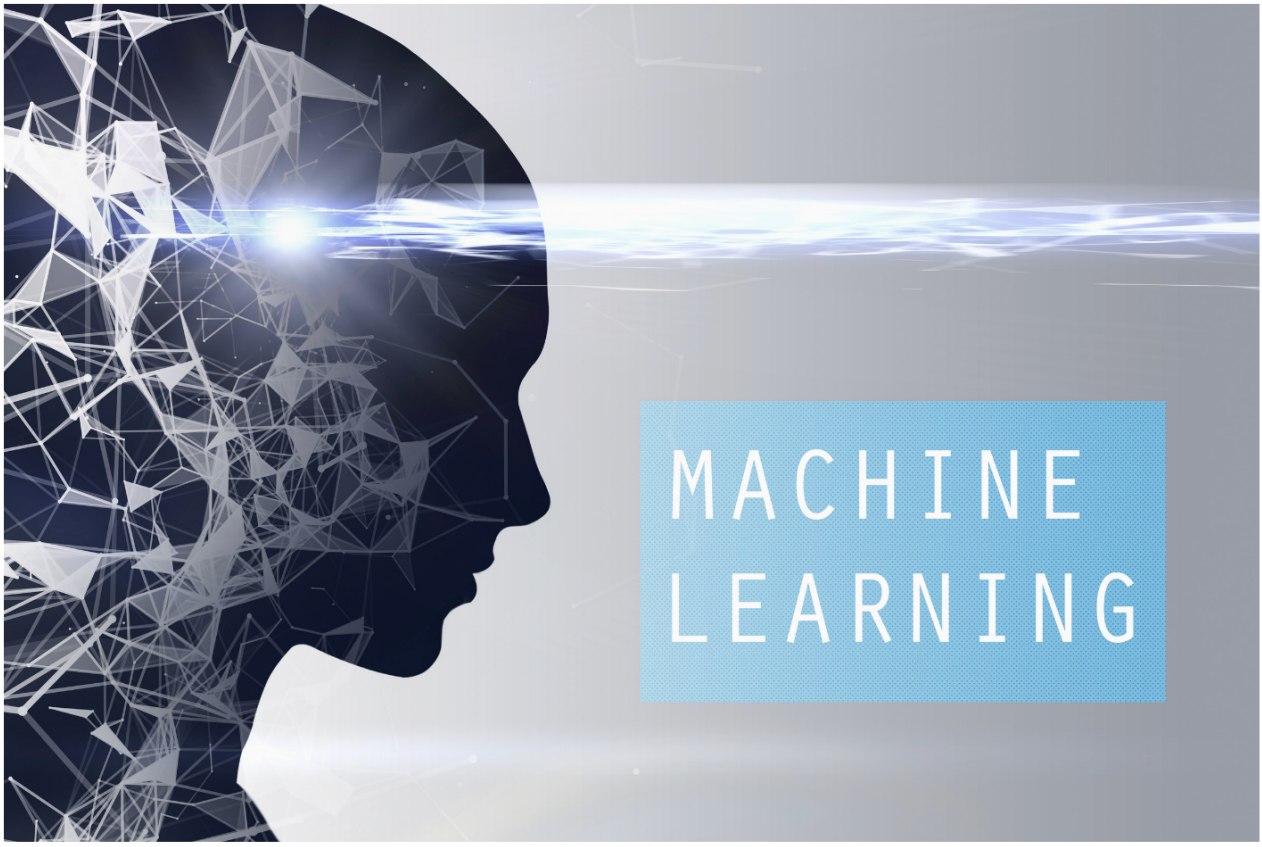
Room Occupancy Detection using SVM



Ing. Agostino Fontana

Trapani 06/12/2023

Sommario

1 | Descrizione dataset

Il dataset[1] oggetto di studio, è relativo all'analisi delle rilevazioni di una collezione di sensori passivi all'interno di una stanza. L'obiettivo dello studio era la rilevazione esatta del numero di persone occupanti una stanza attraverso la stima dei valori rilevati dai sensori, quindi una tecnica di rilevazione delle presenze di tipo non invasiva, come invece lo è un sistema di video-rilevazione. La rilevazione del numero di occupanti di una stanza risulta particolarmente interessante soprattutto in termini di efficientamento energetico in quanto, con l'avvento dell' "Internet delle cose" (IoT), si possono utilizzare dei sensori per temperatura, umidità, livello CO2, luminosità e movimento facilmente collocabili all'interno di una stanza con l'abilità di misurarene i parametri ambientali. I dati rilevati possono quindi essere analizzati utilizzando l'apprendimento automatico (ML) per determinare il livello di occupazione di una stanza evitando l'utilizzo di sistemi basati sulla rilevazione delle immagini. Studi recenti hanno dimostrato che negli edifici in cui è noto il livello occupazionale, attraverso l'utilizzo di sistemi domotici, è possibile ottenere risparmi energetici fino al 30%.

