Heart Attack

Autori: Patrizia Conte(MT:676877), Domenico Cascella(MT:684735), Mariangela Giannini(MT:680636)

Introduzione

Il progetto mira a prevedere la possibilità che un soggetto possa avere un infarto del miocardio. Per fare ciò abbiamo adottato una serie di tecniche sia di apprendimento supervisionato che apprendimento non supervisionato. In più abbiamo creato un grafo per ogni provincia in modo tale da poter effettuare una ricerca su tale grafo per raggiungere il presidio ospedaliero più vicino sul territorio Pugliese della provincia data la posizione in cui il soggetto si trova.

Abbiamo utilizzato dataset diversi

Per quanto riguarda le tecniche di apprendimento supervisionato e non supervisionato abbiamo usato un dataset con queste features

1. Età (age)
2. Sesso (sex) :
    - 1 maschio
    - 0 Femmina
3. Tipo dolore toracico(cp) :
    - 1 Angina Tipica,
    - 2 Angina Atipica,
    - 3 Dolore non Anginoso
    - 4 Asintomatico
4. Pressione Sanguigna a riposo(trtbps)
5. Colesterolo (chol)
6. Elettrocardiogramma a riposo (restecg)
7. Frequenza cardiaca massima raggiunta (thalachh)
8. Depressione del tratto St alla Frequenza cardiaca (slp)

    Con valori :
    - 1 in salita
    - 2 Piatto
    - 3 Discesa
9. Numero di vasi principali (caa)

    Con valori (0-3)
10. Target (output)

    Con valori:
    - 1 Probabilità alta di avere un infarto
    - 2 Probabilità bassa di avere un infarto

Fonte dataset : https://www.kaggle.com/rashikrahmanpritom/heart-attack-analysis-prediction-dataset

Invece per la costruzione dei grafi di ricerca abbiamo usato due dataset con le stesse feature:

1 Comune

2 Provincia

Fonte dataset Comune: https://www.istat.it/it/archivio/6789

Fonte dataset Ospedale: http://www.datiopen.it/it/catalogoopendata/ospedali/?h=search0&search1&search2

Per l' Apprendimento Supervisionato sono stati prodotti anche diversi grafici:

• ROC Curve

• Precision-Recall Curve

• Matrice di Confusione

Algoritmo Rete Neurale (Apprendimento Supervisionato)

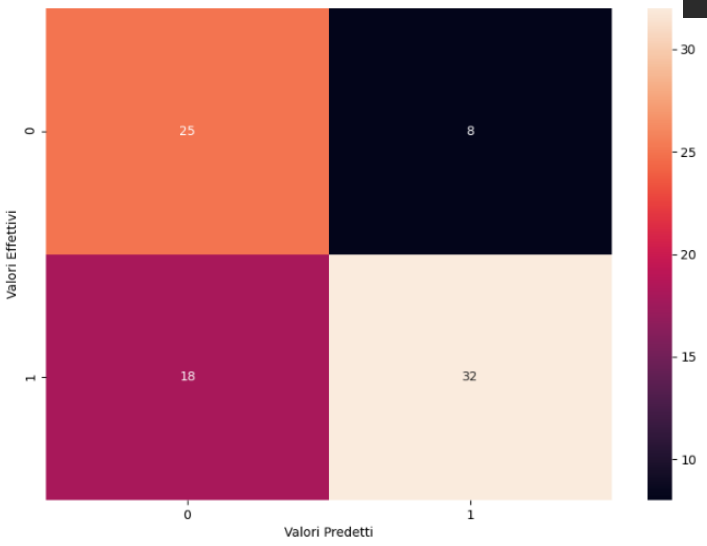
Creazione Rete Neurale

Abbiamo definito il modello della rete neurale passandogli in input il valore 9 che corrisponde al numero delle nostre feature , successivamente definiamo la densità della nostra rete passandogli un valore pari a 25 (valore scelto forfettariamente, dopo aver eseguito diverse prove) e la funzione di attivazione Sigmoidee, abbiamo scelto lei perché facendo delle opportune verifiche ci è apparsa quella più opportuna. Per importare il dataset da un file csv, abbiamo usato la libreria pandas. Dopo di che questo dataset è stato diviso in training set e test set . Successivamente viene creato un modello e con la funzione fit abbiamo potuto addestrare la nostra rete neurale , passandogli le feature di input e di target. Attraverso lo scores valutiamo il test, procediamo con il calcolo della predizione sul test , testando così la nostra rete neurale. Il valore della accuratezza ci dirà quanto la rete neurale sia accurata.

