

Disease Prediction

Documentazione Progetto

Gruppo di lavoro

- Vladut Andrei Mares, 706351, v.mares@studenti.uniba.it

Repository: <https://github.com/Handysaur/IconProject>

Primo appello AA 2022-3 (Iscritto all'AA 2021-22)

Introduzione

Il caso di studio sviluppato ha avuto come scopo quello di creare un applicativo in grado di, dati diversi sintomi relativi ad una malattia, predire quest'ultima nel migliore dei modi.

Per adempiere all'obiettivo sono stati impiegati vari algoritmi di machine learning ad apprendimento supervisionato.

Inoltre si è voluto anche avere altre funzionalità quali visualizzazione di una mappa con medici e ospedale con la quale interagire per poter trovare quello più vicino alla posizione fornita, così da essere in grado di avere aiuto medico in seguito ad una predizione.

Un'altra funzionalità implementata è quella di poter avere informazioni sull'apertura dello studio di un medico, in un determinato periodo scelto, e dato un sintomo specifico, trovare tutte le malattie che hanno quel determinato sintomo.

In tutto l'applicativo è stato utilizzato un dataset fornito tramite il sito [Kaggle](#).

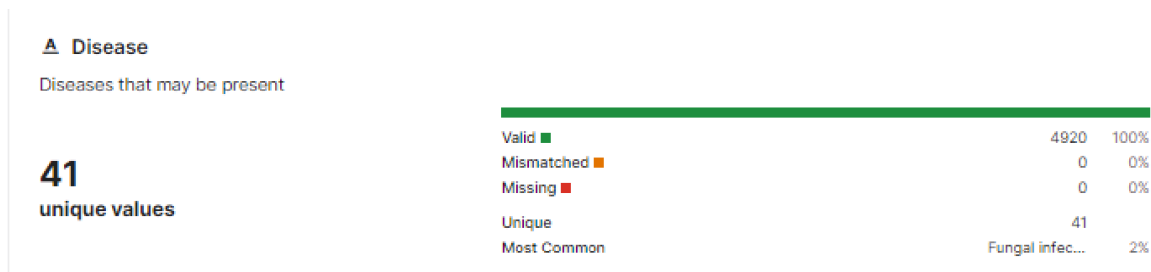
Dataset

Sezione dedicata alla descrizione del dataset e suo utilizzo all'interno dell'applicativo.

Il dataset utilizzato, chiamato "**disease-symptom-description-dataset**" che è possibile recuperare al seguente link: www.kaggle.com/datasets/itachi9604/disease-symptom-description-dataset

Il dataset è organizzato in file con formato csv e in totale sono quattro file, contenenti rispettivamente:

- **dataset**: dataset effettivo che ha come label le malattie e come features i sintomi espressi sottoforma di stringa. Il numero di label uniche è 41, dunque ci sono i dati per 41 diverse malattie. Il numero di sintomi unici invece 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. Dunque una malattia può avere un massimo di 17 sintomi.
- **Symptom-severity**: file contenente il peso che ciascun sintomo ha.
- **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.



dataset: 41 malattie uniche. 4920 righe

Dataset: Rielaborazione

Prima di passare all'utilizzo dei vari classificatori, è stato necessario effettuare delle operazioni di preprocessing sui dataset. Le maggiori operazioni sono state effettuate sul dataset in se.

Difatti è stato effettuato uno shuffle di esso, ovvero una serie di permutazioni dell'array dei dati, utilizzando un fattore *random_state* di 35.

Poi sono stati trasformati tutti i valori "Nan" in 0, rimossi tutti gli spazi andando a sostituirli con il tratto basso '_' e infine il dataset è stato portato al tipo DataFrame, così da poter essere utilizzato con la libreria Pandas. ("pandas")

Sono state poi codificate le severità di ciascun sintomo all'interno del dataset, andando dunque, per ciascuna malattia riportata, a sostituire i vari sintomi con le loro severità.

In questa fase poi sono anche stati sostituiti i sintomi senza rank, dunque rappresentati con il valore “Nan”, con 0 andando ad indicare che quei sintomi non hanno una severità in quel caso per la malattia. Dopo tutte le rielaborazioni, il dataset è stato diviso per dar vita all’insieme delle feature, ovvero i vari sintomi, e le label che sono le varie malattie.

Struttura progetto

Il progetto è organizzato in vari moduli, uno per ogni compito dell’applicativo.

Nel modulo **classifiers** sono state create tutte le funzioni per poter addestrare tutti i vari classificatori. Nel modulo **main** c’è la creazione delle variabili che contengono i riferimenti ai classificatori addestrati, per poterli utilizzare. Inoltre è presente il metodo per poter effettuare il predict di una malattia, dati i sintomi.

Nel modulo **hyperParameterCalc** sono stati creati i metodi per cercare gli iperparametri dei classificatori. Nel modulo **metrics** sono stati creati due metodi per poter rappresentare la performance del predict di un classificatore. Un metodo poi è utilizzato per poter effettuare il confronto tra classificatori, andando a effettuare il plot.

Nel modulo **KB** è descritta la Knowledge Base implementata e il metodo per interagire con essa.

Nel package **Dataset** sono contenuti i file csv del dataset e il modulo **data_Handling** che contiene i metodi di interazione e rielaborazione del dataset.

Nel package **Mapping** è contenuto il file csv che rappresenta le varie locazioni della città di Altamura. Poi sono stati creati i moduli **GeoLocation** che rappresenta l’oggetto **Position**, ovvero un luogo che andrà popolato con i dati dal csv. Il modulo **Graph_Support** che rappresenta le classi utili per la creazione del grafo orientato, ovvero Nodo, Arco e Percorso. Il modulo **SearchProblem** che rappresenta un problema di ricerca. Infine il modulo **Path_Operations** che contiene il metodo per la creazione della mappa, il modulo per visualizzare la mappa e infine per trovare il percorso migliore, data una posizione di inizio e una di destinazione.

