



Anno Accademico 21/22

Progetto di Machine Learning

WATER POTABILITY

Cognome e Nome	Matricola
Colombo Andrea	844557
Sala Mattia	889809

Sommario

1 – Dominio di Riferimento.....	4
2 - Obiettivo.....	4
3 - Scelte di design e assunzioni	5
4 - Pacchetti R Installati e Ambiente di sviluppo.....	6
5 - Descrizione del Training Set: Analisi Esplorativa	6
Formattazione dati e ricerca valori “NA”	6
Correlazione tra variabili.....	9
Pulizia Dati.....	10
Ricerca valori anomali.....	11
Distribuzione di potabilità.....	15
6 - Principal Component Analisys (PCA)	16
7 - Descrizione dei modelli di Machine Learning usati	20
7.1 Modello Base.....	20
7.2 Decision Tree.....	21
7.3 Gradient Boosting	26
8 - Esperimenti.....	29
8.1) 10-Fold cross validation	29
8.2) Confusion Matrix Complessiva, Precision, Recall, F-Measure	32
8.3) ROC & AUC	34
9 – Conclusioni.....	36

1 – Dominio di Riferimento

L'accesso all'acqua potabile è un diritto fondamentale per la salute dell'essere umano. Questo è importante come questione di salute e sviluppo a livello nazionale, regionale e locale. In alcune regioni, è stato dimostrato che gli investimenti nell'approvvigionamento idrico e nei servizi igienico-sanitari possono produrre un beneficio economico, poiché le riduzioni degli effetti negativi sulla salute e dei costi sanitari superano i costi di realizzazione degli interventi.

Il dataset di riferimento su cui si basa questo progetto è “*Water Quality*”, reperibile al seguente link:

<https://www.kaggle.com/datasets/adityakadiwal/water-potability>

Il dataset contiene 3276 record di diversi campioni d'acqua analizzati.

2 - Obiettivo

L'obiettivo dell'elaborato è di risolvere un problema di classificazione legato alla potabilità di un campione d'acqua. Per farlo, verranno impiegati due algoritmi di apprendimento supervisionato e il dataset sopra citato.

3 - Scelte di design e assunzioni

Il dataset di riferimento contiene i seguenti attributi:

Attributo	Descrizione
pH	Il pH è un parametro importante per valutare l'acidità o la basicità dell'acqua. La "WHO" considera potabile l'acqua con un pH che varia tra 6,5 e 8,5. Nel dataset considerato, il range varia tra 6.52 e 6.83, che rientra nei parametri definiti dalla WHO.
Hardness	La durezza è causata principalmente dai sali di calcio e magnesio e viene definita come la capacità dell'acqua di precipitare il sapone causata da calcio e magnesio.
Solids (Total Dissolved Solids)	L'acqua ha la capacità di dissolvere un'ampia gamma di minerali o sali inorganici e alcuni organici. Questi minerali producono un gusto indesiderato e un colore diluito nell'aspetto dell'acqua. Questo parametro è importante per l'uso dell'acqua. L'acqua con un valore TDS elevato indica che l'acqua è altamente mineralizzata. Il limite desiderabile per la TDS è 500 mg/l e il limite massimo è 1000 mg/l prescritto per bere.
Chloramines	Cloro e cloramina sono i principali disinfettanti utilizzati nei sistemi idrici pubblici. Le clorammine si formano più comunemente quando l'ammoniaca viene aggiunta al cloro per trattare l'acqua potabile. I livelli di cloro fino a 4 milligrammi per litro (mg/L o 4 parti per milione (ppm)) sono considerati sicuri nell'acqua potabile.
Sulfate	I solfati sono sostanze presenti in natura che si trovano nei minerali, nel suolo e nelle rocce. La concentrazione di solfato nell'acqua di mare è di circa 2.700 milligrammi per litro (mg/L). Varia da 3 a 30 mg/L nella maggior parte delle riserve di acqua dolce.
Conductivity	La conducibilità elettrica (EC) misura effettivamente il processo ionico di una soluzione che le consente di trasmettere corrente. Secondo gli standard dell'OMS, il valore EC non deve superare i 400 µS/cm.
Organic_carbon	Il carbonio organico totale (TOC) nelle acque di sorgente proviene dalla materia organica naturale in decomposizione (NOM) e da fonti sintetiche. Il TOC è una misura della quantità totale di carbonio nei composti organici in acqua pura. Secondo l'EPA statunitense < 2 mg/L come TOC nell'acqua trattata/potabile e < 4 mg/L nell'acqua di sorgente utilizzata per il trattamento.
Trihalomethanes (TMH)	I THM sono sostanze chimiche che possono essere trovate nell'acqua trattata con cloro. La concentrazione di THM nell'acqua potabile varia in base al livello di materiale organico nell'acqua, alla quantità di cloro necessaria per trattare l'acqua e alla temperatura dell'acqua da trattare. I livelli di THM fino a 80 ppm sono considerati sicuri nell'acqua potabile.
Turbidity	La torbidità dell'acqua è una misura delle proprietà di emissione di luce dell'acqua. Il valore medio di torbidità raccomandato dall'OMS è 5,00 NTU.
Potability	Variabile target che indica se l'acqua è: 0 = Non Potabile 1 = Potabile

4 - Pacchetti R Installati e Ambiente di sviluppo

```
dplyr,forcats,ggplot2,skimr,scales,tidyverse,ggpubr,corrplot,  
RColorBrewer,varImp,GGally,c("FactoMinER","factoextra"),rpart,  
rattle,rpart.plot,RColorBrewer,xgboost,caTools,gbm,irr,C50,caret,  
ROCR,pROC
```

Viene utilizzato RStudio come ambiente di sviluppo per l'intero progetto.

5 - Descrizione del Training Set: Analisi Esplorativa

Formattazione dati e ricerca valori “NA”

Inizialmente è stata eseguita una lettura dei dati del dataset.

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability	
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
1	NA	205.	20791.		7.30	369.	564.	10.4	87.0	2.96	0
2	3.72	129.	18630.		6.64	NA	593.	15.2	56.3	4.50	0
3	8.10	224.	19910.		9.28	NA	419.	16.9	66.4	3.06	0
4	8.32	214.	22018.		8.06	357.	363.	18.4	100.	4.63	0
5	9.09	181.	17979.		6.55	310.	398.	11.6	32.0	4.08	0
6	5.58	188.	28749.		7.54	327.	280.	8.40	54.9	2.56	0
7	10.2	248.	28750.		7.51	394.	284.	13.8	84.6	2.67	0
8	8.64	203.	13672.		4.56	303.	475.	12.4	62.8	4.40	0
9	NA	119.	14286.		7.80	269.	389.	12.7	53.9	3.60	0
10	11.2	227.	25485.		9.08	404.	564.	17.9	72.0	4.37	0
# ... with 3,266 more rows											

Come si può notare, la variabile target è di tipo “double”. Per una migliore operabilità si è convertita in “factor”, in quanto può assumere solo due valori (0 e 1).

Qui sotto il codice:

```
> water_potability$Potability <- as.factor(water_potability$Potability)
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl> <fct>
1	NA	205.	20791.		7.30	369.	564.	10.4	87.0	2.96 0
2	3.72	129.	18630.		6.64	NA	593.	15.2	56.3	4.50 0
3	8.10	224.	19910.		9.28	NA	419.	16.9	66.4	3.06 0
4	8.32	214.	22018.		8.06	357.	363.	18.4	100.	4.63 0
5	9.09	181.	17979.		6.55	310.	398.	11.6	32.0	4.08 0
6	5.58	188.	28749.		7.54	327.	280.	8.40	54.9	2.56 0
7	10.2	248.	28750.		7.51	394.	284.	13.8	84.6	2.67 0
8	8.64	203.	13672.		4.56	303.	475.	12.4	62.8	4.40 0
9	NA	119.	14286.		7.80	269.	389.	12.7	53.9	3.60 0
10	11.2	227.	25485.		9.08	404.	564.	17.9	72.0	4.37 0
# ... with 3,266 more rows										

Prima di iniziare l'analisi, viene effettuato un controllo e pulizia su eventuali dati mancanti (“NA”). Innanzitutto, viene visualizzata la quantità dei dati mancanti su tutti gli attributi del dataset.

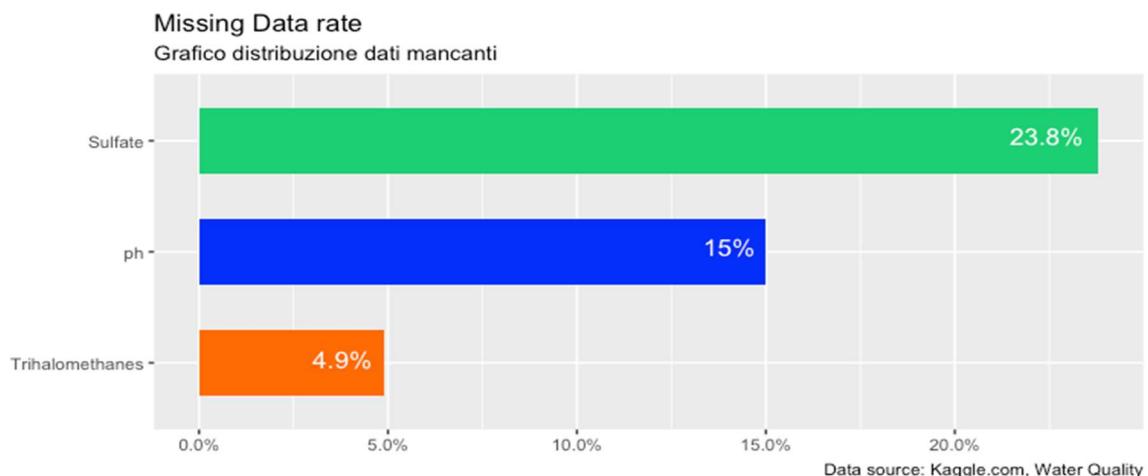
```
> water_potability %>% summarise_all(~ sum(is.na(.)))
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
1	491	0	0	0	781	0	0	162	0	0

Su tre attributi manca un discreto numero di dati. Si analizza quindi il missing data rate (rispetto alle 3276 righe del dataset) e viene visualizzato un grafico con i valori percentuali.