1.1 | Analisi esplorativa dei Dati

Il dataset è formato da 10.129 righe e 19 colonne, pari a 4 giorni di osservazioni rilevate ad intervalli regolari di 30 secondi, non sono presenti valori nulli o mancanti. Per ogni rilevazione si riportano: giorno, ora della rilevazione, i valori rilevati dai sensori passivi (colonne da 2 a 17) l'ultima variabile indica l'effettivo numero di occupanti della stanza che è stato rilevato manualmente.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10129 entries, 0 to 10128
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Date                  10129 non-null  object
 1   Time                  10129 non-null  object
 2   S1_Temp               10129 non-null  float64
 3   S2_Temp               10129 non-null  float64
 4   S3_Temp               10129 non-null  float64
 5   S4_Temp               10129 non-null  float64
 6   S1_Light              10129 non-null  int64
 7   S2_Light              10129 non-null  int64
 8   S3_Light              10129 non-null  int64
 9   S4_Light              10129 non-null  int64
 10  S1_Sound              10129 non-null  float64
 11  S2_Sound              10129 non-null  float64
 12  S3_Sound              10129 non-null  float64
 13  S4_Sound              10129 non-null  float64
 14  S5_CO2                10129 non-null  int64
 15  S5_CO2_Slope          10129 non-null  float64
 16  S6_PIR                10129 non-null  int64
 17  S7_PIR                10129 non-null  int64
 18  Room_Occupancy_Count  10129 non-null  int64
dtypes: float64(9), int64(8), object(2)
memory usage: 1.5+ MB
```

Figura 1 Struttura Dataset

	0	1	2	3	4
Date	2017/12/22	2017/12/22	2017/12/22	2017/12/22	2017/12/22