Poi ci sono i vari moduli relativi all’interfaccia grafica, ovvero: **GUI** che è la finestra principale dell’applicazione, **SecondWindow** che è la finestra relativa al ritrovamento del percorso verso un medico e/o ospedale e infine **KB_GUI** che è la finestra relativa all’interazione con la Knowledge Base.

Elenco argomenti di interesse

- Apprendimento supervisionato
- Problema di ricerca su grafo e implementazione della ricerca
- Interrogazione della base di conoscenza

Sezione Apprendimento supervisionato

Sommario

Per poter avere l'applicativo che sia in grado di predire una malattia, dati dei sintomi, sono stati adottati dei modelli di apprendimento supervisionato.

Inoltre, per ciascun classificatore, è stata effettuata la ricerca del miglior iperparametro.

Strumenti utilizzati

Tutti gli algoritmi e i metodi per interagire con essi, sono presi dalla libreria SkLearn, dalla quale sono stati importati i vari moduli relativi ai classificatori vari.

In particolare si è scelto di adottare:

- 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 quel che riguarda il ritrovamento degli iperparametri, è stata sfruttata la GridSearchCV, che consiste di una ricerca esaustiva su insiemi di parametri di uno stimatore. ("GridSearchCV")

Per la standardizzazione delle features del dataset, è stato utilizzato StandardScaler. ("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(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.

```
KNN_Classifier_Unhyper = classifiers.knnClassifier(X_train, Y_train, False)
BAYESIAN_Classifier_Unhyper = classifiers.bayesianClassifier(X_train, Y_train, False)
RF_Classifier_Unhyper = classifiers.randomForestClassifier(X_train, Y_train, False)
DECISIONTREE_Classifier_Unhyper = classifiers.decisionTreeClassifier(X_train, Y_train, False)
SVC_Classifier_Unhyper = classifiers.svcClassifier(X_train, Y_train, False)
LOGREG_Classifier_Unhyper = classifiers.logisticRegressionClassifier(X_train, Y_train, 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 funzionamento 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.

SVC

```
Grid search 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']}

Best found parameter: {'C': 0.1, 'gamma': 0.25, 'kernel': 'poly'}
```

K-Neighbors

```
Grid search parameters = { 'n_neighbors' : [5, 7, 9, 11, 13, 15, 17, 21],
                           'weights' : ['uniform', 'distance'],
                           'metric' : ['minkowski', 'euclidean', 'manhattan'],
                           'leaf_size' : list(range(1,30))}

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

Logistic Regression

```
Grid search parameters = {'C': np.logspace(-3, 3, 20), 'solver': ['newton-cg', 'liblinear'],
                           'penalty' : ['l2']}

Best found parameter: {'C': 1000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
```

Decision Tree

```
Grid search 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))}

Best found parameter: {'criterion': 'gini', 'max_depth': 17, 'min_samples_leaf': 1,
                       'min_samples_split': 2, 'splitter': 'best'}
```

Random Forest

```
Grid search parameters = {  
    'n_estimators': [500, 900, 1100, 1500],  
    'max_depth': [2, 3, 5, 10, 15, None],  
    'min_samples_split': [2, 5, 10],  
    'min_samples_leaf': [1, 2, 4]}  
  
Best found parameter: {'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 900}
```

Naive Bayes

```
Grid search parameters = {'var_smoothing': np.logspace(0,-9, num=100)}  
  
Best found parameter: {'var_smoothing': 0.005336699231206307}
```

La valutazione dei vari iperparametri è effettuata nella prossima sezione.

