Progetto di Statistica Numerica 2021-2022 Informatica per il management, Università di Bologna

Analisi sulle Top Hits di Spotify dall'anno 2000 al 2019

Membri del gruppo:
Laura Specchiulli, matricola n°: 0000988074,

Samir Jabbar, matricola n°: 0000978246

1) INTRODUZIONE E SELEZIONE DEL DATASET

Il progetto di Machine Learning relativo al corso di Statistica numerica, tenuto dalla docente Elena Loli Piccolomini e dal tutor Andrea Sebastiani, richiede come primo passo la selezione di un dataset su cui basare l'analisi dei dati.

Come gruppo abbiamo deciso di selezionare un dataset che evidenzia tutte le caratteristiche che hanno portato varie canzoni mondiali a posizionarsi, tra l'anno 2000 e 2019, nelle migliori posizioni delle classifiche di Spotify.

Il dataset originalmente utilizzato era composto da 2 mila righe e 18 colonne.

Ogni riga rappresenta una diversa canzone, mentre le colonne descrivono i diversi parametri che le tracce posseggono, che possono indirizzare le preferenze dell'utente.

Per capire meglio, elenchiamo brevemente ogni colonna del dataset con le relative descrizioni:

- ARTIST: il nome dell'artista.
- SONG: il nome della traccia.
- DURATION_MS: la durata della canzone in millisecondi.
- EXPLICIT: le parole o il contenuto di una canzone o del relativo video musicale che potrebbero essere considerate offensive o inadatte a un pubblico di minori.
- YEAR: anno di pubblicazione della canzone.
- POPULARITY: più alto è il valore, maggiore è la popolarità della canzone.
- DANCEABILITY: descrive quanto la traccia è adatta ad essere ballata, sulla base di una combinazione di elementi musicali, compreso il tempo, il ritmo e la potenza del beat. Un valore di 0.0 indica che la canzone trattata è poco ballabile, mentre una traccia che raggiunge un valore di 1.0 sarà altamente ballabile.

- ENERGY: è una misura che va da 0.0 ad 1.0, rappresenta l'intensità della canzone in valore decimale.
- KEY: è la chiave della traccia, la quale viene determinata usando la notazione standard Pitch Class (per esempio: 0=C=Do, 1=C#=Do Diesis, 2=D=Re). Se nell'analisi non è stata rilevata nessuna chiave, il valore restituito è -1.
- LOUDNESS: è il volume totale della traccia in decibels (dB) che determina la qualità di un suono. I relativi valori sono calcolati in media su tutta la durata della traccia e sono utili per confrontare più canzoni tra loro.
  I valori del volume hanno tipicamente un range che va tra i -60 e 0 dB.
- MODE: indica la tonalità (maggiore o minore) di una traccia, con un tipo binario di 0 e 1.
- SPEECHINESS: rileva la presenza di parole pronunciate in una canzone. I valori superiori a 0,66 descrivono tracce che probabilmente sono composte interamente da parole pronunciate. I valori compresi tra 0,33 e 0,66 descrivono tracce che possono contenere sia musica che parlato, inclusi casi come la musica rap. I valori inferiori a 0,33 molto probabilmente rappresentano tracce non vocali.
- ACOUSTICNESS: l'acustica è una misura che varia da 0.0 a un massimo di 1.0
- INSTRUMENTALNESS: indica se una traccia contiene o meno voci. Le tracce rap o parlate sono considerate "vocali". Più il valore della strumentalità è vicino a 1,0, maggiore è la probabilità che la traccia non contenga voci. I valori superiori a 0,5 generalmente rappresentano tracce strumentali.
- LIVENESS: rileva la presenza di un pubblico nella registrazione. Un valore superiore a 0,8 fornisce una forte probabilità che la traccia sia stata eseguita dal vivo.
- VALENCE: è una misura da 0,0 a 1,0 che descrive la positività musicale veicolata da un brano. Più alta è la

valenza e più positivo risulterà il brano (es. Happy, allegro, euforico) e viceversa.

- TEMPO: nella terminologia musicale, il tempo è il ritmo di un brano stimato in battiti al minuto (BPM).
- GENRE: genere della canzone.

2) CARICAMENTO DEL DATASET IN R

Il primo passo, dopo aver scaricato il dataset da Kaggle, è il caricamento del medesimo su R.
Per poter caricare il file su R, si deve settare la working directory tramite il comando “setwd”(path della nostra working directory dove vi è anche il dataset).

Una volta fatto ciò, si può procedere al caricamento del file con il comando “read” sulla variabile df.

Successivamente abbiamo utilizzato le funzioni “head” e “dim” con lo scopo di visualizzare una prima panoramica e la dimensione del nostro dataset selezionato.

È buona norma copiare il dataset in una variabile di backup che in questo caso chiameremo “df_backup”. Questa operazione ci consente di avere sempre conservata una copia del dataset originale.

Prima di questo passo, però, abbiamo eliminato la colonna “key” perché considerata inutile per il proseguimento del lavoro.

La porzione di codice relativa a questa parte sarà quindi:

```
setwd("~/Desktop/R statistica")
# carico sulla variabile df il dataset originale
df <- read.csv("songs_normalize.csv")

# vediamo una prima panoramica del dataset con queste funzioni
# visualizziamo una prima panoramica del dataset
head(df)

# visualizziamo la dimensione del nostro dataset
dim(df)

#cancello la variabile key perchè non mi servono troppo
df$key=NULL

# creiamo una variabile nel quale inserire una copia del dataset, già ripulito da
# eventuali valori che non ci servono o che sono nulli
df_backup <- df;
```

3) PRE PROCESSING

I NaN sono dei dati presenti nel dataset che non hanno un valore, di conseguenza non possono essere presi in considerazione per le analisi. Tramite il comando “na.omit(df)” li abbiamo rimossi dal dataset. Ma già osservando diverse documentazioni su Kaggle relative a questo dataset, confermiamo la non presenza di valori nulli.

Con il comando “summary(df)”, funzione generica che contiene diversi metodi, ci siamo aiutati ad analizzare il dataset completo, constatando diversi valori di ciascuna colonna quali: media, mediana, 1° e 3° e valore minimo e massimo che le variabili assumono.

In seguito abbiamo verificato la tipologia di ciascuna variabile, rilevando che nelle colonne “artist", “song”, “explicit” e “genre" vi erano delle variabili categoriche, che quindi abbiamo trasformato in factor. Inoltre, nella colonna “explicit” abbiamo sostituito le parole intere “TRUE” e ”FALSE” con “T” e “F” per avere una lettura più chiara.

Successivamente abbiamo trasformato la durata delle canzoni da millisecondi a minuti e secondi, arrotondando a 2 cifre decimali, al fine di avere una leggibilità migliore del dataset.

```
df$duration_ms <- df$duration_ms/1000/60
df$duration <- df$duration_ms
df$duration <- round(df$duration, digits=2)
df$duration_ms <- NULL
```

Come detto precedentemente il dataset selezionato non presenta valori nulli, ma vi sono comunque valori in più, o meglio inutili. Perciò in base all’obbiettivo di ricerca che vogliamo perseguire è opportuno ripulire il nostro dataset da tali valori.

Innanzitutto il dataset vuole indicare che le canzoni in esso inserite sono state prodotte nell’arco di tempo 2000-2019, ma

se si osserva bene vi sono tracce pubblicate anche in anni precedenti o successivi, come il 1998-99 e 2020.
Per far ciò assegnamo quindi al nostro dataframe un sottoinsieme subset al fine di visualizzare nella colonna “year” solo valori tra l’anno 2000 e 2019 e non altri superflui.

```
df<-subset(df, subset=(df$year>1999&df$year<2020))
```

Oltre a quanto già fatto, abbiamo ritenuto opportuno svolgere una seconda pulizia alla feature “popularity”, perché essendo un dataset che tratta delle Top Hits di Spotify, non pare logico che esistano delle tracce che hanno valore di popolarità pari a 0.

```
df<-subset(df, subset=(df$popularity!=0))
```

In seguito abbiamo creato un altra serie subset per verificare effettivamente se le features “dancebility”, “energy”, “acousticness” e “valence” hanno solo valori compresi tra 0 e 1 come visto da Kaggle; constatando che i valori dei dati sono corretti.

```
df<-subset(df, subset=(df$energy>=0&df$energy<=1))
df<-subset(df, subset=(df$acousticness>=0&df$acousticness<=1))
df<-subset(df, subset=(df$valence>=0&df$valence<=1))
df<-subset(df, subset=(df$danceability>=0&df$danceability<=1))
```

4) SPLITTING

Dopo aver ripulito il dataset nelle precedenti fasi ed aver controllato i relativi dati, abbiamo ottenuto un numero di righe pari a 1832.