Time	10:49:41	10:50:12	10:50:42	10:51:13	10:51:44
S1_Temp	24.94	24.94	25.0	25.0	25.0
S2_Temp	24.75	24.75	24.75	24.75	24.75
S3_Temp	24.56	24.56	24.5	24.56	24.56
S4_Temp	25.38	25.44	25.44	25.44	25.44
S1_Light	121	121	121	121	121
S2_Light	34	33	34	34	34
S3_Light	53	53	53	53	54
S4_Light	40	40	40	40	40
S1_Sound	0.08	0.93	0.43	0.41	0.18
S2_Sound	0.19	0.05	0.11	0.1	0.06
S3_Sound	0.06	0.06	0.08	0.1	0.06
S4_Sound	0.06	0.06	0.06	0.09	0.06
S5_CO2	390	390	390	390	390
S5_CO2_Slope	0.769231	0.646154	0.519231	0.388462	0.253846
S6_PIR	0	0	0	0	0
S7_PIR	0	0	0	0	0
Room_Occupancy_Count	1	1	1	1	1

Tabella 1 Estrazione righe Dataset

[1] http://archive.ics.uci.edu/dataset/864/room+occupancy+estimation

Dall'osservazione delle principali statistiche descrittive, si nota che la *variabile target "Room Occupancy count*" varia tra un minimo di 0 ed un massimo di 3 occupanti, con una notevole prevalenza di valori 0; ciò detto è necessario tenerne in considerazione nelle successive fasi, poiché il set di dati risulta sbilanciato.

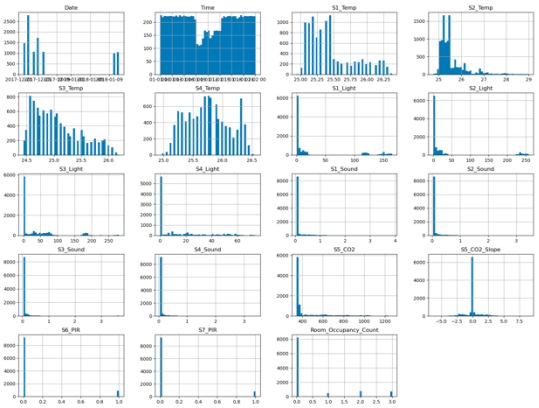


Figura 2 Distribuzione frequenze Variabili

	count	mean	std	min	25%	50%	75%	max
S1_Temp	10129.0	25.454012	0.351351	24.940000	25.190000	25.38	25.63	26.380000
S2_Temp	10129.0	25.546059	0.586325	24.750000	25.190000	25.38	25.63	29.000000
S3_Temp	10129.0	25.056621	0.427283	24.440000	24.690000	24.94	25.38	26.190000
S4_Temp	10129.0	25.754125	0.356434	24.940000	25.440000	25.75	26.00	26.560000
S1_Light	10129.0	25.445059	51.011264	0.000000	0.000000	0.00	12.00	165.000000
S2_Light	10129.0	26.016290	67.304170	0.000000	0.000000	0.00	14.00	258.000000
S3_Light	10129.0	34.248494	58.400744	0.000000	0.000000	0.00	50.00	280.000000
S4_Light	10129.0	13.220259	19.602219	0.000000	0.000000	0.00	22.00	74.000000
