UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO | DIPARTIMENTO DI INFORMATICA

Documentazione Caso di Studio
Ignegneria della Conoscenza
A.A. 2023/24

Disease Prediction

Gruppo di lavoro

• Antonio Caccioppola, 679178, a.caccaccioppola@studenti.uniba.it

Repository GitHub

• https://github.com/antonioruri/Disease-prediction-iconproject

INTRODUZIONE

Il progetto si è incentrato sulla creazione di un'applicazione innovativa finalizzata alla predizione accurata delle malattie, utilizzando informazioni relative a vari sintomi. Per raggiungere questo obbiettivo sono stati impegnati algoritmi di machine learning basati sull'apprendimento supervisionato.

Oltre alla funzione predittiva principale, l'applicativo offre un insieme di caratteristiche per migliorare l'esperienza utente, tra cui:

- **Mappa Interattiva dei Servizi Medici**: L'applicativo consente agli utenti di interagire con le posizisioni di medici e ospedali nelle vicinanze, calcolando la distanza tra un punto e l'altro.
- **Orari apertura degli studi medici**: Gli utenti possono consultare gli orari di apertura degli studi medici in periodi specifici.
- **Informazioni sulle malattie**: E' possibile ottenere informasizoni sulle malattie associate a un sintomo durante un intervallo di tempo.

Il cuore dell'applicativo è rappresentato da un dataset proveniente da Kaggle. Questo insieme di dati è stato fondamentale per addestrare e validare i modelli di machine learning, garantendo una predizione affidabile delle malattie.

DATASET

Nel processo di preparazione del " **disease-symptom-description-dataset** ", sono state eseguite diverse operazioni di preelaborazione per garantire l'affidabilità e la coerenza dei dati. Il set di dati, disponibile qui, è composto da quattro file principali:

- **dataset**: contiene le malattie come etichette e i sintomi espressi sotto forma di stringa come caratteristiche. Il numero di etichette univoche è 41, quindi sono disponibili dati per 41 malattie diverse. Il numero di sintomi unici è 133.
  Il dataset è composto da 18 colonne, di cui 1 contiene la malattia e le restanti 17 i vari sintomi che una malattia può avere. Pertanto una malattia può avere un massimo di 17 sintomi.

- **Symptom-severity:** file contenente una descrizione per ciascuna malattia. Nell'applicativo è stato rinominato in "disease_Description" così da rappresentare le descrizioni delle malattie.

- **Symptom-precaution**: file contenente le precauzioni possibili per ciascuna malattia. Nell'applicativo è stato rinominato in "disease_precaution" così da rappresentare le precauzioni delle malattie. Per ogni malattia, sono possibili 4 precauzioni.
- **Symptom-Description:** file contenente una descrizione per ciascuna malattia. Nell'applicativo è stato rinominato in "disease_Description" così da rappresentare le descrizioni delle malattie.

- **Symptom-precaution:** file contenente le precauzioni possibili per ciascuna malattia. Nell'applicativo è stato rinominato in "disease_precaution " così da rappresentare le precauzioni delle malattie. Per ogni malattia, sono possibili 4 precauzioni.cs

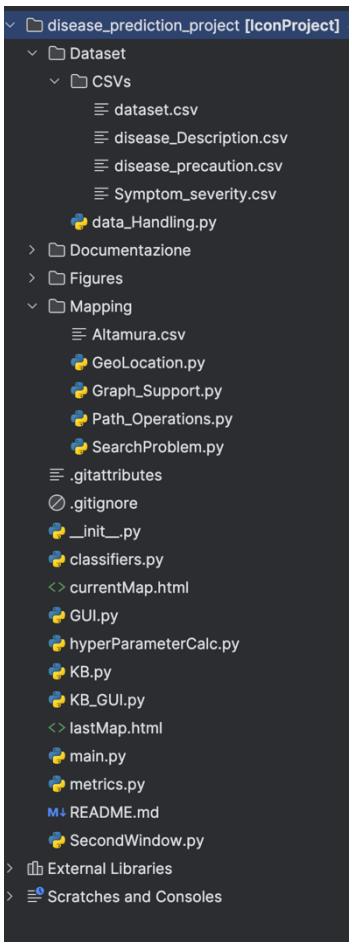
RIELABORAZIONE DATASET:

Prima di utilizzare i classificatori, il set di dati è stato sottoposto a operazioni preelaborazione:

- **Shuffling dei dati**: Una serie di permutazioni casuali sono state applicate all'array di dati utilizzando un random_state pari a 35, garantendo la casualità nelle analisi successive.
- **Trasformazione di valori e spazi mancanti**: Tutti i valori "Nan" sono stati convertiti in 0. Gli spazi sono stati sostituiti con caratteri di sottolineatura ('_'). Il set di dati è stato quindi convertito in un DataFrame per una gestione più efficiente con la libreria 'pandas' di Python.
- **Codifica della gravità dei sintomi**: La gravità di ciascun sintomo è stata codificata, sostituendo i sintomi con la rispettiva gravità per ciascuna malattia segnalata.
- **Gestione dei sintomi senza grado**: I sintomi non classificati, rappresentati come "Nan", sono stati sostituiti con 0, indicando nessuna gravità in quel contesto per la malattia.

Dopo queste rielaborazioni, il dataset è stato suddiviso in caratteristiche (sintomi) ed etichette (malattie), preparando così i dati per il training e la validazione di modelli di machine learning.

STRUTTURA PROGETTO



Il progetto è strutturato in diversi moduli, ciascuno dedicato a specifiche funzionalità dell'applicativo.

Nel modulo "**classifiers**", sono implementate le funzioni per l'addestramento dei vari classificatori utilizzati.
Il modulo "**main**" gestisce la creazione delle variabili che contengono i riferimenti ai classificatori addestrati, consentendo il loro utilizzo. Include anche il metodo per effettuare la predizione di una malattia dati i sintomi.
Il modulo "**hyperParameterCalc**" contiene i metodi per la ricerca degli iperparametri dei classificatori.
Nel modulo "**metrics**" sono presenti due funzioni per rappresentare le prestazioni della predizione di un classificatore. Un metodo è utilizzato per confrontare i vari classificatori, generando un grafico comparativo.
Il modulo "**KB**" descrive la Knowledge Base implementata e offre metodi per interagire con essa.
Nel package "Dataset" sono inclusi i file CSV del dataset e il modulo "**data_Handling**", che contiene le funzioni di interazione e rielaborazione del dataset.
Il package "**Mapping**" comprende il file CSV che rappresenta le diverse locazioni nella città di Altamura. Inoltre, sono presenti moduli come "**GeoLocation**"," che rappresenta l'oggetto Position, e "**Graph_Support**", classi utili per la creazione del grafo orientato (Nodo, Arco e Percorso). Il modulo "**SearchProblem**" rappresenta un problema di ricerca, mentre "**Path_Operations**" contiene metodi per la creazione della mappa, la

visualizzazione e la determinazione del percorso migliore tra due posizioni.
Sono presenti anche moduli relativi all'interfaccia grafica, come "**GUI**", che è la finestra principale dell'applicazione, "**SecondWindow**", dedicato al recupero del percorso verso medici e ospedali, e infine "**KB_GUI**", che gestisce l'interazione con la Knowledge Base.

APPRENDIMENTO SUPERVISIONATO

Al fine di realizzare un'applicativo in grado di predire una malattia sulla base di determinati sintomi, sono stati impiegati modelli di apprendimento supervisionato. Per ottimizzare le prestazioni di ciascun classificatore, è stata condotta una ricerca degli iperparametri migliori.