Come anticipato nell'introduzione per Apprendimento Supervisionato abbiamo usato diverse misure

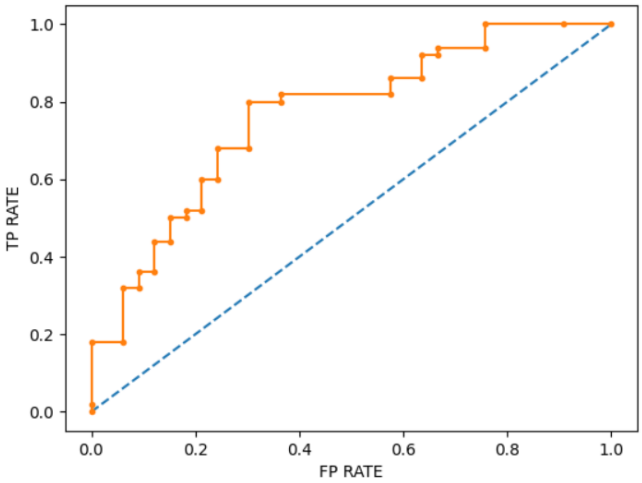
```
#creazione struttura rete neurale
def create_model():
    network = Sequential(np.shape(9, ))
    network.add(layers.Dense(25, activation='sigmoid'))
    network.add(layers.Dense(1, activation='sigmoid'))
    # Compile model
    network.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    return network

np.random.seed(7)
dataset = pd.read_csv("heart2.csv", sep=";",
                      names=["age","sex","cp","trtbps","chol","restecg","thalachh","slp","caa","output"],
                      dtype={'age': object, 'sex': object, 'cp': object,
                             'trtbps': object, 'chol': object,'restecg': object, 'thalachh':object,
                             'slp': object, 'caa': object, 'output':object})
network_data = pd.get_dummies(dataset, columns=["age","sex","cp","trtbps","chol","restecg","thalachh","slp","caa"])

network_data = network_data.drop(["output"], axis=1)

X = network_data
y = pd.get_dummies(dataset["output"], columns=["output"])
y = y["1"]


X, y = sm.fit_resample(X, y.ravel())
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=13)

model = create_model()
# Fit the model
model.fit(X_train, y_train, epochs=100, batch_size=10)

scores = model.evaluate(X_test, y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1] * 100))
```

```
# calcolo della predizione
predictions = model.predict(X_test)
rounded = [round(x[0]) for x in predictions]

accuracy = accuracy_score(y_test, rounded)

print('\nClasification report:\n', classification_report(y_test, rounded))
print('\nConfussion matrix:\n', confusion_matrix(y_test, rounded))
```



Da questa Matrice di confusione si evince che 57 feature (25+32) sono state predette correttamente. Ed invece 26 (18+8) feature non sono state predette correttamente

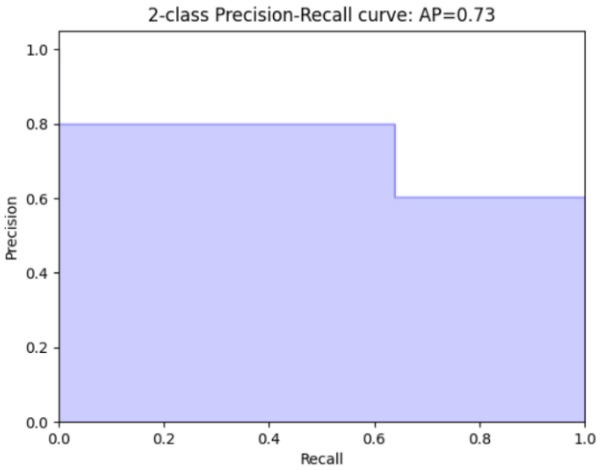
Nella rete neurale la Roc Curve ha AUC pari a 0.761



