

Sistema di diagnostica e di predizione per diabete

Gruppo di lavoro

Pietro de Panizza, 726250, p.depanizza@studenti.uniba.it

https://github.com/pietro495/Icon_project.git

AA 2022-2023

Introduzione

Ho sviluppato un sistema di diagnostica e predizione per la malattia del diabete. Il sistema di diagnostica prevede una base di conoscenza che tramite l'uso delle clausole è in grado di effettuare predizioni e stabilire se il paziente è affetto da diabete, sindrome metabolica o nessuna malattia.

Il sistema di predizione sarà in grado di effettuare predizioni, se il paziente è affetto o meno dal diabete, sulla base dei dati inseriti dall'utente.

Esso sarà inoltre in grado di effettuare predizioni su dati presenti nel dataset.

Elenco argomenti di interesse

Paragrafo su argomento 1 (KnowledgeBase utilizzando forward chaining)

Paragrafo su argomento 2 (es. Apprendimento Supervisionato)

Paragrafo su argomento 3 (K-Fold Cross Validation)

Paragrafo su argomento 4 (ROC e AUC)

Sistema di diagnostica

Sommario

Il sistema del progetto cercherà di diagnosticare la malattia del diabete o della sindrome metabolica chiedendo all'utente se ha o meno i sintomi descritti. Sulla base di alcune clausole il sistema dirà all'utente se ha una malattia o l'altra oppure nessuna.

L'idea generale si basa su una regola di derivazione, una forma generalizzata della regola di inferenza chiamata *modus ponens*:

Se " $h \leftarrow a_1 \wedge \dots \wedge a_m$ " è una clausola definita nella base di conoscenza e ogni a_i è stato derivato, allora h può essere derivato.

Dove:

- h è la testa dell'atomo,
- $a_1 \wedge \dots \wedge a_m$ è il corpo della clausola, formato da a_i atomi

Se $m > 0$, la clausola è detta regola, se $m = 0$, il corpo è vuoto e la clausola è detta clausola atomica o fatto, e tutte le clausole atomiche nella base di conoscenza sono sempre derivate in maniera diretta.

Nel caso del sistema di diagnostica del diabete/sindrome metabolica la base di conoscenza è basata sulle seguenti clausole:

$\text{sintomi_uguali} \leftarrow \text{urinare}$

$\text{sintomi_uguali} \leftarrow \text{sete}$

$\text{sintomi_uguali} \leftarrow \text{iperglicemia}$

diagnosi diabete:

$\text{diabete} \leftarrow \text{urinare} \text{ AND } \text{sete} \text{ AND } \text{iperglicemia} \text{ AND } \text{stanchezza} \text{ AND } \text{fame} \text{ AND } \text{obesità}$

diagnosi sindrome metabolica:

$\text{sindrome_metabolica} \leftarrow \text{urinare} \text{ AND } \text{sete} \text{ AND } \text{iperglicemia} \text{ AND } \text{ipertensione} \text{ AND } \text{mal_di_testa}$

La malattia può manifestarsi in due forme diverse: diabete o sindrome metabolica. Le due sono accomunate da dei sintomi uguali: urinare, sete, iperglicemia.

I sintomi del diabete sono: sintomi_uguali, stanchezza, obesità, fame

Decisioni di Progetto

L'implementazione del sistema si basa su un sistema esperto realizzato in Python.

Utilizzo la libreria Experta, che permette di associare dei fatti accaduti a delle regole riguardo gli stessi. Le regole sono formate da due componenti chiamati LHS (Left-Hand-Side) e RHS(Right-Hand-Side). Il primo descrive le condizioni che si devono verificare affinché la regola venga applicata, mentre il secondo è l'insieme di azioni che vengono compiute quando viene applicata la regola.

Per ogni sintomo che viene chiesto all'utente, lui risponderà "si/no" e in base a questo, il sintomo verrà impostato a True o False e sulla base di questo verranno applicate le altre regole.

Per la diagnosi del diabete, il sistema chiede all'utente i sintomi uguali (descritti sopra) e poi quelli relativi ad esso e sulla base della clausola(sempre descritta sopra) farà la diagnosi. Lo stesso vale per la sindrome metabolica.

```
@Rule(Fact(question=True))
def ask_iperglicemia(self):
    self.declare(Fact(iperglicemia=ask_question("Soffri di iperglicemia?")))
```

```
#diagnosi diabete
@Rule(AND(Fact(urinare=True),Fact(sete=True),Fact(iperglicemia=True),Fact(obesità=True),Fact(fame=True),Fact(stanchezza=True)))
def diagnosi_diabete(self):
    print("Sei diabetico")
    self.reset()
```

```
#diagnosi sindrome
@Rule(AND(Fact(urinare=True),Fact(sete=True),Fact(iperglicemia=True),Fact(ipertensione=True),Fact(mal_di_testa=True)))
def diagnosi_sindrome(self):
    print("Hai sindrome metabolica")
    self.reset()
```

```
➤ Inserisci si se vuoi avviare sistema espertosi
Sei obeso?(si/no):ni
Il valore inserito non è valido
Sei obeso?(si/no):si
Soffri di iperglicemia?(si/no):si
Sei spesso molto stanco?(si/no):si
Urini frequentemente?Ti capita anche di notte?(si/no):si
Hai spesso tanta fame?(si/no):si
Hai spesso sete?(si/no):si
Sei diabetico
```

