Total Sulfur Dioxide
Density
0.000
0.002
0.004
0.006
0
100
200
300
400
total.sulfur.dioxide


Distribuzione di Density
Density
0
20
40
60
80
0.99
1.00
1.01
1.02
1.03
1.04
density


Distribuzione di pH
Density
0.0
0.5
1.0
1.5
2.0
2.5
2.8
3.0
3.2
3.4
3.6
3.8
4.0
pH


Distribuzione di Sulphates
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.5
1.0
1.5
2.0
sulphates


Distribuzione di Alcohol
Density
0.0
0.1
0.2
0.3
0.4
8
9
10
11
12
13
14
15
alcohol

Splitting training set e test set e controllo dati

Sono stati presi in considerazione due dataset, uno per i vini rossi e uno per i vini bianchi, che andremo a raggruppare in un solo dataset principale con l'obiettivo di definire quali tra questi vini siano considerabili sufficienti e quali no.

Per differenziare i due dataset, prima di raggrupparli, è stata aggiunta una colonna *color* di tipo char avente i nomi corrispettivi "*white"* e "*red*".

Una volta che si sono uniti i due dataset, ottenendo il dataset principale, abbiamo normalizzato le seguenti colonne:

1. *Quality* con i valori [1, 2, 3, 4, 5, 6, 7, 8, 9]: abbiamo deciso di assegnare secondo una logica binaria i valori [1;5] al valore "0" per descrivere se non fosse sufficientemente buono, mentre per i valori [6;9] il valore "1" per descrivere se il vino fosse sufficientemente buono;

   La variabile sarà poi trasformata in factor. Essa è inoltre la nostra variabile target.

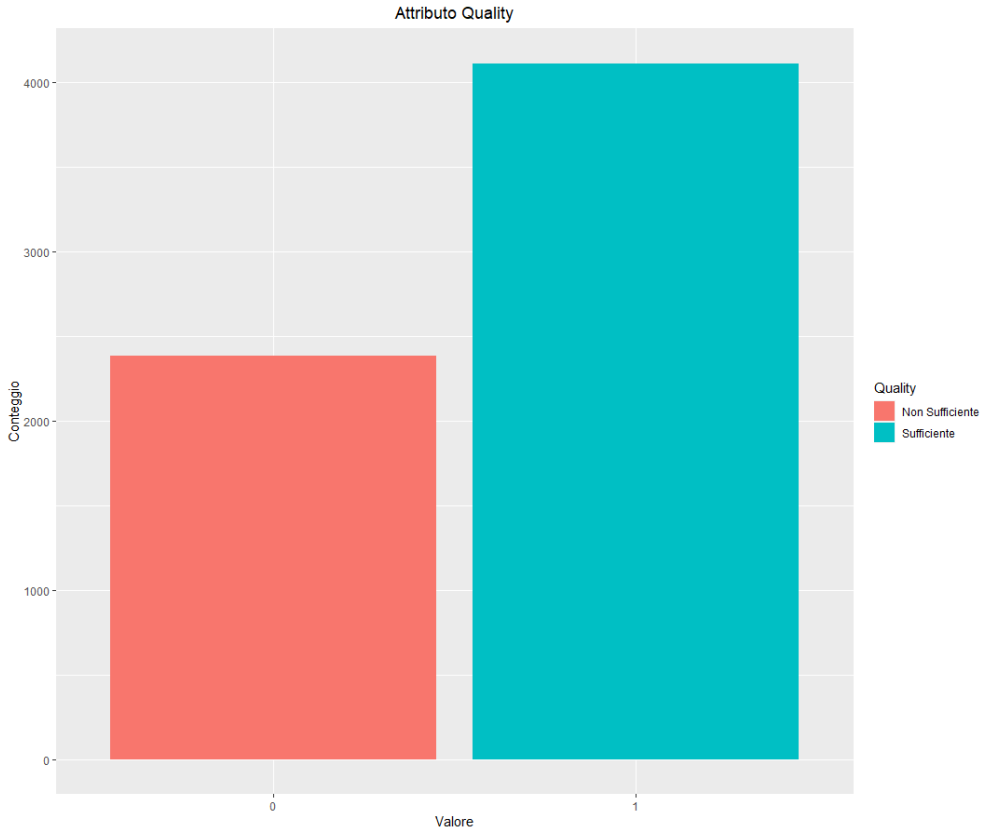
2. *Color*: abbiamo deciso di assegnare secondo una logica binaria all'etichetta "white" il valore "1" e all'etichetta "red" il valore "0".