La Precision-Recall curve ci dice che un'average precision pari a 0.73,

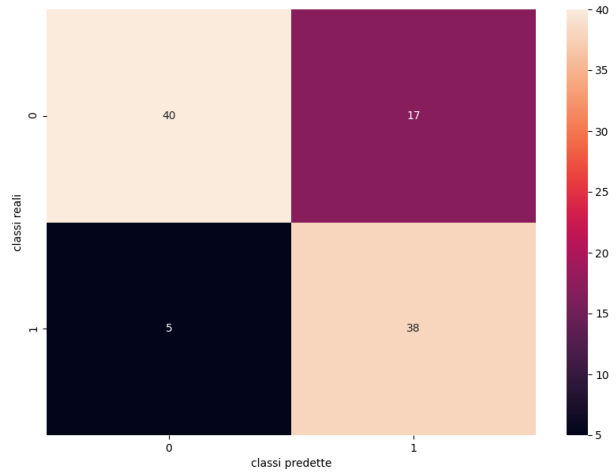
Ha un'accuratezza del 69% in linea con le altre

tecniche trattate e una f1-score di 0.71.



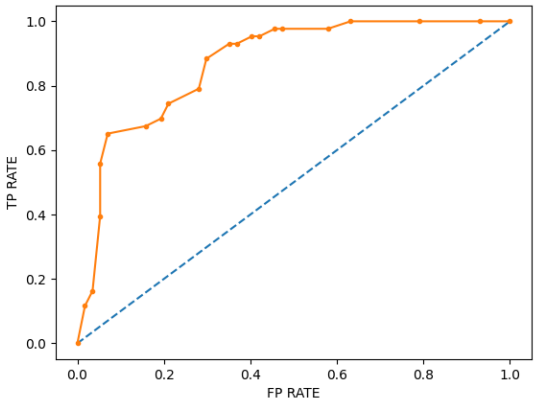
Algoritmo Random Forest (Apprendimento Supervisionato)

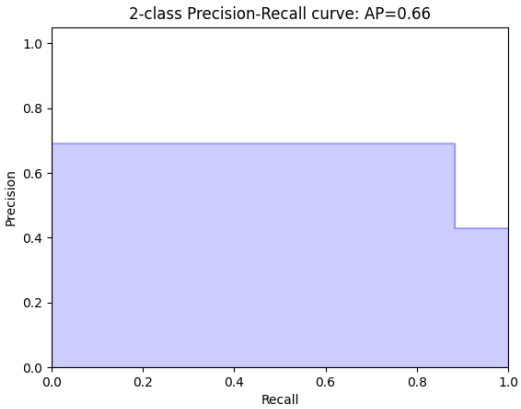
Abbiamo caricato il nostro training set in formato .csv così da acquisire il dataset. Per bilanciare il dataset, abbiamo usato la funzione SMOTE così da evitare l'overfitting. Il risultato di tale funzione è stato usato per addestrare il nostro modello. Fatto ciò, abbiamo diviso il nostro dataset in esempi di training (divisi in feature di input X1_train e y1_train per le feature di output) e test set(divisi in feature X1_test e y1_test per le feature di output da predire). Fatto ciò, abbiamo configurato il nostro modello ad usare 20 alberi e lo addestriamo con i nostri esempi di training. Dopo di che usiamo y1_test e X1_test per fare le predizioni. La matrice di confusione sotto riportata , ci dice il numero di esempi classificati correttamente e il numero di esempi classificati non correttamente:



In tal caso il numero di veri positivi è 38 e il numero di veri negativi è 40 e quindi il maggior numero di valori da predire, è stato predetto correttamente.

