

Anno Accademico 21/22

**Progetto di**

**Machine Learning**

Water potability

|  |  |
| --- | --- |
| **Cognome e Nome** | **Matricola** |
| Colombo Andrea | 844557 |
| Sala Mattia | 889809 |

# Indice

1. **Dominio di riferimento**
2. **Obiettivo**
3. **Scelte di design e assunzioni**
4. **Pacchetti installati**
5. **Descrizione del Training Set**
6. **Principal Component Analisys (PCA)**
7. **Descrizione dei modelli di Machine Learning**
   1. Modello Base
   2. Decision Tree
   3. Gradient Boosting
8. **Esperimenti**
   1. 10 Fold Cross Validation e Stima delle misure di performance
   2. Matrice di Confusione Complessiva ,Precision , Recall, F-Measure
   3. ROC e AUC
9. **Conclusioni Finali**

# **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**

## Pulizia e formattazione dei dati

Inizialmente è stata eseguita una lettura dei dati del dataset.

Immagine che contiene tavolo

Descrizione generata automaticamente

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) |

Immagine che contiene tavolo

Descrizione generata automaticamente

Prima di iniziare l’analisi, viene effettuato un controllo e pulizia su eventuali dati mancanti (“NA”).

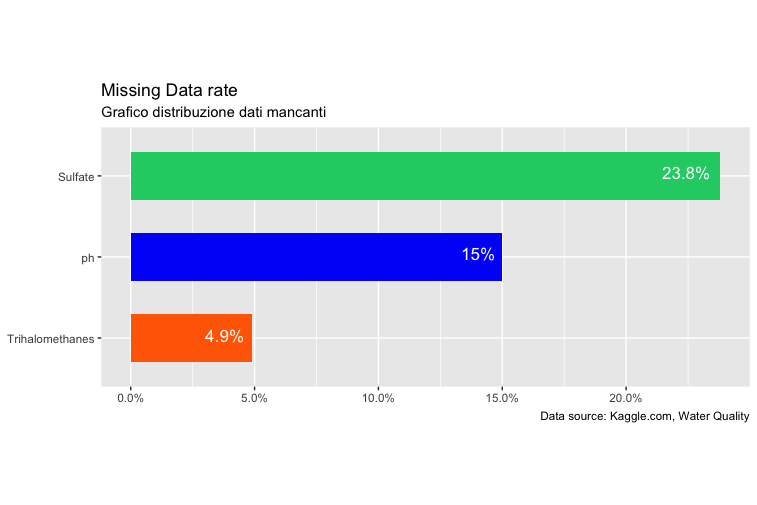
Innanzitutto, viene visualizzata la quantità dei dati mancanti su tutti gli attributi del dataset.

|  |
| --- |
| > water\_potability %>% summarise\_all(~ sum(is.na(.))) |



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:

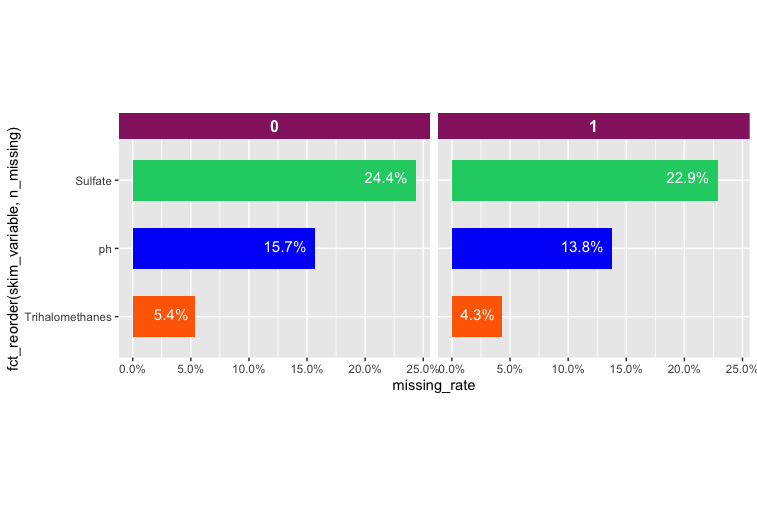
water\_potability %>%  skim() %>%  
    filter(n\_missing != 0) %>%  
    as\_tibble() %>%  
    select(skim\_variable, n\_missing, complete\_rate) %>%  
    mutate(missing\_rate = round(abs(complete\_rate - 1) \* 100, 1)) %>%  
    ggplot(aes(  
        x = fct\_reorder(skim\_variable, n\_missing),  
        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.

È quindi possibile attuare tre diverse strategie, 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.