Strumenti Utilizzati:
Tutti gli algoritmi e i relativi metodi interattivi sono stati estratti dalla libreria SkLearn. Sono stati importati i vari moduli riguardanti i diversi classificatori. In particolare, sono stati adottati i seguenti:

Classificatore K-Neighbors ("KNeighborsClassifier")
Classificatore Naive Bayesiano ("GaussianNB")
Classificatore Decision Tree ("DecisionTreeClassifier")
Classificatore Random Forest ("RandomForestClassifier")
Classificatore SVM, in particolare l'SVC ("SVC")
Classificatore di Regressione Logistica, quindi Logistic Regression ("LogisticRegression")

Per la ricerca degli iperparametri ottimali di ciascun classificatore, è stata impiegata la tecnica della GridSearchCV, che consiste in una ricerca esaustiva su insiemi di parametri di uno stimatore.

Per standardizzare le features del dataset, è stato utilizzato lo "StandardScaler.

DECISIONI DI PROGETTO

Il dataset è stato suddiviso in 4 array ovvero `X_train, X_test, Y_train, Y_test` con una divisione del 65% per il training set e del 35% per il test set. Questa divisione sarà poi spiegata nella prossima sezione. Gli array sopra riportati sono dunque:

- `X_train:` sono contenute le feature di input utilizzate durante la fase di addestramento dei classificatori
- `X_test:` sono contenute le feature di input utilizzate durante la fase di testing dei classificatori
- `Y_train:` sono contenute le feature target utilizzate durante la fase di addestramento dei classificatori
- `Y_test`: sono contenute le feature target utilizzate durante la fase di testing dei classificatori

```
X_train, X_test, Y_train, Y_test = train_test_split( *arrays: data, labels, train_size = 0.65, random_state = 35)
```

Inoltre è stata anche effettuata la standardizzazione del dataset, andando dunque a portare gli array di `X_train` e `X_test` tra valori compresi tra 0 e 1, quindi vengono standardizzate le colonne contenenti le features.

Poi si è passati all'addestramento dei vari classificatori, andando ad utilizzare i set di training `X_train` e `Y_train`. L'addestramento è stato effettuato per ciascun classificatore, andando a creare una variabile per ciascuno di essi così da poterli utilizzare facilmente.

```
#Classificatori non dotati di iperparametro
KNN_Classifier_Unhyper = classifiers.knnClassifier(X_train, Y_train, mode: False)
BAYESIAN_Classifier_Unhyper = classifiers.bayesianClassifier(X_train, Y_train, mode: False)
RF_Classifier_Unhyper = classifiers.randomForestClassifier(X_train, Y_train, mode: False)
DECISIONTREE_Classifier_Unhyper = classifiers.decisionTreeClassifier(X_train, Y_train, mode: False)
SVC_Classifier_Unhyper = classifiers.svcClassifier(X_train, Y_train, mode: False)
LOGREG_Classifier_Unhyper = classifiers.logisticRegressionClassifier(X_train, Y_train, mode: False)
```

Come si vede nello screen sopra riportato, essi sono i vari classificatori addestrati senza l'uso di iperparametri ottimizzati ma usando quelli di default della libreria.

GENERAZIONE DEGLI IPERPARAMETRI

Per poter effettuare la loro ricerca, è stata adottata la GridSearchCV per ciascun classificatore.
Per ciascun classificatore sono stati impiegati gli stessi parametri di funzionamentoo del metodo GridSearchCV, ovvero:

```
GridSearchCV(classifier, parameters, refit=True, verbose=2, scoring='accuracy', n_jobs=-1, cv=10)
```

Con classifier che indica il classificatore per cui si vogliono cercare i parametri, parameters che indica l'insieme dei parametri da valutare, verbose che indica il livello di dettaglio da riportare nel console per ciascun parametro analizzato, n_jobs=-1 che permette di avere il multithreading così da velocizzare le varie fasi di analisi e infine cv=10, ovvero la strategia di cross validation impiegata nella ricerca di iperparametri.

Di seguito sono riportati, per ciascun classificatore, i parametri utilizzati per l'analisi e i parametri migliori trovati.

Support Vector Classification

```
def hyP_SVC():
    parameters = {'C': [0.1, 0.2, 0.4, 0.6, 1], 'gamma': [0.01, 0.1, 0.25, 0.5, 1],
                  'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}
    classifier = SVC()

For the examined diseases dataset with Support Vector Classification:
### Best hyperparameter
The output of the hyP_SVC function is {'C': 0.1, 'gamma': 0.25, 'kernel': 'poly'}
```

K-Neighbors

```
parameters = {'n_neighbors': [5, 7, 9, 11, 13, 15, 17, 21],
              'weights': ['uniform', 'distance'],
              'metric': ['minkowski', 'euclidean', 'manhattan'],
              'leaf_size': list(range(1, 30))}
For the examined diseases dataset with KNeighborsClassifier:
### Best hyperparameter
The output of the hyP_KNN function is {'leaf_size': 1, 'metric': 'minkowski', 'n_neighbors': 5, 'weights': 'distance'}
```

Logistic Regression

```
def hyP_LOGREG():
    parameters = {"C": np.logspace(-3, stop: 3, num: 20), "solver": ['newton-cg', 'liblinear'], 'penalty': ['l2']}
For the examined diseases dataset with Logistic Regression:
### Best hyperparameter
The output of the hyP_LOGREG function is: {'C': 1000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
```

Decision Tree

```
def hyP_DECISIONTREE():
    parameters = {"criterion": ("gini", "entropy"),
                  "splitter": ("best", "random"),
                  "max_depth": (list(range(1, 20))),
                  "min_samples_split": [2, 3, 4],
                  "min_samples_leaf": list(range(1, 20))
                  }
For the examined diseases dataset with Decision Tree:
### Best hyperparameter
The output of the hyP_DECISIONTREE function is {'criterion': 'gini', 'max_depth': 17,
'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
```

Naive Bayes

```
def hyP_NAIVEBAYES():
    parameter = {'var_smoothing': np.logspace( start: 0, -9, num=100)}

For the examined diseases dataset with Naive Bayes:
### Best hyperparameter
The output of the hyP_NAIVEBAYES function is {'var_smoothing': 0.005336699231206307}
```

VALUTAZIONE E CONCLUSIONI

Per la valutazione dei classificatori utilizzati in fase di addestramento, per verificare gli iperparametri trovati e infine per i confronti tra classificatori sono state impiegate le metriche di **accuracy, f1-score e precision score**, tutte facenti parte della libreria **sklearn**.