Oltre a tale grafico, possiamo misurare le prestazioni del nostro modello tramite la ROC curve e la curve di Precision-Recall:

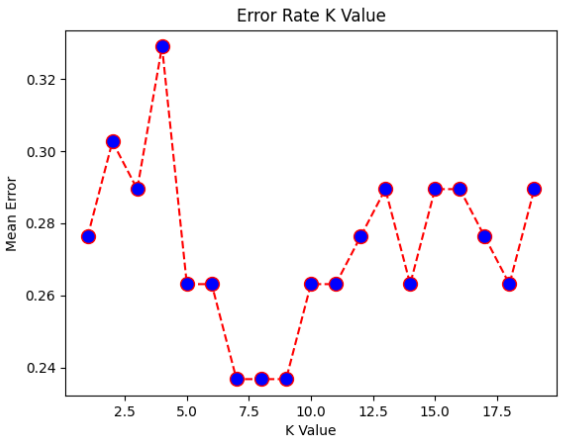




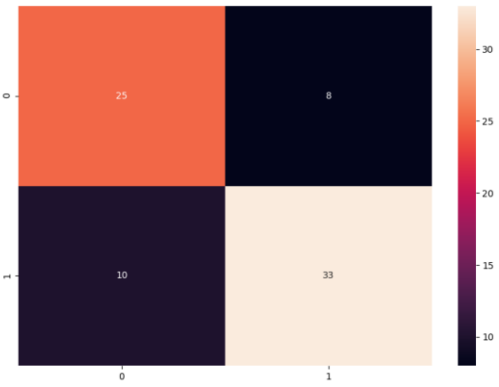
L'area al disotto della curva ROC , è pari a 0.636. Mentre l'area al disotto della curva di precision-recall , che non è altro che l'average precision , è pari a 0.66. L'accuratezza è del 77%

KNN (Apprendimento Supervisionato)

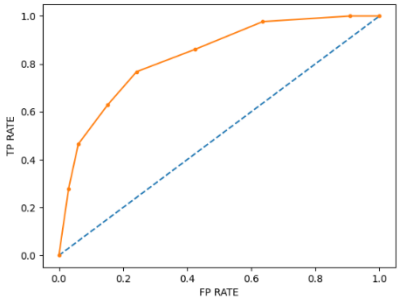
È un modello che consiste nel calcolare la distanza (di solito quella euclidea) tra i primi K esempi più vicini a quello da classificare. Attraverso il fit addestriamo il nostro modello in un range di valori che va da 1 a 20(range scelto in modo arbitrario) e in un vettore salviamo l'errore ossia la differenza tra il valor predetto e quello effettivo. Con l' utilizzo di tale vettore costruiamo un grafico ossia il Error Rate K Value(matplotlib):

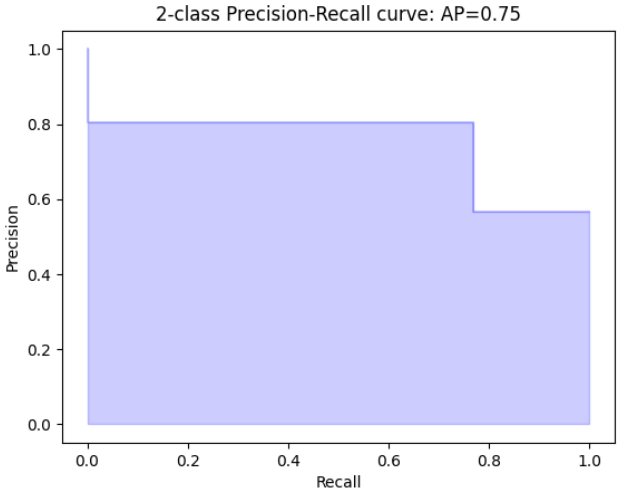


Tale grafico ci mostra come l'errore cambia in base al numero dei K vicini presi in considerazione. Dal grafico possiamo vedere che con K=7 abbiamo un errore minore.



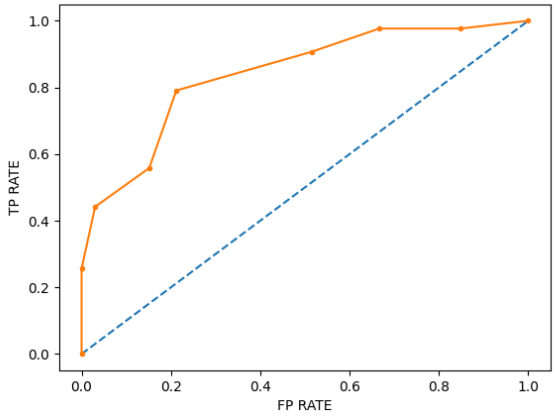
Dalla matrice di confusione abbiamo 58 esempi classificati nel modo corretto, 18 esempi invece sono stati classificati in modo sbagliato.