Di conseguenza si decide di usare la terza strategia; ovvero la sostituzione dei valori nulli con i valor 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.*

Immagine che contiene testo, ricevuta

Descrizione generata automaticamente

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() |

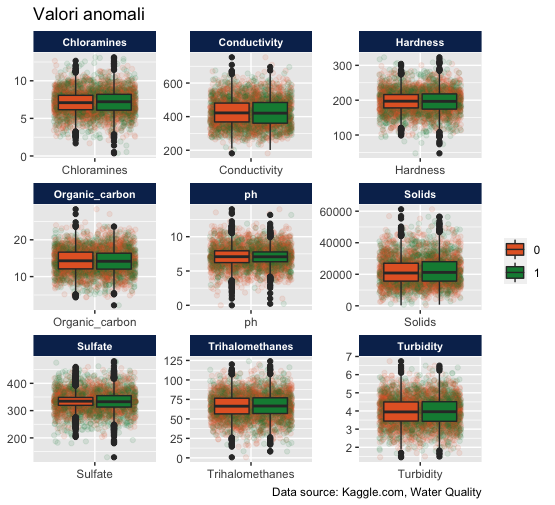
Immagine che contiene tavolo

Descrizione generata automaticamente

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     ) |

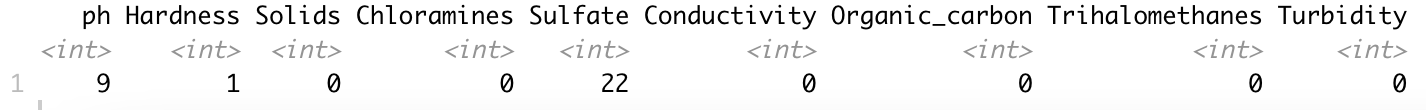


Si nota che potrebbero esserci alcuni outliers, è necessario quindi calcolarne in modo più preciso il numero sfruttandone la definizione stessa:

* 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.

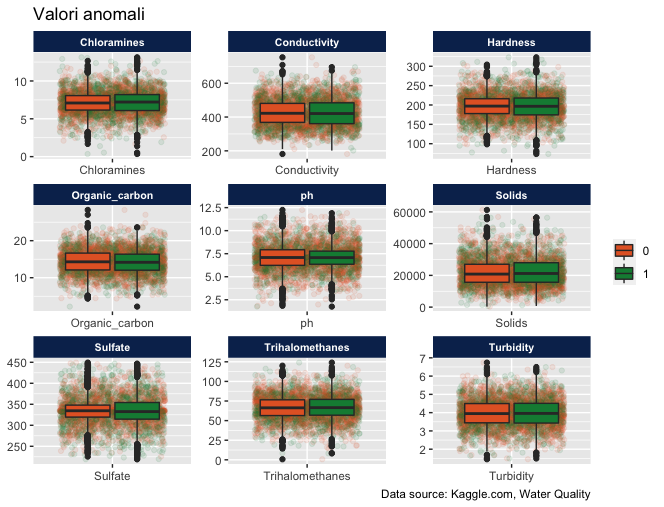
|  |
| --- |
| outliers <- water\_potability %>%    select(-Potability) outliers[] <- lapply(outliers, **function**(x){   qq <- quantile(x, c(0.25, 0.75), na.rm = TRUE)   is.na(x) <-  x < (qq[1] - (3\*(qq[2]-qq[1]))) | x > (qq[2] + (3\*(qq[2]-qq[1])))   x }) outliers %>% summarise\_all(~ sum(is.na(.))) |



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

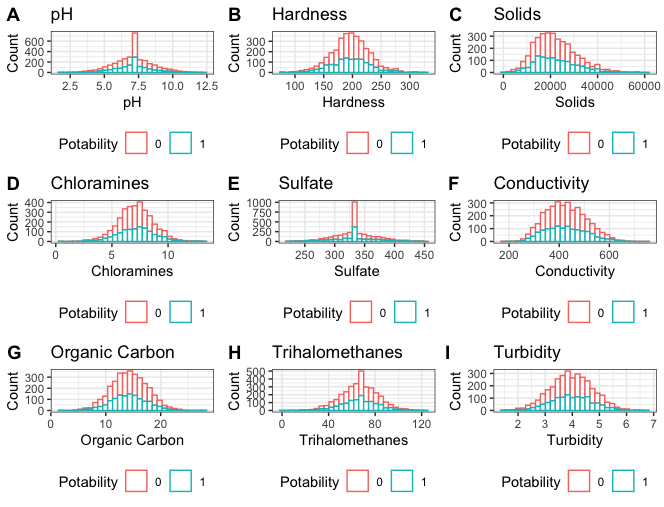
|  |
| --- |
| outliers <- outliers %>%    mutate(across(where(is.numeric),                  ~if\_else(is.na(.),                           mean(., na.rm = T),                             as.numeric(.)))) %>%    ungroup()  outliers$Potability <- water\_potability$Potability water\_potability <- outliers |