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report
```

Lo splitting del dataset è stato prima effettuato con il 70% di split per il training set e poi con il 65% per il training set. Questo cambiamento, come è possibile vedere dai risultati sotto riportati, è stato effettuato poiché si è voluto controllare come si comportassero i vari classificatori con un diverso splitting del dataset. Difatti si è notato un miglioramento delle metriche con la leggera crescita dello splitting del dataset in favore del testing set.
I test sono stati effettuati sia con classificatori senza l'ottimizzazione degli iperparametri sia andando ad utilizzarli una volta trovati.

Splitting al 70% per il training set

Classificatore KNN:

Senza uso di iperparametri ottimizzati:

- F1-score: 99.30%
- Precision: 99.35%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.32%

Con l'uso di iperparametri ottimizzati (stessi risultati):

- F1-score: 99.30%
- Precision: 99.35%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.32%

Classificatore Bayes:

Senza uso di iperparametri ottimizzati:

- F1-score: 86.11%
- Precision: 87.10%
- Accuracy score (Training): 87.69%
- Accuracy score (Testing): 86.86%

Con l'uso di iperparametri ottimizzati:

- F1-score: 99.30%
- Precision: 99.35%
- Accuracy score (Training): 93.23%
- Accuracy score (Testing): 91.80%

Classificatore Random-Forest:

Senza uso di iperparametri ottimizzati (stessi risultati):

- F1-score: 99.30%
- Precision: 99.35%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.32%

Con l'uso di iperparametri ottimizzati (stessi risultati):

- F1-score: 99.30%
- Precision: 99.35%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.32%

Classificatore Decision Tree:

Senza uso di iperparametri ottimizzati:

- F1-score: 99.30%
- Precision: 99.35%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.32%

Con l'uso di iperparametri ottimizzati:

- F1-score: 99.15%
- Precision: 99.19%
- Accuracy score (Training): 99.48%
- Accuracy score (Testing): 99.19%

Classificatore SVC:

Senza uso di iperparametri ottimizzati:

- F1-score: 92.45%
- Precision: 93.05%
- Accuracy score (Training): 93.93%
- Accuracy score (Testing): 92.62%

Con l’uso di iperparametri ottimizzati (stessi risultati):

- F1-score: 99.30%
- Precision: 99.35%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.32%

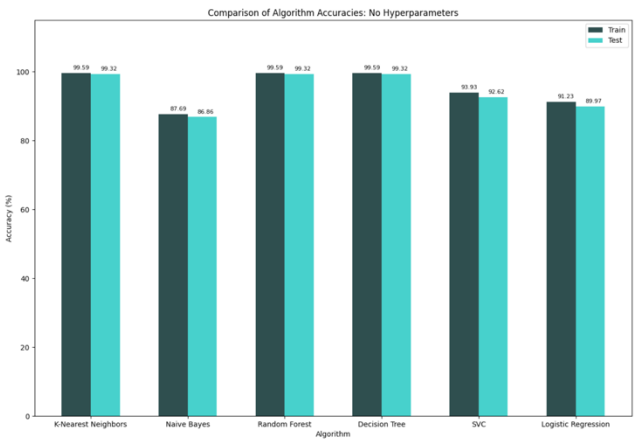
Classificatore Logistic Regression:

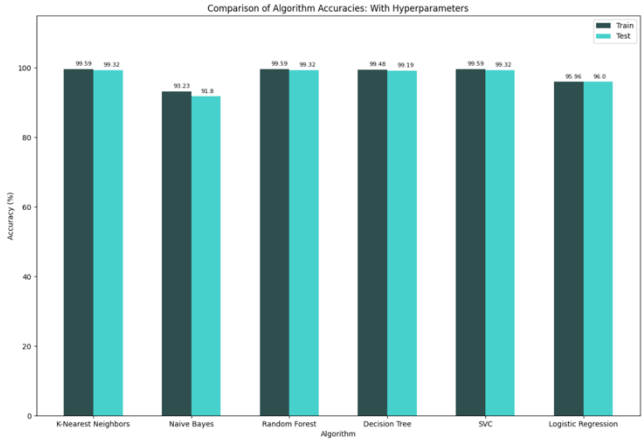
Senza uso di iperparametri ottimizzati:

- F1-score: 89.78%
- Precision: 90.31%
- Accuracy score (Training): 91.23%
- Accuracy score (Testing): 89.97%

Con l’uso di iperparametri ottimizzati:

- F1-score: 95.95%
- Precision: 95.94%
- Accuracy score (Training): 95.96%
- Accuracy score (Testing): 96.00%





Dai risultati ottenuti attraverso la valutazione dei vari classificatori, emerge chiaramente che il **classificatore Bayesiano** ha prestazioni inferiori rispetto agli altri modelli considerati. Senza l'uso di iperparametri ottimizzati, questo classificatore mostra il più basso accuracy score rispetto agli altri. Anche con l'implementazione degli iperparametri ottimizzati, seppur migliorando, l'accuracy score rimane comunque inferiore rispetto agli altri classificatori.
Per quanto riguarda gli altri classificatori, è interessante notare come l'accuracy score tenda a mantenersi relativamente stabile sia senza l'uso di iperparametri che utilizzandoli. Tuttavia, è possibile osservare un miglioramento significativo per tre dei sei classificatori analizzati: il **Naive Bayes**, il **Logistic Regression** e il **Support Vector Classifier (SVC).** Questi modelli hanno beneficiato dell'ottimizzazione degli iperparametri, mostrando un notevole aumento dell'accuracy score compreso tra il 5% e il 7% tra la fase di training e quella di testing. In conclusione, la scelta e l'ottimizzazione degli iperparametri giocano un ruolo fondamentale nel determinare le prestazioni complessive dei modelli di machine learning. Mentre alcuni classificatori

mantengono una buona stabilità nell'accuracy score, è evidente che l'implementazione di strategie di ottimizzazione può portare a miglioramenti significativi.

Splitting al 65% per il training set

Classificatore K-Neighbors:

Senza uso di iperparametri ottimizzati:

- F1-score: 99.34%
- Precision: 99.40%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.36%

Con l'uso di iperparametri ottimizzati (stessi risultati):

- F1-score: 99.34%
- Precision: 99.40%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.36%

Classificatore Bayes:

Senza uso di iperparametri ottimizzati:

- F1-score: 86.34%
- Precision: 87.22%
- Accuracy score (Training): 87.87%
- Accuracy score (Testing): 87.34%

Con l'uso di iperparametri ottimizzati:

- F1-score: 92.28%
- Precision: 93.11%
- Accuracy score (Training): 93.12%
- Accuracy score (Testing): 92.22%

Classificatore Random-Forest:

Senza uso di iperparametri ottimizzati:

- F1-score: 99.34%
- Precision: 99.40%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.36%

Con l'uso di iperparametri ottimizzati (stessi risultati):

- F1-score: 99.34%
- Precision: 99.40%
- Accuracy score (Training): 99.59%

- Accuracy score (Testing): 99.36%

Classificatore Decision Tree:

Senza uso di iperparametri ottimizzati:

- F1-score: 99.34%
- Precision: 99.40%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.36%