Dunque in questa quarta fase il dataset verrà diviso in tre parti: training set, test set e validation set.

Il training set deve essere più grande rispetto alle altre due parti, comprenderà quindi circa a il 70% del dataset ripulito, mentre test set e validation set hanno stessa dimensione, dunque all'incirca un totale del 30%.

Abbiamo optato per la seguente suddivisione:

- Training set = 70% —> 1283 row
- Test set = ~15% —> 280 row
- Validation set = ~15% —> 269 row

Successivamente alla divisione delle parti, dobbiamo "far finta" di non avere a disposizione il test set, in quanto questa parte contiene dati futuri che dobbiamo ancora collezionare, quindi, verranno sfruttati solamente il training e validation set.

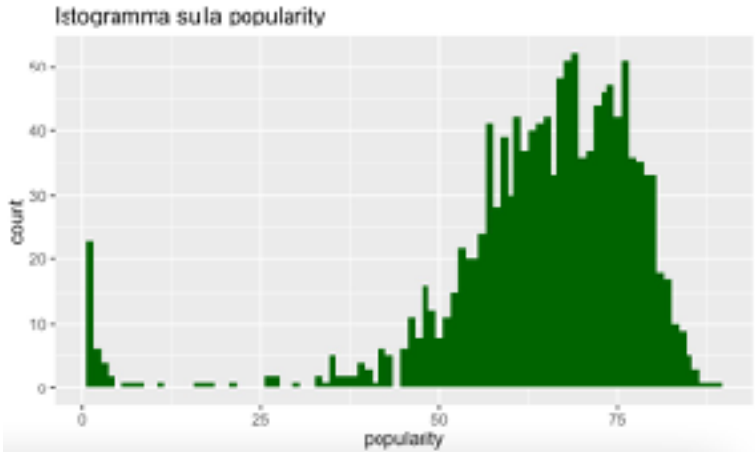
Di conseguenza carichiamo il training set sulla variabile df, vale a dire la variabile che conteneva il dataset originale.

5) EXPLORATORY DATA ANALYSIS (EDA)

Questa parte di lavoro ci è stata utile al fine di prendere dimestichezza con il dataset e le sue variabili, iniziando a capire meglio quali variabili l'utente può ritenere più importarti per la sua soddisfazione e valutazione.

Abbiamo deciso di realizzare un grafico a torta per rappresentare la divisione quantitativa di canzoni che hanno contenuti espliciti all'interno del loro testo.

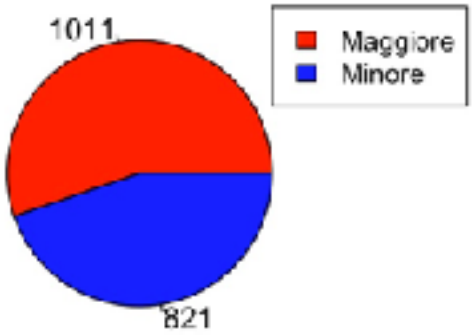
Dalla rappresentazione del grafico, si può notare che nell'insieme di 1283 canzoni ve ne è una maggiore quantità senza contenuti espliciti, per un totale di 926 (fetta del grafico colorata in rosa), rispetto alle tracce con contenuti espliciti, precisamente 357 (fetta del grafico colorata in azzurro).



Mentre, per quanto riguarda la popolarità che una traccia ha conseguito abbiamo deciso di utilizzare un grafico ad istogramma.

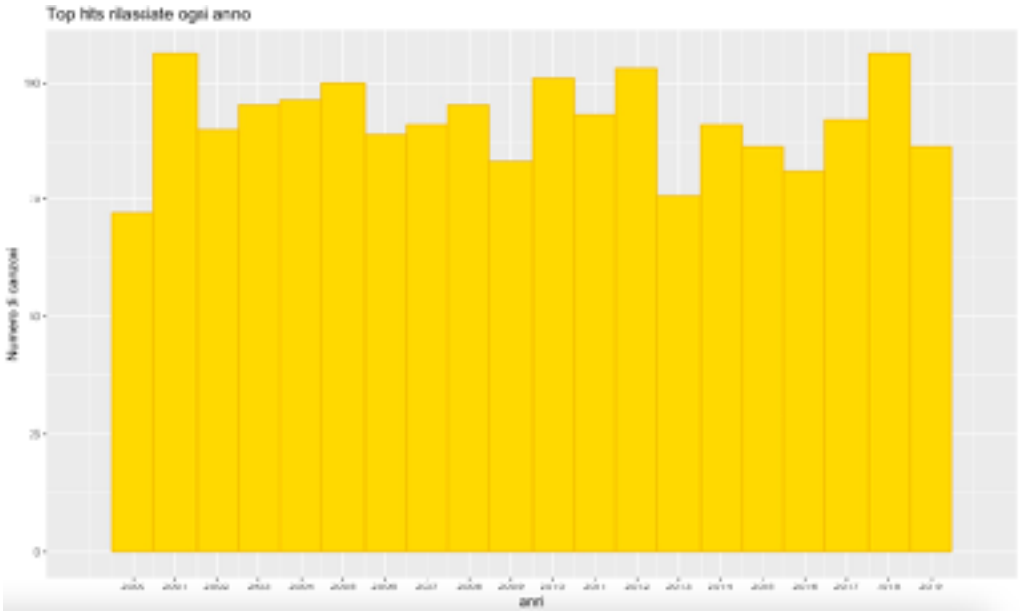
Il range di "popularity" analizzato va da valore 1 a 89, ovvero i massimi e minimi di questa feature. Il grafico ci mostra che la concentrazione maggiore si trova nell'intervallo di popolarità tra 60 e 80, mentre vi si possono trovare meno canzoni nella fascia inferiore a 50, essendo giustamente trattato un dataset sulle Top Hits mondiali.

A seguire abbiamo realizzato un secondo diagramma a torta riguardante la feature “mode”, vale a dire la tonalità di una canzone, la quale può essere espressa in scala maggiore o minore.
Possiamo osservare che quindi il nostro dataset contiene 1011 tracce espresse in scala maggiore (porzione in rosso) e le restanti 821 in scala minore (porzione in blu)

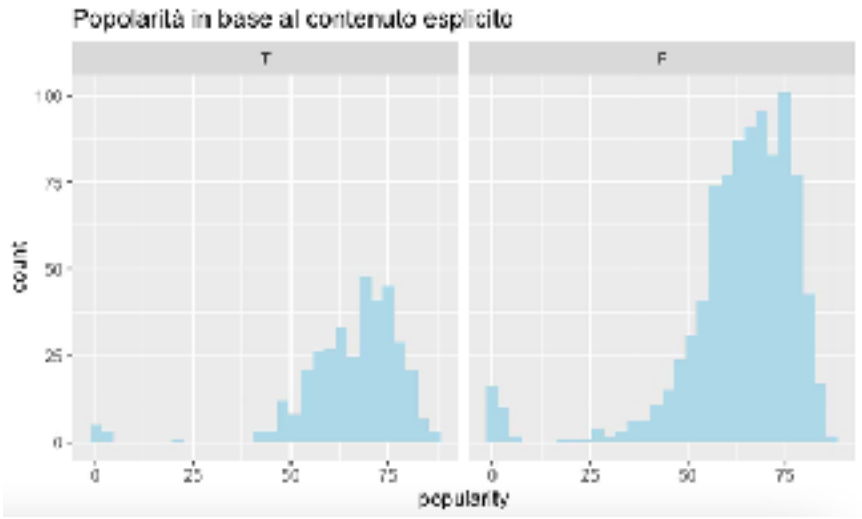


Successivamente alla rappresentazione grafica delle singole variabili, abbiamo deciso di andare ad unire più caratteristiche ponendole a confronto tra di loro.