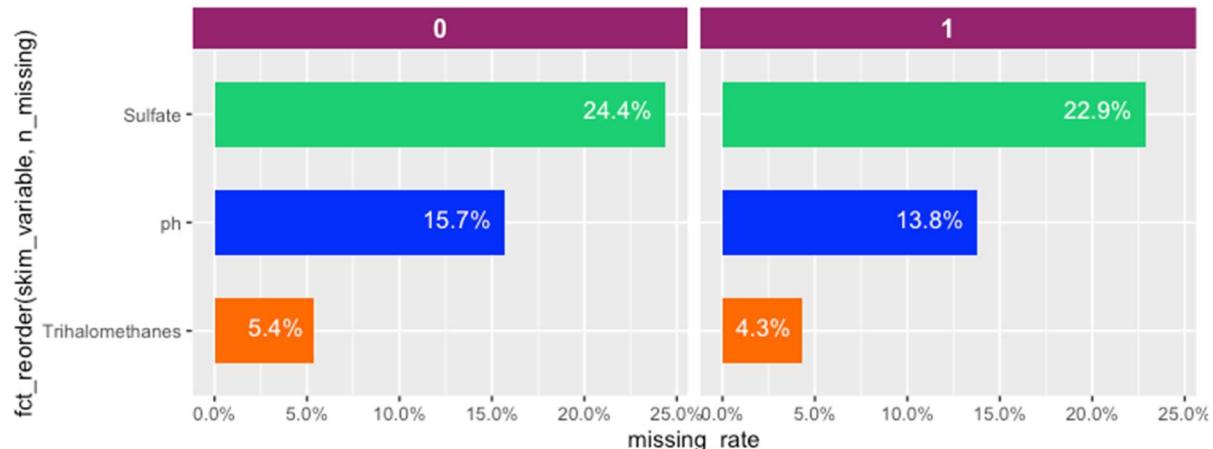
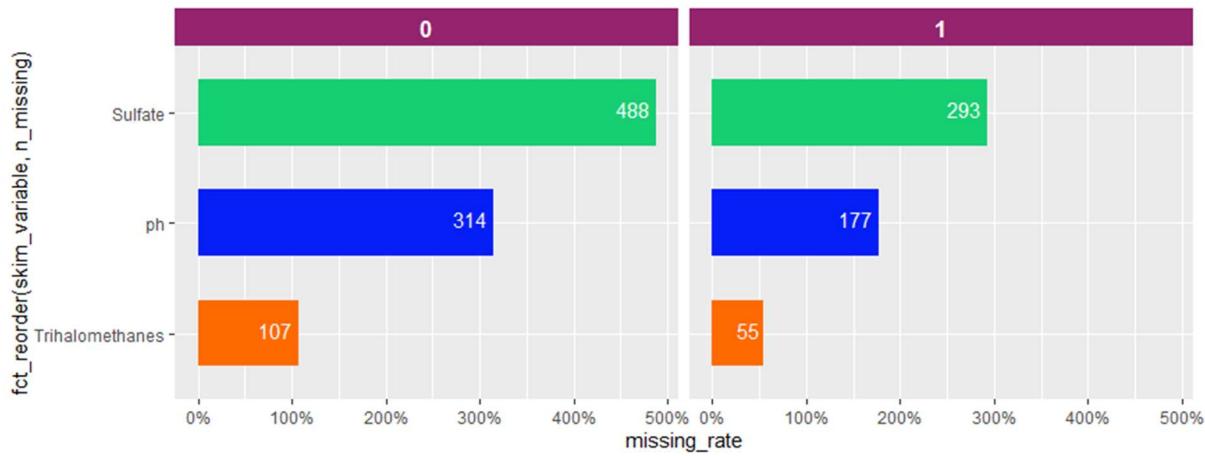
Di seguito viene mostrato il codice:

```
water_potability %>% skim() %>%
  filter(n_missing != 0) %>%
  as_tibble() %>%
  select(skim_variable, n_missing, complete_rate) %>%
  mutate(missing_rate = round(abs(complete_rate - 1) * 100, 1)) %>%
  ggplot(aes(
    x = fct_reorder(skim_variable, n_missing),
    y = missing_rate,
    fill = skim_variable,
    label = paste0(missing_rate, "%")
  )) +
  geom_col(width = .6) +
  geom_text(
    size = 4.5,
    hjust = 1.2,
    vjust = .25,
    col = "white"
  ) +
  coord_flip() + theme(aspect.ratio = .4) +
  theme(
    legend.position = "none"
  ) +
  scale_y_continuous(label = label_percent(scale = 1)) +
  scale_fill_manual(values = c("#071ff7",
                               "#17cf73",
                               "#ff6a00")) +
  labs(
    title = "Missing Data rate",
    subtitle = "Grafico distribuzione dati mancanti",
    caption = "Data source: Kaggle.com, Water Quality",
    x = NULL,
    y = NULL
  )
```



Viene ora studiata la distribuzione dei valori nulli rispetto alla variabile target, in modo da capire la diversa influenza che hanno sul risultato:

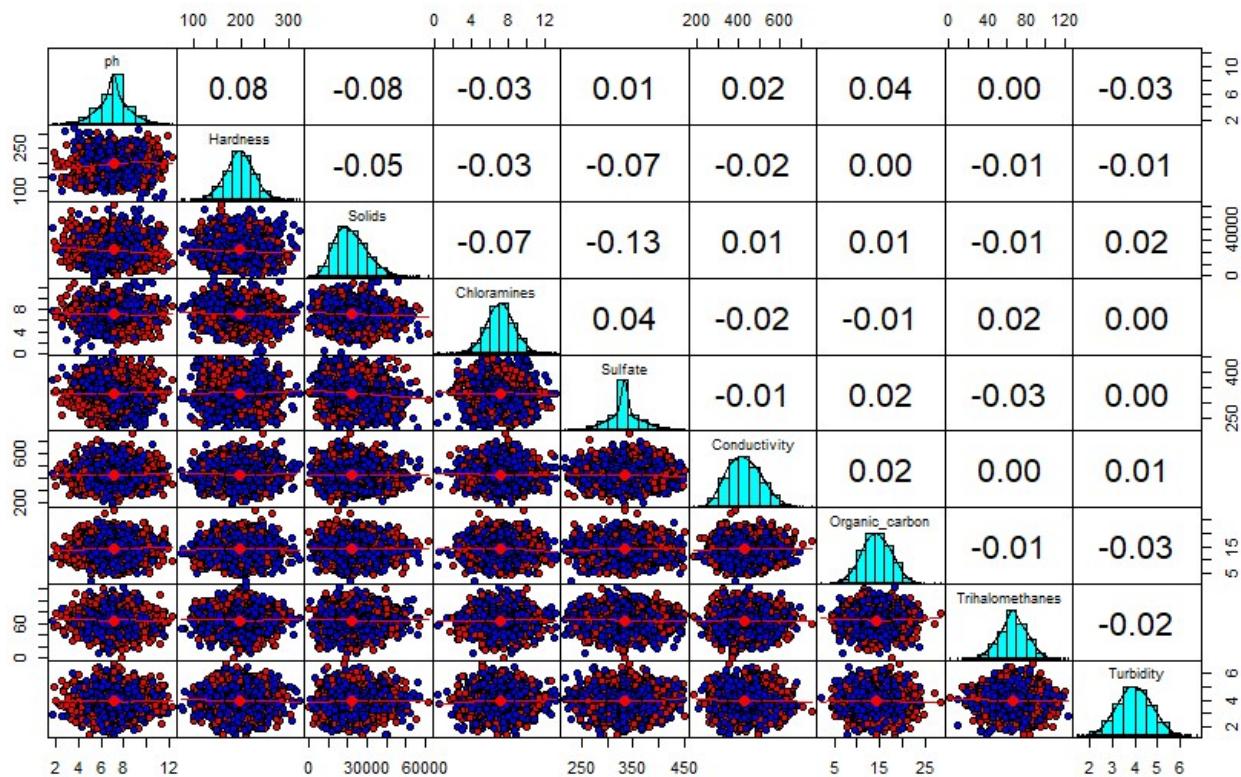
Vengono valutati il numero di dati mancanti per ciascun attributo, a seconda che la variabile target sia 0 oppure 1:



Come si può notare, la distribuzione è molto simile in entrambi i casi; quindi, la presenza di valori nulli non crea evidenti squilibri nella classificazione.

Correlazione tra variabili

Successivamente si osserva la correlazione tra gli attributi del dataset.



In questa matrice, dall'intersezione di due attributi presenti sulla diagonale principale si può ricavare la loro correlazione. Più il valore è vicino a 1, maggiore sarà la correlazione; mentre sarà minore quanto più vicino è a 0.

Come si evince, nessuna delle coppie considerate ha un valore di correlazione vicino a 1. Di conseguenza è possibile sostituire i vari valori mancanti, senza influenzare gli altri attributi.

Pulizia Dati

Le strategie possibili da attuare sono tre, ovvero:

- **Eliminazione delle colonne contenenti valori nulli superiori ad una soglia limite**
- **Eliminazione record contenenti valori nulli**
- **Sostituzione valori nulli con valori che non inficiano la classificazione**

Si è deciso di scartare la prima opzione in quanto si perderebbero molti dati fondamentali alla classificazione.

Si scarta anche la seconda opzione, in quanto verrebbe eliminato un consistente numero di record utili per la classificazione.

Considerando il risultato ottenuto precedentemente sulla correlazione tra gli attributi, si decide di usare la terza strategia; ovvero la sostituzione dei valori nulli con i valori medi degli attributi mancanti. In questo modo manteniamo intatto il numero di record del dataset, inserendo un valore neutro per la classificazione.

Di seguito viene presentato un sommario dei valori statistici principali, in particolare:

media, val. max, val. min, primo quartile, terzo quartile, mediana.

ph	Hardness	Solids	Chloramines	Sulfate	Conductivity
Min. : 0.000	Min. : 47.43	Min. : 320.9	Min. : 0.352	Min. : 129.0	Min. : 181.5
1st Qu.: 6.093	1st Qu.: 176.85	1st Qu.: 15666.7	1st Qu.: 6.127	1st Qu.: 307.7	1st Qu.: 365.7
Median : 7.037	Median : 196.97	Median : 20927.8	Median : 7.130	Median : 333.1	Median : 421.9
Mean : 7.081	Mean : 196.37	Mean : 22014.1	Mean : 7.122	Mean : 333.8	Mean : 426.2
3rd Qu.: 8.062	3rd Qu.: 216.67	3rd Qu.: 27332.8	3rd Qu.: 8.115	3rd Qu.: 360.0	3rd Qu.: 481.8
Max. : 14.000	Max. : 323.12	Max. : 61227.2	Max. : 13.127	Max. : 481.0	Max. : 753.3
NA's : 491					NA's : 781
Organic_carbon	Trihalomethanes	Turbidity	Potability		
Min. : 2.20	Min. : 0.738	Min. : 1.450	0:1998		
1st Qu.: 12.07	1st Qu.: 55.845	1st Qu.: 3.440	1:1278		
Median : 14.22	Median : 66.622	Median : 3.955			
Mean : 14.28	Mean : 66.396	Mean : 3.967			
3rd Qu.: 16.56	3rd Qu.: 77.337	3rd Qu.: 4.500			
Max. : 28.30	Max. : 124.000	Max. : 6.739			
NA's : 162					