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 |

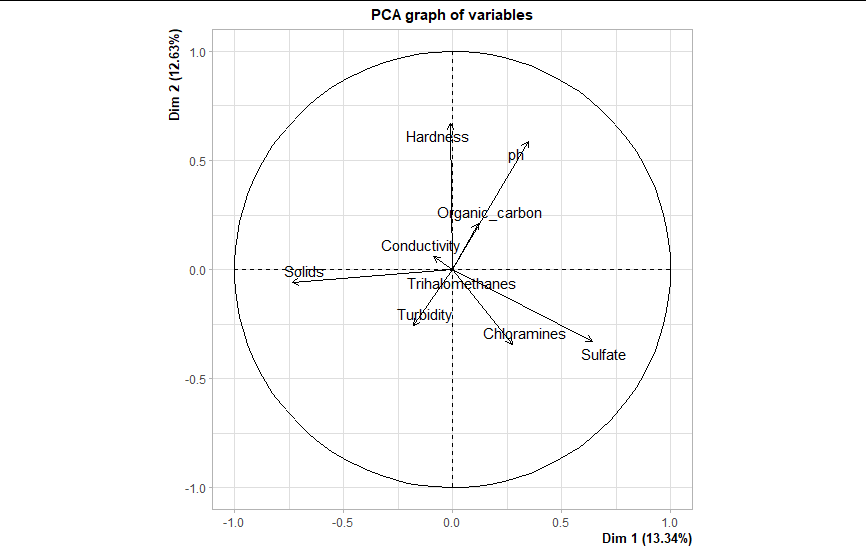


# **6 - Principal Component Analisys (PCA)**

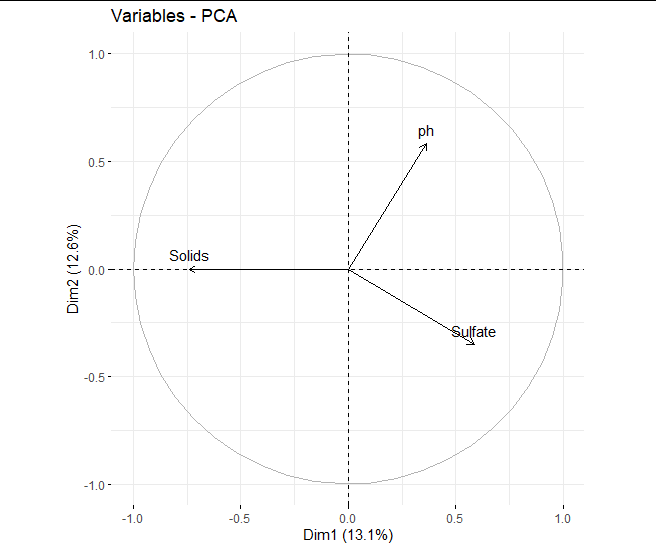
Con la PCA si vuole analizzare quali siano le componenti più incisive del dataset, in modo da utilizzarle in fase di training del modello.

Di seguito viene presentato il codice per computare la PCA. Si prendono le feature “attive” (*tutti gli attributi del dataset ad eccezione della variabile target*). Dopodiché si visualizzano i risultati sul grafico delle componenti principali.

|  |
| --- |
| res.pca <- PCA(as.data.frame(water\_potability[1:9]), graph=TRUE) |



Tra tutte le componenti individuate, si considerano solo quelle più importanti, cioè quelle con più incidenza sulle prime due dimensioni.



Si valutano gli autovalori col comando:

|  |
| --- |
| eig.val <- get\_eigenvalue(res.pca) |

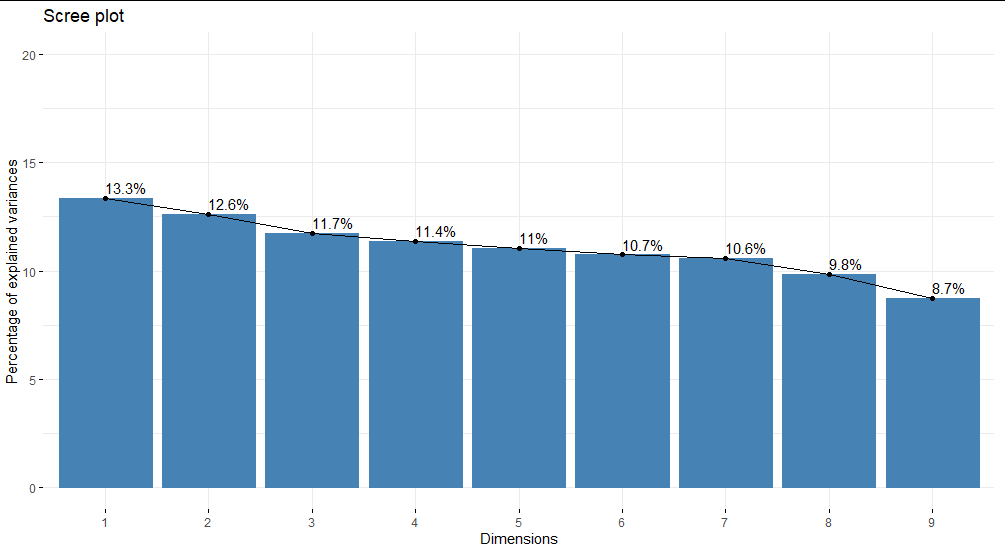
Immagine che contiene testo, monitor, schermo, tabellonesegnapunti

Descrizione generata automaticamente

Il 70% della varianza cumulativa è spiegato dalle prime 6 dimensioni. In particolare, le prime 2 dimensioni spiegano soltanto il 25% cumulativo della varianza (*rispettivamente 13% e 12%*).