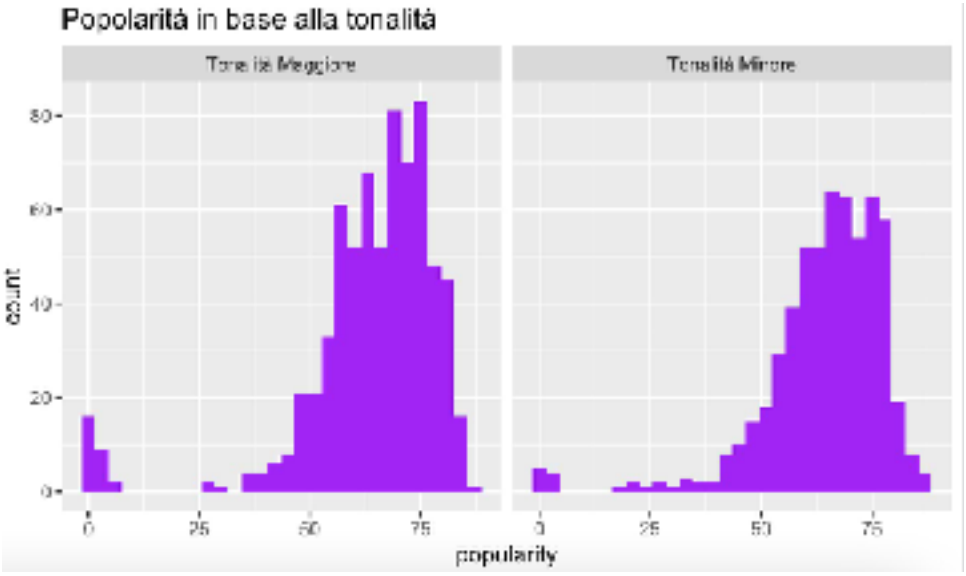
Con il grafico a istogramma sottostante abbiamo deciso di rappresentare la quantità di canzoni prodotte in base al loro anno di rilascio. Notiamo dunque che nell’anno 2001 e 2018 il mercato musicale era saturo di tracce finite nelle Top Hits di Spotify, mentre nel 2000 ve ne sono meno.



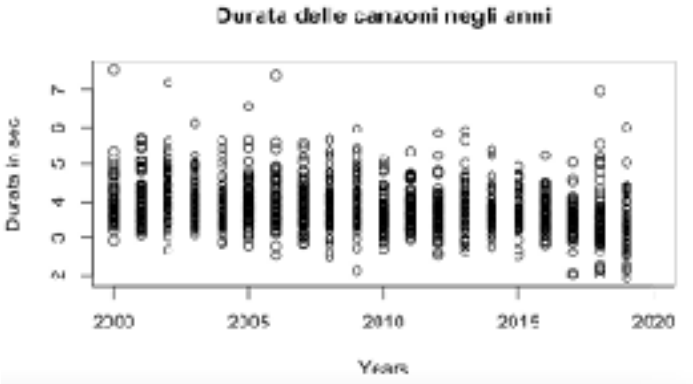
Nell' istogramma riportato qui a lato, invece, abbiamo messo a confronto la popolarità ottenuta dalle canzoni con il loro contenuto esplicito.
In esso si può facilmente vedere che le tracce che non contengono parole offensive o inadatte sono particolarmente numerose e hanno un picco nel range di popolarità che si aggira intorno ai valori di 70-80. A differenza delle canzoni nelle quali il contenuto esplicito si è rivelato "TRUE", che sono decisamente minori per quanto riguarda i valori sopra citati.



Un altro istogramma che abbiamo voluto realizzare riguarda sempre la feauture "popolarità" messa in relazione con la tonalità. Possiamo cosi osservare che il numero di canzoni che posseggono una tonalità maggiore risultano, per il pubblico, più polari, seppur di poco rispetto alle canzoni con una tonalità minore.

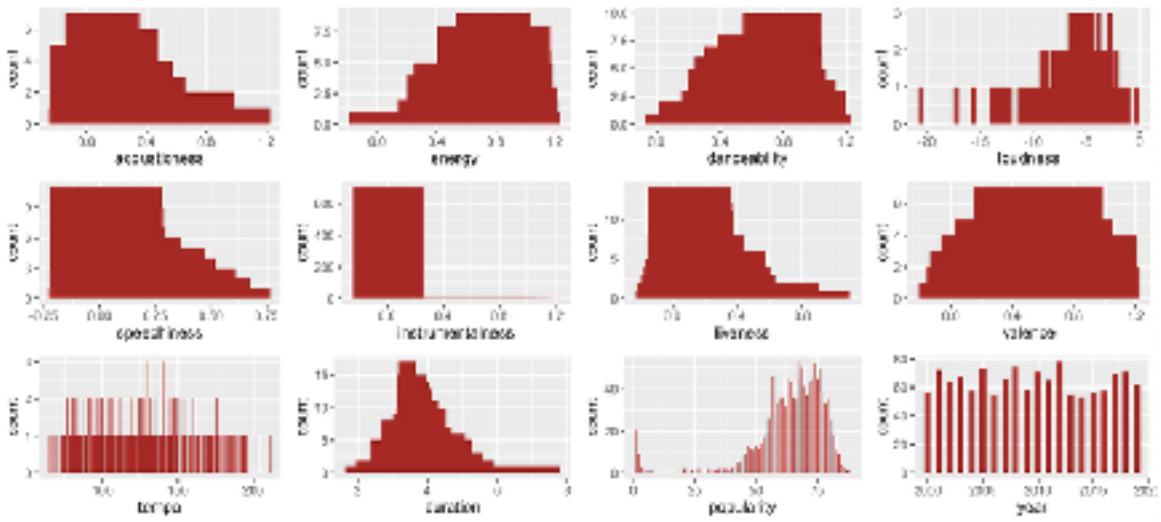


Abbiamo, in seguito, creato un plot per visualizzare come la durata in secondi delle tracce si sia evoluta durante il corso degli anni presi in analisi nel nostro dataset. Si può infatti notare come solitamente le canzoni durino tra un minimo di 3 minuti ed un massimo di 5, ed inoltre come l'avvicinandosi ai giorni nostri, abbia portato ad una tendenza di diminuire il tempo delle canzoni.



Successivamente abbiamo pensato che era importante costruire un'insieme di grafici a barre per ogni caratteristica di una canzone, andando così a vedere, contemporaneamente, in uno schema unico una panoramica generale sulle features delle canzoni.

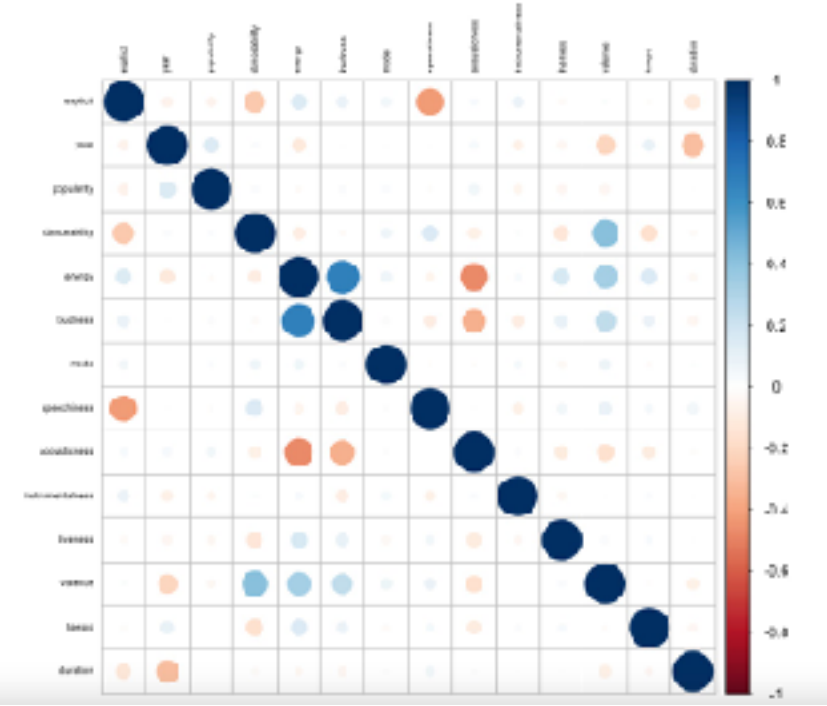
Come primo step andiamo a creare singolarmente, per ogni caratteristica, un grafico a barre, dove viene evidenziato sull'asse y la quantità di canzoni (come "count") mentre sull'asse x la feature singolare.
Successivamente ci avvaliamo di una libreria chiamata "gridExtra" ed utilizziamo la funzione "grid.arrange" che ci permette di organizzare i plot.



Notiamo che i valori dei singoli grafici risultano diversi tra di loro, poiché la densità di valori rappresentati in essi si differenzia. Ed esempio nel grafico relativo ad “energy" sull’asse delle ordinate i valori variano da un minimo di 0.0 a circa 8 canzoni, ed inoltre i valori della stessa feature si differenziano di poco. Di conseguenza, per questi motivi, il grafico risulta più denso; a differenza di altri grafici rappresentanti, per esempio, “popularity” e “tempo” che avendo un range di valori della feature molto ampio, risultanti meno densi.