Si procede con la sostituzione dei valori “**NA**” con il corrispettivo valor medio:

```
water_potability <- water_potability %>%
  group_by(Potability) %>%
  mutate(across(where(is.numeric),
    ~if_else(is.na(.),
      mean(., na.rm = T),
      as.numeric(.)))) %>% ungroup()
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<fct>
1	7.09	205.	20791.	7.30	369.	564.	10.4	87.0	2.96	0
2	3.72	129.	18630.	6.64	335.	593.	15.2	56.3	4.50	0
3	8.10	224.	19910.	9.28	335.	419.	16.9	66.4	3.06	0
4	8.32	214.	22018.	8.06	357.	363.	18.4	100.	4.63	0
5	9.09	181.	17979.	6.55	310.	398.	11.6	32.0	4.08	0
6	5.58	188.	28749.	7.54	327.	280.	8.40	54.9	2.56	0
7	10.2	248.	28750.	7.51	394.	284.	13.8	84.6	2.67	0
8	8.64	203.	13672.	4.56	303.	475.	12.4	62.8	4.40	0
9	7.09	119.	14286.	7.80	269.	389.	12.7	53.9	3.60	0
10	11.2	227.	25485.	9.08	404.	564.	17.9	72.0	4.37	0

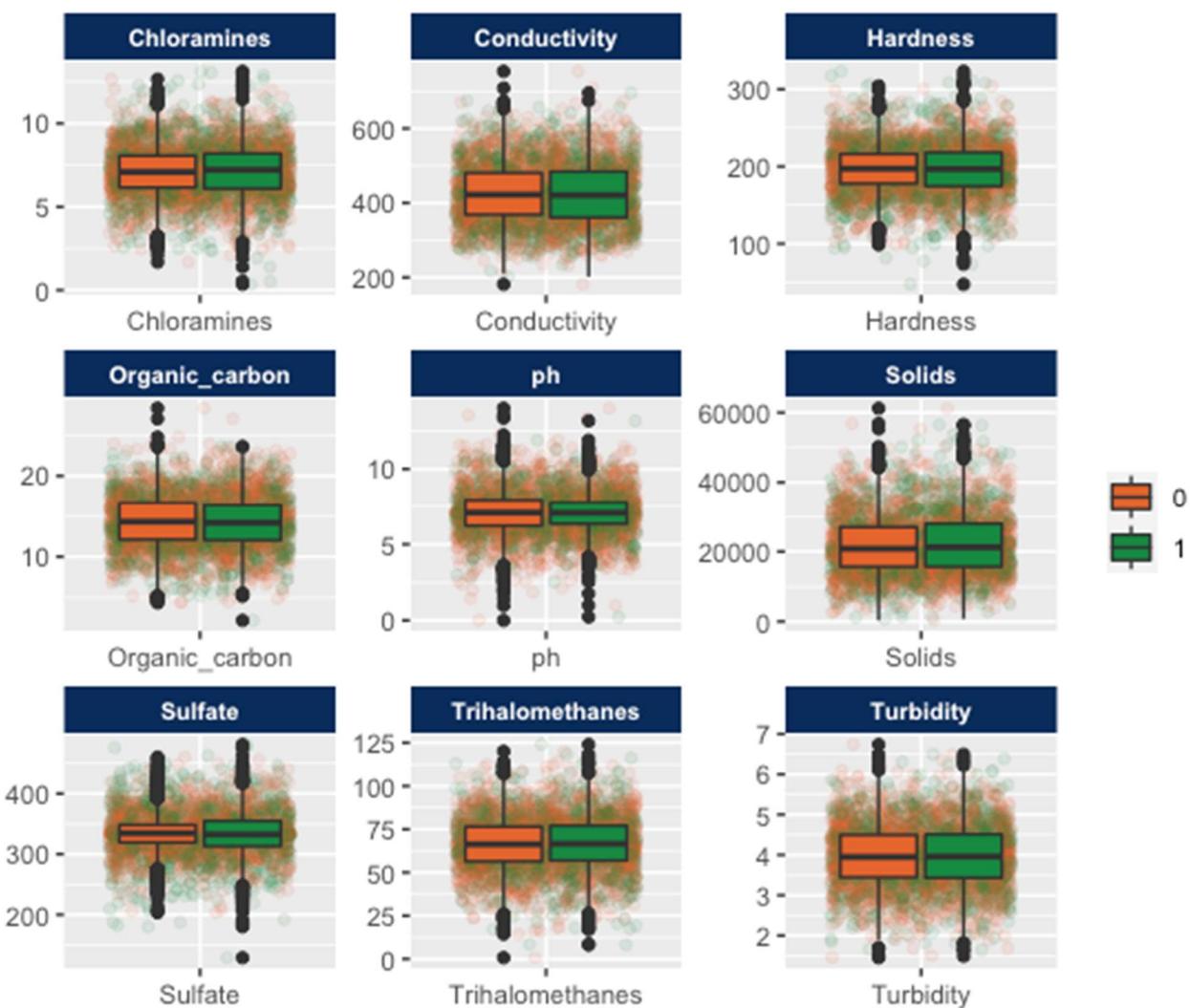
... with 3,266 more rows

Ricerca valori anomali

Viene ora analizzata la presenza di eventuali valori anomali (*Outliers*) che potrebbero compromettere l'analisi dei dati, in quanto non sono indicativi della loro reale distribuzione. Per verificare la presenza di questi valori viene usato un Box Plot, che consente di individuare eventuali dati che si discostano molto dai valori compresi tra 1° e 3° quartile.

```
water_potability %>%
  pivot_longer(cols = -Potability, names_to = "feature") %>%
  ggplot(aes(x = feature, y = value)) +
  geom_jitter(aes(y = value, col = Potability), alpha = 0.1) +
  geom_boxplot(aes(fill = Potability)) +
  facet_wrap(vars(feature), ncol = 3, scales = "free") +
  scale_color_manual(values = c("#E4652E", "#0E8A41")) +
  scale_fill_manual(values = c("#E4652E", "#0E8A41")) +
  theme(
    legend.position = "right",
    strip.background = element_rect(fill = "#0B2D5B"),
    strip.text = element_text(color = "white", face = "bold", size = 8)
  ) +
  labs(
    title = "Valori anomali",
    caption = "Data source: Kaggle.com, Water Quality",
    x = NULL,
    y = NULL,
    fill = NULL,
    color = NULL
  )
```

Valori anomali



Data source: Kaggle.com, Water Quality

Si nota che potrebbero esserci alcuni outliers, è necessario quindi calcolarne in modo più preciso il numero sfruttandone la definizione stessa:

- Nel lato inferiore, un outlier estremo si trova al di sotto del **Primo Quartile – 3 * Scarto Interquartile**
- Nel lato superiore, un outlier estremo si trova al di sopra del **Terzo Quartile + 3 * Scarto Interquartile**

Tramite il codice seguente si sostituiscono con un valore “**NA**”, in modo da poter essere contati.

```
outliers <- water_potability %>%
  select(-Potability)
outliers[] <- lapply(outliers, function(x){
  qq <- quantile(x, c(0.25, 0.75), na.rm = TRUE)
  is.na(x) <- x < (qq[1] - (3*(qq[2]-qq[1]))) | x > (qq[2] + (3*(qq[2]-
```

```

qq[1])))
x
})
outliers %>% summarise_all(~ sum(is.na(.)))

```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity
1	<int>	<int>	1	0	<int>	0	22	<int>	0
	9	1	0	0	0	0	0	0	0

Si nota che sono stati trovati (anche se pochi) degli outliers, i quali verranno sostituiti con il valore medio.

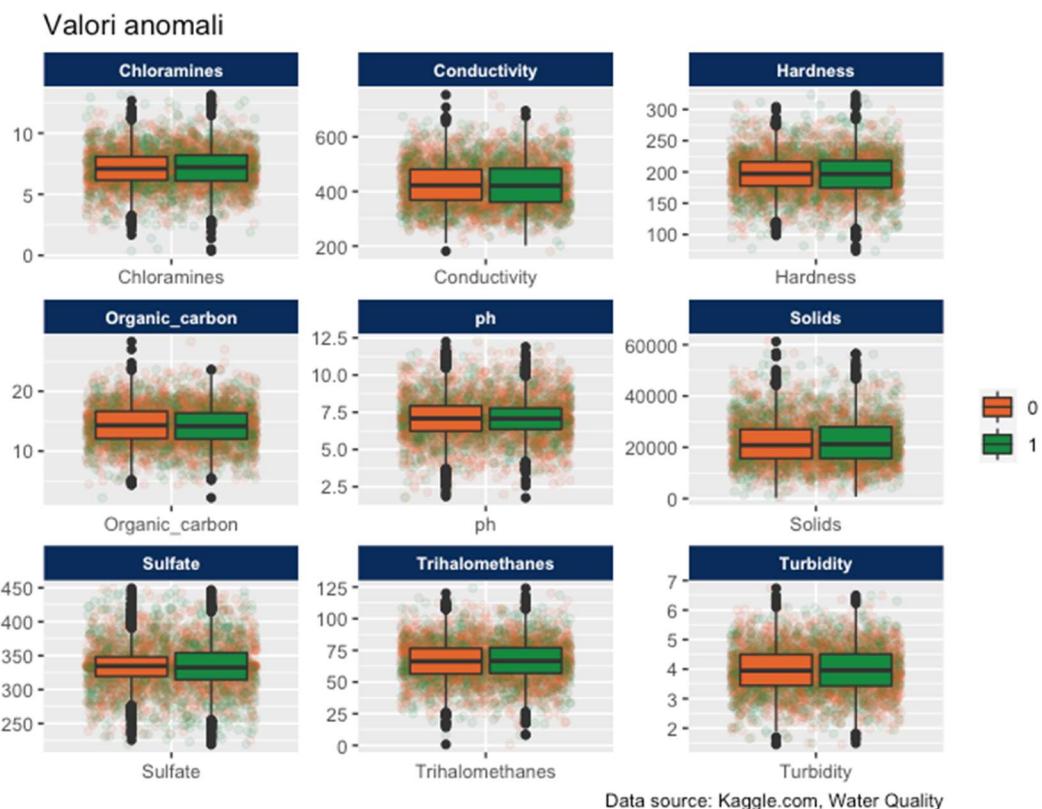
```

outliers <- outliers %>%
  mutate(across(where(is.numeric),
    ~if_else(is.na(.),
      mean(., na.rm = T),
      as.numeric(.)))) %>%
  ungroup()