S1_Sound	10129.0	0.168178	0.316709	0.060000	0.070000	0.08	0.08	3.880000
S2_Sound	10129.0	0.120066	0.266503	0.040000	0.050000	0.05	0.06	3.440000
S3_Sound	10129.0	0.158119	0.413637	0.040000	0.060000	0.06	0.07	3.670000
S4_Sound	10129.0	0.103840	0.120683	0.050000	0.060000	0.08	0.10	3.400000
S5_CO2	10129.0	460.860401	199.964940	345.000000	355.000000	360.00	465.00	1270.000000
S5_CO2_Slope	10129.0	-0.004830	1.164990	-6.296154	-0.046154	0.00	0.00	8.980769
S6_PIR	10129.0	0.090137	0.286392	0.000000	0.000000	0.00	0.00	1.000000
S7_PIR	10129.0	0.079574	0.270645	0.000000	0.000000	0.00	0.00	1.000000
Room_Occupancy_Count	10129.0	0.398559	0.893633	0.000000	0.000000	0.00	0.00	3.000000

Tabella 2 Statistiche descrittive

La tabella successiva rappresenta l'analisi delle correlazioni delle variabili: osservando la *variabile target* è possibile notare una generale correlazione positiva con tutte le altre variabili, tuttavia sembra esserci una maggiore correlazione con le variabili temperatura, luce e CO2.

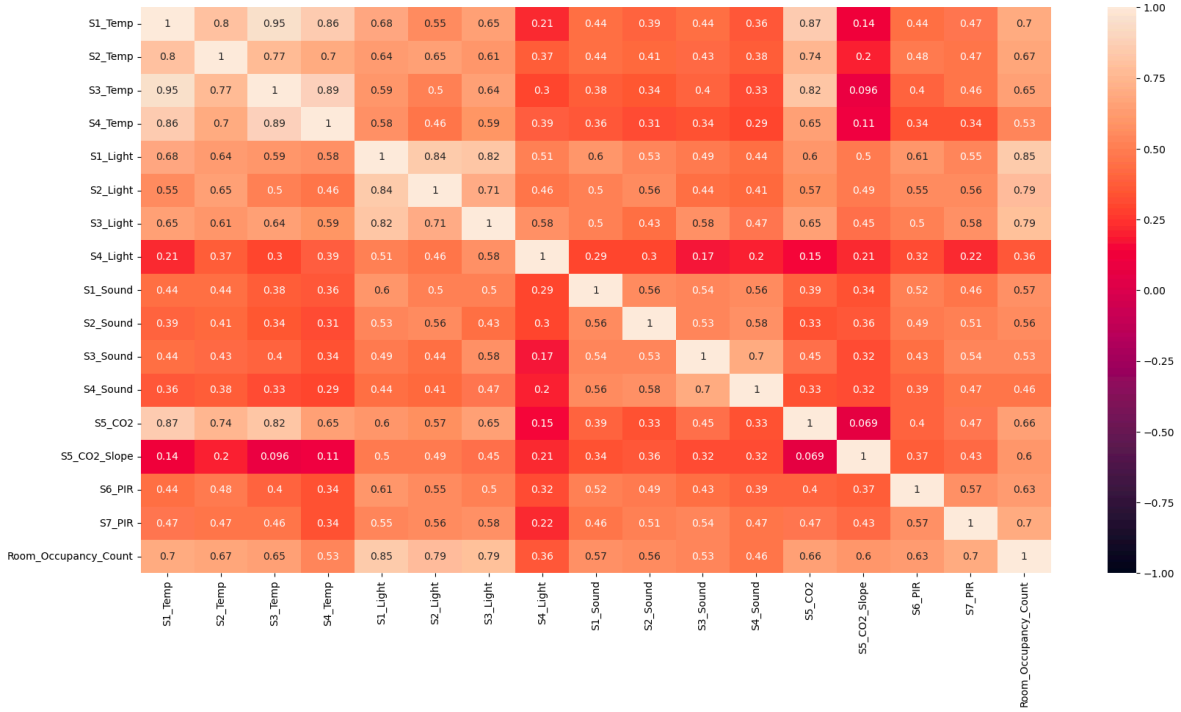


Tabella 3 Matrice delle Correlazioni

Quanto visto nella matrice delle correlazioni trova conferma nella successiva rappresentazione.

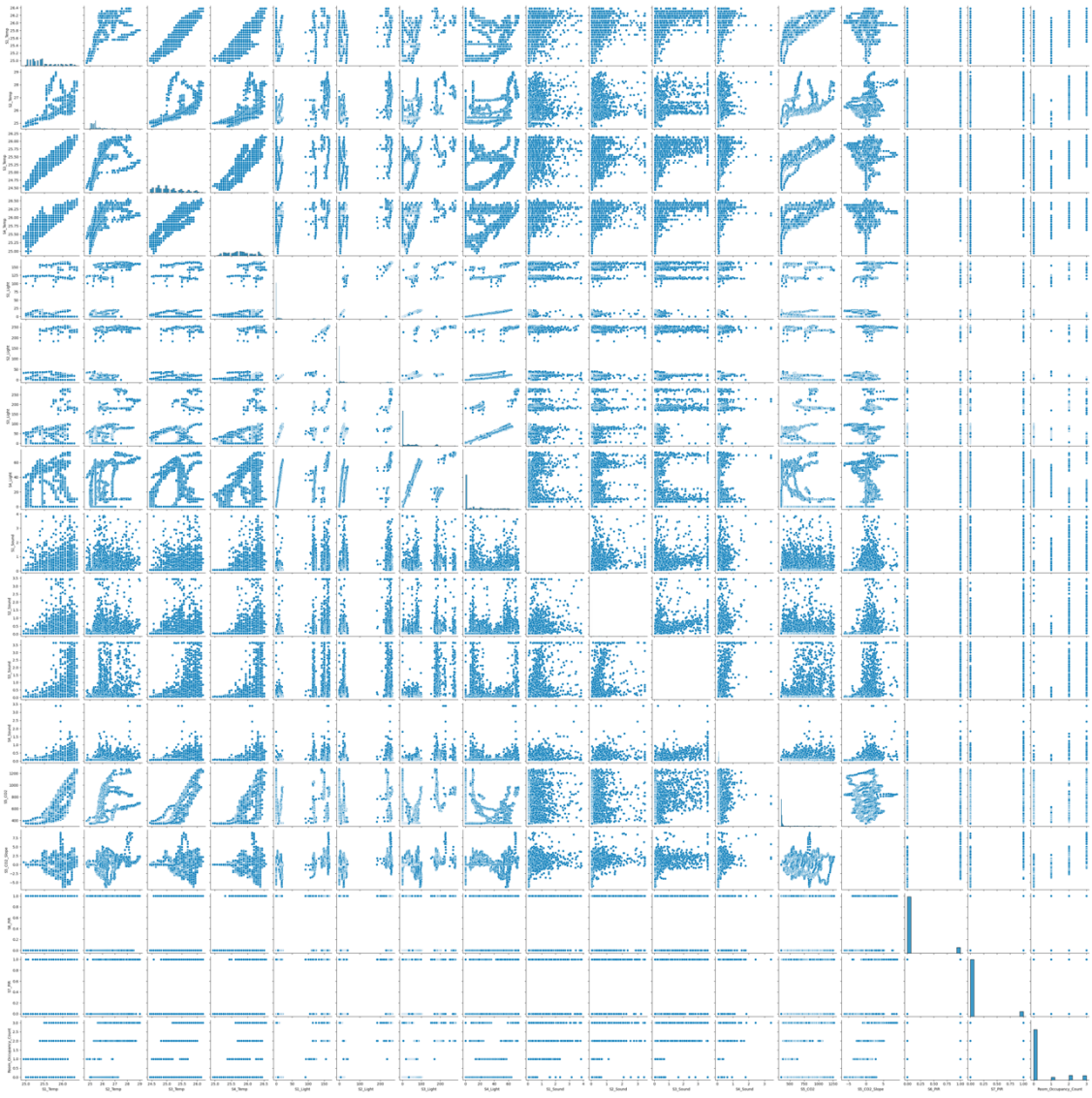


Figura 3 Relazione tra le Variabili

1.2 | Scelta delle variabili

Nel caso in esame la variabile *target* è rappresentata dal numero di occupanti la stanza, (Room_Occupancy_Count) che, come visto, varia tra 0 e 3 con una prevalenza di valori 0. Si assumono rilevanti tutte le variabili rilevate dai sensori, pertanto assumeremo come *features* tutte le variabili del dataset; dato il dataset si procede alla suddivisione in *X_features* e *y_target*.