Con l'uso di iperparametri ottimizzati (stessi risultati):

- F1-score: 99.34%
- Precision: 99.40%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.36%

Classificatore SVC:

Senza uso di iperparametri ottimizzati:

- F1-score: 93.47%
- Precision: 93.97%
- Accuracy score (Training): 94.28%
- Accuracy score (Testing): 93.55%

Con l'uso di iperparametri ottimizzati (stessi risultati):

- F1-score: 99.34%
- Precision: 99.40%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 99.32%

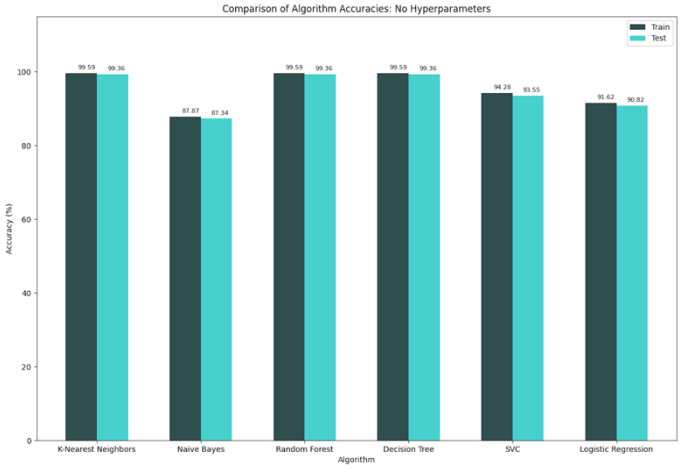
Classificatore Logistic Regression:

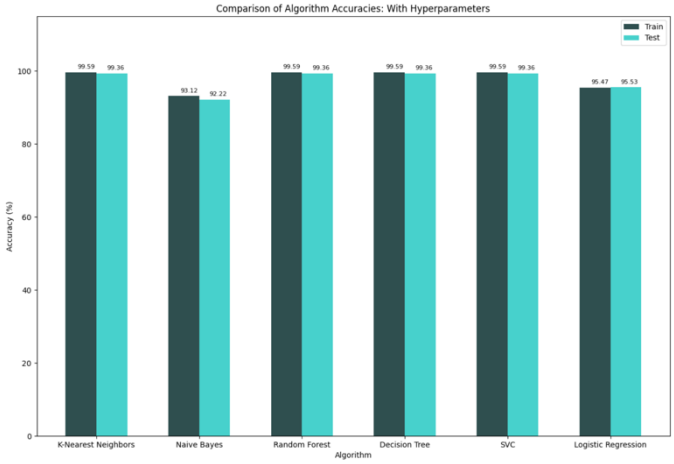
Senza uso di iperparametri ottimizzati:

- F1-score: 89.78%
- Precision: 90.31%
- Accuracy score (Training): 91.23%
- Accuracy score (Testing): 89.97%

Con l'uso di iperparametri ottimizzati:

- F1-score: 95.95%
- Precision: 95.94%
- Accuracy score (Training): 95.96%
- Accuracy score (Testing): 96.00%





Dai risultati ottenuti attraverso la valutazione dei vari classificatori con uno split del dataset al 65%, emerge chiaramente che il classificatore Bayesiano continua a mostrare prestazioni inferiori rispetto agli altri modelli considerati. Senza l'uso di iperparametri ottimizzati, questo classificatore presenta l'accuracy score più basso tra tutti. Anche con l'implementazione degli iperparametri ottimizzati, seppur migliorando, l'accuracy score rimane comunque inferiore rispetto agli altri classificatori.
Per quanto riguarda gli altri classificatori, è interessante notare come l'accuracy score tenda a mantenersi relativamente stabile sia senza l'uso di iperparametri che utilizzandoli. Tuttavia, è possibile osservare un miglioramento significativo per tre dei sei classificatori analizzati: il Naive Bayes, il regressore logistico e il Support Vector Classifier (SVC). Questi modelli hanno beneficiato dell'ottimizzazione degli iperparametri, mostrando un notevole aumento dell'accuracy score compreso tra il 4% e il 6% tra la fase di training e quella di testing.
In conclusione, il classificatore Bayesiano continua a rappresentare il punto debole, mentre gli altri modelli mantengono una stabilità nell'accuracy score. L'ottimizzazione degli iperparametri continua a dimostrare un impatto positivo sulla performance predittiva, come evidenziato dai miglioramenti registrati nei casi del Naive Bayes, del regressore logistico e del Support Vector Classifier.

Volendo concludere l’analisi per quel che riguarda il training split del 70% e del 65%, si è notato un leggerissimo aumento di tutti gli score, con un aumento compreso tra lo 0.5% e l’1.5%.

STANDARDIZZAZIONE DEI DATI

Applicando la standardizzazione dei dati a tutti i classificatori si è notato come non ci siano stati miglioramenti tra le varie metriche per tre dei sei classificatori analizzati.

In particolare, c'è stato un peggioramento sostanzioso per quel che riguarda il classificatore bayesiano. Infatti come si può vedere subito sotto, c'è stato un drastico calo delle metriche nel caso non si usino gli iperparametri. Questo calo poi però non si presenta quando vengono impiegati gli iperparametri ottimizzati. Infatti in questo caso si presenta quasi l'1% di miglioramento delle metriche.

Classificatore Bayes:

Senza uso di iperparametri ottimizzati:

- F1-score: 23.24%
- Precision: 26.31%
- Accuracy score (Training): 87.69%
- Accuracy score (Testing): 28.79%

Con l'uso di iperparametri ottimizzati:

- F1-score: 92.80%
- Precision: 93.75%
- Accuracy score (Training): 93.61
- Accuracy score (Testing): 92.55%

Due classificatori dove però c'è stato un miglioramento dall'uso della standardizzazione sono il Support Vector classifier e il Logistic Regressor, dove c'è stato un boost del 2% nelle varie metriche. In particolare questo boost si vede sempre nell'accuracy-score dei due classificatori ma che non si riflette nelle metriche di F1 e precision score nel caso in cui vengano utilizzati gli iperparametri ottimizzati.

Classificatore SVC:

Senza uso di iperparametri ottimizzati:

- F1-score: 94.66%
- Precision: 95.03%
- Accuracy score (Training): 96.31%
- Accuracy score (Testing): 94.58%

Con l'uso di iperparametri ottimizzati:

- F1-score: 97.14%
- Precision: 98.18%
- Accuracy score (Training): 99.59%
- Accuracy score (Testing): 97.15%

Classificatore Logistic Regression:

Senza uso di iperparametri ottimizzati:

- F1-score: 91.22%
- Precision: 91.53%
- Accuracy score (Training): 92.36%
- Accuracy score (Testing): 91.33%

Con l'uso di iperparametri ottimizzati:

- F1-score: 92.30%
- Precision: 93.64%
- Accuracy score (Training): 95.88%
- Accuracy score (Testing): 92.48%

Per concludere il discorso riguardo la standardizzazione dei dati, andando a confrontare i risultati avuti dalla sua adozione e quelli avuti dall'uso degli iperparametri visti nelle pagine precedenti con lo split al 65%, si evince che non c'è stato un miglioramento effettivo nei vari score.
Dunque non è stata adottata la Standardizzazione.

CONFRONTO IPERPARAMETRI TROVATI

Come detto nella sezione precedente, è stata effettuata l'ottimizzazione degli iperparametri per i vari classificatori. Inoltre è stato anche fatto un test, per ciascun classificatore, con degli iperparametri che non sono ottimizzati ma comunque peggiori rispetto a quelli standard utilizzati nella libreria dei classificatori.

SVC