   Anche questa variabile sarà poi trasformata in factor

Tramite la funzione *str(dataset)* vediamo tutte le variabili presenti all'interno del dataset ed quelle trasformati in fattore.

```
'data.frame':   6497 obs. of  13 variables:
 $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
 $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
 $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
 $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
 $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
 $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
 $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
 $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
 $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
 $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
 $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
 $ quality             : Factor w/ 2 levels "0","1": 1 1 1 2 1 1 1 2 2 1 ...
 $ color               : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

Abbiamo fatto un plotting della variabile categorica che ci permetteva di individuare all'interno del dataset quanti vini fossero etichettati come "Sufficienti" e quanti "Non Sufficienti". Dal grafico notiamo che prevalgono i vini "Sufficienti".

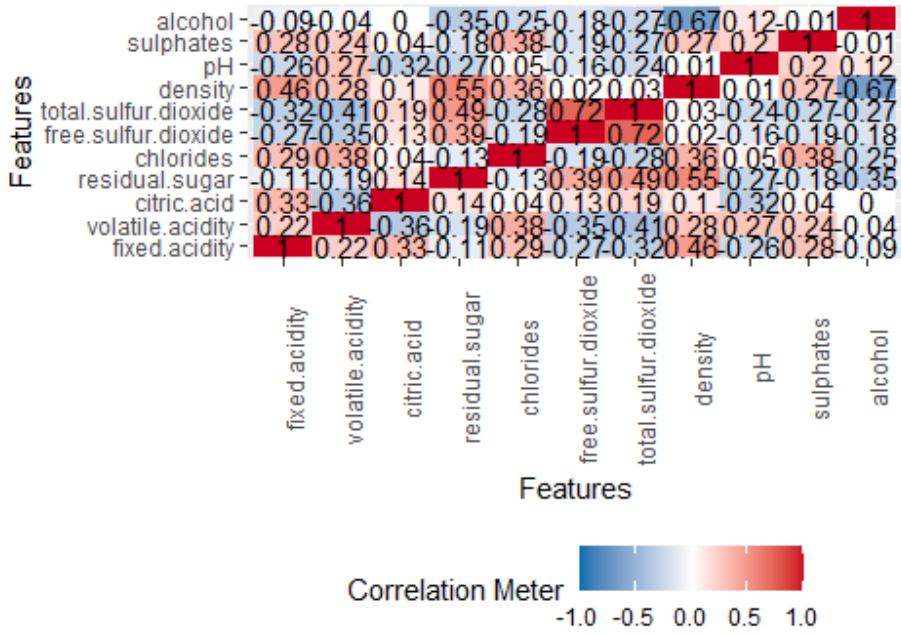


Il dataset principale è stato oggetto di splitting per ottenere:

- Il *training set* pari al *70%* del dataset principale, il quale sarà usato per costruire i modelli di Decision Tree ed SVM,
- Il *test set* pari al *30%* del dataset principale, il quale viene usato per vedere quanto bene il nostro modello funzioni su dati non visibili/visti prima.

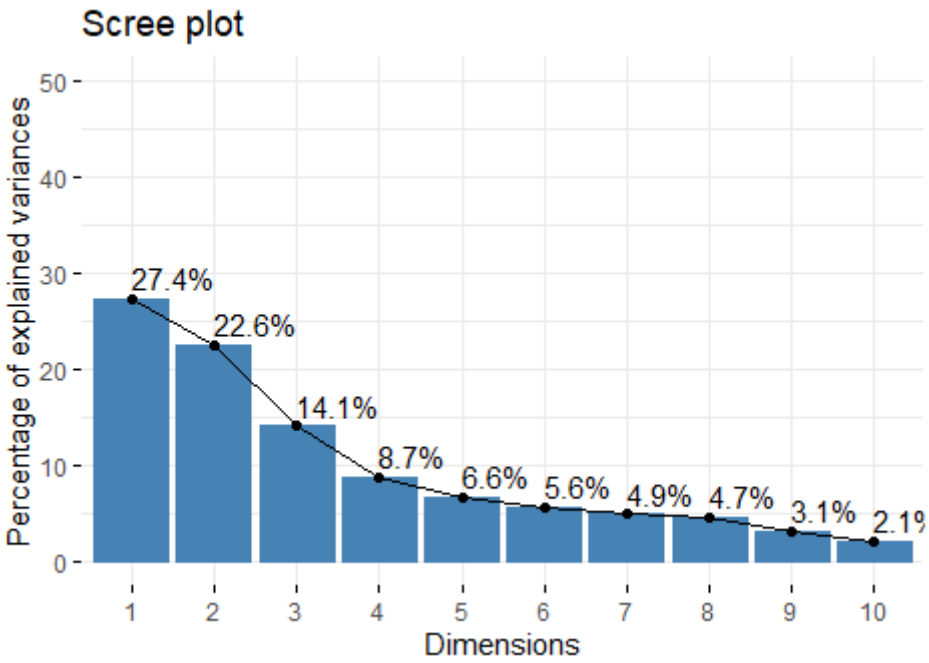
PCA e Modifiche ai train e test

Dopo aver suddiviso il nostro Dataset in Training Set e Test Set, abbiamo deciso di fare una analisi esplorativa del nostro Dataset, in particolar modo per verificare se esistano delle Feature in esso da rimuovere. Per fare ciò, inizialmente ci siamo affidati al calcolo della correlazione di tutte le variabili numeriche presenti nel nostro Dataset:



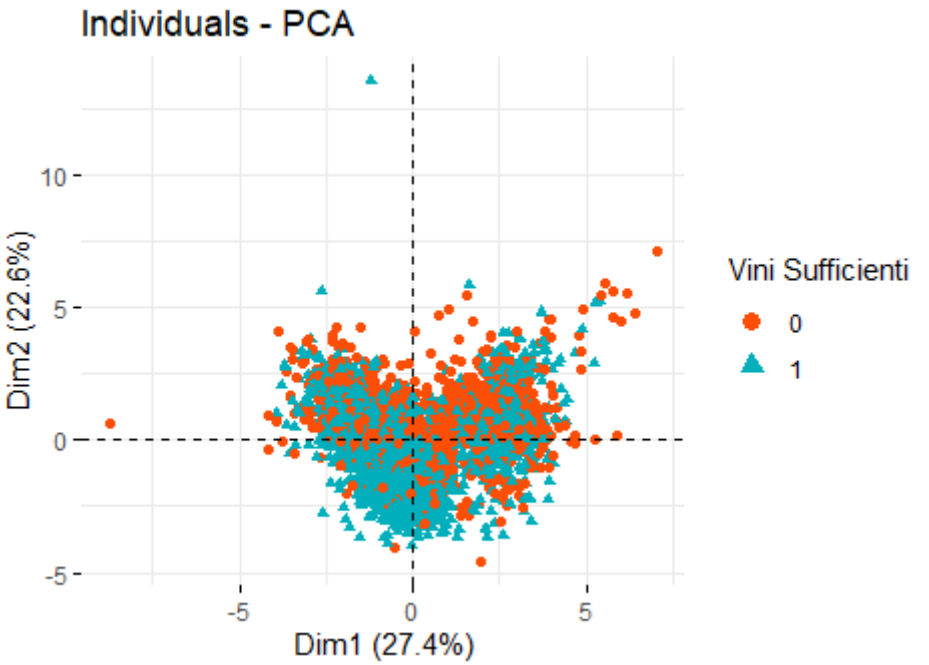
Tramite questo calcolo possiamo notare una forte correlazione tra **total.sulfur.dioxid**e e **free.sulfur.dioxide;** Ciò ha senso dal punto di vista logico: l'anidride solforosa totale presente nel vino è calcolata dalla somma dell'anidride solforosa libera con quella combinata. Inoltre possiamo notare una certa correlazione tra **density** e **residual.sugar**. Prima di procedere all'eliminazione di alcune variabili, però, abbiamo scelto di applicare la PCA al training set per verificare se convenga utilizzare le dimensioni generate dalla stessa. La PCA, infatti, ci servirebbe per effettuare una trasformazione lineare che trasformerebbe le nostre variabili in un nuovo set di componenti principali non correlate tra di loro. Dopo aver effettuato la PCA, questi sono i valori delle nuove componenti:

	eigenvalue	variance.percent	cumulative.variance.percent
Dim.1	3.00908833	27.3553484	27.35535
Dim.2	2.48286933	22.5715394	49.92689
Dim.3	1.54754515	14.0685923	63.99548
Dim.4	0.95965877	8.7241706	72.71965
Dim.5	0.72651640	6.6046945	79.32435
Dim.6	0.61554124	5.5958294	84.92017
Dim.7	0.53892717	4.8993379	89.81951
Dim.8	0.51182561	4.6529601	94.47247
Dim.9	0.34316103	3.1196458	97.59212
Dim.10	0.23091870	2.0992609	99.69138
Dim.11	0.03394827	0.3086206	100.00000

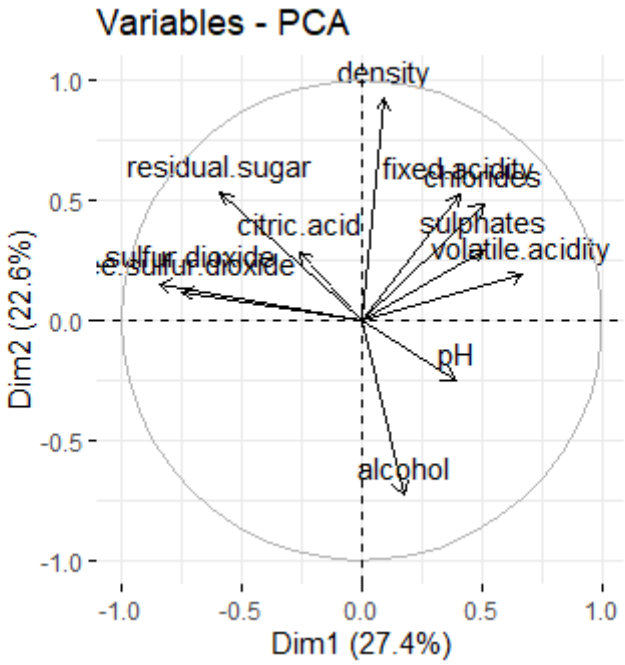


Abbiamo inoltre generato uno Scree Plot dove possiamo scegliere un determinato valore soglia di varianza percentuale spiegata che ci permetta di determinare quali dimensioni tenere e quali scartare.

Possiamo inoltre verificare tramite la PCA dove si posizionerebbe i nuovi valori all'interno di questo nuovo spazio da noi considerato; in questo caso, eccoli presentati utilizzando come riferimento la variabile target:



Un altro aspetto interessante della PCA è la possibilità di determinare quali siano le variabili iniziali che sono maggiormente correlate tra di loro:



Tramite questo grafico possiamo effettuare alcune considerazioni: il nostro dubbio iniziale relativo alla eccessiva correlazione tra **free.sulfur.dioxide** e **total.sulfur.dioxide** è corretto e mostrato anche dalla PCA. Stupisce invece come nello spazio determinato dalla PCA vi sia una correlazione tra **citric.acid** e **residual.sugar**, non mostrata dal plot effettuato precedentemente per determinare una possibile correlazione.

In conclusione, dopo la PCA abbiamo tre diverse opzioni disponibili:

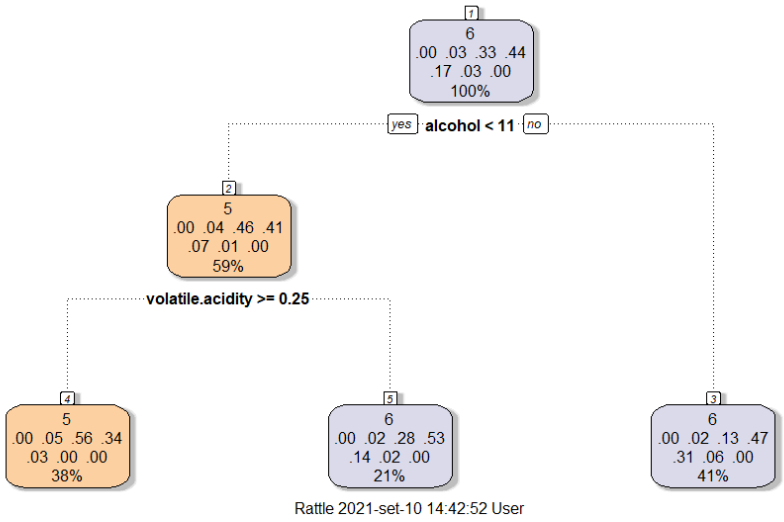
- Utilizzare lo spazio delle nuove variabili descritte dalla PCA
- Utilizzare il plot fatto precedentemente per la correlazione in modo da determinare se vi siano delle variabili da eliminare
- Tenere il Training Set integro

Abbiamo scelto una opzione che possa essere considerata una via di mezzo tra le prime due: abbiamo infatti scelto di utilizzare sia la correlazione normale tra le variabili, sia la loro correlazione nello spazio descritto dalla PCA per rimuovere una delle due variabili tra **free.sulfur.dioxide** e **total.sulfur.dioxide;** nel nostro caso, abbiamo scelto di rimuovere total.sulfur.dioxide, dopo aver notato che nel grafico della correlazione nello spazio della PCA essa è più vicina all'origine, e di conseguenza cattura una minore varianza.

Decision Tree

Cos'è il Decision Tree?

Decision Tree, in italiano alberi di decisione sono un modello supervisionato utilizzato nella ricerca operativa, nella pianificazione strategica e nell'apprendimento automatico. La figura accanto mostra un albero. Ogni quadrato rappresenta un nodo di decisione, di solito più nodi un albero ha e più accurato. Gli ultimi nodi dell'albero decisionale vengono chiamate foglie dell'albero e sono le "decisioni" del modello. Gli alberi decisionali risultano molto intuitivi e facili da costruire.



Perché abbiamo utilizzato Decision Tree?

Abbiamo deciso di utilizzare il modello dell'albero di decisione in quanto volevamo un modello che fosse chiaro nel rappresentare visivamente ed esplicitamente le decisioni prese, così da avere un controllo visivo e semplificato degli attributi con maggiore influenza nella decisione del modello.

Sviluppo 1#

La prima funzione che abbiamo utilizzato è "rpart". Gli argomenti obbligatori della funzione sono la "formula", e "data". Rispettivamente in formula vuole l'attributo che sarà l'output dell'albero di decisione, e tutti gli altri attributi, mentre con data si intende il dataset per il train, cioè tutti quei dati utilizzati per creare e "allenare" il modello.