Viene anche visualizzato lo Scree Plot, ottenibile con il seguente codice:

|  |
| --- |
| fviz\_eig(pca.res, addlabels = TRUE, ylim = c(0, 20)) *#scree plot* |



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) |

Immagine che contiene testo, batteria, targa

Descrizione generata automaticamente

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%

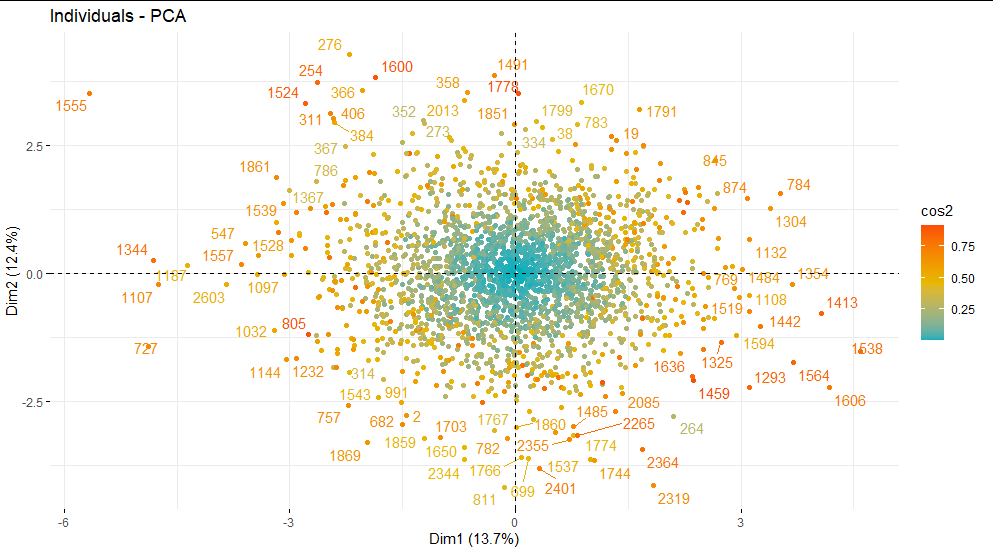
- **Hardness** 39%

- **Chloramines** 10 %

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

Successivamente si visualizzano gli “individui”:

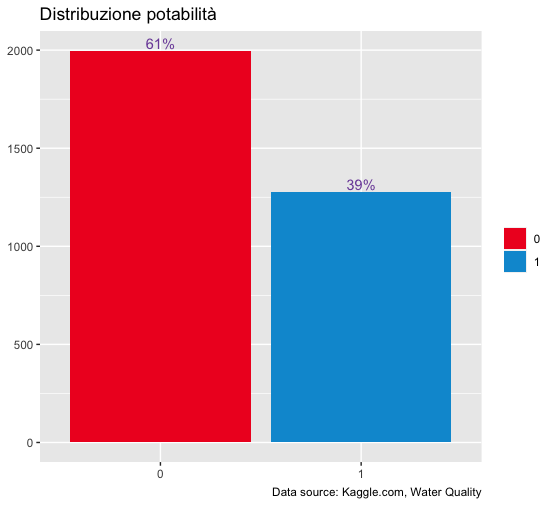
|  |
| --- |
| fviz\_pca\_ind(pca.res, col.ind = "cos2",                      gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),                      repel = TRUE *# Avoid text overlapping (slow if many points)* ) |



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

**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”.

# **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 ≃ **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:

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.

|  |  |  |
| --- | --- | --- |
| **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 % |

Nel dettaglio:

1. **Sulfate + pH:**

Immagine che contiene testo, interni

Descrizione generata automaticamente

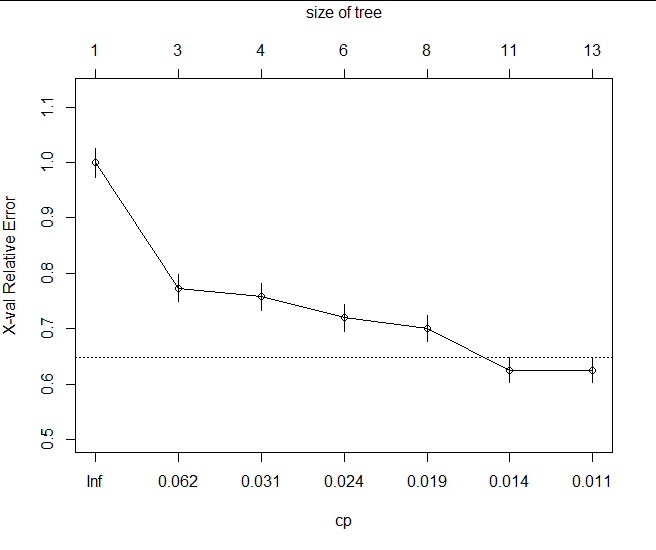


Grafico Complexity Parameter per Sulfate e pH

After prune

**2) Sulfate + Solids + pH:**

****

After prune

Immagine che contiene testo, interni

Descrizione generata automaticamente

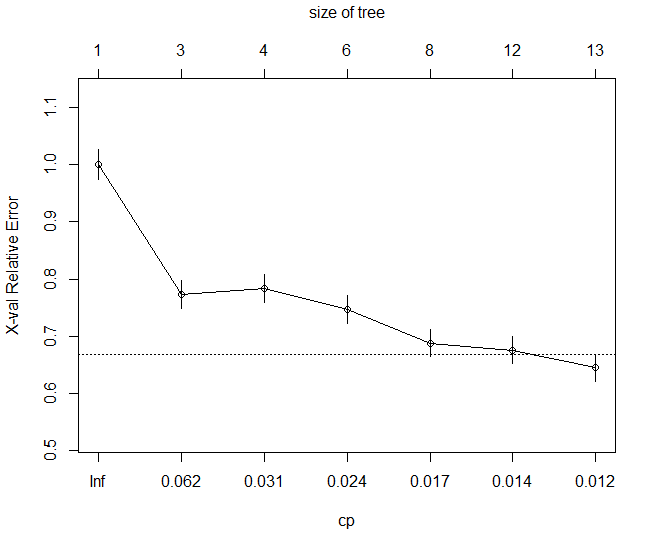
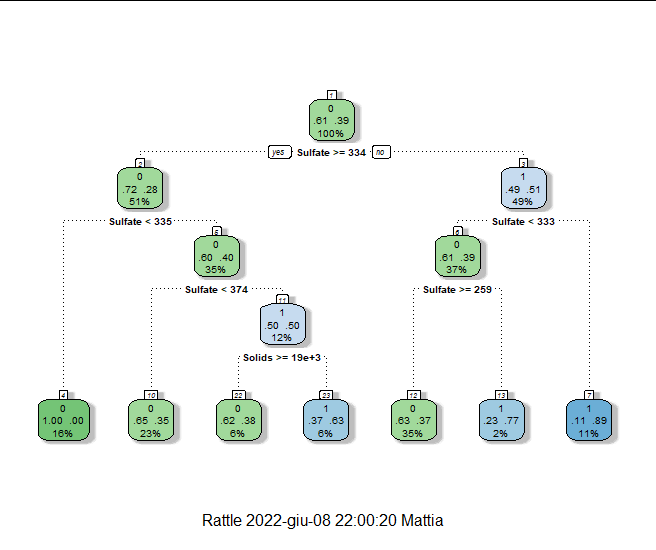


Grafico Complexity Parameter per Solids ,Sulfate e pH

**3) Sulfate + Solids:**



After prune

Immagine che contiene testo, orologio

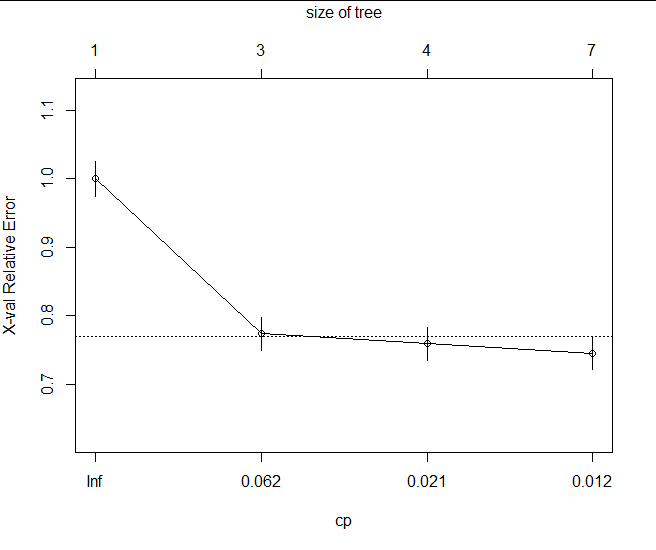
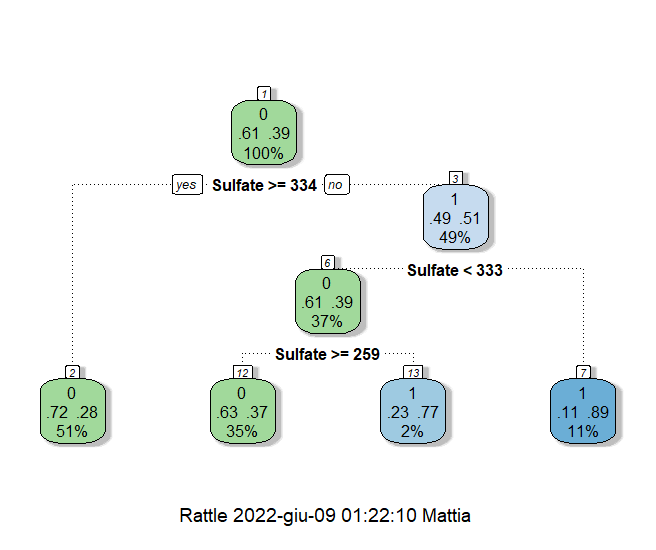
Descrizione generata automaticamente