```
### Best hyperparameter

{'C': 0.1, 'gamma': 0.25, 'kernel': 'poly'}

Accuracy: 0.9936120789779327 %


### Not good hyperparameter

{C=1, gamma=0.5, kernel='sigmoid'}

Accuracy: 0.22195121951219512 %
```

K-Neighbors

```
### Best hyperparameter
```

```
{'leaf_size': 1, 'metric': 'minkowski', 'n_neighbors': 5, 'weights':
'distance'}

Accuracy: 0.9936120789779327 %    Best score: 0.9953115203761754


### Not the best hyperparameter

{'leaf_size': 1, 'metric': 'euclidean', 'n_neighbors': 21, 'weights':
'uniform'}

Accuracy: 0.9422764227642276
```

Logistic Regression

```
### Best hyperparameter

{'C': 1000.0, 'penalty': 'l2', 'solver': 'newton-cg'}

Accuracy: 0.9243902439024391 %


### Not good hyperparameter

{'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
```

Nel caso del regressore logistico, immettendo il "not good hyperparameter" riportato sopra, non è stato possibile concludere i test per valutare l'accuracy score.

Decision Tree

```
### Best hyperparameter

{'criterion': 'gini', 'max_depth': 17, 'min_samples_leaf': 1,
'min_samples_split': 2, 'splitter': 'best'}

Accuracy: 0.9936120789779327 %


### Not good at all hyperparameter

{'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 3,
'min_samples_split': 5, 'splitter': 'random'}

Accuracy: 0.06260162601626017 %
```

Random Forest

```
### Best hyperparameter

{'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2,
 'n_estimators': 900}

Accuracy: 0.9936120789779327 %


### Not good hyperparameter

{'max_depth': 2, 'min_samples_leaf': 4, 'min_samples_split': 10,
 'n_estimators': 1500}

Accuracy: 0.6439024390243903 %
```

Naive Bayes

```
### Best hyperparameter

{'var_smoothing': 0.005336699231206307}

Accuracy: 0.9308943089430894 %


### Not good hyperparameter

{'var_smoothing': 12.74}

Accuracy: 0.6552845528455284 %
```

PROBLEMA DI RICERCA

Nell'applicativo è stata implementata la funzionalità di ricerca del percorso migliore data una posizione di partenza, che può essere una località della città di Altamura, e una posizione di destinazione, sia essa quella di un dottore o dell'ospedale.
Come città sulla quale basare l'intera ricerca è stata adottata la città di Altamura.

STRUMENTI UTILIZZATI

Il file "**Altamura.csv"** contiene tutte le informazioni riguardo le località implementate nell'applicativo.
Le località sono espresse come una strada, quindi da punto A con coordinate(longitudine e latitudine) a punto B con coordinate(longitudine e latitudine).
Il file dunque si struttura in righe e colonne, avendo come righe le varie località. Per ognuna di esse sono presenti:

- Long1,Lat1,Long2,Lat2: due coordinate del tipo (longitudine,latitudine) che rappresentano i punti da dove parte e dove finisce una strada
- PlaceName: nome della località
- Length: lunghezza della strada

L'individuazione della località relativa ad un medico o dell'ospedale, è fatta attraverso il nome della località. Quindi se si tratta di un medico, nel nome verrà posta la parola chiave "Medico", e la parola chiave "Ospedale" nel caso dell'ospedale.

Una volta avviata l'applicazione, il metodo `loadPositions` del modulo GeoLocation va a caricare tutte le località rappresentandole come nodi, espressi dall'oggetto Position creato appositamente, collegati da archi. Così facendo si va a formare il grafo orientato che andrà a rappresentare la mappa all'interno dell'applicazione.

Il grafo in questione è stato implementato nel modulo Graph_Support nella quale viene implementata la classe **Node,** la classe **Arc** e la classe **Path**, ovvero l'insieme degli archi tra nodi che vanno a formare un percorso.

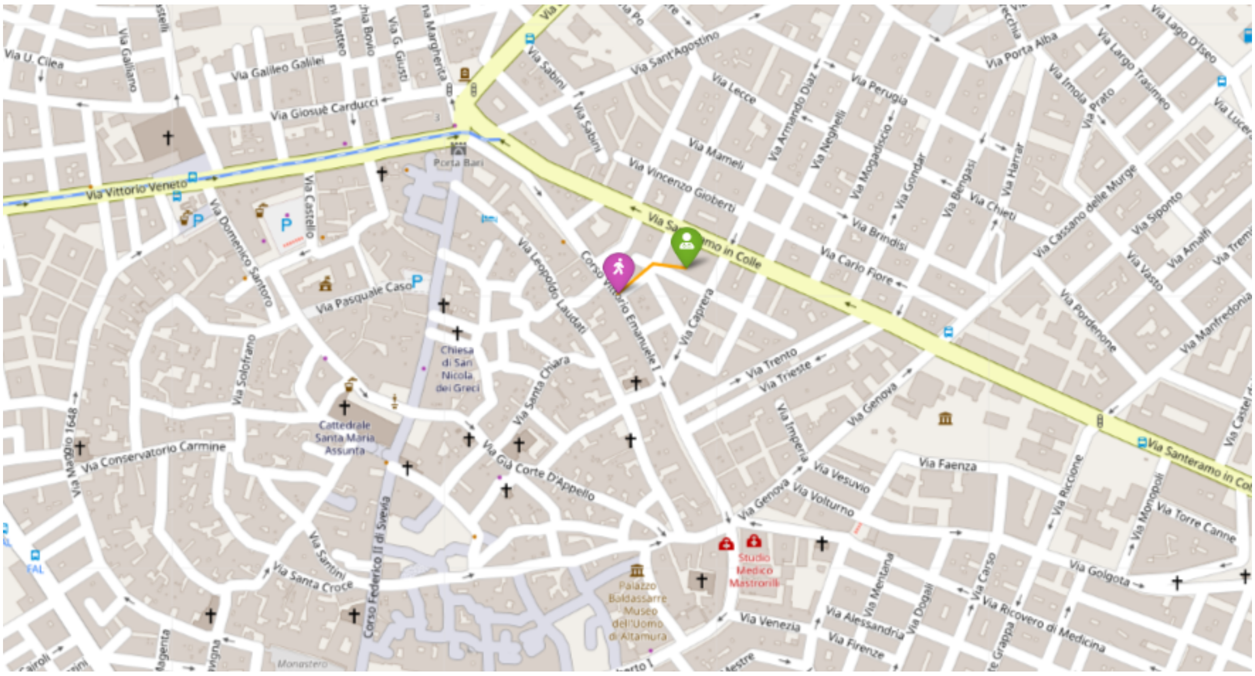
È stato poi implementato il problema di ricerca nel modulo **SearchProblem**.
Si è creata dunque la classe SearchProblem che rappresenta l'oggetto del problema di ricerca, con al suo interno il nodo di partenza e quello obiettivo. Inoltre sono presenti i metodi per recuperare tutti i nodi e gli archi presenti nella struttura.

Per la ricerca sul grafo è stato implementato l'algoritmo di ricerca **A*.** Per poterlo utilizzare inoltre è stata implementata la **frontiera**, rappresentata internamente con una coda con priorità che andrà a recuperare sempre il percorso con più priorità in quel momento, ovvero quello migliore in termini di lunghezza del percorso. ("heapq — Heap queue algorithm")

Per l'implementazione dell'algoritmo A* si è anche implementata l'euristica, in particolare quella della distanza euclidea. Questa euristica va dunque ad essere utilizzata nel calcolo degli score ovvero si va ad addizionare, per un percorso che va dal nodo di partenza ad un certo nodo n al momento in fase di analisi, il

costo di questo percorso più una stima data dall’euristica del costo di un percorso dal nodo n fino al nodo obiettivo.

Una volta trovato il percorso migliore, esso viene visualizzato all’interno del browser sotto forma di mappa. La mappa viene generata tramite la libreria Folium e i dati delle mappe vengono presi da OpenStreetMap. (“Folium”), (“OpenStreetMap”)



Decisioni di Progetto

L'euristica implementata per l'utilizzo con l'algoritmo A* è la distanza euclidea che va dunque a rappresentare la distanza, in linea d'aria, tra due punti. ("Distanza euclidea")