```
decisionTree = rpart(quality ~ fixed.acidity + volatile.acidity + citric.acid +
                     residual.sugar + chlorides + free.sulfur.dioxide +
                     total.sulfur.dioxide + density + pH + sulphates + alcohol
                     + color , data=trainset, method="class")
plot(decisionTree)
text(decisionTree)
```

Sviluppo 2#

Dopo aver testato il nostro modello abbiamo provato a migliorarlo sia per avere una accuratezza migliore sia per avere maggiore flessibilità nell'utilizzare la funzione "rpart".

Abbiamo quindi deciso di utilizzare la funzione "train" per migliorare il nostro modello. Questa funzione imposta una griglia di parametri di ottimizzazione per una serie di routine di classificazione e regressione, si adatta a ciascun modello e calcola una misura delle prestazioni basata sul ricampionamento.

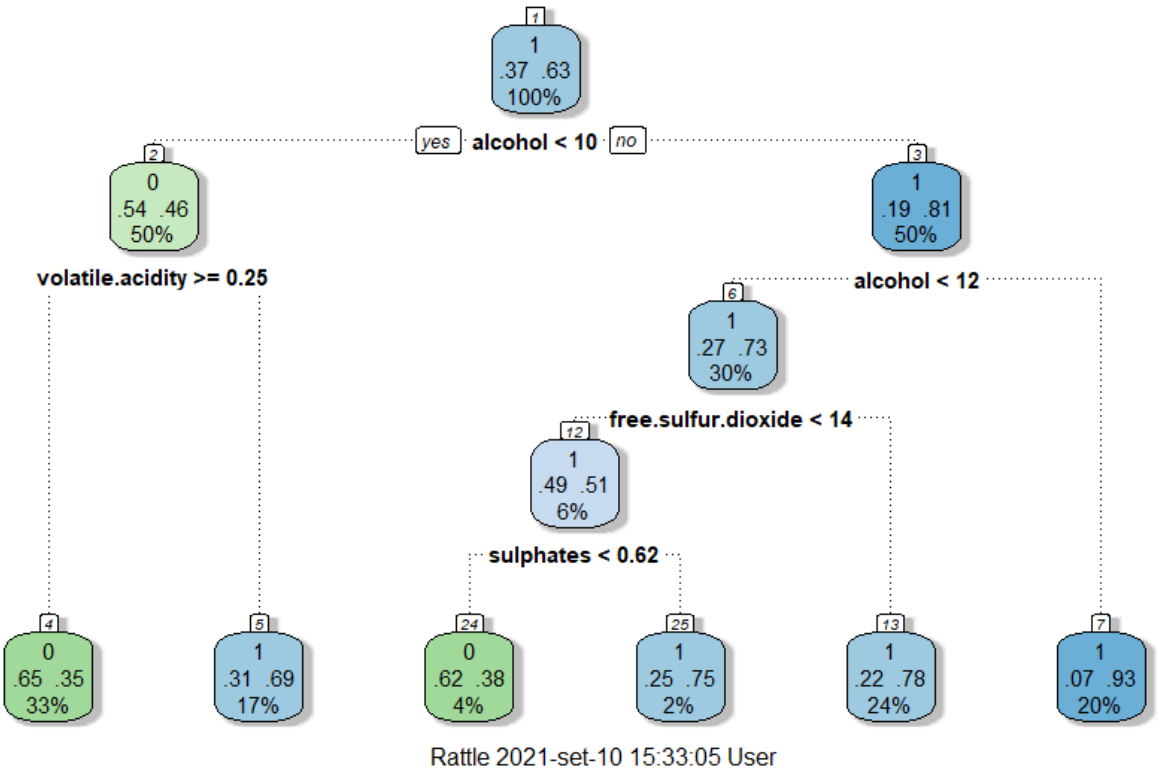
```
dt.control = trainControl(method = "cv", number = 10)
dt.model = train(quality ~ ., data=trainset,
                 method="rpart",
                 weights = NULL,
                 trControl = dt.control,
                 tuneGrid = NULL,
                 tuneLength = 3)
```

La funzione “trainControl” controlla le operazioni computazionali (ricampionamento) che fa la funzione train. L’argomento number della funzione indica quante volte verrà eseguito il ricampionamento.

Abbiamo utilizzato il seguente codice per la predict del nostro modello.