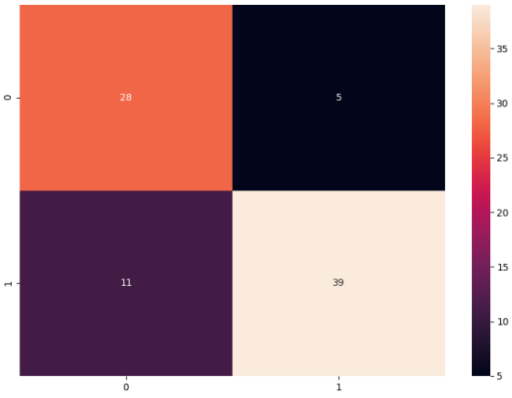




L'area sotto la curva di ROC è 0.832 Tale modello ha un'accuratezza del 76%

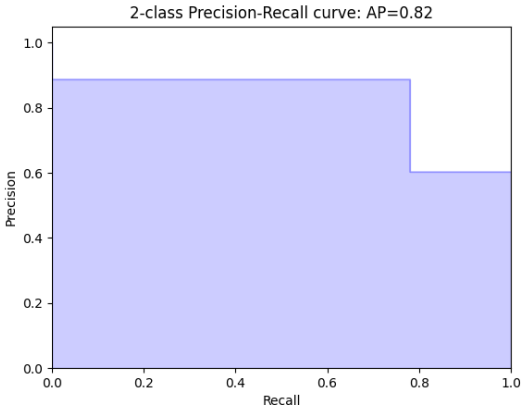
Applicando lo SMOTE abbiamo i seguenti risultati:





Dalla curva di ROC possiamo vedere che l'area è di 0.839.

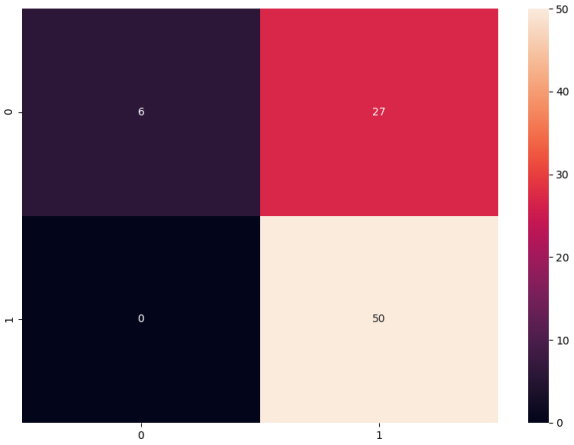
Nella matrice di confusione possiamo vedere che 67 esempi sono stati classificati nel modo corretto, 16 esempi però non sono stati classificati correttamente.



Per la curva di Precision-Recall abbiamo l'Avarage Precision pari a 0.82 e una accuratezza pari a 0.76

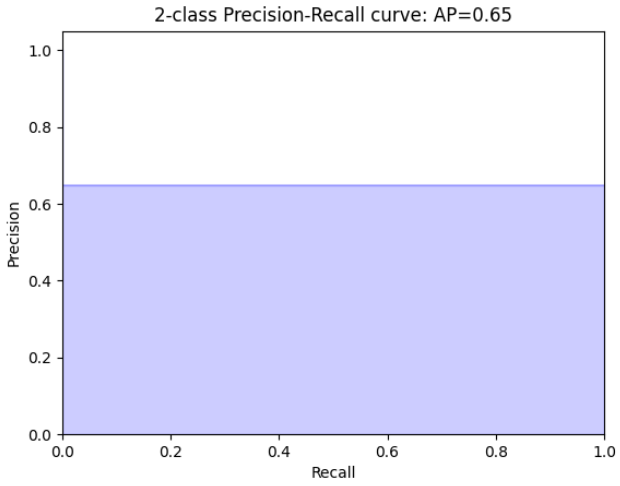
Algoritmo SVM(support-vector machines)