```
y= df.iloc[:, -1]
print(y)
```

```
X=df.iloc[:,2:-1]
print(X)
```

Prima di passare all'implementazione dell'algoritmo è necessario trattare i dati in quanto l'algoritmo risulterà più robusto se le features sono normalizzate.

```
from sklearn.preprocessing import StandardScaler
```

```
  scaler = StandardScaler()
  scaled = scaler.fit_transform(X)
  print(scaled)
```

Eseguita la normalizzazione delle features è possibile effettuare lo split del dataset, in set di training e set di test, in percentuale rispettivamente di 80% e 20%.

```
from sklearn.model_selection import train_test_split
X=scaled

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.2, random_state = 10)
```

Da notare però che la suddivisione del dataset in *train_set* e *test_set*, ha indubbiamente alterato i dati poiché alcune osservazioni, magari rilevanti, risultano escluse dal set di addestramento, quindi si utilizzerà una *K-fold Cross Validation* a 10 fold, in questo modo aumenteremo la probabilità di includere tutti i dati nel set di addestramento.

2 | Implementazione SVM Classification

Nell'implementazione dell'algoritmo di Support Vector Machine Classification è importante limitare i problemi derivanti dall'overfitting, è necessario pertanto effettuare un ottimizzazione degli iperparametri attraverso le tecniche di regolarizzazione. Si utilizza la tecnica della Grid Search, partendo da un set di valori predeterminati e valutando la migliore combinazione, che offre le migliori prestazioni per le metriche di valutazione scelte, quali l'accuratezza.

2.1 | Definizione parametri SVM

Il primo kernel che si utilizza è quello lineare, più veloce e computazionalmente meno complesso del kernel RBF, si stabilisce quindi una griglia di partenza per il parametro C da 10^-3 a 10^5.

```
Dataset di partenza splittato :
X_features = np.array(scaled)
y_target = np.array(df.iloc[:, -1].to_numpy())

# Grid values for C
gridHyperPar = {"C": [0.001, 0.1, 1, 10, 100, 10000, 100000]}
```

Quindi si definisce una Kfold Cross-validation a 10 fold, definita "outerCV" poiché sarà applicata per la validazione del best_model ottenuto con la GridSearch.

```
# Configure the outer cross-validation class

outerCV = KFold(n_splits=10)
```

Si definiscono i vettori contenenti i valori di accuratezza dei due modelli.

```
outerScores_Linear = np.array([])
```

```
outerScores_RBF = np.array([])
```

Inizializzazione del best_model per il Kernel Lineare

```
# Linear Kernel
best_model_linear = None
best_score_linear = 0.0  # Initialize with a low value
```

Quindi si crea un ciclo *for* per la Kfold CrossValidation esterna al best_model ottenuto dall'ottimizzazione degli iperparametri

```
for trainOuter, testOuter in outerCV.split(X_features):
    current_X_Outer_TrainingSet = X_features[trainOuter]
    current_X_Outer_TestSet = X_features[testOuter]
    current_y_Outer_TrainingSet = y_target[trainOuter]
    current_y_Outer_TestSet = y_target[testOuter]
```

2.2 | Ottimizzazione degli iperparametri

Per la regolarizzazione degli iperparametri, come già anticipato, si utilizza la tecnica di Grid Search, quindi definito il modello da utilizzare (lineare in questo caso) si applica nuovamente una Kfold cross validation per ottimizzare la ricerca degli iperparametri.