Valutazione

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

("accuracy_score"), ("f1_score"), ("precision_score")
```

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 K-Neighbors: Senza uso di iperparametri ottimizzati

F1-score% = 99.29691543850312 | Precision% = 99.34741822334611

Accuracy score:= Training: 99.593496% Testing: 99.322493%

Classificatore K-Neighbors: Con l'uso di iperparametri ottimizzati

F1-score% = 99.29691543850312 | Precision% = 99.34741822334611

Accuracy score:= Training: 99.593496% Testing: 99.322493%

Classificatore Bayes: Senza uso di iperparametri ottimizzati

F1-score% = 86.11288116531628 | Precision% = 87.09622773507019

Accuracy score:= Training: 87.688734% Testing: 86.856369%

Classificatore Bayes: Con l'uso di iperparametri ottimizzati

F1-score% = 99.29691543850312 | Precision% = 99.34741822334611

Accuracy score:= Training: 93.234611% Testing: 91.802168%

Classificatore Random-Forest: Senza uso di iperparametri ottimizzati

F1-score% = 99.29691543850312 | Precision% = 99.34741822334611

Accuracy score:= Training: 99.593496% Testing: 99.322493%

Classificatore Random-Forest: Con l'uso di iperparametri ottimizzati

F1-score% = 99.29691543850312 | Precision% = 99.34741822334611

Accuracy score:= Training: 99.593496% Testing: 99.322493%

Classificatore Decision Tree: Senza uso di iperparametri ottimizzati

F1-score% = 99.29691543850312 | Precision% = 99.34741822334611

Accuracy score:= Training: 99.593496% Testing: 99.322493%

Classificatore Decision Tree: Con l'uso di iperparametri ottimizzati

F1-score% = 99.15410115478328 | Precision% = 99.1933040448417

Accuracy score:= Training: 99.477352% Testing: 99.186992%

Classificatore SVC: Senza uso di iperparametri ottimizzati

F1-score% = 92.45336673635668 | Precision% = 93.04901126359162

Accuracy score:= Training: 93.931475% Testing: 92.615176%

Classificatore SVC: Con l'uso di iperparametri ottimizzati

F1-score% = 99.29691543850312 | Precision% = 99.34741822334611

Accuracy score:= Training: 99.593496% Testing: 99.322493%

Classificatore Logistic Regression: Senza uso di iperparametri ottimizzati

F1-score% = 89.78088080761452 | Precision% = 90.31305889413659

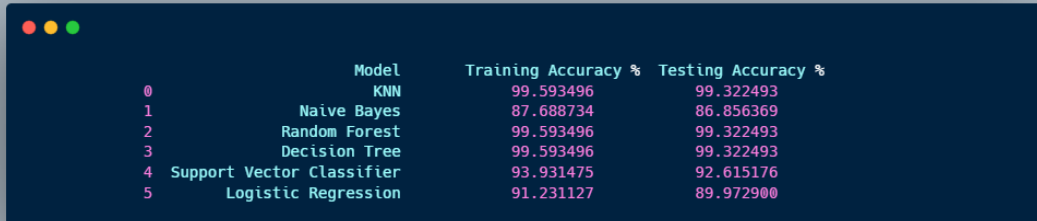
Accuracy score:= Training: 91.231127% Testing: 89.972900%

Classificatore Logistic Regression: Con l'uso di iperparametri ottimizzati

F1-score% = 95.95183079054553 | Precision% = 95.94486270251905

Accuracy score:= Training: 95.963995% Testing: 96.002710%

Confronto accuracy-score tra i vari classificatori