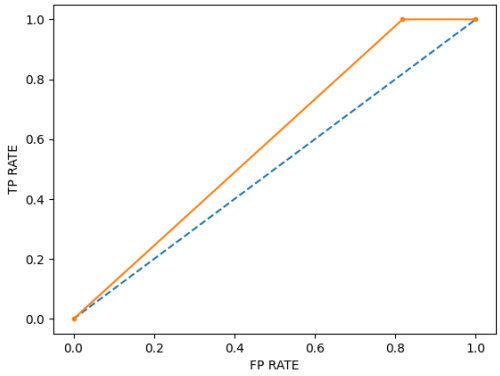
Un altro algoritmo utilizzato è il SVM, tale algoritmo sa trovare l'iperpiano che massimizza il margine che separa opportunamente dalla superficie che ci fa prendere una decisione su cui al di sopra classifichiamo gli esempi come una certa categoria, al di sotto come la categoria opposta. In questo modello abbiamo usato l'algoritmo SMOTE per bilanciare i dati del dataset.



Da tale matrice di confusione possiamo verificare che 56 esempi sono stati classificati correttamente invece 27 sono stati classificati come falsi positivi.

Il grafico di precision-recall mostra un buon andamento della precisione al variare del recall con un grado di average precision pari a 0.65 e una accuracy pari a 0.70



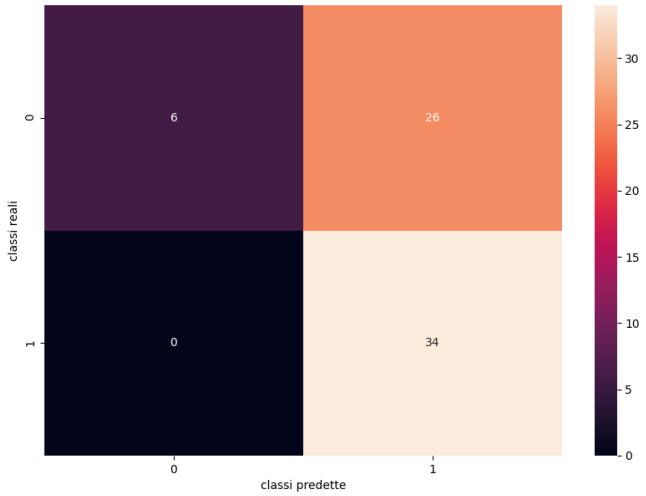


L'area sotto la curva di ROC è pari a 0.591. L'accuratezza di tale modello è del 67%

Multinomial Naive Bayes

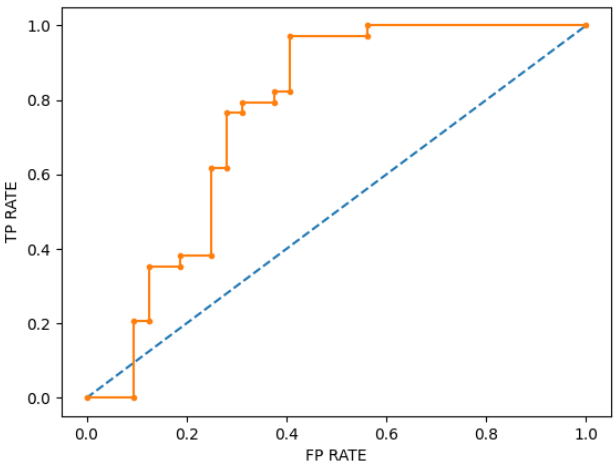
Con un modello di eventi multinomiali, gli esempi (vettori di feature) rappresentano le frequenze con cui certi eventi sono stati generati da una distribuzione polinomiale (p1, ⋯, pn) dove pi è la probabilità che l'evento i si verifichi.