```
fancyRpartPlot(dt.model$finalModel)
dt.pred = predict(dt.model, testset[,! names(testset) %in% c("quality")])
table(dt.pred, testset$quality)
```

Grazie alla funzione “train” abbiamo ottenuto il seguente albero.



Attraverso la Confusion Matrix l’accuratezza e le statistiche del nostro albero sono state:

```
                Accuracy : 0.7476
                  95% CI : (0.7277, 0.7667)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : <2e-16

                   Kappa : 0.4563

 Mcnemar's Test P-Value : 0.7523

              Precision : 0.6592
                 Recall : 0.6518
                     F1 : 0.6555
             Prevalence : 0.3684
         Detection Rate : 0.2401
   Detection Prevalence : 0.3643
      Balanced Accuracy : 0.7276
```

Sviluppo 3#

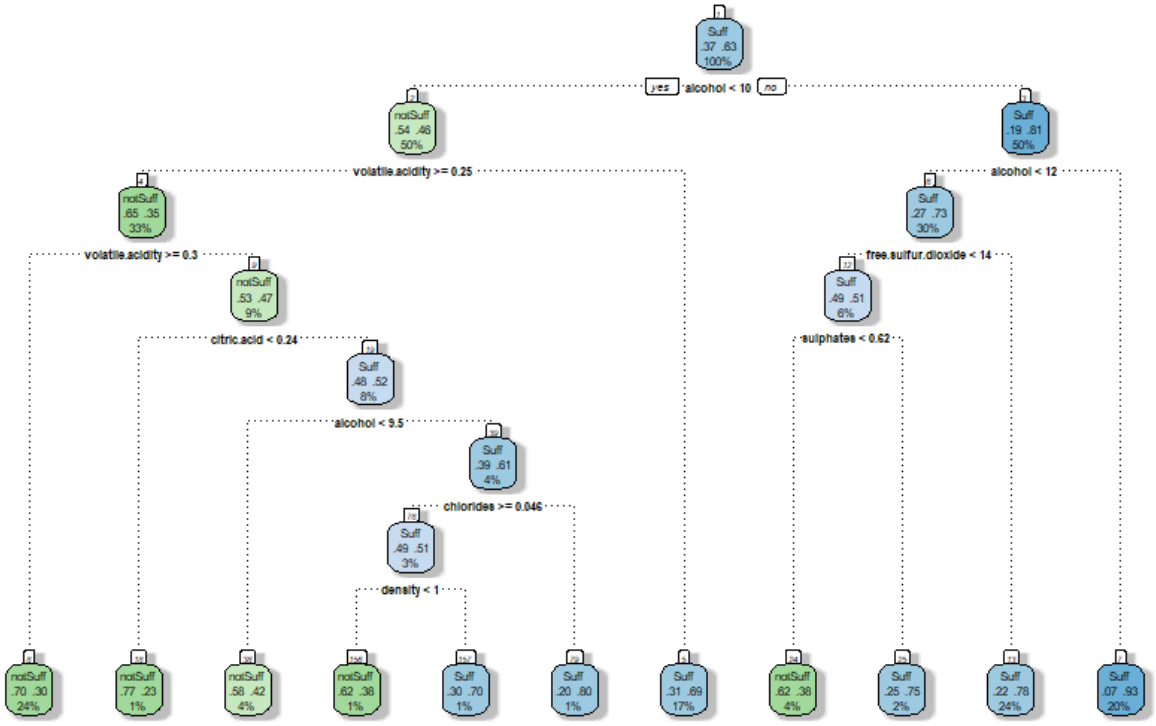
Dopo una attenta valutazione dei dati, ci siamo concentrati sul migliorare l'accuratezza del nostro modello.