```
    # Configure the inner cross-validation class

    innerCV = KFold(n_splits=10, shuffle=True, random_state=1)

    svmModel = SVC(kernel="linear")

    # The inner cross-validation is performed by the GridSearch function

    gridSearch = GridSearchCV(svmModel,
                              gridHyperPar,
                              scoring='accuracy',
                              cv=innerCV, refit=True)

    # Execute search

    searchResult = gridSearch.fit(current_X_Outer_TrainingSet,
                                  current_y_Outer_TrainingSet.ravel())
```

2.3 | Applicazione del best_model

Ottenuti i parametri per il *best_model* lineare è possibile utilizzarli per effettuare la prediction del modello e stimarne la bontà attraverso il calcolo dell'accuratezza.

```
#  Given  that  refit=True,  gridSearch.fit  returns  the  best  hyperparameter  over  all
current_X_Outer_TrainingSet

    best_model = searchResult.best_estimator_

    bestHyperParameterC = gridSearch.best_params_['C']

    # We can now evaluate the current model on the hold-out dataset current_X_Outer_TestSet
```

```
    current_y_Predictions = best_model.predict(current_X_Outer_TestSet)

    # Compute accuracy

    currentScore = accuracy_score(current_y_Outer_TestSet,
                                  current_y_Predictions)
```

2.4 | Visualizzazione dei risultati

Per ogni fold esterno ottenuto si visualizzano quindi i *best value* di: accuratezza e di regolazione degli iperparametri.

```
# Append to array of scores

    outerScores_Linear = np.append(outerScores_Linear, currentScore)

    # Check if this model has the highest accuracy so far
    if currentScore > best_score_linear:
        best_score_linear = currentScore
        best_model_linear = best_model

    # Report progress

    print('Accuracy (Linear Kernel) = %.3f, Best Hyper Parameter C = %.3f' %
(currentScore, bestHyperParameterC))
```

I risultati ottenuti per il kernel Lineare sono i seguenti :

```
Accuracy (Linear Kernel) = 0.770, Best Hyper Parameter C = 10.000
Accuracy (Linear Kernel) = 1.000, Best Hyper Parameter C = 1.000
Accuracy (Linear Kernel) = 0.998, Best Hyper Parameter C = 1.000
Accuracy (Linear Kernel) = 0.976, Best Hyper Parameter C = 10.000
Accuracy (Linear Kernel) = 1.000, Best Hyper Parameter C = 10.000
Accuracy (Linear Kernel) = 1.000, Best Hyper Parameter C = 1.000
Accuracy (Linear Kernel) = 1.000, Best Hyper Parameter C = 10.000
Accuracy (Linear Kernel) = 1.000, Best Hyper Parameter C = 10.000
Accuracy (Linear Kernel) = 0.942, Best Hyper Parameter C = 1.000
Accuracy (Linear Kernel) = 1.000, Best Hyper Parameter C = 1.000
```

```
Best Model (Linear Kernel):
SVC(C=1, kernel='linear')
```

2.5 | Kernel RBF

Il Kernel RBF differisce da quello lineare poiché aumenta la dimensione delle variabili, rendendo linearmente separabili anche quelle che non lo erano. Quindi introduciamo un ulteriore fattore tra gli iperparametri “gamma” e definiamo la nuova griglia.

```
# RBF Kernel
gridHyperPar = {"C": [0.001, 0.1, 1, 10, 100, 10000, 50000, 100000],
                "gamma": [0.001, 0.01, 0.1]}
```

Mentre gli attributi del modello “RBF” sono il fattore di confronto di tipo One vs Rest ed il bilanciamento attivo delle classi, necessario in casi di sbilanciamento tra le classi.

```
  svmModel = SVC(kernel="rbf",
                  decision_function_shape='ovr',
                  class_weight='balanced')
```

I risultati ottenuti sono i seguenti :