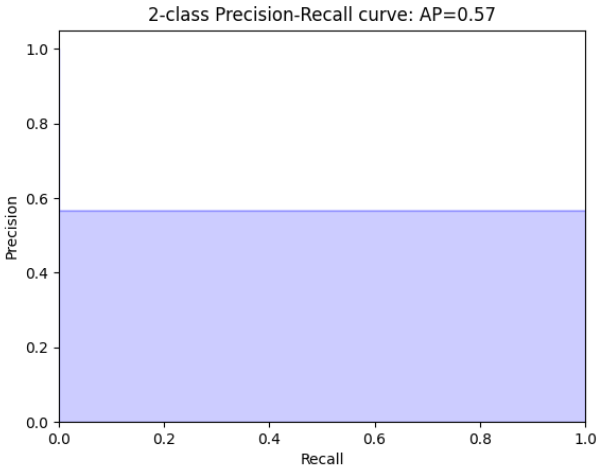
Tale modello nel nostro caso non è stato molto performante infatti dai seguenti grafici possiamo notare:



Nella matrice di confusione possiamo vedere che il modello è riuscito a classificare nel modo giusto 40 esempi, ha però classificato come falsi positivi 26 esempi.

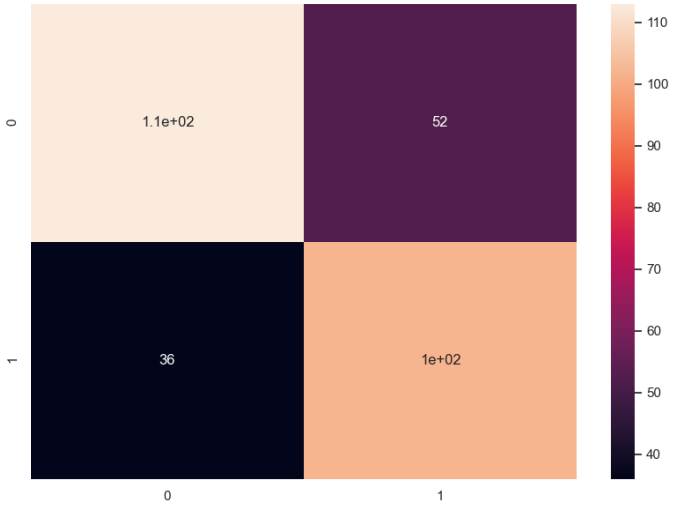
L’area sotto la curva di ROC è pari a 0.760



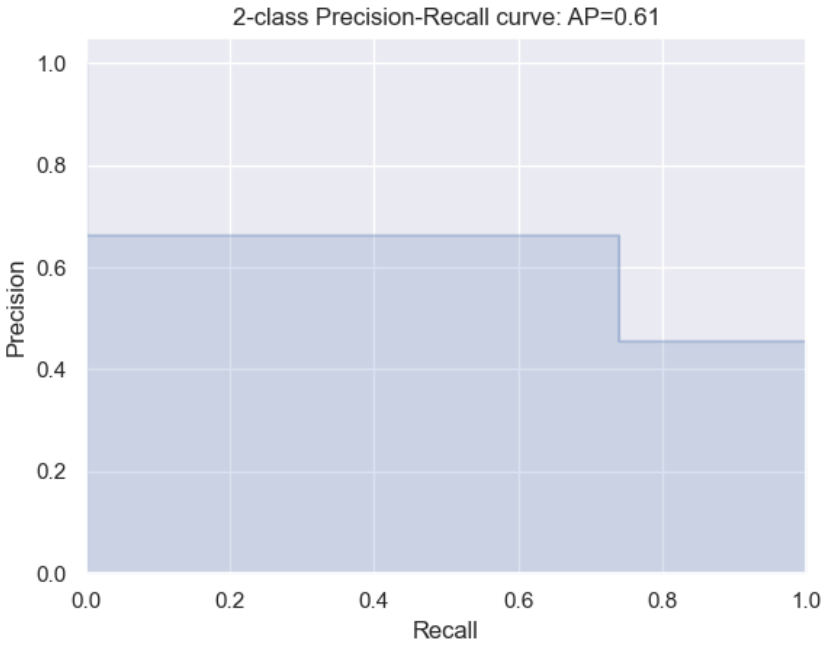


Nonostante l’uso dell’algoritmo SMOTE, l’average-precision è pari a 0.57. L’accuratezza è del 59%

KMeans (Apprendimento non supervisionato)



Dalla matrice di confusione possiamo vedere che 210 esempi sono stati classificati correttamente, invece 88 esempi non sono stati classificati nel modo corretto.



