Introduzione

Creatori

Massimo Tubito 717440 m.tubito5@studenti.uniba.it

Ferrulli Nunzio 719508 n.ferrulli5@studenti.uniba.it

Link di GitHub per scaricare il progetto: https://github.com/FerrulliNunzio/Progetto_Icon_TF.git

Strumenti software

IDE

E' stato utilizzato l'IDE **PyCharm** sviluppato da **JetBrains** ed è un **ambiente di sviluppo integrato** (IDE), ovvero un software che, in fase di programmazione, supporta i programmatori nello sviluppo e debugging del codice sorgente di un programma.

Linguaggio utilizzato

Il software è sviluppato interamente in python che è un linguaggio di programmazione di "alto livello", orientato a oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing. **La versione di python utilizzata è la 3.8**

Librerie utilizzate

Le librerie utilizzate nel progetto sono **Sklearn** e **PySwip** (per poter utilizzare la libreria <pyswip> è necessario aver installato [swi-prolog](https://www.swi-prolog.org/Download.html)

Il Progetto

Il software simula una situazione in cui si verifica un **incidente** all'interno di un centro abitato, dove è necessario l'intervento da parte di una delle 3 **stazioni dei vigili del fuoco** presenti all'interno del centro**.** Verrà generato un evento casuale il quale verrà classificato (Algoritmi di reggressione) e gli verrà associato un valore da 1 a 4 che rappresenta il grado di emergenza. Grazie al valore di emergenza vengono definite le truppe necessarie dell'**intervento** per risolvere l'incidente. Successivamente si **determinerà** (Knowledge base) **quali delle stazioni dei vigili del fuoco ha le risorse per intervenire**. Infine per ogni stazione dei vigili del fuoco, che soddisfa i requisiti per intervenire, verrà calcolato il **percorso migliore** (Ricerca del percorso) per arrivare sul luogo dell'incidente, tenendo conto del tempo e distanza al luogo dell'evento.

Algoritmi di regressione

Per valutare la gravità dell'incidente e determinare che tipo di intervento bisognerà eseguire, utilizziamo un regressore che prenderà in input un incidente generato secondo 5 features numeriche, e restituirà valori numerici. Il regressore restituirà in output il grado di emergenza associato all'incidente.

		Boolean			
Input features	Numero dei feriti	Esplosioni	Incendi	Calamità naturali	Incidente stradale
Dominio	{0,1,2,3}	{0,1,2}	{0,1,2}	{0,1,2,3}	{0,1,2}

Numero dei feriti: ciascun valore del dominio indica un range del numero dei feriti

- 0 : 0
- 1 : 1->3
- 2 : 4->7
- 3 : 8+

Esplosioni: ciascun valore del dominio indica il grado dell'esplosione o l'assenza di essa

- 0 : indica la mancanza di esplosioni nell'incidente;
- 1 : indica la presenza di esplosioni di grado 1 nell'incidente
- 2 : indica la presenza di esplosioni di grado 2 nell'incidente

Incendi: ciascun valore del dominio indica il rischio dell'incendio o l'assenza di esso

- 0 : indica la mancanza di un incendio nell'incidente
- 1 : indica la presenza di incendi di rischio basso nell'incidente
- 2 : indica la presenza di incendi di rischio medio alto nell'incidente

Calamità naturali: ciascun valore del dominio indica il rischio di calamità naturali o l'assenza di esse

- 0 : indica la mancanza di calamità naturali nell'incidente
- 1 : indica la presenza di calamità naturali di rischio lieve nell'incidente
- 2 : indica la presenza di calamità naturali di rischio medio nell'incidente
- 3 : indica la presenza di calamità naturali di rischio alto nell'incidente

Incidente stradale: ciascun valore del dominio indica il numero di macchine coinvolte incidente stradale o l'assenza da esso

- 0 : indica la mancanza di incidente stradale nell'incidente
- 1 : indica la presenza di incidente stradale di una sola macchina nell'incidente
- 2 : indica la presenza di incidente stradale di più macchine nell'incidente

Per l'implementazione del regressore abbiamo deciso di utilizzare gli **Alberi Decisionali** in quanto possono essere visualizzati e interpretati facilmente. Richiedono poca preparazione dei dati, infatti per l'addestramento abbiamo usato un **dataset** di esempi creato da noi. Inoltre sono in grado di gestire dati sia numerici che categoriali e problemi con più output.