A terminal window with a dark blue background and light blue text. It displays a table comparing the training and testing accuracy of five different machine learning models. The models are KNN, Naive Bayes, Random Forest, Decision Tree, and Support Vector Classifier. The accuracies are listed as percentages.

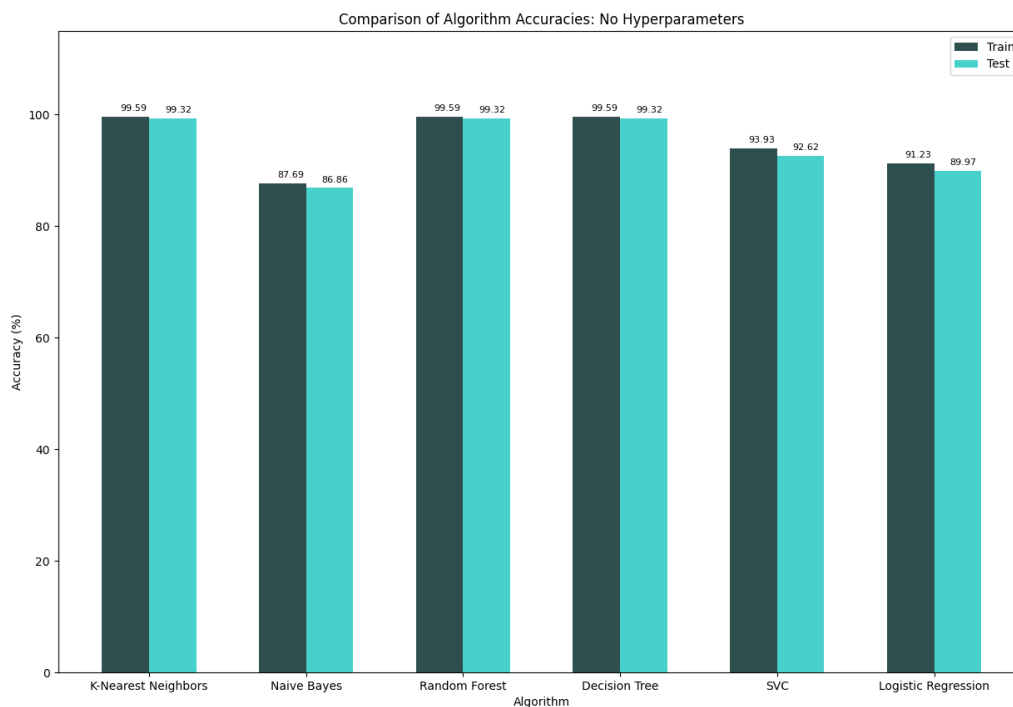
	Model	Training Accuracy %	Testing Accuracy %
0	KNN	99.593496	99.322493
1	Naive Bayes	87.688734	86.856369
2	Random Forest	99.593496	99.322493
3	Decision Tree	99.593496	99.322493
4	Support Vector Classifier	93.931475	92.615176
5	Logistic Regression	91.231127	89.972900

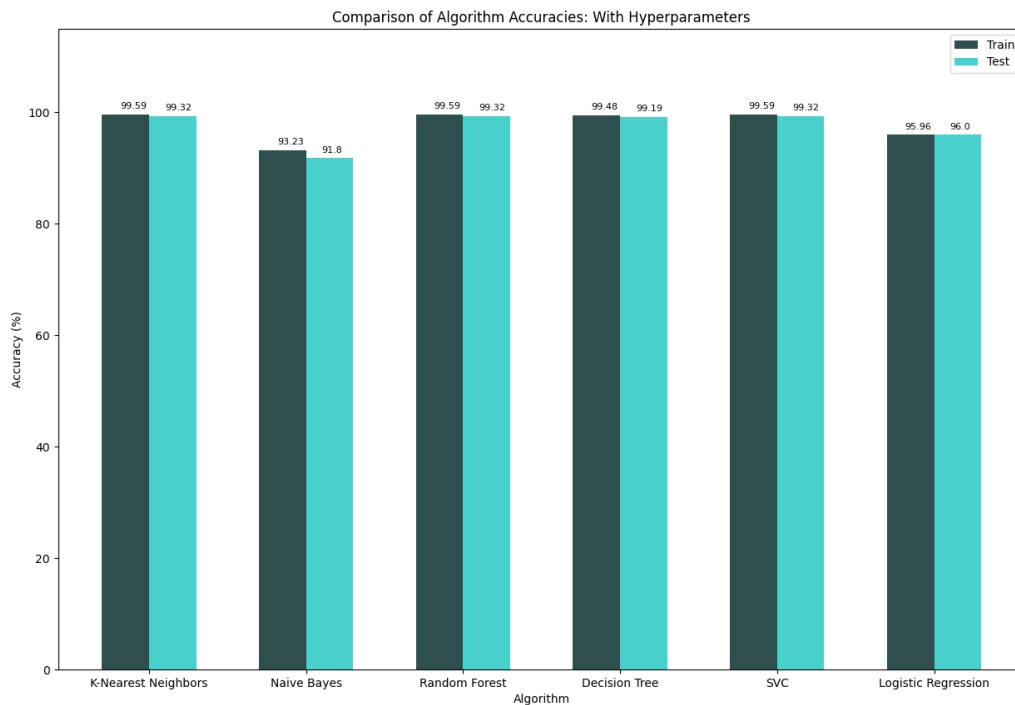
	Model	Training Accuracy %	Testing Accuracy %
0	Hyperparameter KNN	99.593496	99.322493
1	Hyperparameter Naive Bayes	93.234611	91.802168
2	Hyperparameter Random Forest	99.593496	99.322493
3	Hyperparameter Decision Tree	99.477352	99.186992
4	Hyperparameter Support Vector Classifier	99.593496	99.322493
5	Hyperparameter Logistic Regression	95.963995	96.002710

Come si può vedere dai vari report per ciascun classificatore, il classificatore bayesiano è quello che performa meno rispetto agli altri. Infatti nel caso in cui questo classificatore viene utilizzato senza l'uso di iperparametri ottimizzati, ha l'accuracy score minore di tutti. Discorso che non cambia anche quando vengono impiegati gli iperparametri ottimizzati, andando sì ad alzare l'accuracy score ma restando comunque minore rispetto a tutti gli altri classificatori.

Riguardo essi, si può notare come l'accuracy score risulti pressoché sempre lo stesso per la maggior parte dei classificatori, sia senza l'uso di iperparametri sia utilizzandoli.

Un miglioramento effettivo però c'è stato per tre dei sei classificatori analizzati. Infatti si può vedere come sia il Naive Bayes sia il regressore logistico e sia il Vector Support classifier abbiano beneficiato dell'ottimizzazione degli iperparametri, avendo un boost dell'accuracy-score compreso tra il 5% e il 7% tra la fase di training e quella di testing.





Splitting al 65% per il training set

Classificatore K-Neighbors: Senza uso di iperparametri ottimizzati

F1-score% = 99.34291398032053 | Precision% = 99.40155363397831

Accuracy score:= Training: 99.593496% Testing: 99.361208%

Classificatore K-Neighbors: Con l'uso di iperparametri ottimizzati

F1-score% = 99.34291398032053 | Precision% = 99.40155363397831

Accuracy score:= Training: 99.593496% Testing: 99.361208%

Classificatore Bayes: Senza uso di iperparametri ottimizzati

F1-score% = 86.33745321293328 | Precision% = 87.21806544122317

Accuracy score:= Training: 87.867417% Testing: 87.340302%