outliers$Potability <- water_potability$Potability
water_potability <- outliers

```

Si procede visualizzando nuovamente il Box Plot relativo:



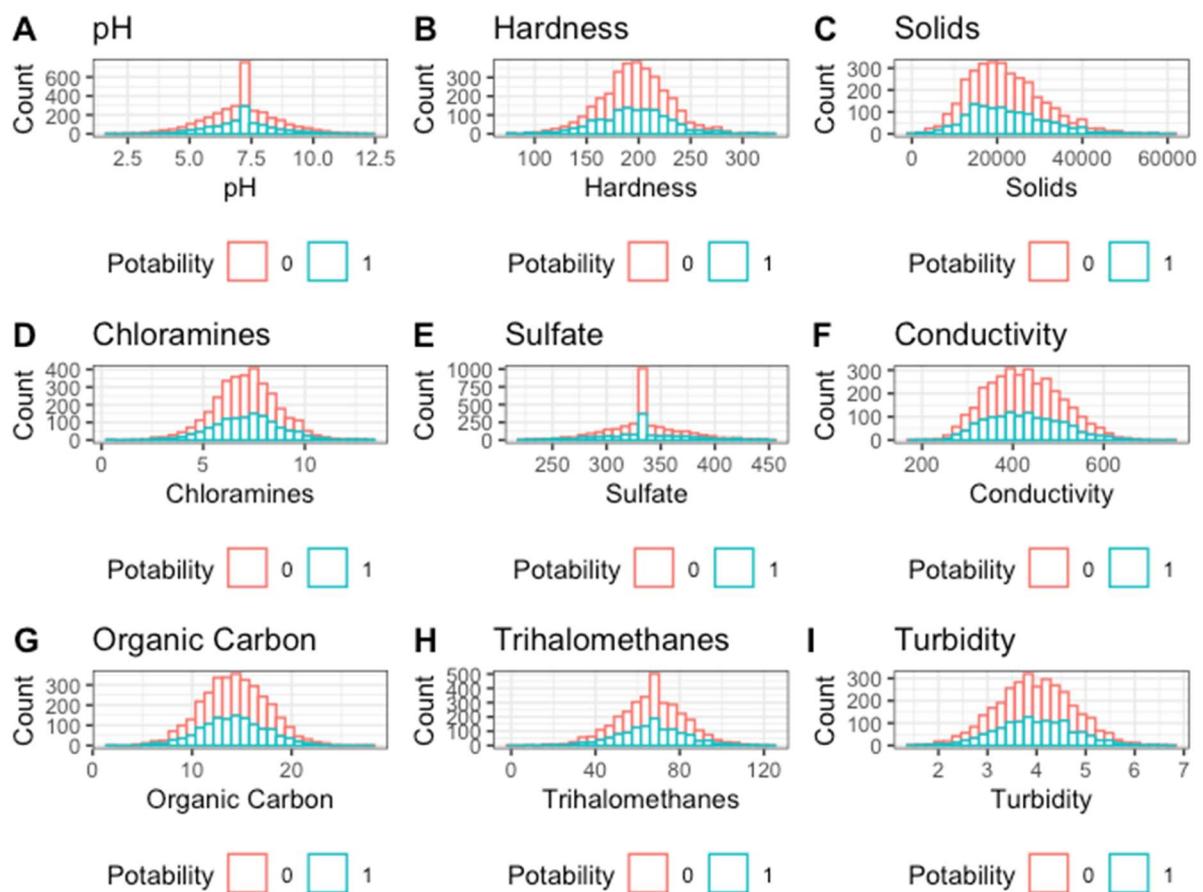
Viene ora visualizzato l'istogramma con la distribuzione dei dati di ciascun attributo, per avere un ulteriore riscontro dell'assenza di valori anomali.

```
p1 <- ggplot(water_potability, aes(ph, color = as.factor(Potability)))+
  geom_histogram(bins = 30, fill = "white") +
  labs(x = "pH", y = "Count", col = "Potability") +
  theme_bw() +
  theme(legend.position = "bottom")+
  labs(title = "pH")

p2 <- ...
...
p9 <- ...

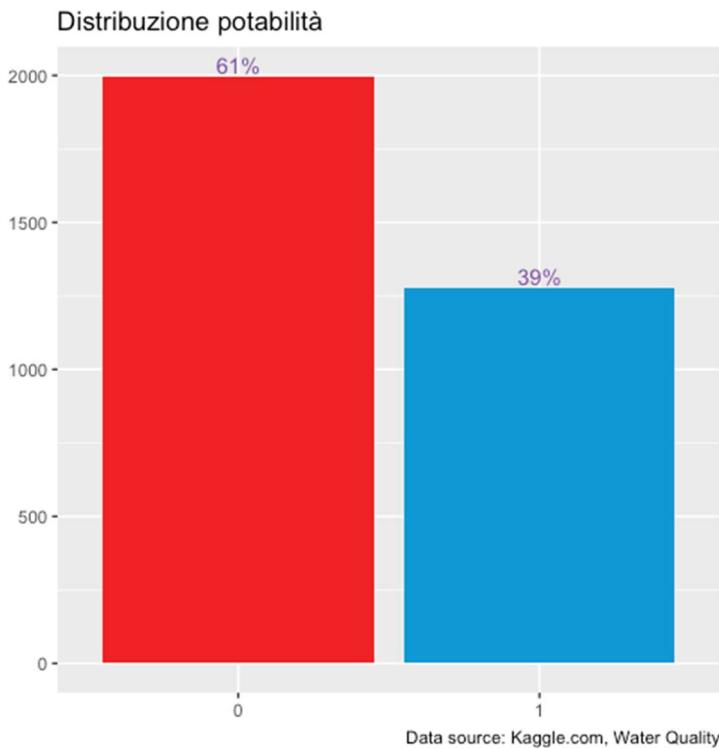
figure <- ggarrange(p1, p2, p3, p4, p5, p6, p7, p8, p9, nrow = 3, ncol = 3,
labels = "AUTO")

figure
```



Distribuzione di potabilità

```
water_potability %>%
  select(Potability) %>%
  count(Potability) %>% mutate(percent = paste0(round(n / sum(n) * 100),
"%%"), 2) %>%
  ggplot(aes(
    x = Potability,
    y = n,
    label = percent,
    fill = Potability
  )) +
  geom_col() +
  geom_text(vjust = -0.2, color = "#7C4EA8") +
  scale_fill_manual(values = c("#EF1A25", "#0099D5")) +
  labs(
    title = "Distribuzione potabilità",
    caption = "Data source: Kaggle.com, Water Quality",
    x = NULL,
    y = NULL,
    fill = NULL
  )
```



Viene calcolata la distribuzione della potabilità sulla variabile target.

Come si può osservare, il dataset è più sbilanciato verso la non potabilità. Questo risultato verrà tenuto in considerazione in fase di creazione del “modello base”.

6 - Principal Component Analisys (PCA)

In questa sezione viene svolta la Principal Component Analisys (PCA).

L'obiettivo principale è di individuare quali sono le componenti che spiegano la maggior parte della varianza cumulativa e, su queste, quali attributi incidono maggiormente.

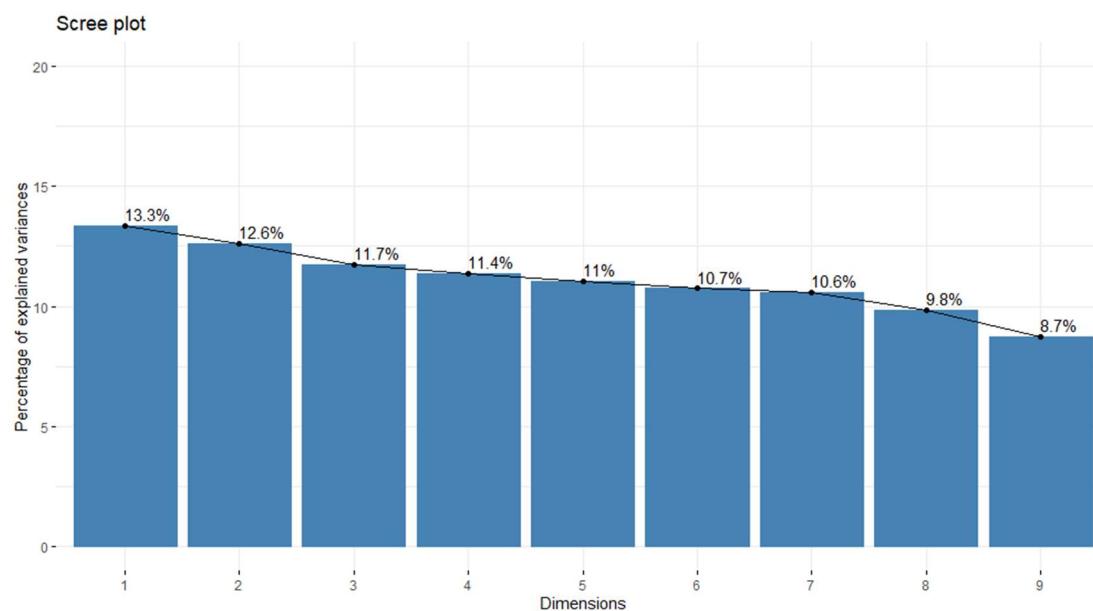
In questo modo è possibile utilizzarle in fase di training del modello, a supporto dell'apprendimento.

Di seguito viene presentato il codice per computare la PCA, proiettare il grafico per visualizzare la varianza di ciascuna componente e gli autovalori associati a ciascuna dimensione.