Infine, abbiamo eseguito il plot più importante tra tutti, costruendo così una matrice di correlazione. La matrice di correlazione è uno strumento a noi utile che ci permette di costruire una matrice in cui sia sull’asse delle ascisse che su quello delle ordinate sono riportate tutte le variabili del dataset e, tramite un valore e un “circle” più o meno grosso e marcato, mi permette di visualizzare quale è il grado di correlazione di ciascuna variabile con le altre.

Per poterla stampare bisogna innanzitutto installare la libreria “corrplot” e creare un dataframe che contiene solo valori numerici del dataframe originale al quale passeremo la funzione “cor" per creare la matrice di correlazione.
Ma la funzione “cor" accetta solo dataframe di tipo numerico, dunque le colonne che non erano numeriche vengono trasformate in valori numerici.



Ovviamente la diagonale centrale ha valore unicamente uguale ad 1 (vale a dire che ha il massimo della correlazione) ma possiamo anche notare che le variabili “energy" e “loudness” sono a loro volta correlate tra di loro; questo perché ci si può aspettare che una canzone con un livello di energia elevato possa avere anche un volume totale molto alto.

6) ADDESTRAMENTO DEL MODELLO

Per addestrare il nostro modello (una Support Vector Machine) inizialmente occorre installare il pack “e1071”, il quale fornisce funzioni utili per svolgere operazioni statistiche e probabilistiche.

In seguito all’installazione, è necessario scegliere la tipologia di kernel tra le 4 possibili: linear, polynomial, radial, sigmoid.

Dopo numerosi tentativi con diversi kernel abbiamo considerato il più adatto quello di tipo “polynomial” che ha come iperparametri “cost” e “degree”.

Per trovare gli iperparametri “migliori”, ovvero quelli che restituiscono un MR il più piccolo possibile e un accuracy il più possibile vicino al 100%, si deve eseguire la SVM numerose volte cambiando cost, ed analizzando in seguito MR e Acc.

Dopo questi numerosi tentativi è stato constatato che la SVM più precisa è quella con cost=10 e degree=10.

```
# installo il pacchetto e1071
install.packages("e1071")
library(e1071)


model.SVM <- svm(explicit~., df, kernel = "polynomial", cost=10, degree=10)
print(model.SVM)
summary(model.SVM)
```

7) VALUTAZIONE DEL MODELLO

Una volta definito un modello, bisogna valutarne la performance. La valutazione del modello verrà basata su due valori: il Misclassification Rate (MR) e l'Accuracy (Acc).

Si inizia definendo le funzioni utili a calcolare l'MR e l'Acc, dove:

MR (frequenza di misclassificazione) —> un numero decimale compreso tra 0 e 1 e rappresenta il numero medio di predizioni accurate fatte dal predittore.

Acc (accuratezza) —> un numero decimale compreso tra 0 e 1; più è grande, più il modello sarà migliore.

```
# creo la funzione MR (primo strumento per valutare il modello)
MR <- function(y.pred, y.true) {
  res <- mean(y.pred != y.true)
  return (res)
}

# creo la funzione Acc (secondo strumento per valutare il modello)
Acc <- function(y.pred, y.true) {
  res <- 1 - mean(y.pred != y.true)
  return (res)
```

Successivamente, con l'ausilio di "model.SVM", andiamo a predire i valori del test set e, una volta fatto ciò, valutiamo la qualità della predizione.

```
y.pred <- predict(model.SVM,df.test)
MR(y.pred, df.test$explicit)
Acc(y.pred, df.test$explicit)
```

```
> MR(y.pred, df.test$explicit)
[1] 0.3285714
> Acc(y.pred, df.test$explicit)
[1] 0.6714286
>
```

Abbiamo ottenuto questi valori di MR e Acc, ma ad ogni esecuzione del codice questi valori variano, in quanto la fase di splitting è stata fatta in maniera casuale.

8) HYPERPARAMETER TUNING

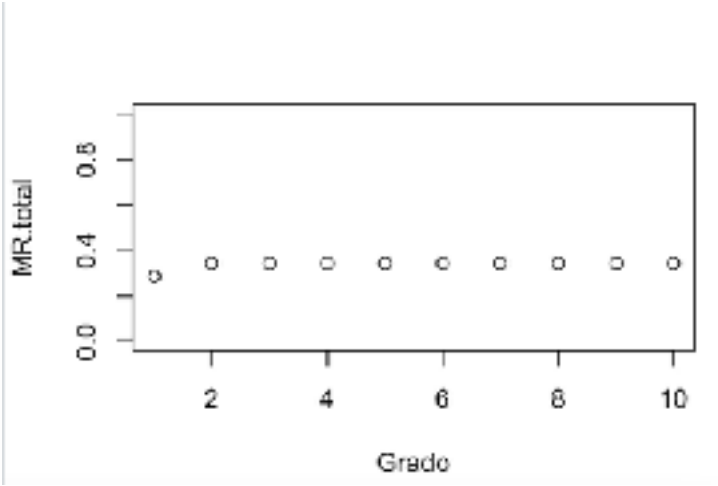
Lo scopo di questa fase è quello di trovare il costo e grado ottimale in modo che l'MR sia più vicino a 0 e l'accuracy più vicino ad 1 possibile.

Per far ciò useremo il validation set provando a predire gli iperparametri ideali, iterando i procedimenti.

Dopo aver realizzato i relativi cicli for , abbiamo trovato il costo e grado ideali, caricandoli nelle variabili "c" e "d" per poi continuare le analisi con i valori migliori. Abbiamo infine visualizzato dei plot che analizzeremo qui sotto.

Questo è il codice e i grafici relativi al grado:

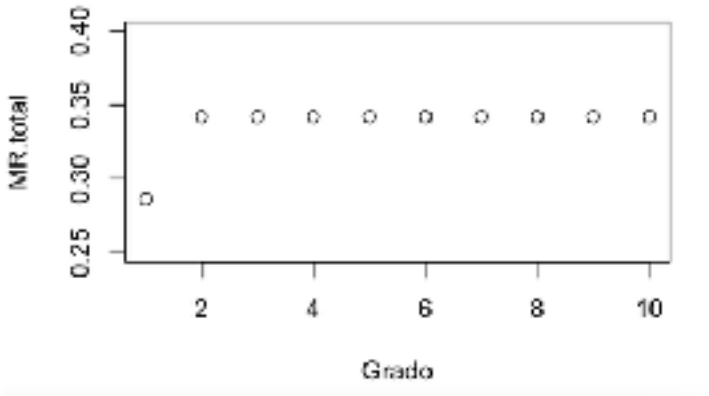
```
#Creo un vettore di 10 elementi e lo assegno alla variabile MR.total
MR.total <- 1:10

#itero il ciclo 10 volte al fine di trovare il valore di degree migliore a costo 10
for(d in 1:10){
  model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=10, degree=d)
  y.pred <- predict(model.SVM, df.val)
  MR.poly <- MR(y.pred, df.val$explicit)
  MR.total[d] <- MR.poly
}
# carico in d il degree migliore
d <- which.min(MR.total)

#visualizzo un plot per vedere il deegre ad ogni ciclo
plot(MR.total, type='p', xlab="Grado", ylim=c(0,1))
#faccio uno zoom sulla zona interessata
plot(MR.total, type='p', xlab="Grado", ylim=c(0.25,0.4))
```



Questo è il grafico riguardante i valori contenuti del vettore MR.total in corrispondenza dei vari gradi. Possiamo notare che i “pallini vuoti” sono parecchio allineati, dunque abbiamo ristretto i limiti per determinare il grado ideale.

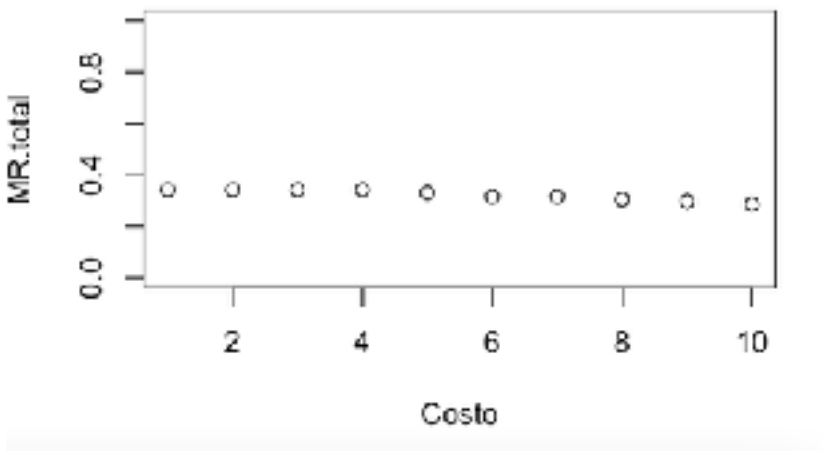
Andiamo quindi ad impostare i nuovi limiti dell’asse y, che saranno 0.25 quello inferiore e 0.40 quello superiore.