Grafico Complexity Parameter per Sulfate e Solids

**4) Sulfate**:



After prune

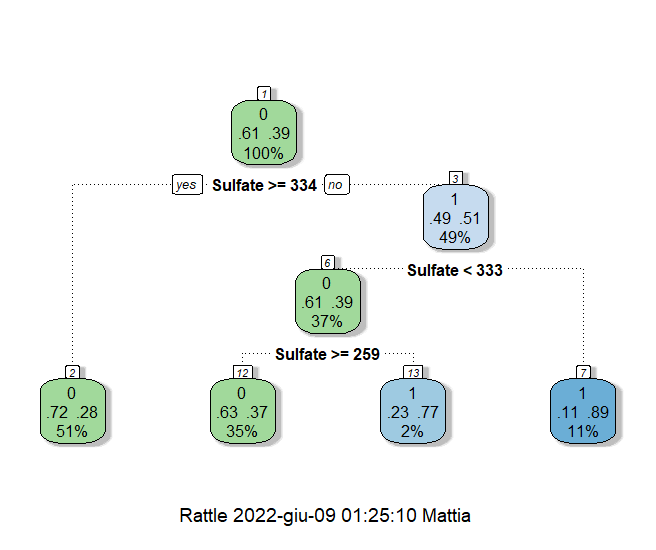
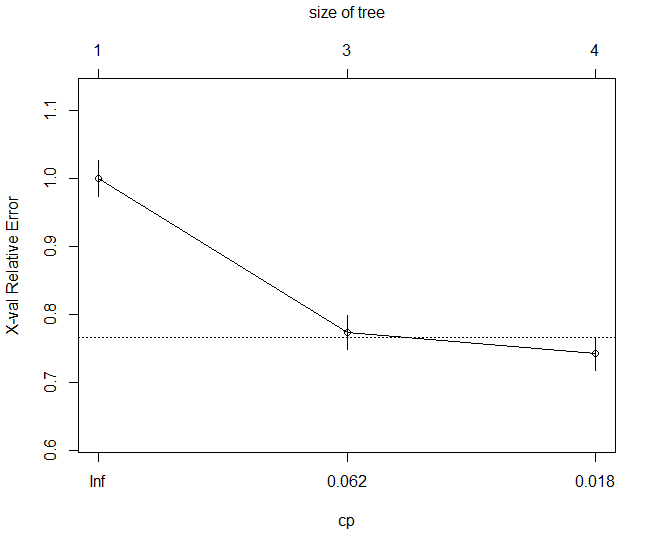


Grafico Complexity Parameter per Sulfate

## **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 richede 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:

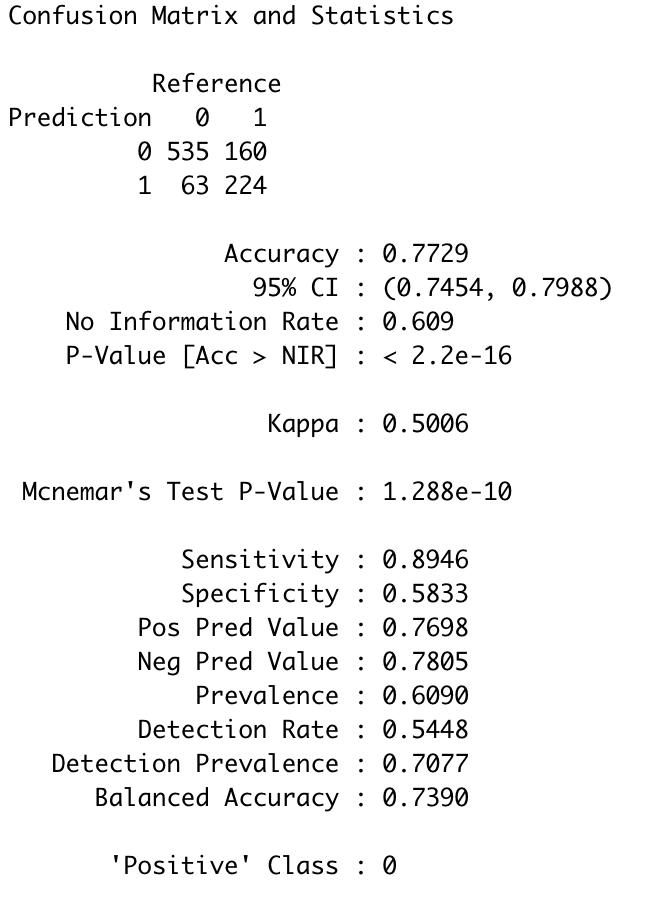
* **Booster**: tipologia di booster da utilizzare (tree based o funzione lineare)
* **Objective**: Specificare il learning task e il learning objective corrispondente (*in questo caso classificazione multiclass*)
* **Eta**: riduce i pesi delle funzionalità per rendere il processo di potenziamento più 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)) |