```
dt.control = trainControl(method = "cv", number = 10, classProbs = TRUE, summaryFunction = twoClassSummary)
dt.model = train(quality ~ ., data=trainset, method="rpart", metric = "ROC", tuneLength = 5,
                 trControl = dt.control)
```

Nella funzione "trainControl" abbiamo aggiunto gli argomenti:

- **classProbs,** se a true, il metodo train ricalcola le classi di probabilità per la classificazione del modello per ogni ricampionamento.
- **summyFunction,** una funzione per calcolare le metriche delle prestazioni attraverso i ricampionamenti.

Nella funzione "train" invece ci siamo concentrati sul trovare quale fosse il miglior **tuneLength**, ovvero il numero di valori diversi da provare per ogni parametro di sintonizzazione. Dopo vari test abbiamo constatato che il migliore fosse 5. Il codice per disegnare l'albero e calcolare le varie statistiche è rimasto invariato.



Rattle 2021-set-10 21:25:46 User

Grazie a queste modifiche abbiamo ottenuto un albero diverso, con più nodi e l'accuratezza e le altre statistiche sono leggermente migliorate.

```
               Accuracy : 0.7481
                 95% CI : (0.7282, 0.7672)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.4499

 Mcnemar's Test P-Value : 0.01481

              Precision : 0.6712
                 Recall : 0.6198
                     F1 : 0.6445
             Prevalence : 0.3684
         Detection Rate : 0.2283
   Detection Prevalence : 0.3402
      Balanced Accuracy : 0.7213
```

Support Vector Machines

Le SVM è un algoritmo di apprendimento automatico supervisionato che può essere utilizzato per scopi di classificazione e di regressione. Ottiene la sua massima efficacia nei problemi di classificazioni binarie. L'idea di base è di trovare il miglior iperpiano separatore per separare un insieme di dati linearmente separabili.

Dopo aver suddiviso il nostro Dataset in Training Set e Test Set, siamo andati a studiare il modello con classificatore SVM che utilizza: *kernel lineare* ed *kernel non lineare*.

Per allenare il modello abbiamo utilizzato la libreria *Caret* la quale ci permette di utilizzare la funzione *train()* legato al metodo *trainControl()* che ci permette di impostare una convalida incrociata di K-fold. Inoltre, tale pacchetto ci permette di scegliere automaticamente i valori ottimali per i parametri di ottimizzazione del modello.

SVM Kernel Linear

Nel modello SVM Kernel Linear viene presa in considerazione la variabile *C,* chiamata anche *costo*, che permette di determinare i possibili misclassifications, ovvero impone una penalità al modello per aver commesso un errore; più il valore di *C* è alto ed minore è la possibilità che l'algoritmo SVM classifichi erroneamente un punto.

Di seguito riportiamo il codice utilizzato per l'allenamento del modello avente costo ***C=1***

```
> svm.model = train(
                    quality ~.,  #prende in considerazione tutte le variabili
                    data = trainset,
                    trControl = svm.control,
                    preProcess = c("center","scale"), #var scalabile per rendere comparabile la loro scala
                     method = "svmLinear"
                    )
> > svm.model$times
                                    $everything
                                      utente   sistema trascorso
                                       6.51     0.00     6.52
                                    $final
                                      utente   sistema trascorso
                                       0.61     0.00     0.61
>  svm.model
```

```
Support Vector Machines with Linear Kernel

4547 samples
  11 predictor
   2 classes: '0', '1'

Pre-processing: centered (11), scaled (11)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 4091, 4093, 4092, 4093, 4091, 4092, ...
Resampling results:

  Accuracy   Kappa
  0.7391572  0.412328

Tuning parameter 'C' was held constant at a value of 1
```

Come possiamo notare abbiamo una Accuracy del modello pari a 0.7391.

Abbiamo definito un algoritmo che ci permette di calcolare automaticamente SVM per valori diversi di C e scegliere quello ottimale che massimizza l' Accuracy.