Attraverso questo zoom sul grafico deduciamo che il grado ottimale è di valore 1.



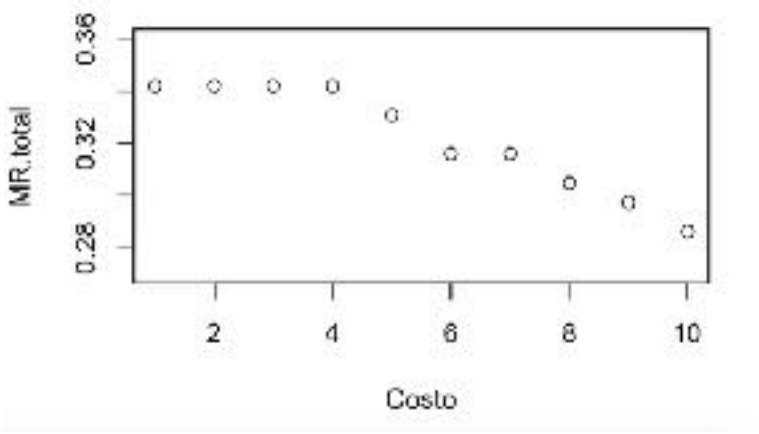
Questo, invece, è il codice e i grafici relativi al <u>costo</u>:

```
#applico la stessa procedura fatta per il grado anche per il costo
for(c in 1:10){
  model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=c, degree=d)
  y.pred <- predict(model.SVM, df.val)
  MR.poly <- MR(y.pred, df.val$explicit)
  MR.total[c] <- MR.poly
}
# carico in c il costo migliore
c <- which.min(MR.total)

#visualizzo un plot per vedere il cost ad ogni ciclo
plot(MR.total, type='p', xlab="Costo", ylim=c(0,1))
plot(MR.total, type='p', xlab="Costo", ylim=c(0.27,0.36))
```



Come per il grado, anche in questo grafico i “pallini vuoti” sono molto allineati tra di loro, quindi procediamo nuovamente a “zoommare” per dichiarare il valore ottimale di costo.

Da questo grafico possiamo notare che il valore di costo ottimale è pari a 10.



9) VALUTAZIONE DELLA PERFORMANCE

In questa nona fase l'obbiettivo è di andare a valutare il modello sul test set. Lo faremo riaddestrando il modello con i valori ottimali di cost e degree calcolati nella fase di hypermarameter-tuning.

```
model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=c, degree=d)
y.pred <- predict(model.SVM,df.test)
```

Andiamo quindi a stabilire le due funzioni che calcoleranno l'Acc e l'MR sul dataframe di test set.

```
MR.test <- MR(y.pred, df.test$explicit)
MR.test
Acc.test <- Acc(y.pred, df.test$explicit)
Acc.test
```

Una volta applicato il modello stampiamo i nostri "strumenti" di valutazione del modello:

```
> MR.test <- MR(y.pred, df.test$explicit)
> MR.test
[1] 0.2607143
> Acc.test <- Acc(y.pred, df.test$explicit)
> Acc.test
[1] 0.7392857
>
```

I valori ottenuti non sono uguali a quelli ottenuti nel validation set ma possiamo comunque constatare che il nostro modello è ottimo anche sul test set.

10) INTERPRETAZIONE PROBABILISTICA

In questa fase ci troviamo nel caso di una classificazione binaria in cui la variabile y di output (nel nostro caso è explicit) può assumere due valori:

0 = la traccia non contiene parole esplicite.
1= la traccia contiene parole esplicite.

Quello che vogliamo è calcolare la probabilità che explicit sia uguale a 0 oppure 1 ad ogni predizione.

```
#addestriamo la svm probabilistica con kernel polynomial e i valori di
#cost e deegre ottimali calcolati in precedenza
SVM.probs <- svm(explicit ~ ., df, kernel = "polynomial", cost=c, degree=d,
                 probability = TRUE)

#usiamo la svm e facciamo predizioni
y.pred <- predict(SVM.probs, df.test, probability = TRUE)
y.pred

#estraggo una singola colonna, poichè mi sarà utile per creare il vettore seguente
y.probs <- attr(y.pred, "probabilities")
y.probs <- y.probs[,1]
y.probs
```

Successivamente prendiamo la colonna di explicit uguale a 1.

Fatto questo è stata settata una percentuale che rappresenta un valore sopra il quale si può considerare che la canzone abbia contenuti espliciti. Questo valore, detto treshold, è stato scelto pari al 50% (0.05).

```
#prendo il vettore costruito e lo converto in un vettore di predizioni
#questo vettore conterrà solo valori binari di 0 e 1
y.total <- rep(0, N.test)
y.total[y.probs > 0.5] <- 1     # treshold = 0.5
y.total
```

Stampiamo dunque una tabella che ci indicherà che la probabilità in ogni caso sia o 0 o 1.

```
  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [42] 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 [83] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1
[124] 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 1 1 0 0 0 1 1
[165] 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1
[206] 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0
[247] 0 0 1 1 1 0 0 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
```

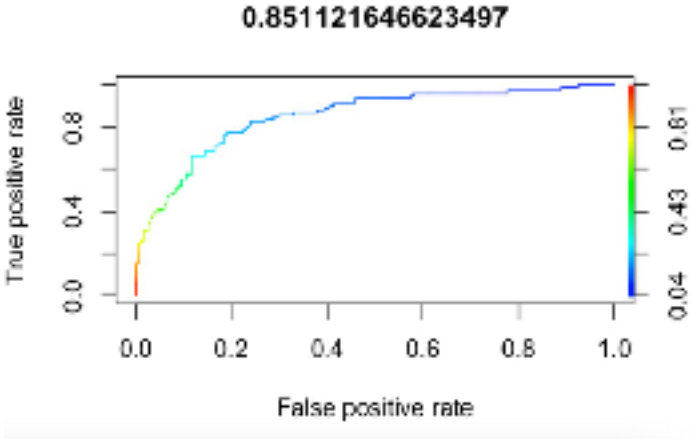
Andiamo poi a creare la matrice di confusione, ovvero una tabella che ci permette di visualizzare il numero di Falsi Negativi e Falsi Positivi del nostro modello; con il comando:

```
#creiamo una tabella che rappresenta la matrice di confusione
table(y.pred, df.test$explicit)
```

y.pred	T	F
T	41	17
F	38	184

Un'altra prova per concludere al meglio la valutazione del nostro modello è la curva di ROC. E come primo passo installiamo il pacchetto "ROCR".

Abbiamo, in seguito, valutato due unità di misura: il true positive rate e il false positive rate. La curva ottenuta, la quale rappresenta la treshold è la seguente:



Il numero soprastante il grafico è il valore dell'AUC (Area under the curve) che corrisponde a circa l' 85%, che consideriamo una buon valore poiché più si avvinca ad 1 più il modello sarà buono.

11) STUDIO STATISTICO SUI RISULTATI DELLA VALUTAZIONE

L'undicesimo punto di questo lavoro consiste nel ripetere le fasi di addestramento e testing k>10 volte, per dare una valutazione ancora più corretta del modello.

```
#stabilisco quante volte voglio fare ciclare i calcoli successivi
#scelgo 15 poiché è un valore >10 e permette di avere un calcolo "veloce"
k <- 15

# creo i vettori MR.SRS e Acc.SRS di lunghezza pari a k
MR.SRS <- 1:k
Acc.SRS <- 1:k

#ripeto il codice fatto in precedenza, usando il backup poiché
#il nostro dataset originale era stato usato per il train set
N <- nrow(df_backup)
N.train <- 1283
N.test <- 280
N.val <- 269

#ripeto la fase 6 per addestrare il nostro modello
for (i in 1:k){
  df <- df_backup
  train.sample <- sample(N,N.train)
  df.train <- df[train.sample,]
  df.test <- df[-train.sample,]

  val.sample <- sample(N.test + N.val, N.val)
  df.val <- df.test[val.sample,]
  df.test <- df.test[-val.sample,]
  df <- df.train
  MR.total <- 1:10

  #ripeto la fase 8 per prendere gli iperparametri ottimali
  for(d in 1:10){
    model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=5, degree=d)
    y.pred <- predict(model.SVM, df.val)
    MR.poly <- MR(y.pred, df.val$explicit)
    MR.total[d] <- MR.poly
  }
  d <- which.min(MR.total)

  for(c in 1:10){
    model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=c, degree=d)
    y.pred <- predict(model.SVM, df.val)
    MR.poly <- MR(y.pred, df.val$explicit)
    MR.total[c] <- MR.poly
  }
  c <- which.min(MR.total)

  model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=c, degree=d)
  y.pred <- predict(model.SVM, df.test)

  #scrivo sui vettori MR.SRS e Acc.SRS i valori delle valutazioni del modello
  MR.SRS[i] <- MR(y.pred, df.test$explicit)
  Acc.SRS[i] <- Acc(y.pred, df.test$explicit)
}
```

Come si può vedere dalla relativa parte di codice prima di tutto si è settato il valore di k, impostato a k=15, per cercare di avere anche un tempo di esecuzione non troppo elevato, ma comunque con un valore di k>10.