Classificatore Bayes: Con l'uso di iperparametri ottimizzati

F1-score% = 92.28031973116411 | Precision% = 93.10832893276547

Accuracy score:= Training: 93.120700% Testing: 92.218351%

Classificatore Random-Forest: Senza uso di iperparametri ottimizzati

F1-score% = 99.34291398032053 | Precision% = 99.40155363397831
Accuracy score:= Training: 99.593496% Testing: 99.361208%

Classificatore Random-Forest: Con l'uso di iperparametri ottimizzati
F1-score% = 99.34291398032053 | Precision% = 99.40155363397831
Accuracy score:= Training: 99.593496% Testing: 99.361208%

Classificatore Decision Tree: Senza uso di iperparametri ottimizzati
F1-score% = 99.34291398032053 | Precision% = 99.40155363397831
Accuracy score:= Training: 99.593496% Testing: 99.361208%

Classificatore Decision Tree: Con l'uso di iperparametri ottimizzati
F1-score% = 99.34291398032053 | Precision% = 99.40155363397831
Accuracy score:= Training: 99.593496% Testing: 99.361208%

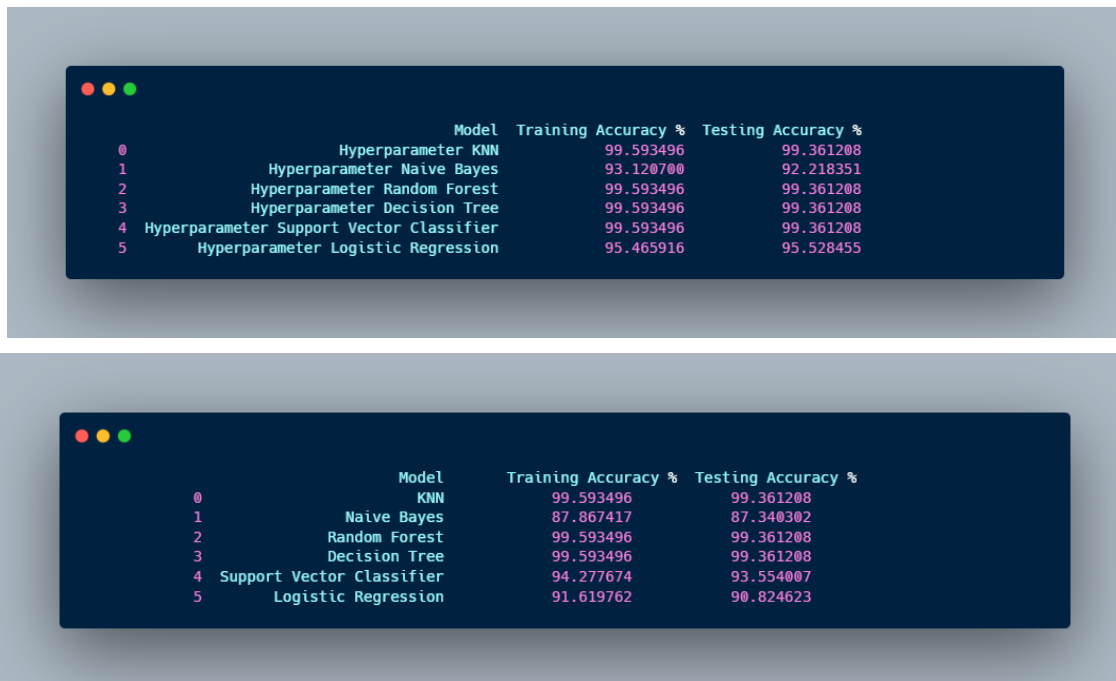
Classificatore SVC: Senza uso di iperparametri ottimizzati
F1-score% = 93.47432425071028 | Precision% = 93.9689845281356
Accuracy score:= Training: 94.277674% Testing: 93.554007%

Classificatore SVC: Con l'uso di iperparametri ottimizzati
F1-score% = 99.34291398032053 | Precision% = 99.40155363397831
Accuracy score:= Training: 99.593496% Testing: 99.322493%

Classificatore Logistic Regression: Senza uso di iperparametri ottimizzati
F1-score% = 89.78088080761452 | Precision% = 90.31305889413659
Accuracy score:= Training: 91.231127% Testing: 89.972900%

Classificatore Logistic Regression: Con l'uso di iperparametri ottimizzati
F1-score% = 95.95183079054553 | Precision% = 95.94486270251905
Accuracy score:= Training: 95.963995% Testing: 96.002710%

Confronto accuracy-score tra i vari classificatori



The image displays two terminal windows side-by-side, each showing a table of accuracy scores for six different classifiers. The top window shows results for models with hyperparameter tuning, while the bottom window shows results for the same models without tuning. In both cases, the Naive Bayes classifier has the lowest accuracy, while the other four classifiers (KNN, Random Forest, Decision Tree, and Support Vector Classifier) show significantly higher and more consistent performance.

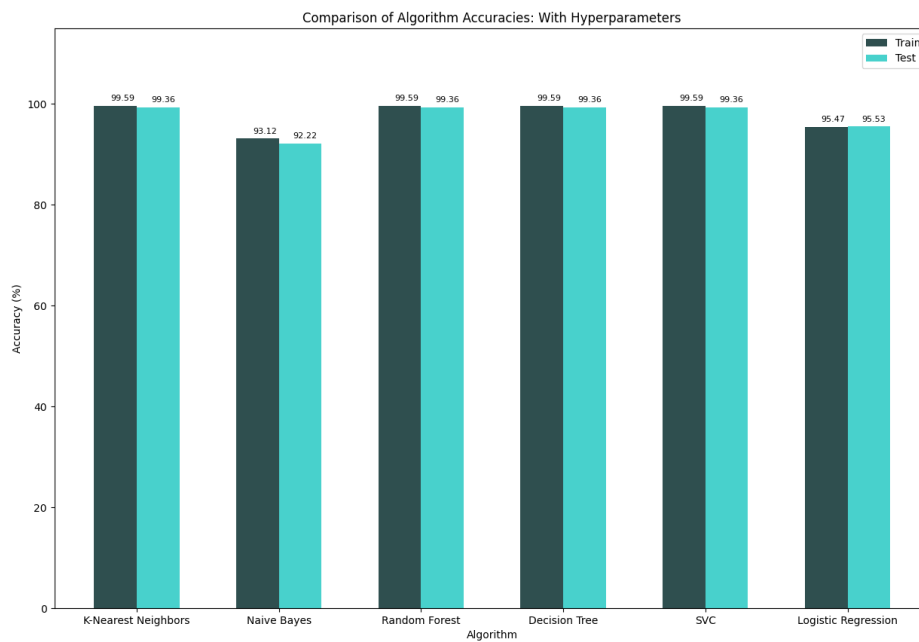
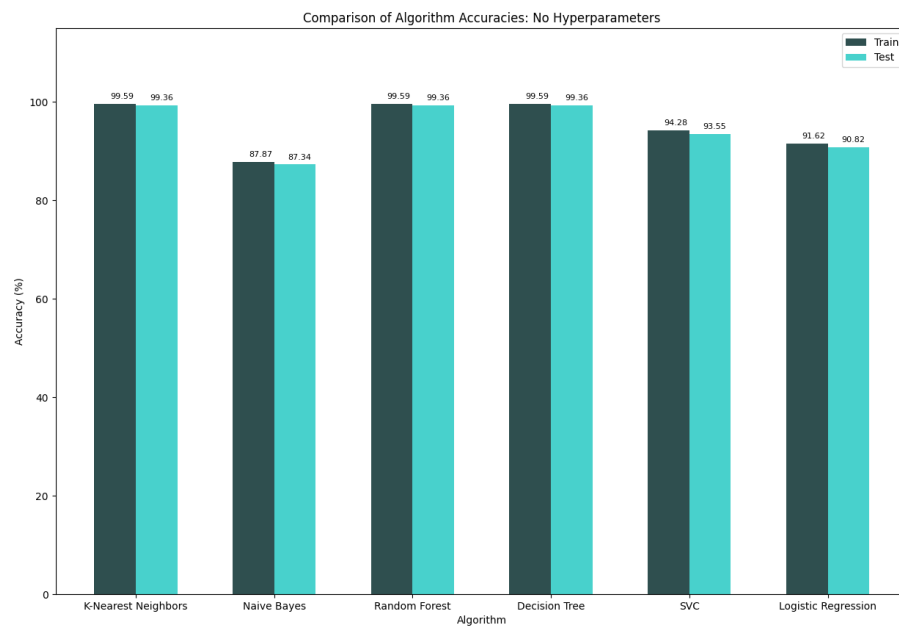
	Model	Training Accuracy %	Testing Accuracy %
0	Hyperparameter KNN	99.593496	99.361208
1	Hyperparameter Naive Bayes	93.120700	92.218351
2	Hyperparameter Random Forest	99.593496	99.361208
3	Hyperparameter Decision Tree	99.593496	99.361208