```
def eucledian_Heuristic(positionA: GeoLocation.Position, positionB: GeoLocation.Position):
    return math.sqrt((positionA.getContent().getLatitude() - positionB.getContent().getLongitude()) **2
                     + abs(positionA.getContent().getLatitude() - positionB.getContent().getLatitude())**2)
```

Knowledge Base

L'ultima funzionalità dell'applicativo è quella di permettere l'interrogazione della base di conoscenza al fine di ottenere varie informazioni, tra cui: orario di apertura e/o chiusura di uno studio medico e anche le malattie possibili, dato un singolo sintomo espresso. Dunque dato un sintomo, vengono recuperate tutte le malattie che contengono quel sintomo.

Strumenti utilizzati

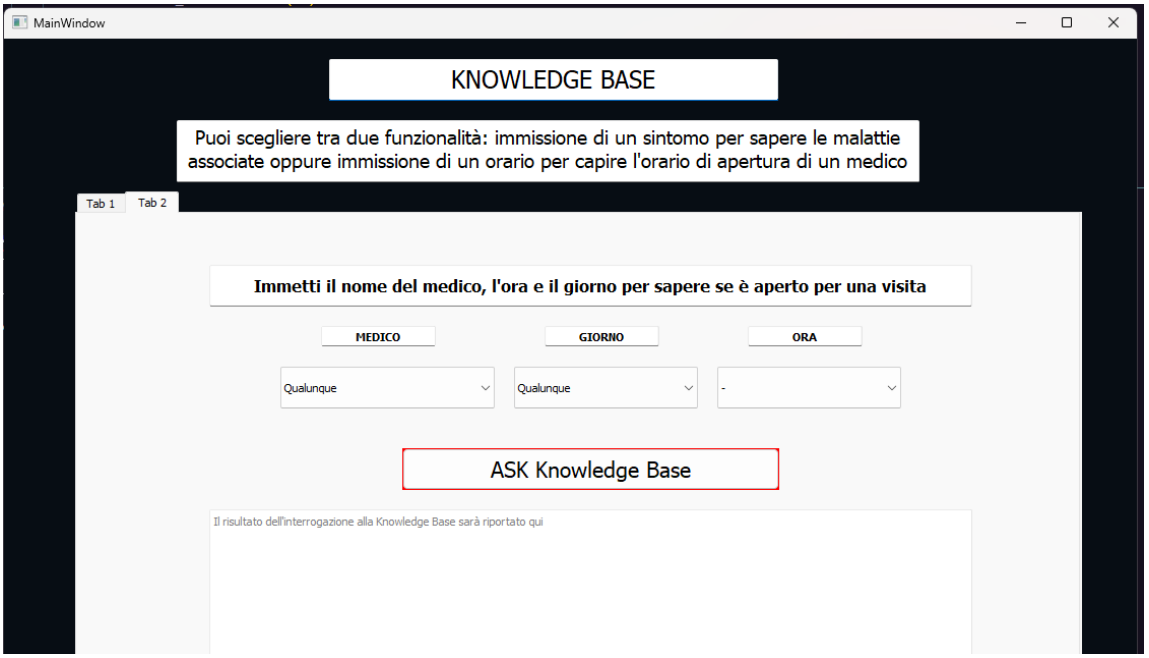
La base di conoscenza è stata implementata direttamente all'interno di python grazie alla libreria *pytholog* che fornisce i metodi per poter popolare la base di conoscenza con i fatti (facts) e il metodo per poter effettuare query sulla base di conoscenza. ("pytholog")

Decisioni di Progetto

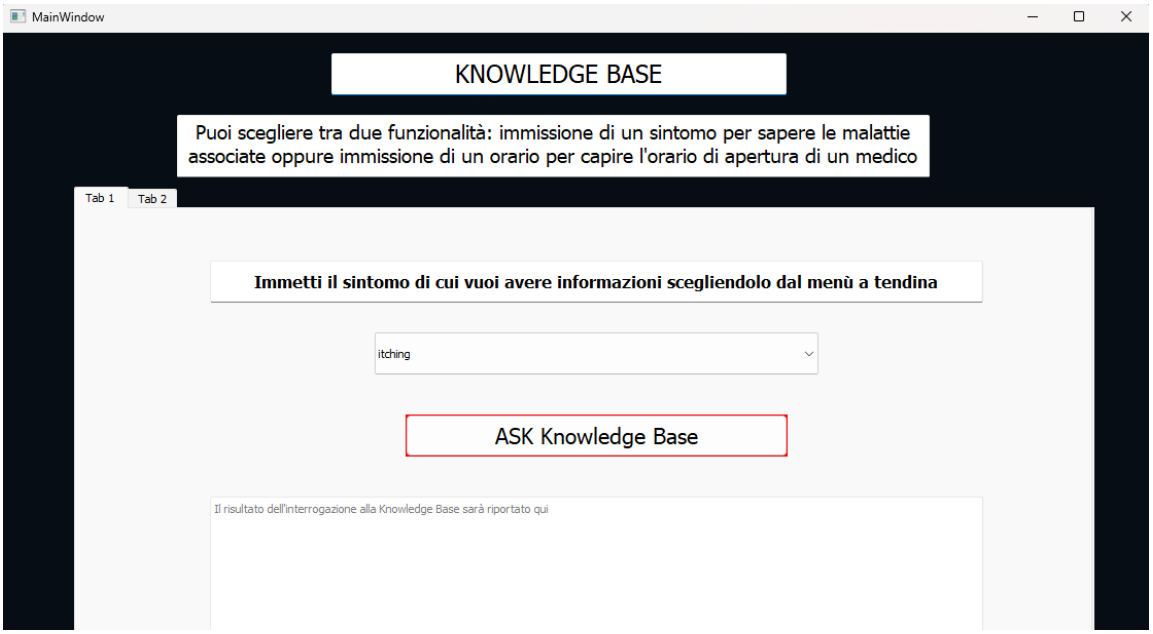
Gli assiomi che sono stati implementati all'interno della base di conoscenza sono del tipo:

```
#diseasehassymp(symptom,disease)

#doctorstarthour(doctor,start_hour,day)