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){...}) |

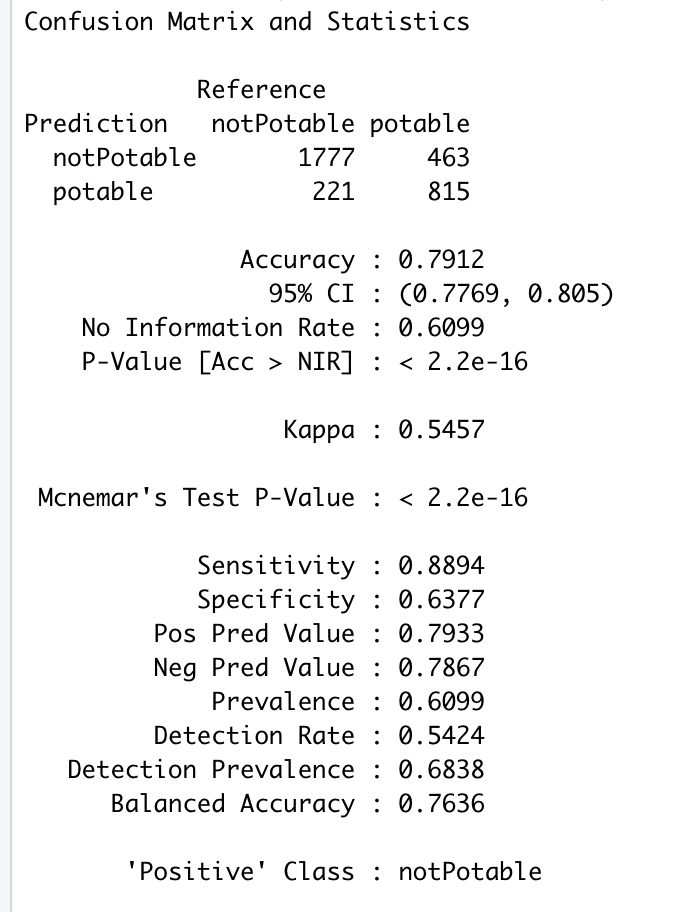
Il parametro **function**(x){...} utilizzano ha una diversa implementazione a seconda dell’algoritmo di apprendimento da utilizzare.

#### 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:



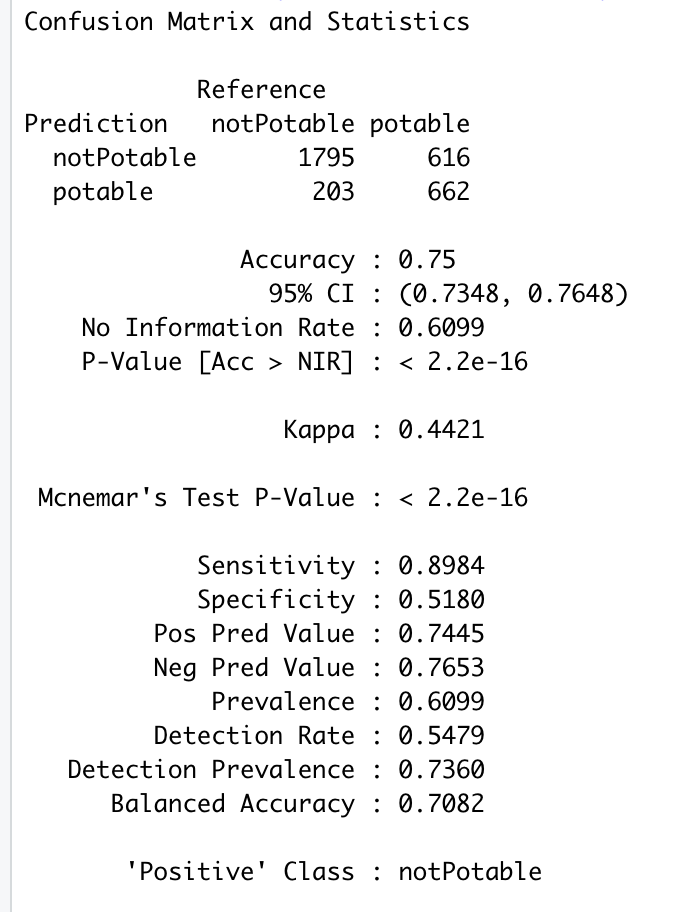
È da subito evidente l’aumento dell’accuratezza media ricavato dall’esperimento, cioè circa del 2% (79%).

#### 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:



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

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

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.