```
Accuracy (RBF Kernel) = 0.746, Best Hyper Parameter C = 10000.000, Best Hyper
Parameter Gamma = 0.001
Accuracy (RBF Kernel) = 1.000, Best Hyper Parameter C = 100.000, Best Hyper Parameter
Gamma = 0.001
Accuracy (RBF Kernel) = 1.000, Best Hyper Parameter C = 10.000, Best Hyper Parameter
Gamma = 0.010
Accuracy (RBF Kernel) = 0.963, Best Hyper Parameter C = 100.000, Best Hyper Parameter
Gamma = 0.001
Accuracy (RBF Kernel) = 1.000, Best Hyper Parameter C = 100.000, Best Hyper Parameter
Gamma = 0.001
Accuracy (RBF Kernel) = 1.000, Best Hyper Parameter C = 10.000, Best Hyper Parameter
Gamma = 0.010
Accuracy (RBF Kernel) = 1.000, Best Hyper Parameter C = 100.000, Best Hyper Parameter
Gamma = 0.001
Accuracy (RBF Kernel) = 1.000, Best Hyper Parameter C = 100.000, Best Hyper Parameter
Gamma = 0.010
Accuracy (RBF Kernel) = 0.945, Best Hyper Parameter C = 100.000, Best Hyper Parameter
Gamma = 0.010
Accuracy (RBF Kernel) = 1.000, Best Hyper Parameter C = 100.000, Best Hyper Parameter
Gamma = 0.001
```

```
Best Model (RBF Kernel): SVC(C=100, class_weight='balanced', gamma=0.001)
```

Si nota che il Kernel SBF risulta mediamente più stabile, e presenta valori di accuratezza più alti.

3 | Confusion Matrix

Ottenuti i parametri dei best_model per i Kernel Lineare e RBF analizziamo anche la matrice di confusione per confrontare i risultati ottenuti.

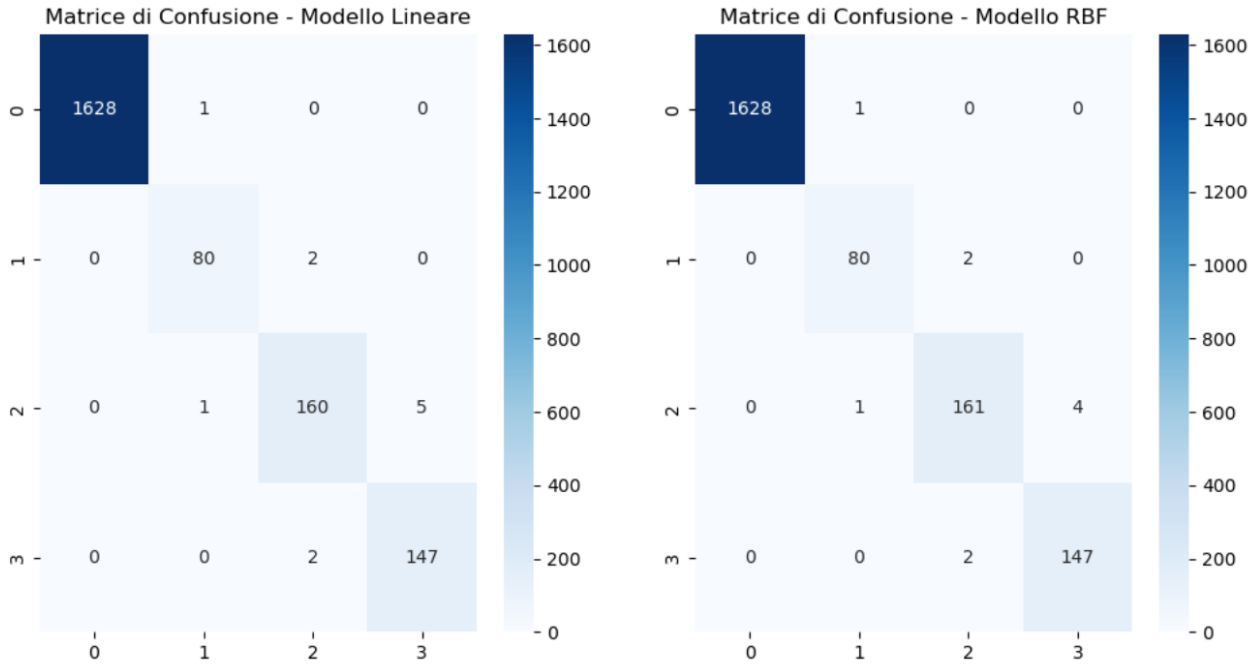


Figura 4 Confusion Matrix Kernel Lineare Vs RBF

Si può notare che entrambi i modelli ottengono buone performance di classificazione: in ascissa sono indicati i valori predetti dal modello, mentre l'ordinata rappresenta il valore effettivo, pertanto l'unica differenza che si nota, in favore del Kernel RBF, è nella classe 2 dove riporta un errore di previsione inferiore.