In seguito, abbiamo settato anche i due vettori MR.SRS e Acc.SRS ed abbiamo realizzato anche una fase di splitting in modo da non aver gli stessi valori quando si andava a generare l'SRS.

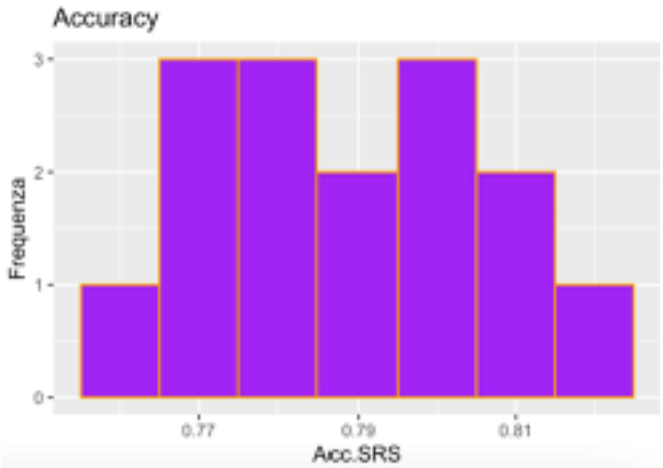
Dentro al ciclo for su k è stata anche inserita la parte di hyperparameter-tuning in cui venivano calcolati il costo e il grado ottimali.

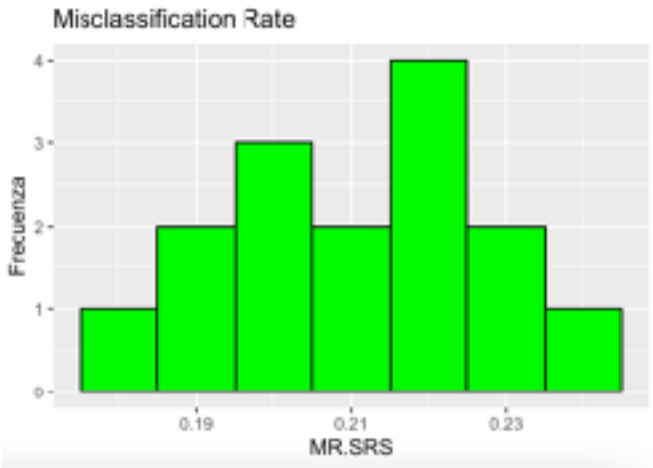
Successivamente salviamo nei vettori MR.SRS e Acc.SRS gli indici MR e Acc, che ad ogni ciclo verranno calcolati.

Per poter stampare i grafici, usiamo la libreria ggplot creando un dataframe (SRS.data) che contiene i vettori con gli MR.SRS e Acc.SRS.

```
# per poter usare la libreria ggplot viene richiesta la creazione di un dataset
# contenente i due array con le valutazioni del modello
SRS.data <- data.frame(MR.SRS=MR.SRS, Acc.SRS=Acc.SRS)
```

Di conseguenza riusciamo a visualizzare i contenuti dei due vettori, sotto forma di grafici a istogramma.





Per poter avere qualche dettaglio in più riguardo i due vettori usiamo i comandi summary(MR.SRS) e summary(Acc.SRS):

```
> summary(MR.SRS)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1786  0.1946  0.2214  0.2183  0.2357  0.2786
> summary(Acc.SRS)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.7214  0.7643  0.7786  0.7817  0.8054  0.8214
```

In secondo luogo vogliamo fare inferenza riguardo alla distribuzione cui appartiene il campione. In particolare, stimando la media e calcolando l' intervallo di confidenza.

La media nel nostro caso corrisponde a circa 0.218 come possiamo vedere dal summary di MR.SRS. Abbiamo però arrotondato il suo valore a 0.22

```
#inferenza riguardo alla distribuzione a cui appartiene il campione
#stima della media
sigma <- sd(MR.SRS)
mu0 <- 0.22
alpha <- 0.05
x <- MR.SRS
m <- mean(x)

#calcolo intervallo di confidenza
zalfa <- qnorm(1 - alpha /2,0,1)
c1 <- m - zalfa * sigma / sqrt(x)
c2 <- m - zalfa * sigma / sqrt(x)

#test verifica di ipotesi
if (mu0 < c1 | mu0 > c2) {
  message("Ipotesi: Rejected")
} else {
  message("Ipotesi: Not Rejected")
}
```

```
Ipotesi: Rejected
```

Essendo fuori dall'intervallo di confidenza l'ipotesi risulta RIGETTATA.

Questo ultimo passaggio, anziché farlo con il MR, avremmo potuto farlo con l'Acc semplicemente mettendo come x=Acc.SRS e assegnando a mu0 la media dell'accuratezza rilevata grazie al comando summary precedentemente (μ = ~0.79).

12) FEATURE SELECTION

Per procedere ad un ulteriore studio sul modello, come ultimo passo, effettuiamo l'operazione di feature selection, che consiste nell'individuare l'influenza di alcune caratteristiche ritenute più importanti, ai fini della qualità del modello.

Nonostante le feature possono essere eliminate in maniera casuale, abbiamo scelto di toglierne alcune che abbiamo notato essere particolarmente legate a explicit nella matrice di correlazione effettuata al punto 5, quali:

- la variabile loudness

```
#modello senza la feature loudness
model.SVM <- svm(explicit ~ .-loudness, df, kernel = "polynomial", cost=c, degree=d)
y.pred <- predict(model.SVM, df.test)
MR(y.pred, df.test$explicit)
Acc(y.pred, df.test$explicit)
```

Valori con loudness —> Valori senza loudness

```
> MR(y.pred, df.test$explicit)
[1] 0.3285714
> Acc(y.pred, df.test$explicit)
[1] 0.6714286
>
```

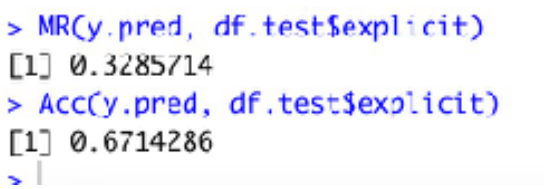
```
> MR(y.pred, df.test$explicit)
[1] 0.1892857
> Acc(y.pred, df.test$explicit)
[1] 0.8107143
```

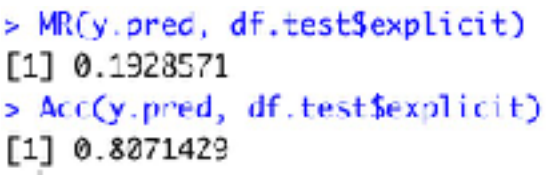
Osserviamo come la variabile loudness, se tolta ha un impatto rilevante sia su entrambi i valori di MR e Acc.

- la variabile energy

```
model.SVM <- svm(explicit ~ .-energy, df, kernel = "polynomial", cost=c, degree=d)
y.pred <- predict(model.SVM, df.test)
MR(y.pred, df.test$explicit)
Acc(y.pred, df.test$explicit)
```

Valori con energy —> Valori senza energy





Anche in questo caso i valori di MR e Acc variano, ma notiamo che la variabile energy ha impatto minore rispetto a loudness, seppur di poco.

In conclusione affermiamo che nel dataset da noi selezionato non sono presenti variabili superflue ma più che altro alcune più influenti di altre sul risultato. Questo perché ciascuna variabile contribuisce, a suo modo, all'incremento dell'Acc e all'abbattimento del MR.