#doctorendhour(doctor,end_hour,day)
```

Per effettuare delle query, si selezionano le varie opzioni direttamente da interfaccia grafica e sarà poi il sistema a creare la query adatta per poter interrogare la base di conoscenza.



In particolare le query permettono di verificare se una clausola è conseguenza logica dello stato della base di conoscenza, dunque si andrà ad avere una risposta del tipo “Yes” oppure “No” direttamente nella console python. Nell’interfaccia grafica invece verrà visualizzata una risposta testuale adeguata al contesto, quindi del tipo: “`Il medico da te cercato al momento risulta aperto.`”



Per quel che riguarda le query riguardo i sintomi all’interno di una malattia, il sintomo viene scelto da una comboBox e poi il sistema va ad effettuarle sfruttando anche variabili così che la risposta della base di conoscenza abbia tutti i valori che questa variabile può assumere.

Dunque la query sarà del tipo: “`diseasehassymp("nome sintomo,MALATTIA)`” dove MALATTIA sarà la variabile da usare per le risposte.

Le possibili risposte saranno del tipo: {'MALATTIA': 'fungal_infection'}, {'MALATTIA':'chronic_cholestasis'} all'interno della console python mentre nell'interfaccia grafica saranno del tipo:

- Input query: diseasehassymp(itching,MALATTIA)

- Output delle possibili malattie dato il sintomo:

{'MALATTIA': 'drug_reaction'}

{'MALATTIA': 'chronic_cholestasis'}

{'MALATTIA': 'chicken_pox'}

{'MALATTIA': 'fungal_infection'}

{'MALATTIA': 'hepatitis_b'}

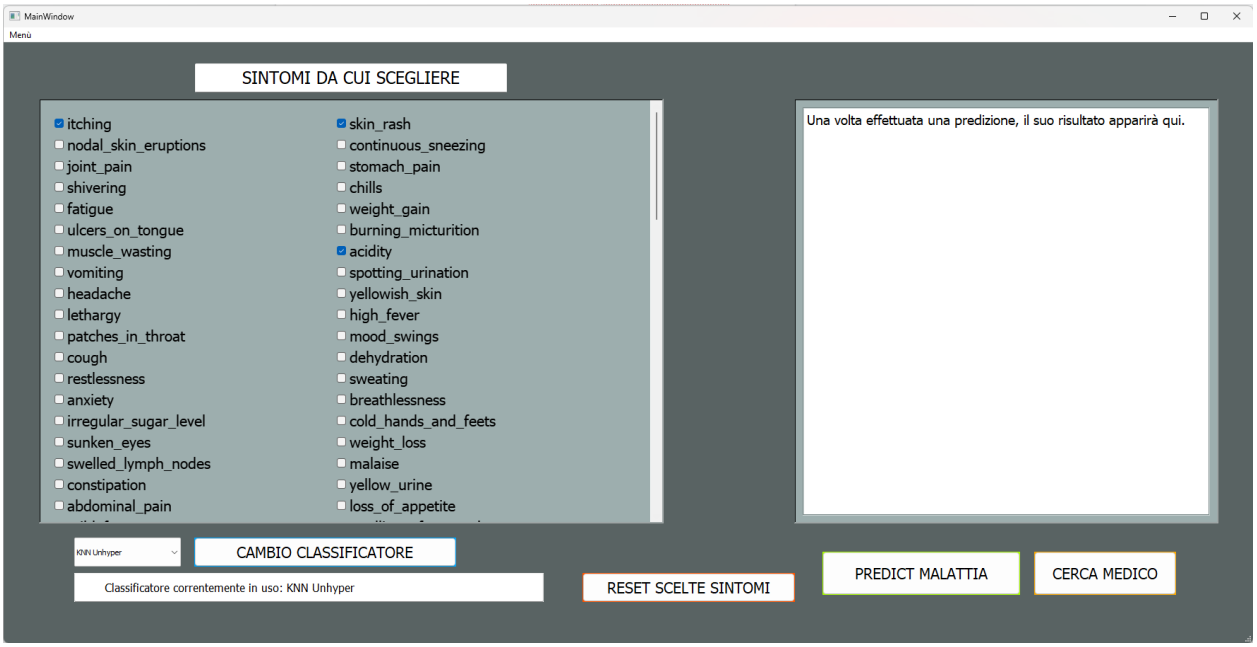
{'MALATTIA': 'jaundice'}

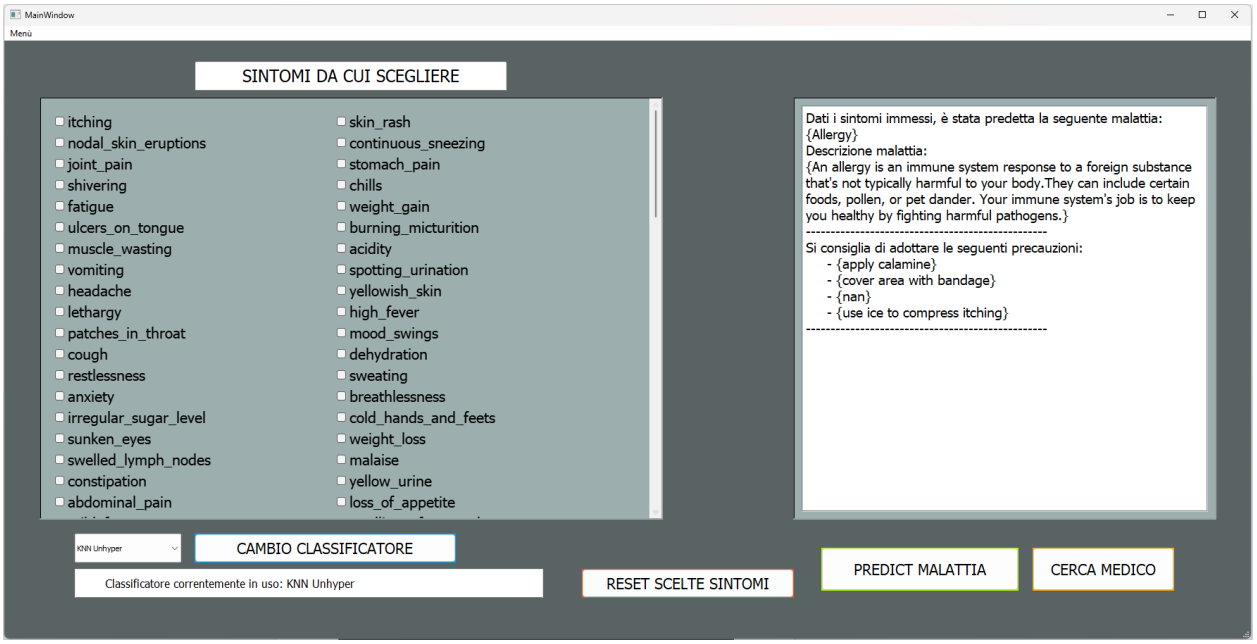
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-__-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_

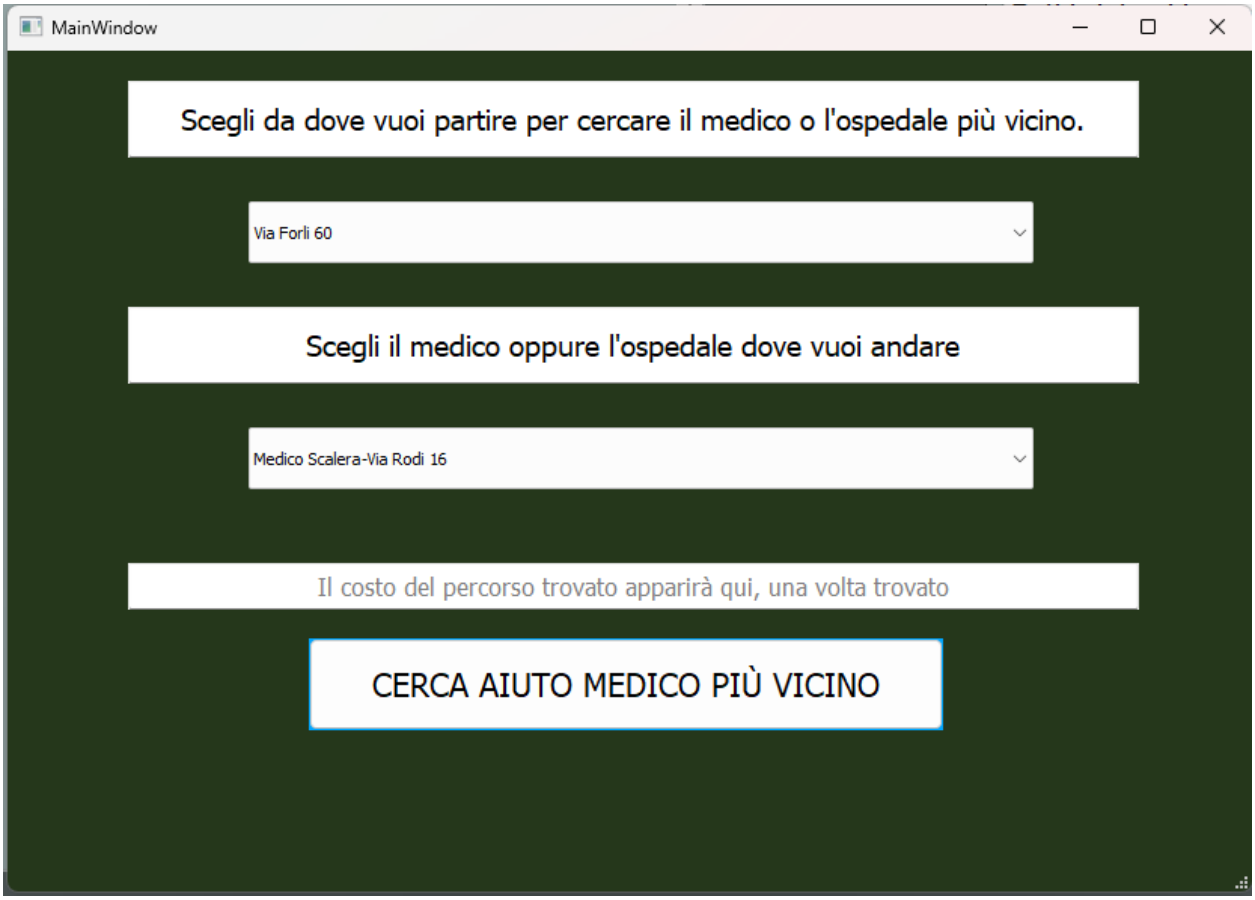
INTERFACCIA GRAFICA

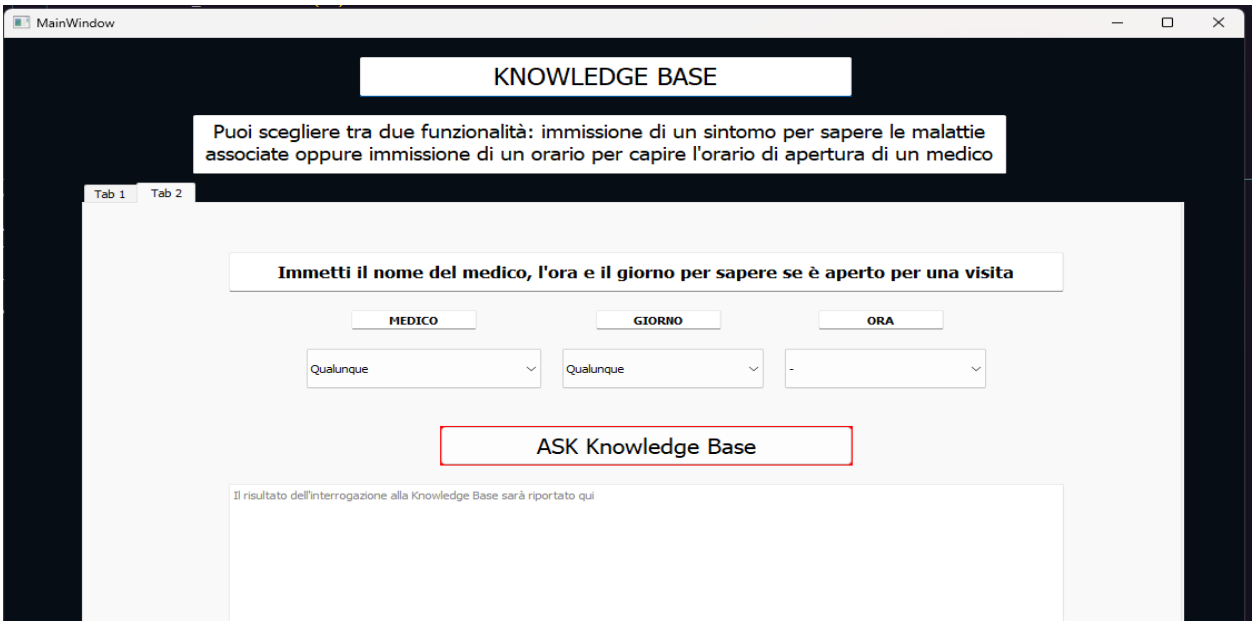
L'interfaccia grafica è stata sviluppata anticipatamente tramite il tool "**designer**" della libreria PyQT5.

Da interfaccia grafica è anche possibile cambiare il classificatore utilizzato per la predizione.






MainWindow
Scegli da dove vuoi partire per cercare il medico o l'ospedale più vicino.
Via Forli 60
Scegli il medico oppure l'ospedale dove vuoi andare
Medico Scalera-Via Rodi 16
Il costo del percorso trovato apparirà qui, una volta trovato
CERCA AIUTO MEDICO PIÙ VICINO


MainWindow
KNOWLEDGE BASE
Puoi scegliere tra due funzionalità: immissione di un sintomo per sapere le malattie associate oppure immissione di un orario per capire l'orario di apertura di un medico
Tab 1
Tab 2
Immetti il nome del medico, l'ora e il giorno per sapere se è aperto per una visita