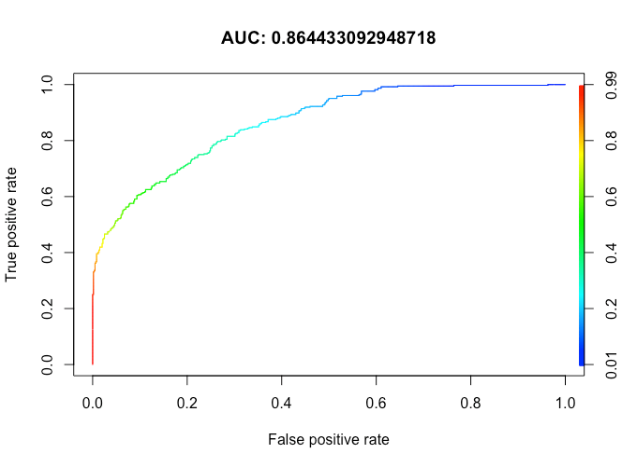
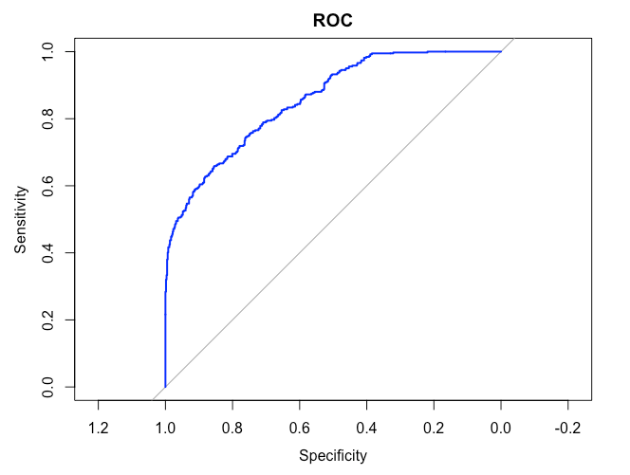
## 8.3) ROC & AUC

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.

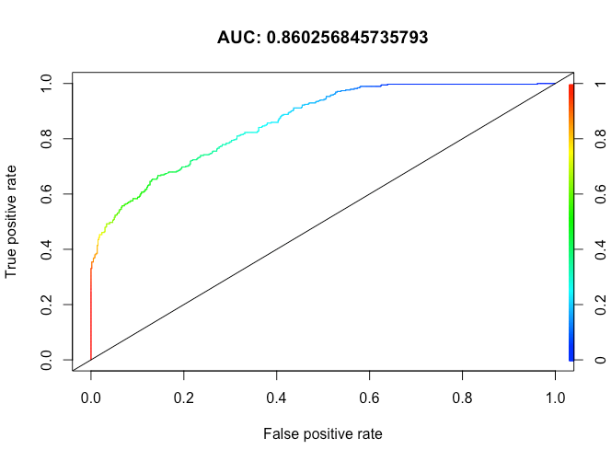
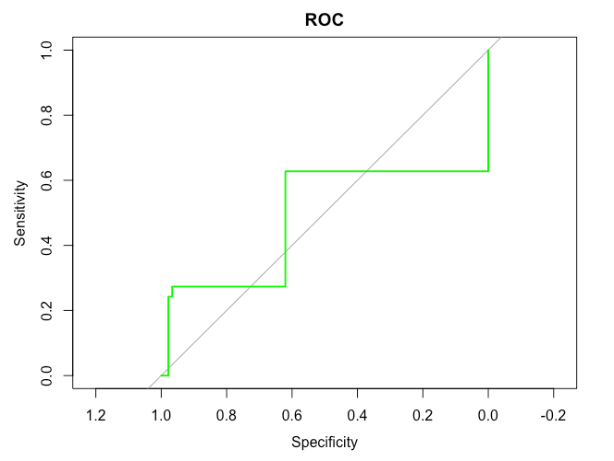
Più questa curva si avvicina all'angolo superiore sinistro, migliori sono le prestazioni del classificatore, in quanto si comporterebbe come un “Classificatore Perfetto”. Le curve che si trovano vicino alla diagonale del Classificatore Random risultano dei classificatori che tendono a fare delle stime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

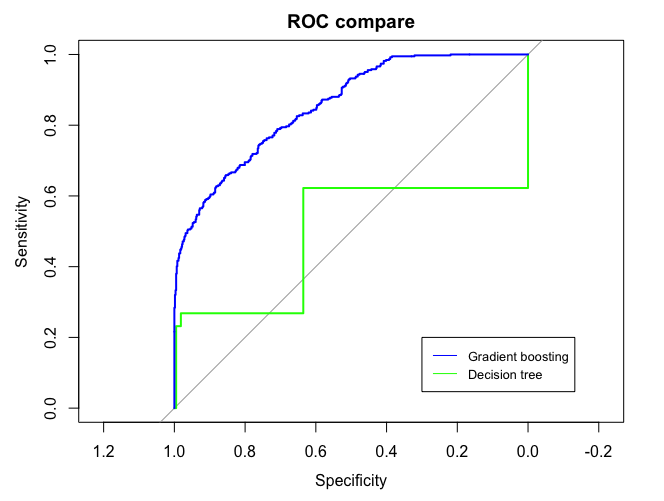
 

##### Decision Tree

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.