4	Hyperparameter Support Vector Classifier	99.593496	99.361208
5	Hyperparameter Logistic Regression	95.465916	95.528455

	Model	Training Accuracy %	Testing Accuracy %
0	KNN	99.593496	99.361208
1	Naive Bayes	87.867417	87.340302
2	Random Forest	99.593496	99.361208
3	Decision Tree	99.593496	99.361208
4	Support Vector Classifier	94.277674	93.554007
5	Logistic Regression	91.619762	90.824623

Come si può vedere dai vari report per ciascun classificatore, il classificatore bayesiano è quello che performa meno rispetto agli altri. Infatti nel caso in cui questo classificatore viene utilizzato senza l'uso di iperparametri ottimizzati, ha l'accuracy score minore di tutti. Discorso che non cambia anche quando vengono impiegati gli iperparametri ottimizzati, andando sì ad alzare l'accuracy score ma restando comunque minore rispetto a tutti gli altri classificatori.

Riguardo essi, si può notare come l'accuracy score risulti pressoché sempre lo stesso per la maggior parte dei classificatori, sia senza l'uso di iperparametri sia utilizzandoli.

Un miglioramento effettivo però c'è stato per tre dei sei classificatori analizzati. Infatti si può vedere come sia il Naive Bayes sia il regressore logistico e sia il Vector Support classifier abbiano beneficiato dell'ottimizzazione degli iperparametri, avendo un boost dell'accuracy-score compreso tra il 4% e il 6% tra la fase di training e quella di testing.



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% = 23.24288313173004 | Precision% = 26.309613870348308

Accuracy score:= Training: 87.688734% Testing: 28.794038%

Classificatore Bayes: Con l'uso di iperparametri ottimizzati

F1-score% = 92.79698146178535 | Precision% = 93.75414198452742

Accuracy score:= Training: 93.612079% Testing: 92.547425%

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% = 94.6586879399184 | Precision% = 95.03201279791293

Accuracy score:= Training: 96.312427% Testing: 94.579946%

Classificatore SVC: Con l'uso di iperparametri ottimizzati

F1-score% = 97.13800309307715 | Precision% = 98.18251105188635

Accuracy score:= Training: 99.593496% Testing: 97.154472%

Classificatore Logistic Regression: Senza uso di iperparametri ottimizzati

F1-score% = 91.22486979918524 | Precision% = 91.52807537429794

Accuracy score:= Training: 92.363531% Testing: 91.327913%

Classificatore Logistic Regression: Con l'uso di iperparametri ottimizzati

F1-score% = 92.29696376963386 | Precision% = 93.64235182027298

Accuracy score:= Training: 95.876887% Testing: 92.479675%

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 %
```