MEDICO
GIORNO
ORA
Qualunque
Qualunque
-
ASK Knowledge Base
Il risultato dell'interrogazione alla Knowledge Base sarà riportato qui

CONCLUSIONI

In ottica di sviluppo futuro, una prospettiva interessante potrebbe consistere nell'adozione di un'ontologia che contenga informazioni su un ampio spettro di malattie. Ciò consentirebbe un notevole ampliamento della conoscenza dell'applicativo, permettendo di incorporare un maggior numero di malattie e migliorare la copertura medica dell'applicazione.

Un'altra possibilità di sviluppo futuro, non implementata in questo progetto per mancanza di difficoltà tecniche, potrebbe riguardare la creazione di una mappa generica della città di riferimento. Questa mappa potrebbe essere utilizzata per effettuare ricerche sulla posizione iniziale e di destinazione relative ai medici e/o ospedali. L'obiettivo sarebbe permettere il recupero automatico delle posizioni senza la necessità di immetterle manualmente nel codice.

In questo modo, entrambi gli sviluppi futuri contribuirebbero a arricchire l'applicazione, migliorando la sua utilità e la sua praticità per gli utenti.

RIFERIMENTI BIBLIOGRAFICI:

1. **accuracy_score:** Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html]

2. **DecisionTreeClassifier**: Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html]

3. **Distanza euclidea:** Wikipedia. [https://it.wikipedia.org/wiki/Distanza_euclidea]

4. **f1_score**: Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html]

5. **Folium**: GitHub Pages. [https://python-visualization.github.io/folium/]

6. **GaussianNB**: Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html]

7. **GridSearchCV:** Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html]

8. **heapq — Heap queue algorithm:** Python Docs. [https://docs.python.org/3/library/heapq.html]

9. **KNeighborsClassifier**: Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html]

10. **LogisticRegression**: Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html]

11. **OpenStreetMap**: OpenStreetMap. [https://wiki.openstreetmap.org/wiki/Main_Page]

12. **pandas**: pandas. [https://pandas.pydata.org]

13. **precision_score**: Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html]

14. **pytholog**: pytholog (Write Prolog in Python). [https://mnoorfawi.github.io/pytholog/]

15. **RandomForestClassifier:** Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html]

16. **StandardScaler**: Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html]

17. **SVC**: Scikit-learn. [https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html]