Dal grafico si evince che l'area sotto la curva è di 0.61, l'accuratezza è del 71%

CONCLUSIONE

In conclusione prendendo in considerazione gli algoritmi supervisionati e osservando i risultati ottenuti con i relativi grafici e accuratezza ci siamo resi conto che il miglior modello è il Random Forest con un'accuratezza del 77%, al secondo posto abbiamo il KNN con il 76%. Il modello che ci ha dato il risultato peggiore è stato il Multinomial Naive Bayes con un'accuratezza del 59%.

Ricerca nel grafo

La seconda parte del nostro progetto riguarda il calcolo del percorso partendo da un paese della regione Puglia per arrivare in una città in cui è presente un ospedale. La ricerca inizia chiedendo all'utente la provincia in cui risiede e procede chiedendo il comune di partenza, il comune di arrivo e l'algoritmo da utilizzare. Come esempio abbiamo selezionato la provincia di Foggia, con comune di partenza Peschici, come comune di arrivo San Giovanni Rotondo e eseguiamo i diversi algoritmi.

Come primo algoritmo abbiamo usato il DLS e come percorso il nostro algoritmo restituisce:

['Peschici', 'Vieste', 'Mattinata', 'Monte Sant Angelo', 'Vieste', 'Mattinata', 'Monte Sant Angelo', 'Vico del Gargano', 'Rodi Garganico', 'Vico del Gargano', 'Monte Sant Angelo', 'Vieste', 'Monte Sant Angelo', 'San Giovanni Rotondo']

```
FG:Foggia, BA: Bari, BT: Barletta-Andria-Trani
LE:Lecce, TA:Taranto, BR:Brindisi
Inserisci provincia(Sigla):
FG
I comuni da cui puoi partire sono: dict_keys(['Accadia', 'Alberona', 'Anzano di Puglia', 'Apricena', 'Ascoli Satriano', 'Biccari', 'Bovino', 'Cagnano Varano', 'Candela', 'Carapelle
Inserisci comune di partenza:
Peschici
Gli ospedali di provincia sono: {'San Giovanni Rotondo', 'San Severo', 'Cerignola', 'Foggia', 'Manfredonia'}
Inserisci comune di arrivo:
San Giovanni Rotondo
Quale algorimto vuoi usare?:
1)DLS
2)UCS
3)DFS
4)BFS
1
['Peschici', 'Vieste', 'Mattinata', 'Monte Sant Angelo', 'Vieste', 'Mattinata', 'Monte Sant Angelo', 'Vico del Gargano', 'Rodi Garganico', 'Vico del Gargano', 'Monte Sant Angelo',
Vuoi Ricominciare?(si/no):
si
```

Il secondo algoritmo (UCS) restituisce il seguente percorso:

```
Quale algorimto vuoi usare?:
1)DLS
2)UCS
3)DFS
4)BFS
2
['Peschici', 'Vico del Gargano', 'Carpino', 'Cagnano Varano', 'San Marco in Lamis', 'San Giovanni Rotondo']
Vuoi Ricominciare?(si/no):
si
```

Il terzo algoritmo (DFS) restituisce il seguente percorso: ['Peschici', 'Vieste', 'Mattinata', 'Monte Sant Angelo',⋯,'San Giovanni Rotondo']

```
Quale algorimto vuoi usare?:
1)DLS
2)UCS
3)DFS
4)BFS
3
['Peschici', 'Vieste', 'Mattinata', 'Monte Sant Angelo', 'Mattinata', 'Vieste', 'Vico del Gargano', 'Rodi Garganico', 'Vico del Gargano', 'Rodi Garganico', 'Ischitella', 'Cagnano
Vuoi Ricominciare?(si/no):
```

Il quarto algoritmo(BFS) restituisce il seguente percorso:

```
San Giovanni Rotondo
Quale algorimto vuoi usare?:
1)DLS
2)UCS
3)DFS
4)BFS
4
['Peschici', 'Vico del Gargano', 'Monte Sant Angelo', 'San Giovanni Rotondo']
Vuoi Ricominciare?(si/no):
```

Mettendo a confronto i diversi algoritmi possiamo notare che per il percorso Peschici-San Giovanni Rotondo, si deduce che l'algoritmo più efficiente è il BFS poiché restituisce un numero nodi visitati molto più limitato rispetto agli altri.