Sistema di predizione

Sommario

Il sistema di predizione si occupa di predire se un paziente presenta o meno il diabete tramite la SVM(support vector machine).

Il sistema opera online con l'acquisizione dei dati inseriti dall'utente ed effettua su di questi le predizioni. Per realizzare quest'ultima funzione ho sviluppato una web app interattiva con Streamlit chiamata "App Web per la previsione del diabete".



App Web per la previsione del diabete

Numero di gravidanze

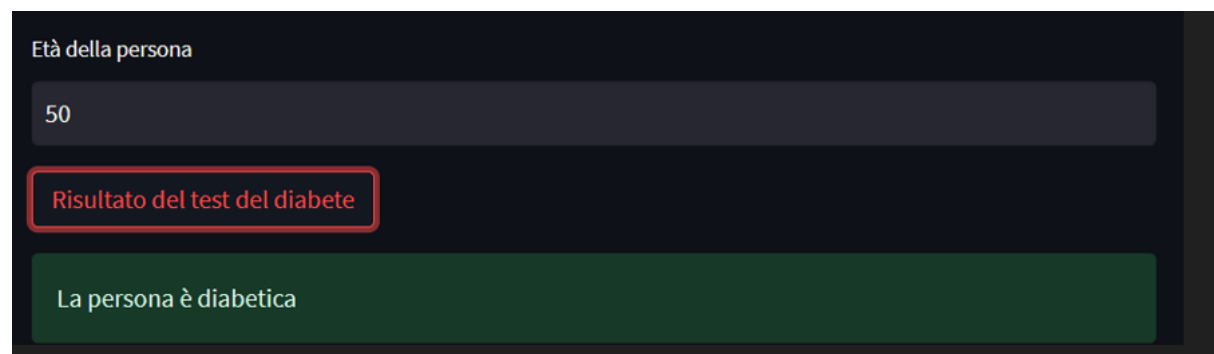
Livello di glucosio

Valore della pressione sanguigna

Valore dello spessore della pelle

Livello di insulina

Valore BMI



Età della persona

50

Risultato del test del diabete

La persona è diabetica

Strumenti utilizzati

Il sistema di predizione utilizza la SVM. E' stata implementata anche la K Fold Cross validation. Essa è stata utilizzata anche su altri modelli quali: KNN, Logistic Regression, RandomForest.

Decisioni di Progetto

Come librerie ho utilizzato la sklearn,pandas e numpy e tutte quelle che mi permettevano di utilizzare gli altri modelli.

La libreria utilizzata per la predizione è sklearn. Contiene algoritmi di classificazione, regressione e clustering (raggruppamento) e macchine a vettori di supporto, regressione logistica, classificatore bayesiano e k-mean ed è progettato per operare con le librerie NumPy e SciPy.

E' una libreria open-source di apprendimento automatico per il linguaggio di programmazione python. Ci ha permesso di analizzare il dataset e effettuare le varie predizioni con classificatori diversi.

Valutazione

```
Cross Validation accuracies for LogisticRegression(solver='liblinear') = [0.72727273 0.76623377 0.80519481 0.71428571 0.75324675 0.75324675
0.80519481 0.80519481 0.75      0.82894737]
Accuracy % of the LogisticRegression(solver='liblinear') 77.09
-----
Cross Validation accuracies for SVC(C=1000) = [0.67532468 0.72727273 0.5974026 0.64935065 0.74025974 0.7012987
0.67532468 0.7012987 0.64473684 0.76315789]
Accuracy % of the SVC(C=1000) 68.75
-----
Cross Validation accuracies for KNeighborsClassifier(metric='manhattan', n_neighbors=30, weights='distance') = [0.77922078 0.74025974 0.74025974 0.67532468 0.71428571 0.75324675
0.77922078 0.79220779 0.68421053 0.78947368]
Accuracy % of the KNeighborsClassifier(metric='manhattan', n_neighbors=30, weights='distance') 74.48
-----
Cross Validation accuracies for RandomForestClassifier(max_depth=30, min_samples_split=16) = [0.72727273 0.79220779 0.80519481 0.68831169 0.74025974 0.76623377
0.79220779 0.85714286 0.71052632 0.82894737]
Accuracy % of the RandomForestClassifier(max_depth=30, min_samples_split=16) 77.08
-----
```

Qui possiamo vedere i risultati ottenuti in seguito all'applicazione della k-fold cross validation. Quest'ultima è stata applicata prendendo in considerazione i classificatori: regressione logistica, SVC, KNN e random forest. Il modello più performante è stato la regressione logistica con un'accuratezza del 77.09% seguito dal random forest che presenta un'accuracy del 77.08%

Per la scelta degli iperparametri più performanti è stata applicata la gridsearchcv. Quest'ultima mi permette di performare i parametri che vado ad elencare per il modello.