```
res.pca <- PCA(as.data.frame(water_potability[1:9]), graph=TRUE)
```

```
fviz_eig(res.pca, addlabels = TRUE, ylim = c(0, 20)) #scree plot
```

```
eig.val <- get_eigenvalue(res.pca)
```

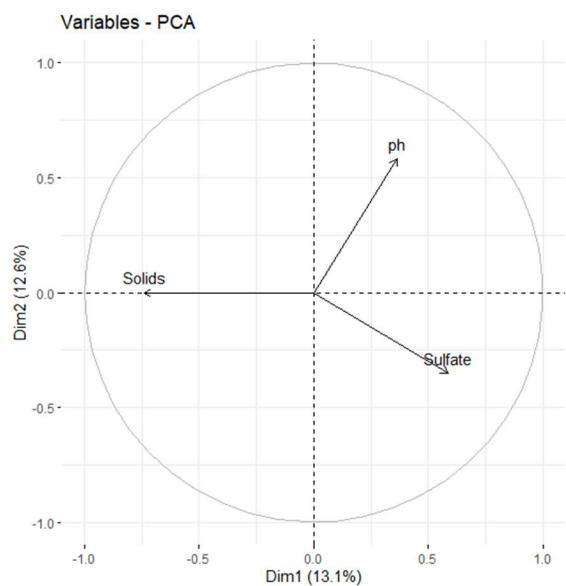
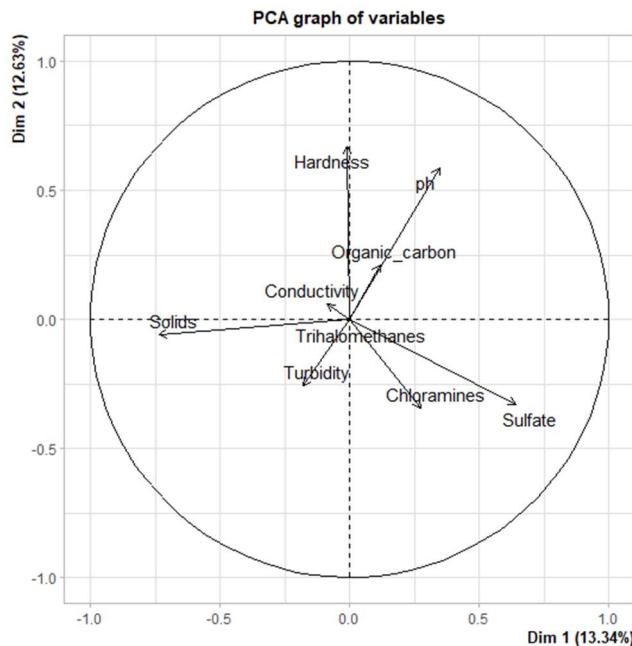


	eigenvalue	variance.percent	cumulative.variance.percent
Dim.1	1.2004998	13.338887	13.33889
Dim.2	1.1367111	12.630124	25.96901
Dim.3	1.0561206	11.734674	37.70368
Dim.4	1.0234416	11.371574	49.07526
Dim.5	0.9939919	11.044354	60.11961
Dim.6	0.9669696	10.744107	70.86372
Dim.7	0.9510282	10.566980	81.43070
Dim.8	0.8864939	9.849932	91.28063
Dim.9	0.7847432	8.719369	100.00000

In totale sono state individuate 9 dimensioni che spiegano il 100% della varianza.

Il 60% della varianza cumulativa è spiegato dalle prime cinque. Tra queste, le due più importanti – in termini di varianza “spiegata” – sono le prime due dimensioni. Queste spiegano però soltanto il 25% cumulativo della varianza (*rispettivamente 13% e 12%*).

Qui sotto vengono mostrati i grafici relativi all’incidenza di ciascuna variabile rispetto alle prime due dimensioni e, le top 3 variabili.



Dopodiché si vuole osservare che incidenza ha ciascun attributo sulle varie dimensioni:

```
# Contributions to the principal components
var <- get_pca_var(pca.res)
head(var$contrib)

> print(var$contrib)
      Dim.1      Dim.2      Dim.3      Dim.4      Dim.5
ph    10.06470865 30.393925813 0.7225635 0.00350143 3.8525823
Hardness 0.00744774 39.289165176 13.6011682 4.24244822 0.1774123
Solids 44.77028901 0.300224871 3.5306739 0.63331403 3.9691553
Chloramines 6.36135500 10.394958086 13.1186427 3.14559234 0.2940083
Sulfate 34.35780980 9.557580173 6.6014599 1.34842650 0.3506378
conductivity 0.61876989 0.300968664 24.1805278 3.44293943 50.5606717
organic_carbon 1.16319708 3.956313281 31.1385057 1.44445276 10.5007194
Trihalomethanes 0.00930994 0.001483623 6.5295975 58.01764088 5.4882900
Turbidity 2.64711288 5.805380314 0.5768609 27.72168440 24.8065229
> # Contributions to the principal components
> head(var$contrib)
      Dim.1      Dim.2      Dim.3      Dim.4      Dim.5
ph    10.06470865 30.3939258 0.7225635 0.00350143 3.8525823
Hardness 0.00744774 39.2891652 13.6011682 4.24244822 0.1774123
Solids 44.77028901 0.3002249 3.5306739 0.63331403 3.9691553
chloramines 6.36135500 10.3949581 13.1186427 3.14559234 0.2940083
Sulfate 34.35780980 9.5575802 6.6014599 1.34842650 0.3506378
conductivity 0.61876989 0.3009687 24.1805278 3.44293943 50.5606717
> |
```

Come si può vedere, gli attributi con più incidenza sulla “Dimensione 1”, sono:

- **Solids** 44%
- **Sulfate** 34%
- **pH** 10%

responsabili del **88%** della varianza sulla prima dimensione.

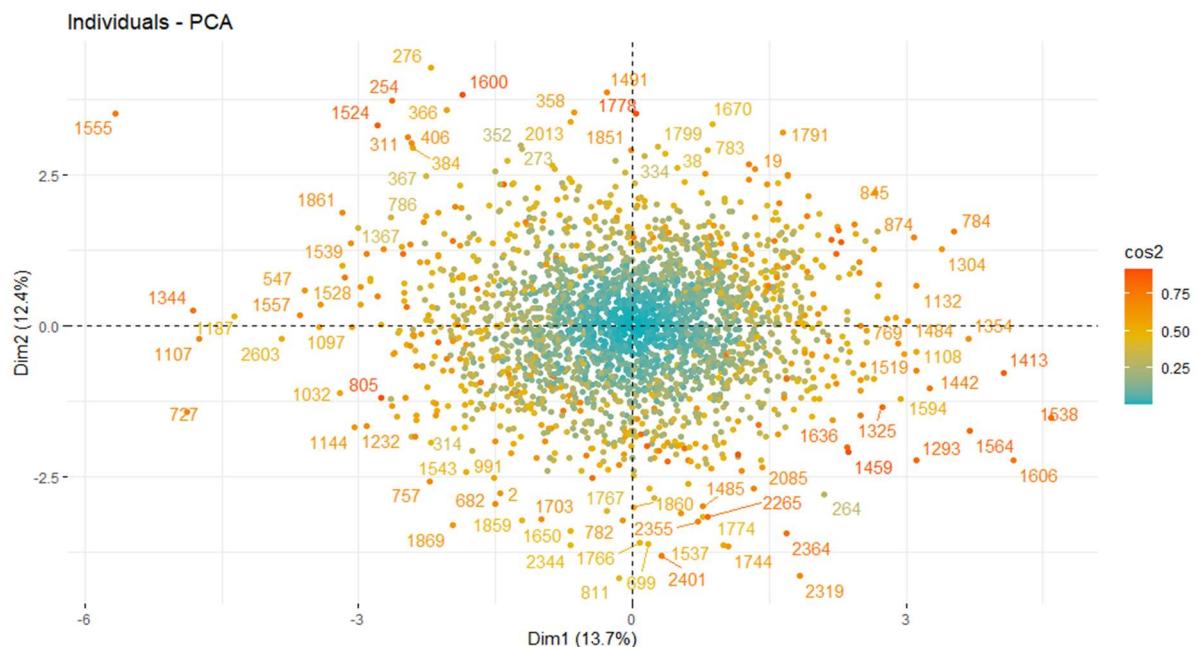
Per quanto riguarda la “Dimensione 2” invece, si ha:

- **pH** 30%
- **Hardness** 39%
- **Chloramines** 10 %

Responsabili del **79%** della varianza sulla seconda dimensione.

Successivamente si visualizzano gli “individui”:

```
fviz_pca_ind(pca.res, col.ind = "cos2",
              gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
              repel = TRUE # Avoid text overlapping (slow if many
points)
)
```



Il parametro “cos2” consente di distinguere gli individui se ben rappresentati dalle PCs (*valore maggiore*), da quelli non ben rappresentati (*valore minore*).

Come è possibile notare, la densità di individui ben classificati è nettamente inferiore a quella degli individui non ben classificati.

In conclusione di questa fase, è stato possibile individuare quali attributi hanno una maggiore incidenza sulle dimensioni con più varianza. I risultati ottenuti verranno tenuti in considerazione durante il training dei modelli scelti.

7 - Descrizione dei modelli di Machine Learning usati

La scelta di un algoritmo di apprendimento automatico su cui costruire un modello è un'operazione delicata e che richiede uno studio profondo del dominio di riferimento e dei dati a disposizione.

La prima cosa da fare è categorizzare il problema che si sta affrontando, in modo da escludere e priori algoritmi poco consoni.

Per questo motivo, essendo un problema di classificazione con solo 2 possibili classi, con valori numerici ed un numero relativamente basso di record e features si è deciso di utilizzare come primo algoritmo i Decision Tree.

La distribuzione dei valori delle varie features risulta molto concentrata in un range molto piccolo, condizione che porta all'esclusione di algoritmi che utilizzano un iperpiano per separare le classi.

Si è deciso dunque di utilizzare un algoritmo derivante sempre dai Decision Tree ma che ne consente un incremento sostanziale delle performance, il Gradient Boosting.

7.1 Modello Base

Si utilizza un modello base, che si basa sulla distribuzione di 0 e 1 della potabilità nel trainset e, la si utilizza per effettuare una predizione iniziale sul testset. Dopodichè si confronterà questa predizione con l'effettiva variabile target (Potability) del testset, calcolando l'accuratezza. Il testset viene inizializzato con variabile target a 0 (non potabile) in quanto secondo lo studio precedentemente effettuato, è il valore con la maggior distribuzione di probabilità all'interno del trainset.

```
testset$Prediction = rep(0, 982)
testset$Prediction = factor(testset$Prediction)

confusion.matrix = table(testset$Potability, testset$Prediction)

sum(diag(confusion.matrix))/sum(confusion.matrix)
```

L'**accuratezza** ottenuta dopo il calcolo della Confusion Matrix, è: 0.6089613 $\simeq \mathbf{0.61}$.

Il risultato ottenuto con un modello base non è molto accurato e rispecchia – ovviamente - la distribuzione della variabile target.

7.2 Decision Tree

Dopo aver suddiviso il dataset in: 70% Train e 30% Test, si procede con il training attraverso il modello degli **Alberi di Decisione** (DT).

Dai risultati ottenuti durante la fase di *Analisi delle componenti* (PCA), si allena il modello tenendo conto solo del seguente insieme di attributi:

```
attributes = { Sulfate; Solids; ph }
```

Dopodiché si effettuano le predizioni sul test set e si calcola l'accuratezza.

Infine, si pota l'albero impostando il valore del *Complexity Parameter* “**cp**”, ricavato dal corrispondente grafico e, si calcola nuovamente l'accuratezza.