13) CODICE COMPLETO

Qui sotto riportiamo il codice completo:

```
#### 1-2. SELEZIONARE E CARICARE IL DATASET ####
# Setto la working directory in modo che posso caricare il dataset
setwd("~/Desktop/R statistica")

# carico sulla variabile df il dataset originale
df <- read.csv("songs normalize.csv")

# vediamo una prima panoramica del dataset con queste funzioni
# visualizziamo una prima panoramica del dataset
head(df)
# visualizziamo la dimensione del nostro dataset
dim(df)

# cancello la caratteristiche inutilizzate
df$key=NULL

# creiamo una variabile nel quale inserire una copia del dataset, già ripulito da
# eventuali valori che non ci servono o che sono nulli
df_backup <- df;


#### 3. PRE-PROCESSING #####
# rimuovo i NaN dal dataset
# non cambia niente poiché nella descrizione del dataset viene evidenziato che non vi sono valori nulli
df <- na.omit(df)

# con l'ausilio del comando "summary(df)" si possono verificare diversi valori di ciascuna
# variabile del dataset, tra cui media, mediana, 1° e 3° quantile ma soprattutto il valore
# minimo e massimo che questa assume.
summary(df)

# trasformo in factor le opportune variabili
df$artist <- factor(df$artist)
df$song <- factor(df$song)
df$explicit <- factor(df$explicit, c("True","False"), c("T","F"))
df$genre <- factor(df$genre)
df$mode <- factor(df$mode, c("1","0"), c("Tonalità Maggiore","Tonalità Minore"))

# trasformo la durata delle canzoni da millisecondi corrispondendo a minuti e secondi al fine di avere una leggibilità migliore del dataset
df$duration_ms <- df$duration_ms/100000
df$duration <- df$duration_ms
df$duration <- round(df$duration, digits=2)
df$duration_ms <- NULL

# creo un subset da assegnare al database al fine di visualizzare solo le song dal 2000 al 2019 come interessa a noi
df<- subset(df, subset=(df$year>1999&df$year<2020))

# creo un subset per vedere se la dancebility ha solo valori compresi tra 0 e 1 come su kaggle
df<-subset(df, subset=(df$danceability>=0&df$danceability<=1))
# itero il concetto sulle altre variabili
df<-subset(df, subset=(df$energy>=0&df$energy<=1))
df<-subset(df, subset=(df$acousticness>=0&df$acousticness<=1))
df<-subset(df, subset=(df$valence>=0&df$valence<=1))
# la loudness ha un range solitamente che fa da -60 a 0 decibell, quindi considero quello
df<-subset(df, subset=(df$loudness>=-61&df$loudness<1))

# abbiamo scelto di ripulire il dataset da tutte le osservazione con la variabile
# popularity=0 perché essendo un dataset sulle hophits non ci sembra logico che
# esistano delle valori simili
df<-subset(df, subset=(df$popularity!=0))

# mi copio il df ripulito su df_backup
summary(df);
df_backup=df


#### 4 - SPLITTING ####
N  <- nrow(df)

# stabilisco il numero di cui da estrarre in n.train, n.test, n.val
N.train <- 1788
N.test <- 238
N.val <- 269

# creo in train.sample togliendo N.test e N.val
train.sample<-sample(N,N.train)
df.train<-df[train.sample,]
df.test<-df[-train.sample,]

# creo il val.sample prendendo solo N.val
val.sample <- sample(N.test,N.val, N.val)
df.val <- df.test[val.sample,]
df.test <- df.test[-val.sample,]

# copio il train su df in modo da utilizzare solo quello
df <- df.train
N <- nrow(df)
```

```r
##### 5 - Exploratory Data Analysis (EDA) #####
# Installo il pacchetto corrplot e verifico la presenza del pacchetto ggplot2
library(ggplot2)
install.packages ("corrplot")
library (corrplot)

# creo un grafico a torta sul contenuto esplicito
pie(summary(df$explicit), summary(df$explicit),  main="Contenuto Esplicito", col=c("light blue","pink"))
legend(1.8, 1.8, cex = 0.9, legend = c("Esplicit", "NoEsplicit"), fill = c("light blue","pink"))

# creo un grafico a torta sulla tonalità maggiore o minore
pie(summary(df$mode), summary(df$mode),  main="Tonalità", col=c("red","blue"))
legend(1.8, 1.8, cex = 0.9, legend = c("Maggiore", "Minore"), fill = c("red","blue"))

# creo un istogramma sulle top hits per anno
ggplot(df, aes(x = year)) + geom_histogram(aes(y = ..count..), binwidth = 1,
colour = "goldenrod2", fill = "gold1") + scale_x_continuous(name = "anni",
breaks = seq(2000, 2019, 1),
limits=c(1999, 2020)) +
scale_y_continuous(name = "Numero di canzoni") + ggtitle("Top hits rilasciate ogni anno")

# creo un plot per visualizzare la durata in sec delle canzoni durante gli anni
plot(df$year,df$duration, xlab = "Years", ylab = "Durata in sec", main="Durata delle canzoni negli anni", xlim=c(2000,2020))

# creo un istogramma sulla popolarità
ggplot(df, aes(x=popularity)) + geom_histogram(fill = "darkgreen", binwidth = 1) + labs(title="Istogramma sulla popularity")

# istogramma sulla popolarità in base al contenuto esplicito
# ci permette di vedere che il numero di canzoni più popolari sono quelle che non contengono del contenuto esplicito
ggplot(df, aes(x=popularity)) + geom_histogram(fill = "light blue") + labs(title="Popolarità in base al contenuto esplicito") + facet_wrap(~df$explicit)

# istogramma sulla popolarità in base alla tonalità
# ci permette di vedere che il numero di canzoni con una tonalità maggiore sizno di più, se pur di poco rispetto alle canzoni con tonalità minore
ggplot(df, aes(x=popularity)) + geom_histogram(fill = "purple") + labs(title="Popolarità in base alla tonalità") + facet_wrap(~df$mode)

# in questo primo confronto si può visualizzare e dedurre che le canzoni preferite siano canzoni che non contengono
# contenuto esplicito e che abbiano una tonalità maggiore.


# installo un pacchetto per visualizzare più grafici contemporaneamente
# prima creo tutte le singole componenti, ovvero i singoli grafici
p1 <- ggplot(df, aes(x=acousticness)) + geom_bar(fill = 'brown', width = 0.5)
p2 <- ggplot(df, aes(x=energy)) + geom_bar(fill = 'brown', width = 0.5)
p3 <- ggplot(df, aes(x=danceability)) + geom_bar(fill = 'brown', width = 0.5)
p4 <- ggplot(df, aes(x=loudness)) + geom_bar(fill = 'brown', width = 0.5)
p5 <- ggplot(df, aes(x=speechiness)) + geom_bar(fill = 'brown', width = 0.5)
p6 <- ggplot(df, aes(x=instrumentalness)) + geom_bar(fill = 'brown', width = 0.5)
p7 <- ggplot(df, aes(x=liveness)) + geom_bar(fill = 'brown', width = 0.5)
p8 <- ggplot(df, aes(x=valence)) + geom_bar(fill = 'brown', width = 0.5)
p9 <- ggplot(df, aes(x=tempo)) + geom_bar(fill = 'brown', width = 0.5)
p10 <- ggplot(df, aes(x=duration)) + geom_bar(fill = 'brown', width = 0.5)
p11 <- ggplot(df, aes(x=popularity)) + geom_bar(fill = 'brown', width = 0.5)
p12 <- ggplot(df, aes(x=year)) + geom_bar(fill = 'brown', width = 0.5)

library(gridExtra)
grid.arrange(p1,p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, ncol=4, nrow=4)

# installo il pacchetto per fare matrice correlazione
install.packages("corrplot")
library(corrplot)

# creo un dataframe che poi passerò alla funzione cor per creare la matrice di correlazione.
# cor vuole un dataframe solo numerico quindi le colonne che non erano numeriche vengono trasformate in numeri e tolte
df_cor <- df
df_cor$explicit <- as.numeric(factor(df$explicit, c("T","F"), c(1, 0)))
df_cor$mode <- as.numeric(factor(df$mode, c("Tonalità Maggiore","Tonalità Minore"), c(1, 0)))
df_cor$song <- NULL
df_cor$artist <- NULL
df_cor$genre <- NULL

# creo la matrice di correlazione e poi la stampo
cor.matrix = cor(df_cor)