```
> svm.model2 = train(
                quality ~.,
                data = trainset,
                trControl = svm.control,
                method = "svmLinear",
                tuneGrid = expand.grid(C = seq(0, 2, length = 20))
               )
```

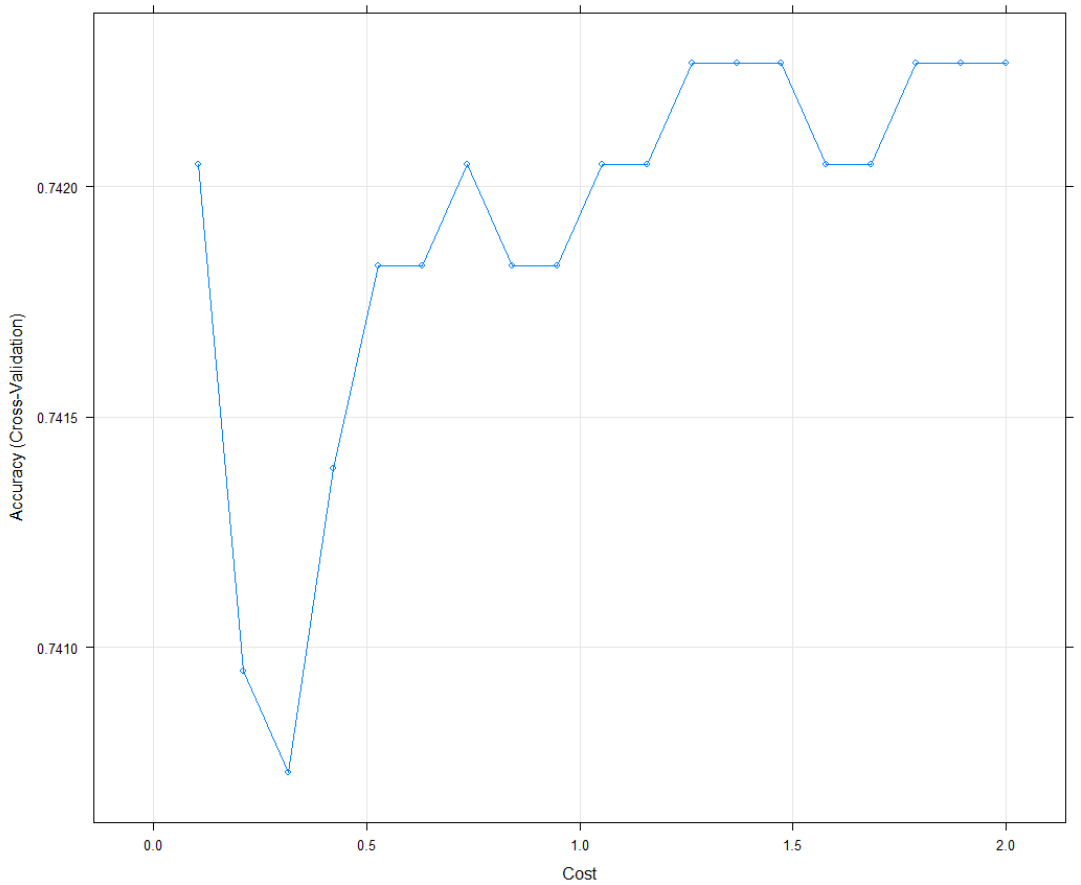
```
> svm.model2$times
                $everything
                  utente   sistema trascorso
                  106.00     0.22    106.38
                $final
                  utente   sistema trascorso
                   0.86     0.00     0.86
```

Di seguito vengono riportate i calcoli ottenuti di Accuracy per il range di costi [0; 20].

```
> svm.model2
```

C	Accuracy	Kappa
0.0000000	NaN	NaN
0.1052632	0.7420475	0.4169724
0.2105263	0.7409481	0.4150494
0.3157895	0.7407283	0.4146510
0.4210526	0.7413876	0.4163829
0.5263158	0.7418282	0.4175227
0.6315789	0.7418282	0.4176778
0.7368421	0.7420479	0.4180984
0.8421053	0.7418282	0.4176778
0.9473684	0.7418282	0.4176778
1.0526316	0.7420479	0.4180984
1.1578947	0.7420475	0.4180856
1.2631579	0.7422677	0.4186736
1.3684211	0.7422677	0.4186736
1.4736842	0.7422677	0.4186736
1.5789474	0.7420484	0.4182657
1.6842105	0.7420475	0.4180856
1.7894737	0.7422677	0.4186736
1.8947368	0.7422677	0.4186736
2.0000000	0.7422682	0.4186864

> plot(svm.model2)



Come possiamo notare dai due diversi grafici, abbiamo una migliore Accuracy, pari a 0.742, legata al costo *C=2*.

SVM Kernel Not Linear

Per studiare un modello SVM non lineare, possiamo utilizzare una funzione kernel radiale o polinomiale.

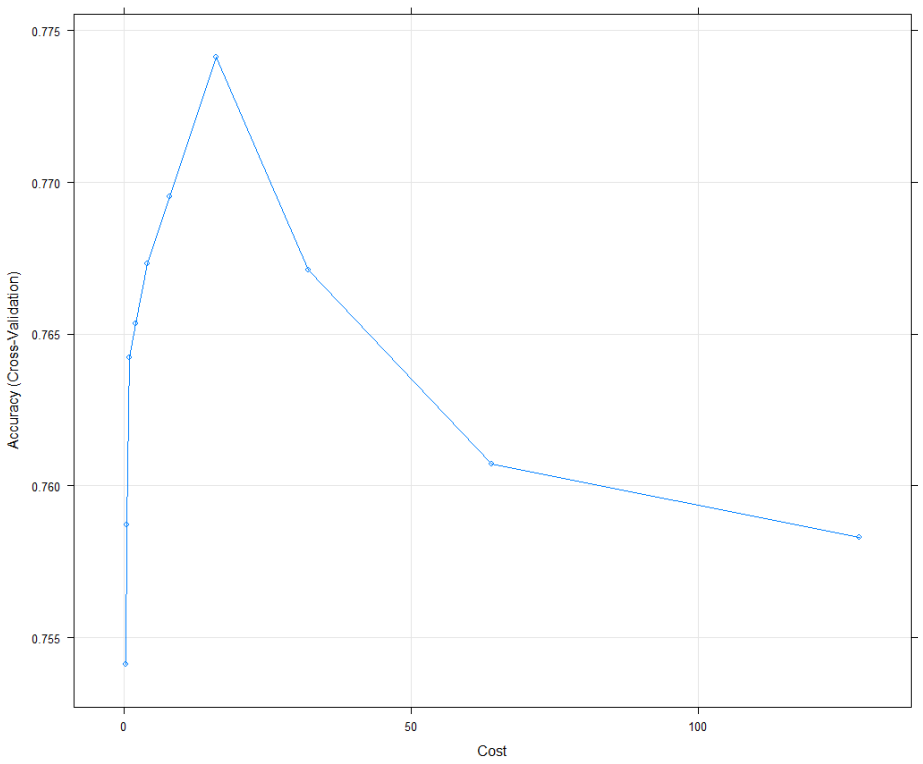
La funzione da noi utilizzata è la kernel radiale, di seguito riportiamo il codice utilizzato:

```
> svm.model3 = train(
                quality ~.,
                data = trainset,
                trControl = svm.control,
                method = "svmRadial",
                tuneLength = 10 #n. di valori diversi da provare per ogni parametro di sintonizzazione
                )
> svm.model3$times
                        $everything
                          utente   sistema trascorso
                          170.28     1.49    172.16
                        $final
                          utente   sistema trascorso
                           1.94      0.04     1.98
```

> svm.model3

```
Support Vector Machines with Radial Basis Function Kernel

4547 samples
  11 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 4092, 4092, 4093, 4092, 4092, 4093, ...
Resampling results across tuning parameters:

  C       Accuracy   Kappa
    0.25  0.7541083  0.4476559
    0.50  0.7587276  0.4618262
    1.00  0.7642269  0.4767669
    2.00  0.7653321  0.4810783
    4.00  0.7673169  0.4882103
    8.00  0.7695191  0.4944328
   16.00  0.7741364  0.5048588
   32.00  0.7671010  0.4917887
   64.00  0.7607245  0.4779352
  128.00  0.7583064  0.4741292

Tuning parameter 'sigma' was held constant at a value of 0.1000997
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.1000997 and C = 16.
```

> plot (svm.model3)



Come possiamo notare dai due grafici, con un valore costante di sigma = 0.1000997 ed un costo C = 16 abbiamo una migliore Accuracy pari a 0.7741362

Confronto Modelli SVM

Dall'analisi fatte riportiamo qui sotto un sommario dei dati relativi ad ogni modello studiato, portando all'attenzione non solo l'accuratezza ma bensì il tempo di esecuzione per allenare il modello che risulta essere almeno 10 volte maggiore rispetto a quello iniziale.

In conclusione il miglior modello risulta essere SVM Linear poiché ci rende un'accuratezza di circa 74% in un tempo accettabile.

Model <chr>	**Accuracy**	**Time**
SVM Radial	0.7741362	172.16
SVM Linear scegliendo il costo	0.7422682	106.38
SVM Linear	0.7391572	6.52

SVM viene utilizzato anche per la previsione numerica ed ora che abbiamo allenato il nostro modello con *SVM Linear*, possiamo prevedere le classi per il nostro Test Set usando il metodo *predict()* avente come primo parametro il modello allenato ed il secondo parametro il nostro frame di dati di test. I valori predittivi vengono poi comparati con i valori effettivi per analizzare com'è andata la predizione.

> svm.pred = predict(svm.model, testset[,! names(testset) %in% c("quality")])

Come possiamo notare dalla tabella abbiamo ottenuto dei valori accettabili della predizione.

svm.pred	*0*	*1*
0	*405*	*176*
1	*313*	*1055*

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0  405  176
         1  313 1055

               Accuracy : 0.7491
                 95% CI : (0.7292, 0.7682)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4385

 Mcnemar's Test P-Value : 7.742e-10

              Precision : 0.6971
                 Recall : 0.5641
                     F1 : 0.6236
             Prevalence : 0.3684
         Detection Rate : 0.2078
   Detection Prevalence : 0.2981
      Balanced Accuracy : 0.7105

       'Positive' Class : 0
```

Analisi dei dati raccolti (Roc)

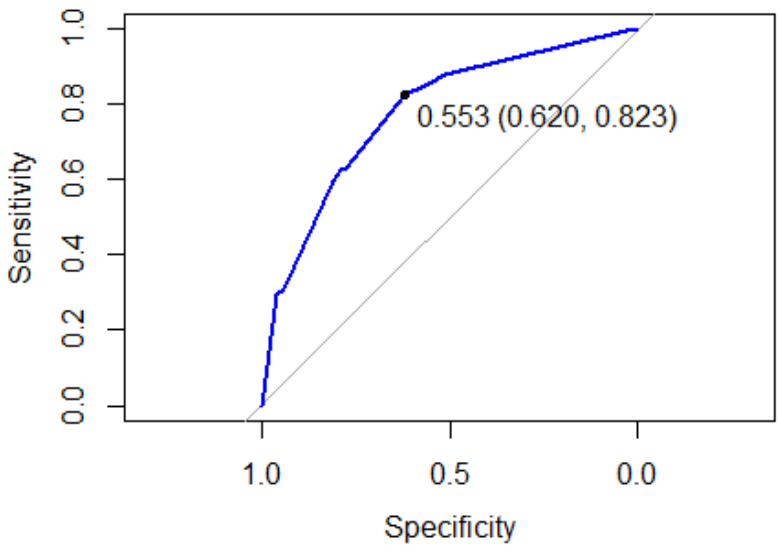
Per Analizzare i due modelli e poi effettuare una comparazione, iniziamo a controllare i valori ricevuti dalle confusion Matrix di entrambi i modelli. Per ottenerle, è stata effettuata precedentemente un processo di 10-fold Cross Validation ad entrambi i modelli, per poi successivamente effettuare i calcoli per ottenere le confusion Matrix di ognuno dei modelli stessi; dopo aver effettuato una modifica per permettere loro di ottenere dati anche sulle curve ROC. Per Il Decision Tree abbiamo:

```
          Reference
Prediction notSuff Suff
   notSuff     445  218
   Suff        273 1013

               Accuracy : 0.7481
                 95% CI : (0.7282, 0.7672)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.4499

 Mcnemar's Test P-Value : 0.01481

              Precision : 0.6712
                 Recall : 0.6198
                     F1 : 0.6445
             Prevalence : 0.3684
         Detection Rate : 0.2283
   Detection Prevalence : 0.3402
      Balanced Accuracy : 0.7213

       'Positive' Class : notSuff
```

```
          Reference
Prediction notSuff Suff
   notSuff     445  218
   Suff        273 1013

               Accuracy : 0.7481
                 95% CI : (0.7282, 0.7672)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.4499

 Mcnemar's Test P-Value : 0.01481

              Precision : 0.7877
                 Recall : 0.8229
                     F1 : 0.8049
             Prevalence : 0.6316
         Detection Rate : 0.5198
   Detection Prevalence : 0.6598
      Balanced Accuracy : 0.7213

       'Positive' Class : Suff
```

Nel Decision Tree abbiamo dunque una accuratezza del 74,81%. Possiamo inoltre notare una differenza sostanziale tra il valore di recall nella classe minoritaria (notSuff) e la classe maggioritaria (Suff). Ciò potrebbe essere dato dal fatto che la nostra variabile target è sbilanciata verso la classe “sufficiente”.