```
decisionTree <- rpart(Potability ~ {attributes}, data=trainset, method="class")
fancyRpartPlot(decisionTree)
printcp(decisionTree)
plotcp(decisionTree)

#accuracy with trained DT
testset$Prediction <- predict(decisionTree, testset, type = "class")
confusion.matrix = table(testset$Potability, testset$Prediction)
sum(diag(confusion.matrix))/sum(confusion.matrix)

#after prune of tree
myPruned = prune(decisionTree, cp=.x)
fancyRpartPlot(myPruned)

testset$Prediction <- predict(myPruned, testset, type = "class")
confusion.matrix = table(testset$Potability, testset$Prediction)
sum(diag(confusion.matrix))/sum(confusion.matrix)
```

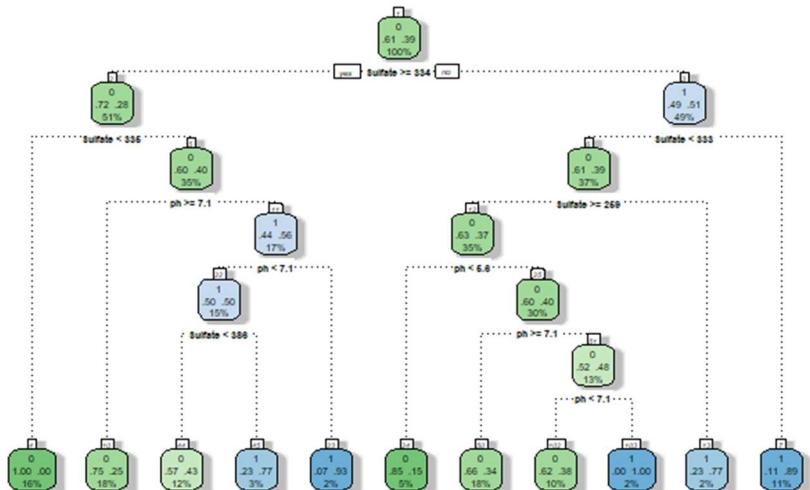
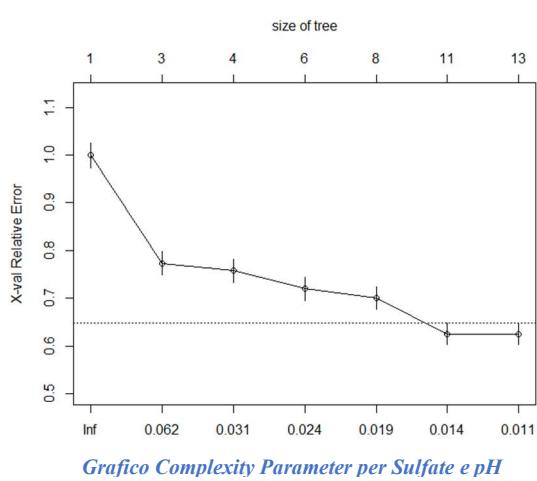
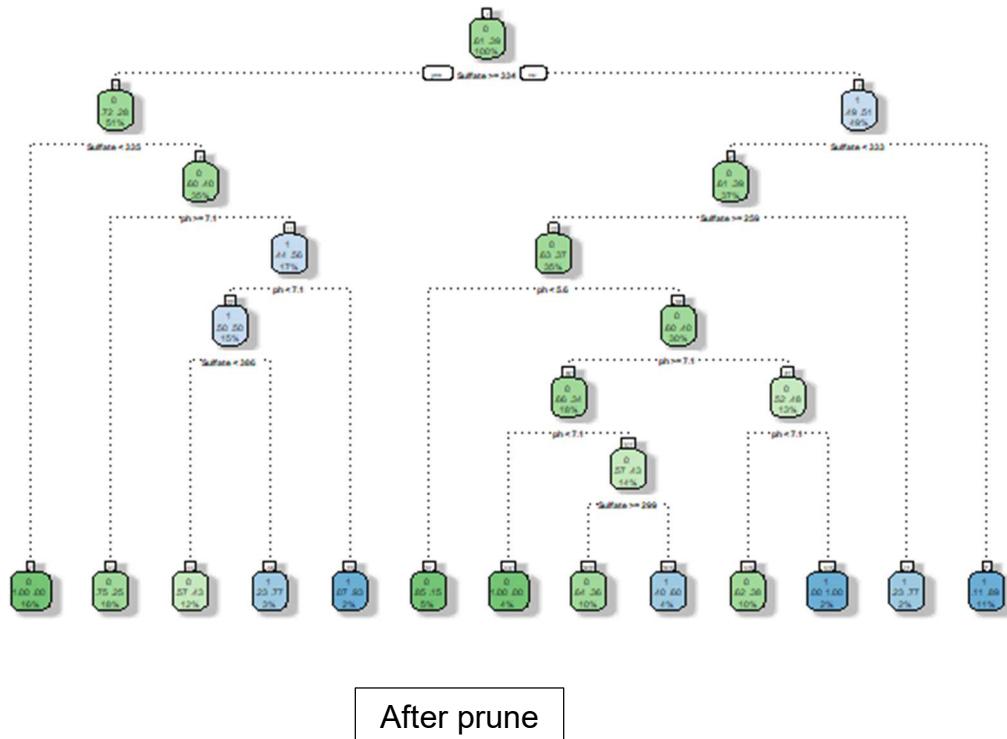
In particolare, sono stati valutati questi sottoinsiemi di attributi. Di seguito viene mostrata l'accuratezza prima e dopo la potatura:

Training su Variabili	Accuracy (Pre-Pruning)	Accuracy (After-Pruning)
Sulfate + pH	75,45 %	73,82 %
Sulfate + Solids +pH	75,05 %	72,81 %
Sulfate + Solids	69,24 %	69,55 %
Sulfate	69,55 %	69,55 %

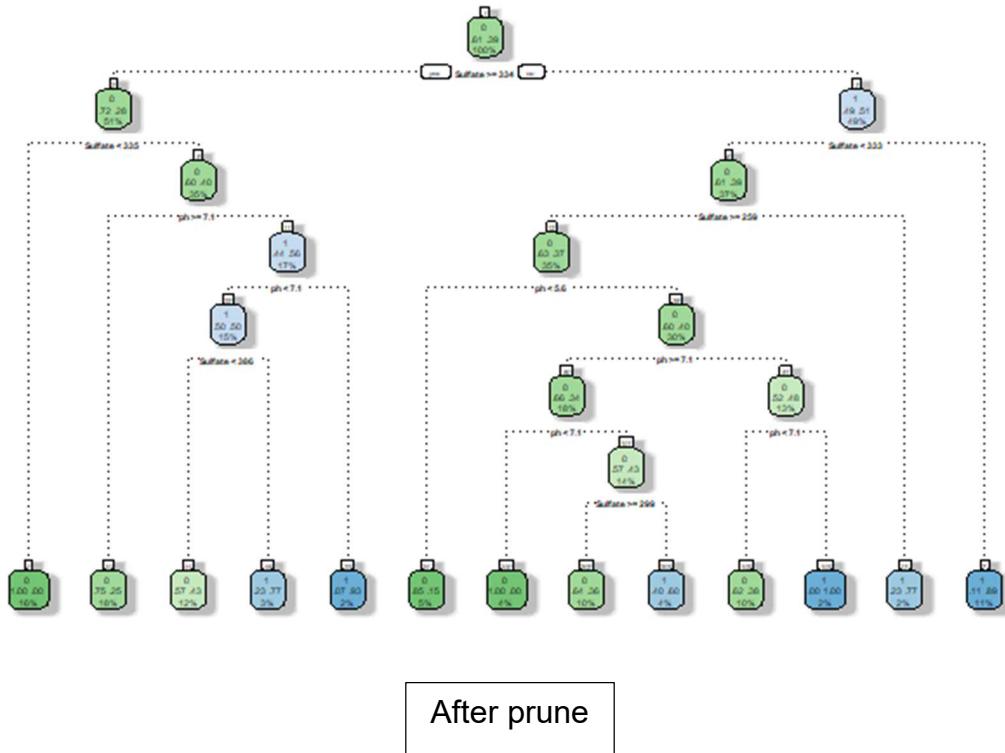
A fronte dei risultati ottenuti, si sceglie come sottoinsieme di attributi per il training del modello la coppia: {Sulfate, pH}; in quanto l'albero presenta una maggiore accuratezza.

Nel dettaglio:

1) Sulfate + pH:



2) Sulfate + Solids + pH:



After prune

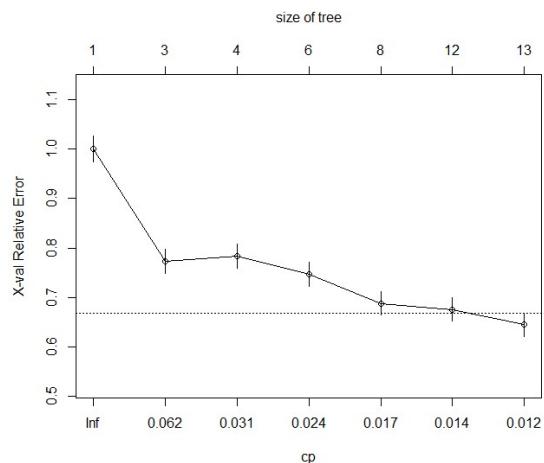
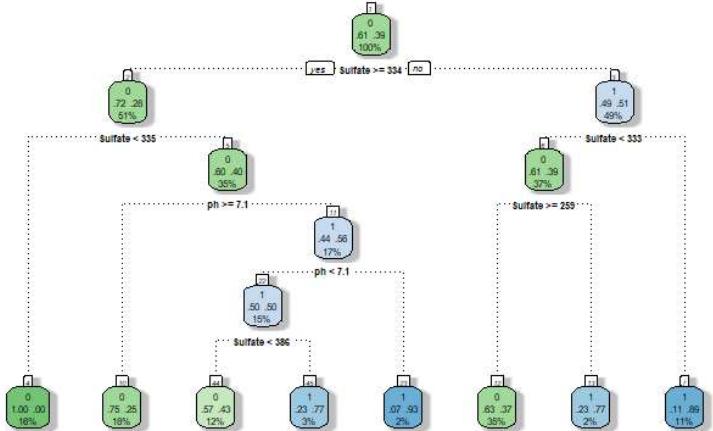
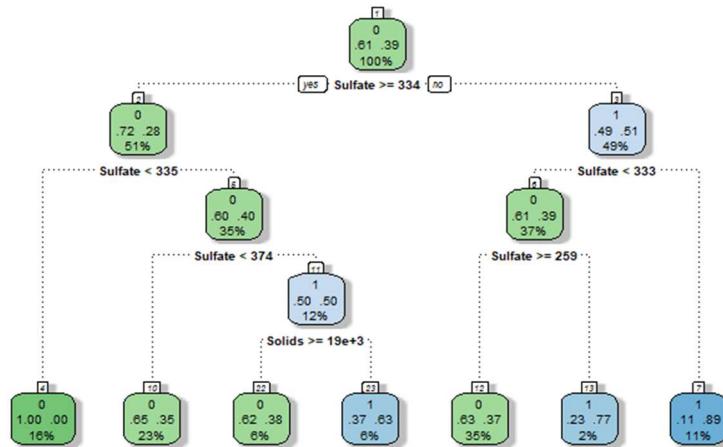


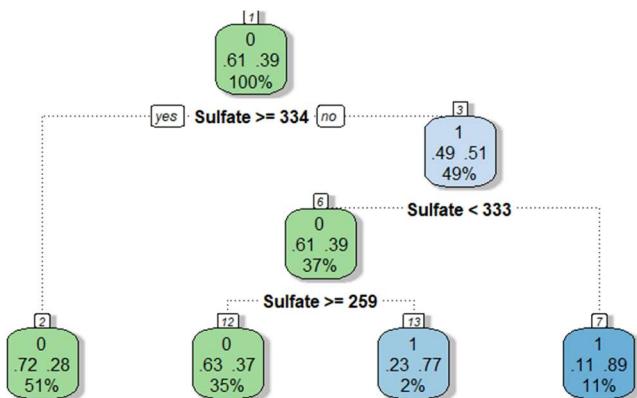
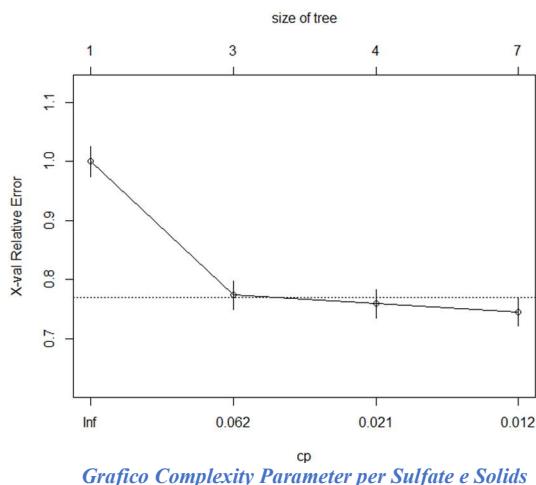
Grafico Complexity Parameter per Solids ,Sulfate e pH



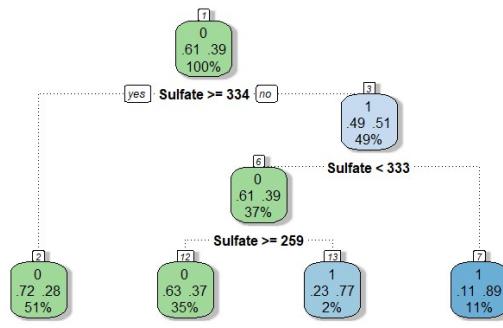
3) Sulfate + Solids:



After prune



4) Sulfate:



After prune

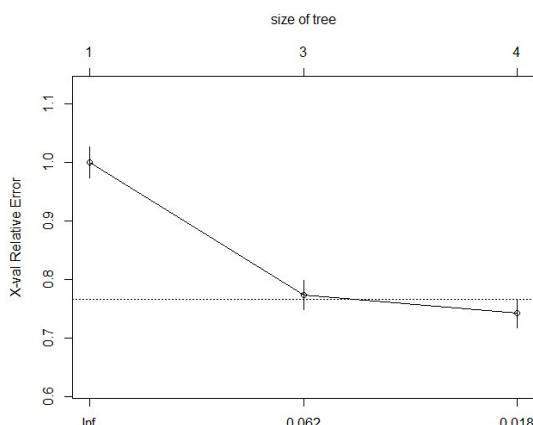
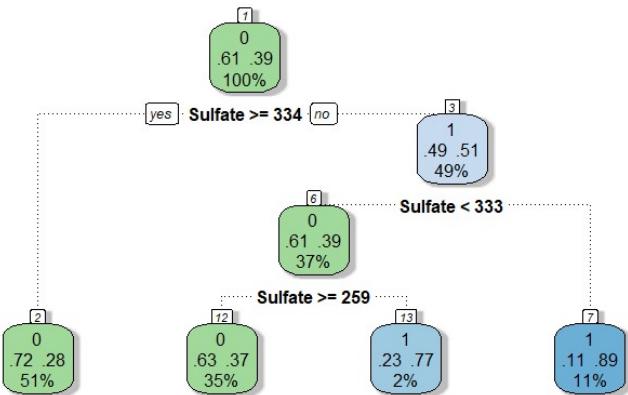


Grafico Complexity Parameter per Sulfate



7.3 Gradient Boosting

Come per Decision Tree si procede allo split del dataset 70%-30%.

Si procede poi a convertire le classi di potabilità in intero, in quanto l'algoritmo richiede delle classi intere per la rappresentazione della variabile target.

Essendo che il comando `as.integer` restituisce un intero nel range [1, n], mentre il comando `xgb.DMatrix` richiede valori in un range [0, n] sottraiamo 1 al valore convertito.

Viene poi rimossa la colonna potability dal trainset e dal testset.