Sezione Problema di ricerca

Sommario

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.

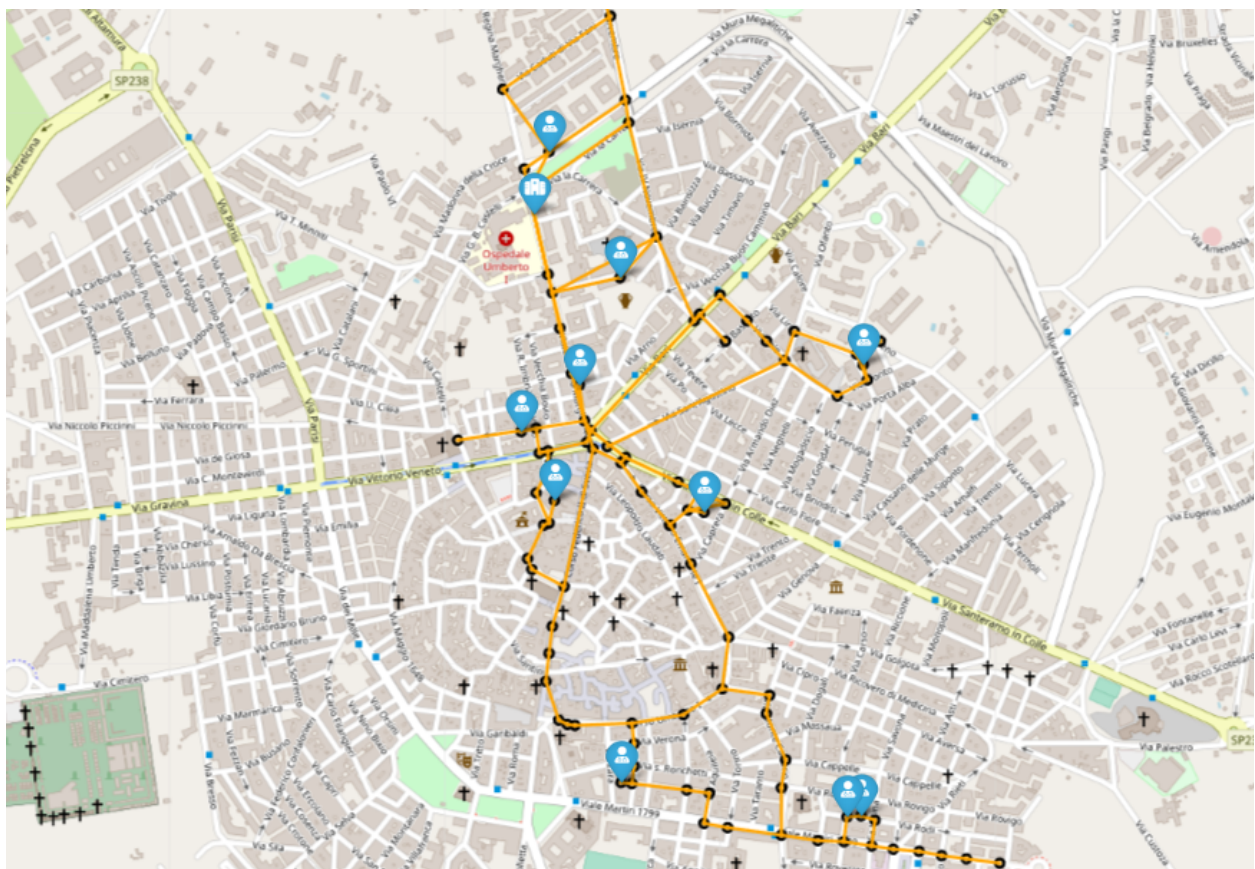
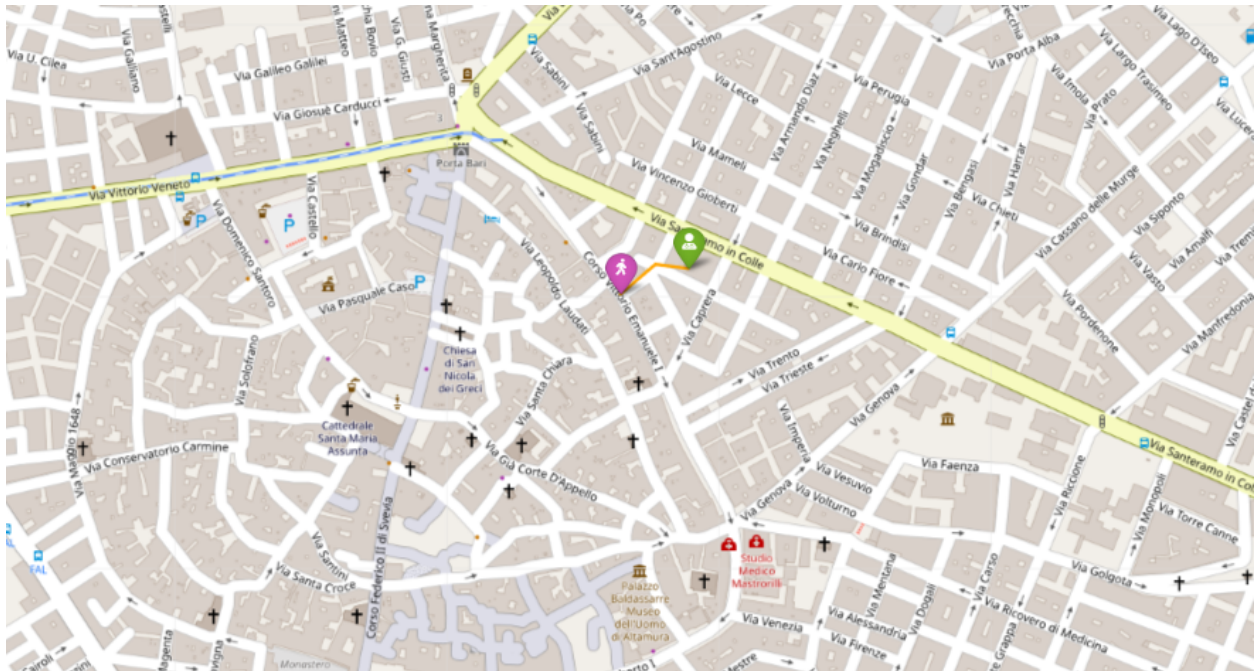
È stato poi implementato il problema di ricerca nel modulo **SearchProblem**.