Di seguito viene mostrata anche la curva ROC con threshold migliore relativa al decision Tree:



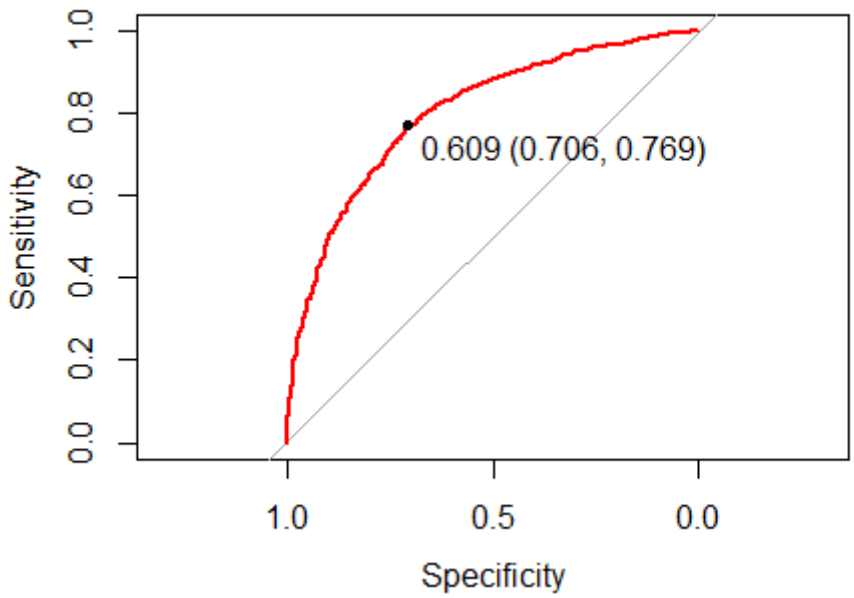
Per quanto riguarda la SVM invece, ecco la relativa Confusion Matrix:

```
          Reference
Prediction notSuff Suff
   notSuff     387  167
   Suff        331 1064

               Accuracy : 0.7445
                 95% CI : (0.7245, 0.7637)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4235

 Mcnemar's Test P-Value : 2.789e-13

              Precision : 0.6986
                 Recall : 0.5390
                     F1 : 0.6085
             Prevalence : 0.3684
         Detection Rate : 0.1986
   Detection Prevalence : 0.2842
      Balanced Accuracy : 0.7017

       'Positive' Class : notSuff
```

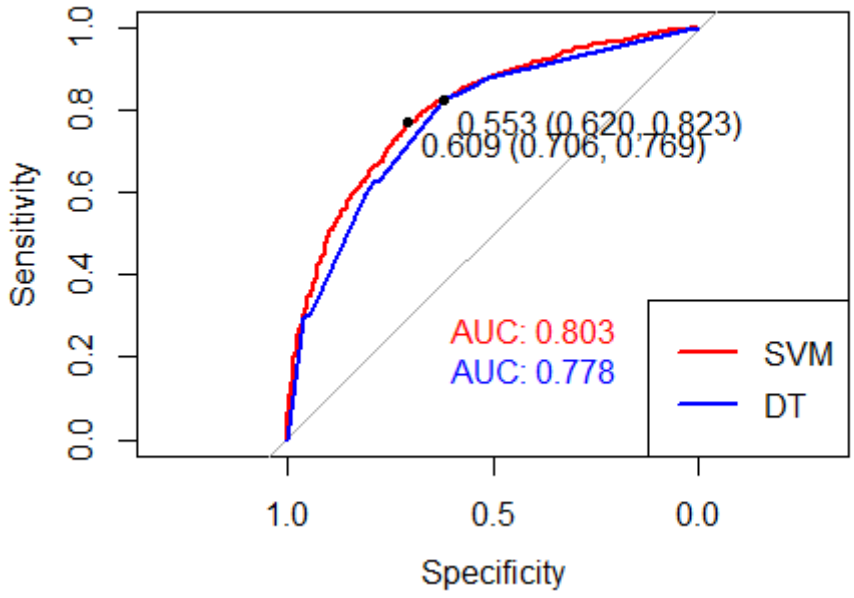
```
          Reference
Prediction notSuff Suff
   notSuff     387  167
   Suff        331 1064

               Accuracy : 0.7445
                 95% CI : (0.7245, 0.7637)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4235

 Mcnemar's Test P-Value : 2.789e-13

              Precision : 0.7627
                 Recall : 0.8643
                     F1 : 0.8104
             Prevalence : 0.6316
         Detection Rate : 0.5459
   Detection Prevalence : 0.7158
      Balanced Accuracy : 0.7017

       'Positive' Class : Suff
```

Possiamo notare che abbiamo una accuratezza del 74,45%, leggermente inferiore rispetto all'accuratezza mostrata dal Decision Tree; inoltre, notiamo come nella classe minoritaria (notSuff) vi sia un valore di recall del 53,90%, ben diverso dal valore di Recall di 86,43% presente nella classe maggioritaria. Anche in questo caso ciò può essere considerato a causa del fatto che i due valori della variabile target sono sbilanciati l'uno rispetto all'altro.

La curva ROC con threshold migliore per la SVM invece risulta essere questa:



Questo è invece il grafico per la AUC-ROC del nostro modello:



Conclusioni

Tramite la nostra analisi, possiamo stabilire che tra i due modelli da noi descritti, quello che si presta maggiormente al compito di prevedere se un vino sia sufficiente o meno è la SVM; In primo luogo, possiamo notare come l'area sottesa dalle curve AUC sia maggiore per la SVM rispetto a quella del Decision Tree. In secondo luogo possiamo notare come tra i due, la SVM presenta un F-score maggiore per la positive class "sufficiente", mentre il Decision Tree presenta un F-score maggiore per la positive class "non Sufficiente"; tra i due, riteniamo che sia più importante la possibilità di prevedere correttamente quali siano le bottiglie sufficienti.

Una ulteriore considerazione deve essere fatta sul valore di Accuracy, in quanto nonostante sia leggermente più alto per il Decision Tree, bisogna considerare che il modello di SVM utilizzato per la comparazione tra i due è quello più semplice, con Kernel lineare non ottimizzato; riteniamo infatti che nel caso di una successiva ottimizzazione del modello (da noi non fatta per cercare di tenere i valori di time paragonabili tra SVM e DT) il valore di Accuracy possa migliorare.

L'ultima nota riguarda il tempo di generazione di modello, unico punto in cui eccelle il modello del Decision Tree rispetto alla SVM.