```
split.data = function(data, p = 0.7, s = 1){  
  set.seed(s)  
  index = sample(1:dim(data)[1])  
  train = data[index[1:floor(dim(data)[1] * p)], ]  
  test = data[index[((ceiling(dim(data)[1] * p)) + 1):dim(data)[1]], ]  
  return(list(train=train, test=test))  
}  
  
allset = split.data(water_potability, p=0.7)  
trainset = allset$train  
testset = allset$test  
  
y_train <- as.integer(trainset$Potability) - 1  
y_test <- as.integer(testset$Potability) - 1  
X_train <- trainset %>% select(-Potability)  
X_test <- testset %>% select(-Potability)
```

A questo punto creiamo 2 matrici (train e test) che verranno utilizzate dall'algoritmo per l'apprendimento e la verifica.

```
xgb_train <- xgb.DMatrix(data = as.matrix(X_train), label = y_train)  
xgb_test <- xgb.DMatrix(data = as.matrix(X_test), label = y_test)
```

Passiamo ora alla fase di definizione dei parametri di default, che studieremo per poi effettuare il tuning migliore possibile cercando di aumentare l'efficienza dell'algoritmo.

```
#default parameters  
xgb_params <- list(  
  booster = "gbtree",  
  objective = "multi:softprob",  
  eta=0.3,  
  max_depth=6,  
  num_class = length(levels(water_potability$Potability)),  
)  
  
#printa il valore ideale per nround con parametri default  
xgbcv <- xgb.cv(  
  params = xgb_params,  
  data = xgb_train,
```

```

nrounds = 100,
nfold = 5,
showsd = T,
stratified = T,
print_every_n = 10,
early_stopping_rounds = 20,
maximize = F)

```

In particolare, i parametri indicano:

- **Booster**: tipologia di booster da utilizzare (tree based o funzione lineare)
- **Objective**: Specificare il learning task e il learning objective corrispondente (*in questo caso classificazione multiclass*)
- **Eta**: riduce i pesi delle funzionalità per rendere il processo di potenziamento più conservativo (*più è basso e meglio resiste all'overfitting*)
- **Max_depth**: altezza massima di un albero
- **Num_class**: numero di classificazioni possibili

Tramite la funzione `xgb.cv` otteniamo il numero ideale di iterazioni per cui l'errore ricavato con il train set è minore dell'errore ricavato nel test set.

In questo caso il numero ideale di iterazioni è 20.

Eseguiamo ora il train con i nuovi parametri e mostriamo la confusion matrix finale.

```

xgb_model <- xgb.train(
  params = xgb_params,
  data = xgb_train,
  nrounds = 20,
  verbose = 1,
)

xgb_preds <- predict(xgb_model, as.matrix(X_test), reshape = TRUE)

xgb_preds <- as.data.frame(xgb_preds)

colnames(xgb_preds) <- levels(water_potability$Potability)

xgb_preds$PredictedClass <- apply(xgb_preds, 1, function(y)
  colnames(xgb_preds)[which.max(y)])
xgb_preds$ActualClass <- levels(water_potability$Potability)[y_test + 1]

confusionMatrix(as.factor(xgb_preds$PredictedClass),
  as.factor(xgb_preds$ActualClass))

```

Confusion Matrix and Statistics

		Reference
Prediction	0	1
0	535	160
1	63	224

Accuracy : 0.7729
95% CI : (0.7454, 0.7988)
No Information Rate : 0.609
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.5006

Mcnemar's Test P-Value : 1.288e-10

Sensitivity : 0.8946
Specificity : 0.5833
Pos Pred Value : 0.7698
Neg Pred Value : 0.7805
Prevalence : 0.6090
Detection Rate : 0.5448
Detection Prevalence : 0.7077
Balanced Accuracy : 0.7390

'Positive' Class : 0

Dalla confusion matrix si ricava un'accuratezza del 77%, che risulta ottima data anche la tipologia di dati contenuti nel dataset scelto.

8 - Esperimenti

Si eseguono ora una serie di esperimenti sul dataset per avere un miglior riscontro sui risultati ottenuti con gli algoritmi d'apprendimento utilizzati.

In particolare, verrà eseguita una K-fold cross validation (*con K=10*) per meglio verificare l'accuratezza dei due modelli usati.

8.1) 10-Fold cross validation

Come primo passo si procede alla suddivisione del dataset in 10 “fold”.

```
folds <- createFolds(water_potability$Potability, k=10)
```

Successivamente viene applicata iterativamente per ogni fold.

In particolare, la funzione permette di utilizzare la fold presa in esame in quell'istante come Test set; mentre la restante parte del dataset come Train set.

Dopo la definizione dei due set, si passa al train del modello e all'elaborazione dei risultati, producendo in output 10 “Confusion Matrix”, una per ogni fold.

```
results <- lapply(folds, function(x){...})
```

Il parametro `function(x){...}` utilizzato ha una diversa implementazione a seconda dell'algoritmo di apprendimento da utilizzare.

Gradient Boosting

Con il Gradient Boosting l'implementazione sarà la seguente:

```
results <- lapply(folds, function(x) {  
  fold_train <- water_potability[-x, ]  
  fold_test <- water_potability[x, ]  
  y_fold_train <- as.integer(fold_train$Potability) - 1  
  complete_test_set <- fold_test  
  
  fold_train <- fold_train %>% select(-Potability)  
  fold_test <- fold_test %>% select(-Potability)  
  
  xgb_fold_train<- xgb.DMatrix(data = as.matrix(fold_train), label =  
y_fold_train)  
  
  credit_model <- xgb.train(  
    params = xgb_params,  
    data = xgb_fold_train,  
    nrounds = 20,  
    verbose = 1,
```

```

)
preds <- predict(credit_model, as.matrix(fold_test), reshape = TRUE)
preds <- as.data.frame(preds)

colnames(preds) <- levels(water_potability$Potability)

preds$PredictedClass <- apply(preds, 1, function(y)
colnames(preds)[which.max(y)])
preds$ActualClass <- complete_test_set$Potability
preds$PredictedClass <- as.factor(preds$PredictedClass)
preds$ActualClass <- as.factor(preds$ActualClass)

return(confusionMatrix(preds$PredictedClass, preds$ActualClass))
}

```

Una volta ottenute le confusion matrix si procede alla loro somma e al calcolo della matrice complessiva ottenendo:

```

Confusion Matrix and Statistics

Reference
Prediction    notPotable    potable
  notPotable       1777       463
    potable          221       815

Accuracy : 0.7912
95% CI  : (0.7769, 0.805)
No Information Rate : 0.6099
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.5457

McNemar's Test P-Value : < 2.2e-16

Sensitivity : 0.8894
Specificity  : 0.6377
Pos Pred Value : 0.7933
Neg Pred Value : 0.7867
Prevalence   : 0.6099
Detection Rate : 0.5424
Detection Prevalence : 0.6838
Balanced Accuracy : 0.7636

'Positive' Class : notPotable

```

È da subito evidente l'aumento dell'accuratezza media ricavato dall'esperimento, cioè circa del 2% (79%).

Decision Tree

Con il Decision Tree, l'implementazione sarà la seguente:

```
results <- lapply(folds, function(x) {  
  credit_train <- water_potability[-x, ]  
  credit_test <- water_potability[x, ]  
  
  credit_model <- rpart(Potability ~ Sulfate + ph, data=credit_train,  
    method="class")  
  
  preds <- predict(credit_model, credit_test, reshape = TRUE)  
  preds <- as.data.frame(preds)  
  colnames(preds) <- levels(water_potability$Potability)  
  
  preds$PredictedClass <- apply(preds, 1, function(y) colnames(preds)[which.max(y)])  
  preds$ActualClass <- credit_test$Potability  
  preds$PredictedClass <- as.factor(preds$PredictedClass)  
  preds$ActualClass <- as.factor(preds$ActualClass)  
  
  return(confusionMatrix(preds$PredictedClass, preds$ActualClass))  
})
```

I risultati dell'esperimento per il Decision Tree, sono:

Confusion Matrix and Statistics

		Reference	
		notPotable	potable
Prediction	notPotable	1795	616
	potable	203	662

Successivamente viene calcolata l'accuratezza complessiva, che risulta essere del 75%. Risultato che non si discosta troppo da quanto ottenuto alla sezione 7.2.

Accuracy : 0.75
95% CI : (0.7348, 0.7648)

No Information Rate : 0.6099
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.4421

Mcnemar's Test P-Value : < 2.2e-16

Sensitivity : 0.8984
Specificity : 0.5180
Pos Pred Value : 0.7445
Neg Pred Value : 0.7653
Prevalence : 0.6099
Detection Rate : 0.5479
Detection Prevalence : 0.7360
Balanced Accuracy : 0.7082

'Positive' Class : notPotable

8.2) Confusion Matrix Complessiva, Precision, Recall, F-Measure

La Confusion Matrix Complessiva di ciascun algoritmo di apprendimento è stata calcolata andando a sommare tra loro tutte e 10 le Confusion Matrix ricavate dal 10 Fold Cross Validation.

```
rfConfusionMatrixFinal <- results$Fold01$table +  
  results$Fold02$table +  
  results$Fold03$table +  
  results$Fold04$table +  
  results$Fold05$table +  
  results$Fold06$table +  
  results$Fold07$table +  
  results$Fold08$table +  
  results$Fold09$table +  
  results$Fold10$table  
  
confusionMatrix(rfConfusionMatrixFinal)
```

Successivamente sono state calcolate le metriche di performance per ciascun modello, in particolare:

- **PRECISION:** Indica il rapporto fra le previsioni corrette (*true positive - TP*) e le previsioni positive totali effettuate (*true positive + false positive*).

$$precision = \frac{TP}{TP + FP}$$

- **RECALL:** Indica il rapporto fra i valori attuali true positive ed il totale dei valori attuali positivi per ogni classe.

$$recall = \frac{TP}{TP + FN}$$

- **F-MEASURE:** è invece una misura che bilancia il peso delle misure precision e recall in quanto rappresenta la media armonica fra le due. È quindi un buon modo per riassumere la valutazione di precision e recall in un unico numero.

$$fmeasure = 2 \frac{precision * recall}{precision + recall}$$

Di seguito viene mostrato il codice per calcolare le misure di performance sopra citate:

```
#accuracy
accuracy <- (rfConfusionMatrixFinal[1,1] + rfConfusionMatrixFinal[2,2])/
  (rfConfusionMatrixFinal[1,1] +
   rfConfusionMatrixFinal[2,2] +
   rfConfusionMatrixFinal[1,2] +
   rfConfusionMatrixFinal[2,1])

#precision
precision <- rfConfusionMatrixFinal[1,1]/(rfConfusionMatrixFinal[1,1] +
                                             rfConfusionMatrixFinal[1,2])

#recall
recall <- rfConfusionMatrixFinal[1,1]/(rfConfusionMatrixFinal[1,1] +
                                         rfConfusionMatrixFinal[2,1])

#f-measure
f_measure <- ((2*precision*recall)/(precision + recall))
print(f_measure)
```

Ottenendo i seguenti risultati:

	Decision Tree	Gradient Boosting
Accuracy	75,67 %	79,12 %
Precision	74,04 %	79,33 %
Recall	89,34 %	88,93 %
F-Measure	81,42 %	83,86 %

Tranne per la Recall, il Gradient Boosting risulta avere misure di performance migliori rispetto al Decision Tree.

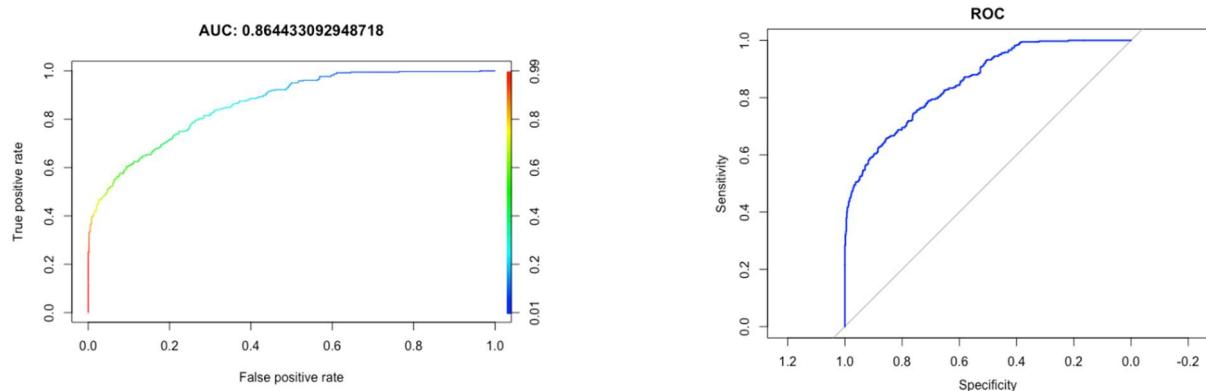
8.3) ROC & AUC

Come ultima misura di performance, per ciascun classificatore viene confrontato il tasso di veri positivi e il tasso di falsi positivi nella curva **ROC** (*Receiver Operating Characteristic*) e il valore **AUC** (*Area Under the Curve*) corrispondente.

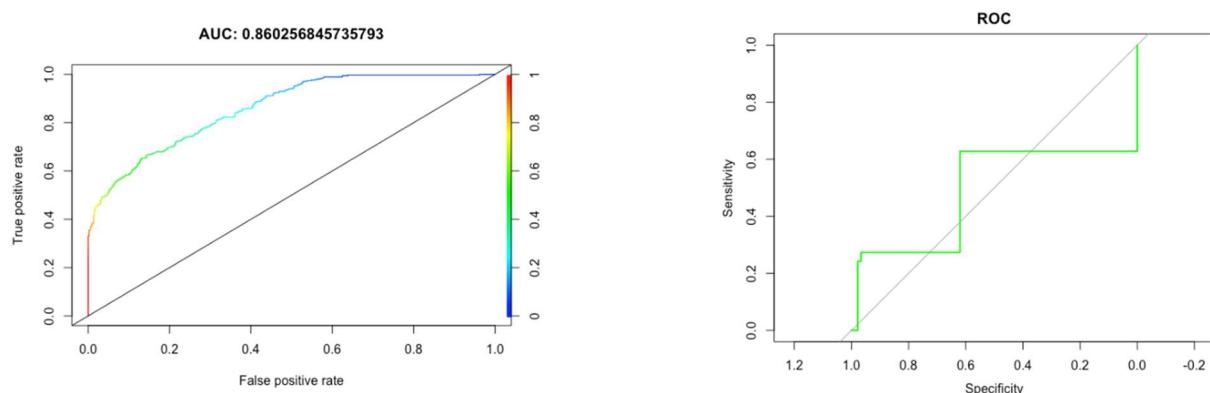
Più questa curva si avvicina all'angolo superiore sinistro, migliori sono le prestazioni del classificatore, in quanto si comporterebbe come un “Classificatore Perfetto”. Le curve che si trovano vicino alla diagonale del Classificatore Random risultano dei classificatori che tendono a fare delle stime al limite della casualità.

Nel caso in esame, è possibile notare come il classificatore del Gradient Boosting sia leggermente più accurato di quello del Decision Tree, avendo un AUC lievemente maggiore.

Gradient Boosting

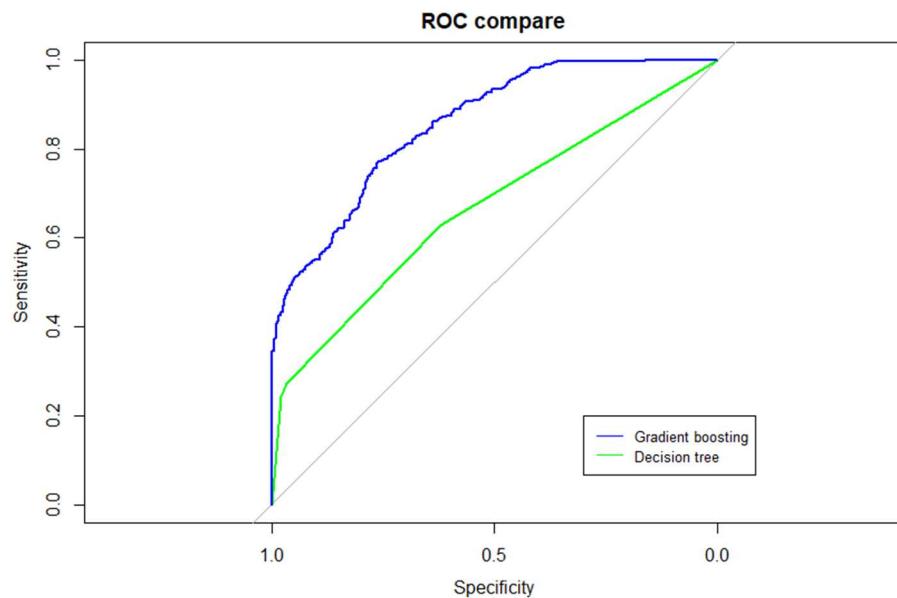


Decision Tree



Vediamo ora un confronto diretto tra i 2 modelli in esame:

Comparazione ROC



Si nota che la curva generata dal Gradient Boosting risulta molto più alta di quella generata da Decision Tree e di conseguenza avrà un AUC maggiore, indicatore di una maggior accuratezza e di minor tasso di errore.

9 – Conclusioni

In conclusione, tra i due classificatori utilizzati sul dataset “Water Potability”, il Gradient Boosting ha riscontrato risultati migliori rispetto al Decision Tree. Questo sia per quanto riguarda l’accuracy, sia per le restanti misure di performance.

Il tutto è dovuto al fatto che l’algoritmo stesso si basa su l’elaborazione seriale di più Decision Tree, in cui l’apprendimento di ognuno di essi è condizionato dal risultato del precedente, migliorandosi dopo ogni iterazione.