Il metodo funziona in questo modo:

Il metodo `.GridSearchCV()` è disponibile nella classe scikit-learn **model_selection**. Può essere avviato creando un oggetto di `GridSearchCV()`:

```
clf= GridSearchCv(stimatore, param_grid, cv, punteggio)
```

In primo luogo, sono necessari 4 argomenti, ovvero **estimator** , **param_grid** , **cv** e **scoring** . La descrizione degli argomenti è la seguente:

1. **estimatore** – Un modello scikit-learn
2. **param_grid** – Un dizionario con nomi di parametri come chiavi ed elenchi di valori di parametro.
3. **punteggio** – La misura della performance. Ad esempio, ' **r2** ' per i modelli di regressione, ' **precision** ' per i modelli di classificazione.
4. **cv** – Un numero intero che è il numero di pieghe per la convalida incrociata K-fold.

GridSearchCV può essere utilizzato su diversi iperparametri per ottenere i valori migliori per gli iperparametri specificati.

Come metriche ho calcolato la ROC e AUC, precision

La curva **ROC** (Receiver Operating Characteristic curve) è una curva che mostra le prestazioni di un modello di classificazione a diverse soglie di probabilità. AUC è l'acronimo di (**A**rea **U**nder **T**he **C**urve) e misura l'intera area bidimensionale sotto l'intera curva ROC da (0,0) a (1,1).

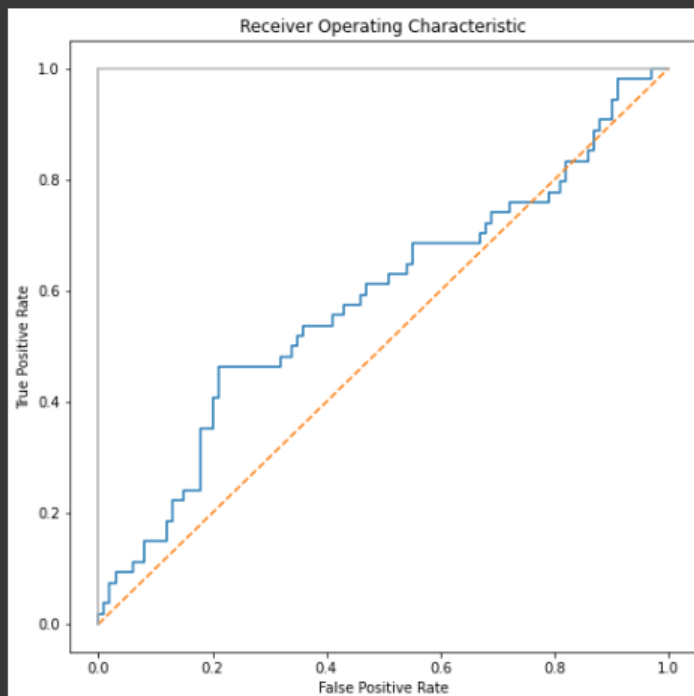
```
[ ] auc=metrics.roc_auc_score(Y_test,X_test_prediction)
    print(auc)
    #plt.show(auc)

0.607962962962963
```

In particolare dobbiamo capire e conoscere la metrica di Recall o TPR (True Positive Rate) o anche conosciuta come Sensitivity e la sua duale FPR (False Positive Rate) = $1 - \text{Sensitivity}$.

Il grafico ROC si ottiene pertanto tracciando FPR e TPR sul sistema di riferimento cartesiano, dove FPR (False Positive Rate) è tracciato sull'asse delle x e TPR (True Positive Rate) è tracciato sull'asse y per diversi valori di soglia di probabilità compresi tra 0,0 e 1,0.

```
y_score1 = classifier.predict_proba(X_test)[: ,1]
false_positive_rate1, true_positive_rate1, threshold1 = roc_curve(Y_test, y_score1)
plt.subplots(1, figsize=(8,8))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate1, true_positive_rate1)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0], c=".7"), plt.plot([1, 1], c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In definitiva se noi andassimo ad effettuare dei test di inferenza sul nostro DATASET campione con l'obiettivo di calcolare il valore totale dei TPR e degli FPR al cambiare della soglia di valutazione su una classificazione binaria otterremmo la nostra AUROC.