corrplot(cor.matrix, method="circle", tl.col = "black", tl.cex = 0.5)

# dimostro alcne correlazioni trovate nel corrplot tramite due grafici
plot(df$popularity,df$year, xlab = 'Popolarità', ylab = 'Anno')
plot(df$popularity,df$duration, xlab = 'Popolarità', ylab = 'Durata')
```

```r
### 6) ADDESTRAMENTO DEL MODELLO ###
# installo il pacchetto e1071
install.packages("e1071")
library(e1071)

# addestriamo il modello
# scegliamo di procedere inserendo come cost e degree i medesimi valori della traccia per il progetto
model.SVM <- svm(explicit~., df, kernel = "polynomial", cost=10, degree=10)
print(model.SVM)
summary(model.SVM)


### 7) VALUTAZIONE DELLA PERFORMANCE ###
# creo la funzione MR (primo tool per valutare il modello)
MR <- function(y.pred, y.true) {
  res <- mean(y.pred != y.true)
  return (res)
}

# creo la funzione Acc (secondo tool per valutare il modello)
Acc <- function(y.pred, y.true) {
  res <- 1 - mean(y.pred != y.true)
  return (res)
}

# facciamo una prova valutando il modello
y.pred <- predict(model.SVM,df.test)
MR(y.pred, df.test$explicit)
Acc(y.pred, df.test$explicit)



#### 8) HYPER-PARAMETER TUNING ####

# Creo un vettore di 10 elementi e lo assegno alla variabile MR.total
MR.total <- 1:10

# itero il ciclo 10 volte al fine di trovare il valore di degree migliore a costo 10
for(d in 1:10){
  model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=10, degree=d)
  y.pred <- predict(model.SVM, df.val)
  MR.poly <- MR(y.pred, df.val$explicit)
  MR.total[d] <- MR.poly
}
# carico in d il degree migliore
d <- which.min(MR.total)

# visualizzo un plot per vedere il deegre ad ogni ciclo
plot(MR.total, type='p', xlab="Grado", ylim=c(0,1))
#faccio uno zoom sulla zona interessata
plot(MR.total, type='p', xlab="Grado", ylim=c(0.25,0.4))

# applico la stessa procedura fatta per il grado anche per il costo
for(c in 1:10){
  model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=c, degree=d)
  y.pred <- predict(model.SVM, df.val)
  MR.poly <- MR(y.pred, df.val$explicit)
  MR.total[c] <- MR.poly
}
# carico in c il costo migliore
c <- which.min(MR.total)

# visualizzo un plot per vedere il cost ad ogni ciclo
plot(MR.total, type='p', xlab="Costo", ylim=c(0,1))
plot(MR.total, type='p', xlab="Costo", ylim=c(0.27,0.36))
```

```r
#### 9) VALUTAZIONE DELLA PERFORMANCE ####

# In questa fase di valutazione prendiamo il modello e lo consideriamo sul test set
model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=c, degree=d)
y.pred <- predict(model.SVM,df.test)

MR.test <- MR(y.pred, df.test$explicit)
MR.test
Acc.test <- Acc(y.pred, df.test$explicit)
Acc.test



#### 10) INTERPRETAZIONE PROBABILISTICA ###

# addestriamo la svm probabilistica con kernel polynomial e i valori di
# cost e deegre ottimali calcolati in precedenza
SVM.probs <- svm(explicit ~ ., df, kernel = "polynomial", cost=c, degree=d, probability = TRUE)

# usiamo la svm e facciamo predizioni
y.pred <- predict(SVM.probs, df.test, probability = TRUE)
y.pred

# estraggo una singola colonna, poichè mi sarà utile per creare il vettore seguente
y.probs <- attr(y.pred, "probabilities")
y.probs <- y.probs[,1]
y.probs

# prendo il vettore costruito e lo converto in un vettore di predizioni
# questo vettore conterrà solo valori binari di 0 e 1
y.total <- rep(0, N.test)
y.total[y.probs > 0.5] <- 1 # treshold = 0.5
y.total

# creiamo una tabella che rappresenta la matrice di confusione
table(y.pred, df.test$explicit)


# installo il pacchetto ROCR
install.packages("ROCR")
library(ROCR)

# costruisco la curva di ROCR
pred <- prediction(y.probs, df.test$explicit)
perf <- performance(pred, "tpr", "fpr")

# calcolo l'AUC
auc <- performance(pred, "auc")
auc <- auc@y.values[[1]]

# stampo curva di ROC e AUC
plot(perf, colorize=TRUE, main=auc)
```

```r
#### 11) STUDIO PROBABILISTICO SUI RISULTATI DELLA VALUTAZIONE ####

# stabilisco quante volte voglio fare ciclare i calcoli successivi
# scelgo 15 poichè è un valore >10 e permette di avere un calcolo "veloce"
k <- 15

# creo i vettori MR.SRS e Acc.SRS di lunghezza pari a k
MR.SRS <- 1:k
Acc.SRS <- 1:k

# ripeto il codice fatto in precedenza, usando il backup poichè
# il nostro dataset originale era stato usato per il train set
N <- nrow(df_backup)
N.train <- 1283
N.test <- 280
N.val <- 269

# ripeto la fase 6 per addestrare il nostro modello
for (i in 1:k){
  df <- df_backup
  train.sample <- sample(N,N.train)
  df.train <- df[train.sample,]
  df.test <- df[-train.sample,]

  val.sample <- sample(N.test + N.val, N.val)
  df.val <- df.test[val.sample,]
  df.test <- df.test[-val.sample,]
  df <- df.train
  MR.total <- 1:10

  # ripeto la fase 8 per prendere gli iperparametri ottimali
  # utilizzo cost=5 per velocizzare il calcolo
  for(d in 1:10){
    model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=5, degree=d)
    y.pred <- predict(model.SVM, df.val)
    MR.poly <- MR(y.pred, df.val$explicit)
    MR.total[d] <- MR.poly
  }
  d <- which.min(MR.total)
```

```
    for(c in 1:10){
      model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=c, degree=d)
      y.pred <- predict(model.SVM, df.val)
      MR.poly <- MR(y.pred, df.val$explicit)
      MR.total[c] <- MR.poly
    }
    c <- which.min(MR.total)

    model.SVM <- svm(explicit ~ ., df, kernel = "polynomial", cost=c, degree=d)
    y.pred <- predict(model.SVM, df.test)

    #carico sui vettori MR.SRS e Acc.SRS i valori delle valutazioni del modello
    MR.SRS[i] <- MR(y.pred, df.test$explicit)
    Acc.SRS[i] <- Acc(y.pred, df.test$explicit)
  }

  # per poter usare la libreria ggplot viene richiesta la creazione di un dataset
  # contenente i due array con le valutazioni del modello
  SRS.data <- data.frame(MR.SRS=MR.SRS, Acc.SRS=Acc.SRS)

  # vediamo attraverso degli istogrammi i valori di MR e Acc del nostro modello
  # grafico per MR
  ggplot(SRS.data, aes(x=MR.SRS)) +
    geom_histogram(col="black", fill="green", binwidth=0.01) +
    labs(title="Misclassification Rate", y="Frequenza")

  # grafico per ACC
  ggplot(SRS.data, aes(x=Acc.SRS)) +
    geom_histogram(col="orange", fill="purple", binwidth=0.01) +
    labs(title="Accuracy", y="Frequenza")

  summary(MR.SRS)
  summary(Acc.SRS)

  # inferenza riguardo alla distribuzione a cui appartiene il campione
  # stima della media
  sigma <- sd(MR.SRS)
  mu0 <- 0.22
  alpha <- 0.05
  x <- MR.SRS
  n <- mean(x)

  # calcolo intervallo di confidenza
  zalfa <- qnorm(1 - alpha /2,0,1)
  c1 <- n - zalfa * sigma / sqrt(k)
  c2 <- n + zalfa * sigma / sqrt(k)
```

```
# test verifica di ipotesi
if (mu0 < c1 | mu0 > c2) {
  message("Ipotesi: Rejected")
} else {
  message("Ipotesi: Not Rejected")
}

#### 12) FEATURE SELECTION ####

#vediamo inizialmente i valori
MR(y.pred, df.test$explicit)
Acc(y.pred, df.test$explicit)

# Feature selection
# modello senza la feature loudness
model.SVM <- svm(explicit ~ .-loudness, df, kernel = "polynomial", cost=c, degree=d)
y.pred <- predict(model.SVM, df.test)
MR(y.pred, df.test$explicit)
Acc(y.pred, df.test$explicit)

# modello senza la feature energy
model.SVM <- svm(explicit ~ .-energy, df, kernel = "polynomial", cost=c, degree=d)
y.pred <- predict(model.SVM, df.test)
MR(y.pred, df.test$explicit)
Acc(y.pred, df.test$explicit)
```