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 aggiungere, 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 euclidian_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)
```

Sezione KB

Sommario

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.

The screenshot shows a window titled 'MainWindow' with a dark background. At the top, a white box contains the text 'KNOWLEDGE BASE'. Below it, another white box contains the text: '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'. Below this, there are two tabs labeled 'Tab 1' and 'Tab 2'. The 'Tab 1' content area has a white background and contains a form with the title 'Immetti il nome del medico, l'ora e il giorno per sapere se è aperto per una visita'. The form has three sections: 'MEDICO' with a dropdown menu showing 'Qualunque', 'GIORNO' with a dropdown menu showing 'Qualunque', and 'ORA' with a dropdown menu showing '-'. Below these is a red-outlined button labeled 'ASK Knowledge Base'. At the bottom of the form area, there is a text box with the placeholder text 'Il risultato dell'interrogazione alla Knowledge Base sarà riportato qui'.

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.”

The screenshot shows the same 'MainWindow' as the previous one. The 'Tab 1' content area now has a different form with the title 'Immetti il sintomo di cui vuoi avere informazioni scegliendolo dal menù a tendina'. Below the title is a dropdown menu showing 'itching'. Below the dropdown is a red-outlined button labeled 'ASK Knowledge Base'. At the bottom of the form area, there is a text box with the placeholder text 'Il risultato dell'interrogazione alla Knowledge Base sarà riportato qui'.

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'}

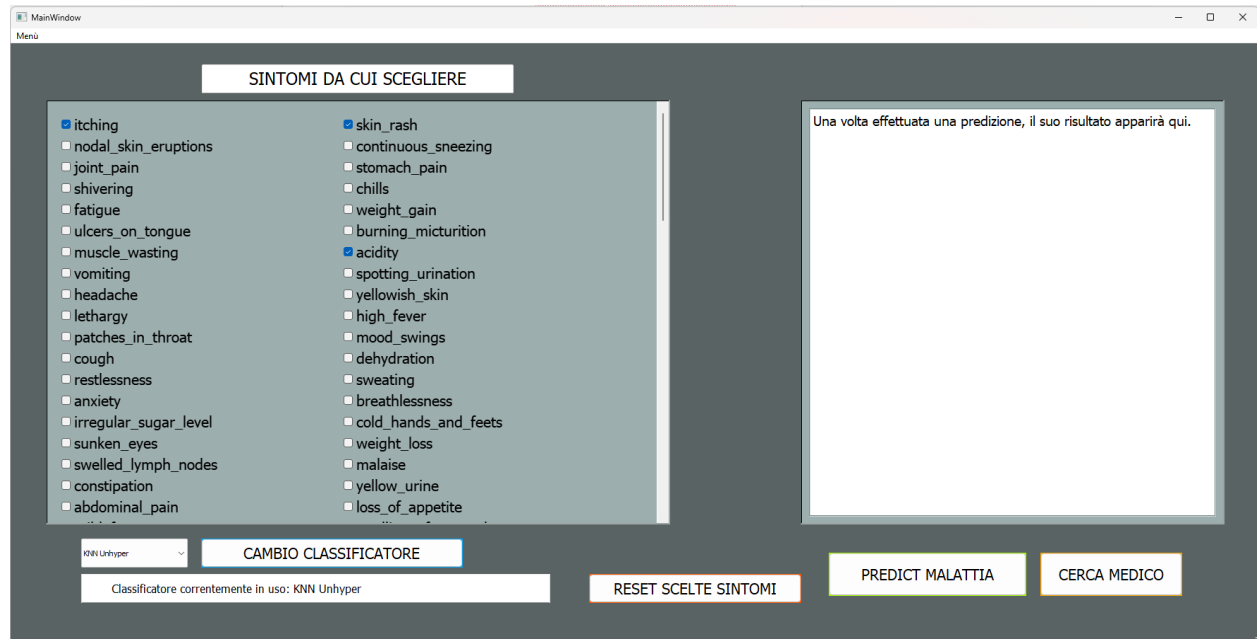
{'MALATTIA': 'hepatitis_b'}

{'MALATTIA': 'jaundice'}

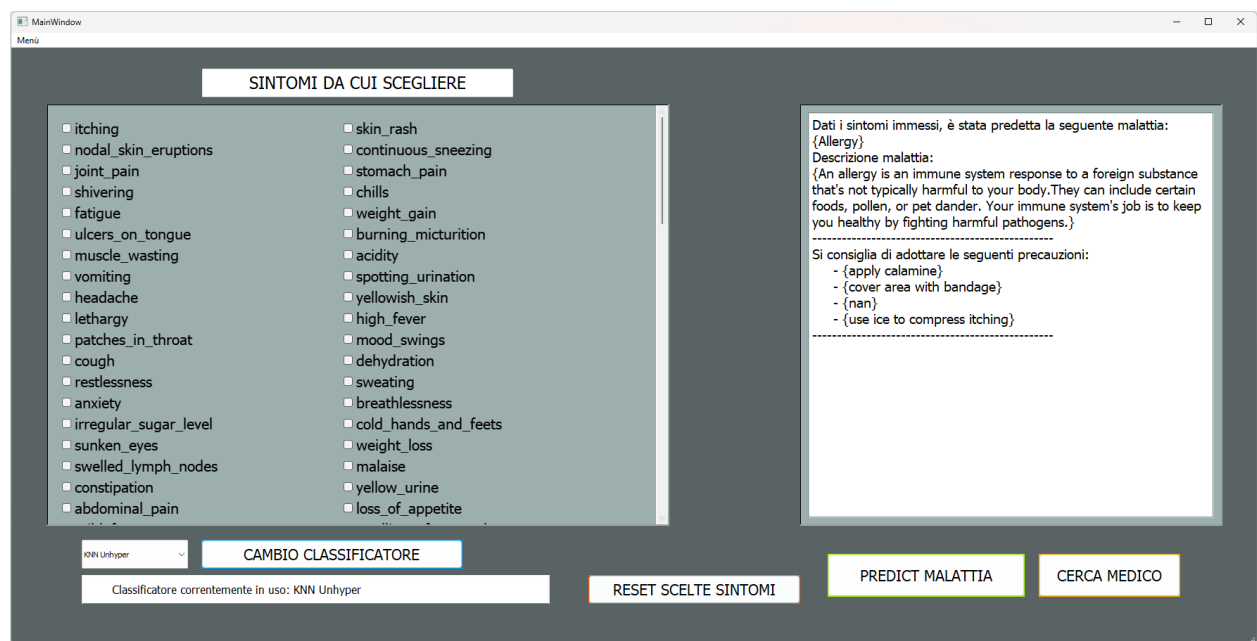
Interfaccia Grafica

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.



Malattia predetta



Malattia predetta

Scegli da dove vuoi partire per cercare il medico o l'ospedale più vicino.

Via Forlì 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

Una volta trovato il medico e/o ospedale richiesto, viene visualizzato il costo del percorso.

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

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

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

Immetti il sintomo di cui vuoi avere informazioni scegliendolo dal menù a tendina

Sintomo

ASK Knowledge Base

Il risultato dell'interrogazione alla Knowledge Base sarà riportato qui

Schermata di interazione con la base di conoscenza accessibili tramite il menù in alto nella finestra principale

Conclusioni

Uno sviluppo futuro potrebbe essere quello di adottare un'ontologia contenente informazioni su diverse malattie, andando così ad ampliare la conoscenza dell'applicativo. Inoltre così facendo sarebbe possibile avere molte più malattie e dunque arricchire quella che è la copertura dal punto di vista medico dell'applicazione.

Un altro spunto per uno sviluppo futuro, che per mancanza di difficoltà tecniche, non ha visto la sua implementazione in questo progetto, è quella di avere una mappa generica della città di riferimento sulla quale effettuare una ricerca per quel che riguarda la posizione iniziale da cui si vuole partire e la posizione di destinazione, dunque relativa ai medici e/o ospedali. Dunque avere la possibilità di recuperare automaticamente le varie posizioni senza immetterle a priori nel codice.

Riferimenti Bibliografici

“accuracy_score.” *Scikit-learn*,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html.

“DecisionTreeClassifier.” *Scikit-learn*,

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>.

“Distanza euclidea.” *Wikipedia*, https://it.wikipedia.org/wiki/Distanza_euclidea.

“f1_score.” *Scikit-learn*,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

“Folium.” *GitHub Pages*, <https://python-visualization.github.io/folium/>.

“GaussianNB.” *Scikit-learn*,

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html.

“GridSearchCV.” *Scikit-learn*,

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

“heapq — Heap queue algorithm.” *Python Docs*, <https://docs.python.org/3/library/heapq.html>.

“KNeighborsClassifier.” *Scikit-learn*,

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>.

“LogisticRegression.” *Scikit-learn*,

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

“OpenStreetMap.” *OpenStreetMap*, https://wiki.openstreetmap.org/wiki/Main_Page.

“pandas.” *pandas*, <https://pandas.pydata.org>.

“precision_score.” *Scikit-learn*,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html.

“pytholog.” *pytholog (Write Prolog in Python)*, <https://mnoorfawi.github.io/pytholog/>.

“RandomForestClassifier.” *Scikit-learn*,

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.

“StandardScaler.” *Scikit-learn*,

<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>.

“SVC.” *Scikit-learn*, <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>.