Nel progetto è utilizzata la libreria **Sklearn** per la costruzione del **Decision Tree Regressor** con i seguenti parametri:

```
clf_entropy = DecisionTreeRegressor(
 criterion="friedman_mse", random_state=0,
    max_depth=4, max_features='sqrt')
```

I parametri passati nella funzione rappresentano:

- criterion: può assumere i deguenti valori{“squared_error”, “friedman_mse”, “absolute_error”, “poison”}, di defaul assume il valore “squared_error”
- random_state: Controlla la casualità dello stimatore.
- max_depth: La profondità massima dell'albero.
- max_features*, int, float o {“auto”, “sqrt”, “log2”},*
  Il numero di funzioni da considerare quando si cerca la suddivisione migliore, se "sqrt", allora max_features=sqrt(n_features).

I parametri passati in input alla costruzione del classificatore sono i valori migliori trovati con dei test statistici per tenere conto dell’affidabilità del modello, usando le metriche R-quadro Score e Mean Squared Error.

I test sono stati effettuati dividendo il dataset in due parti la prima parte (75% del dataset) per il training e la seconda parte (25% del dataset) per il test. Successivamente, abbiamo usato le funzioni di metrics di sklearn: r2_score() e mean_squared_error().

Il nostro classificatore nei test ha ottenuto le seguenti valutazioni:

- Mean Squares Error = 0.2876984126984127
- R-quadro Score = 0.792687908496732

È stata eseguita un’altra valutazione usando la cross validation utilizzando la funzione cross_val() di sklearn che prende in input il nostro classificatore, il nostro dataset e una strategia di suddivisione di 5. La funzione restituisce i valori dell’accuratezza di 0,63 e deviazione di 0,17.

Knowledge base

La **knowledge base** è stata utilizzata per rappresentare le varie caserme e le varie truppe disponibili per ognuna di essa. Ogni caserma dispone di tre tipi di truppe

- Agenti: numeri di agenti del fuoco disponibili in una caserma
- Veicoli: numero dei veicoli disponibili in una caserma (si intende con veicoli l’autopompa e il fuoristrada)
- Veicoli speciali: numero dei veicoli speciali disponibili in una caserma (si intende con veicoli speciali i veicoli di supporto come l’autobotte che viene utilizzata solo in casi speciali)

Una volta classificato l’**incidente**, in base al grado di gravità verrà generato un **intervento** che andrà a definire quali sono le truppe e i mezzi necessari per essere eseguito. Questo servirà per determinare quali sono le caserme in grado di intervenire secondo le necessità di ciascun grado di emergenza descritte di seguito.

Interventi in base al grado di pericolosità:

grado 1

-numero agenti richiesti 2

-numero veicoli 2

-numero veicoli speciali 0

-tempo 40

grado 2

-numero agenti richiesti 7

-numero veicoli 5

-numero veicoli speciali 0

-tempo 30

grado 3

-numero agenti richiesti 10

-numero veicoli 7

-numero veicoli speciali 1

-tempo 25

grado 4

-numero agenti richiesti 15

-numero veicoli 10

-numero veicoli speciali 5

-tempo 20

Per l'implementazione della **knowledge base** abbiamo utilizzato **PySwip** è un bridge Python - SWI-Prolog che consente di interrogare **SWI-Prolog** nei tuoi programmi Python. Quindi la nostra base di conoscenza sarà definita in prolog.

La base di conoscenza è stata definita nel seguente modo:

```
agentiInCaserma(cas1,1).
agentiInCaserma(cas1,2).
agentiInCaserma(cas1,3).
agentiInCaserma(cas1,4).
agentiInCaserma(cas1,5).
agentiInCaserma(cas2,1).
agentiInCaserma(cas2,2).
agentiInCaserma(cas2,3).
agentiInCaserma(cas2,4).
agentiInCaserma(cas2,5).
agentiInCaserma(cas2,6).
agentiInCaserma(cas2,7).
```

```
agentiInCaserma(cas2,8).
agentiInCaserma(cas2,9).
agentiInCaserma(cas2,10).
agentiInCaserma(cas3,1).
agentiInCaserma(cas3,2).
agentiInCaserma(cas3,3).
agentiInCaserma(cas3,4).
agentiInCaserma(cas3,5).
agentiInCaserma(cas3,6).
agentiInCaserma(cas3,7).
agentiInCaserma(cas3,8).
agentiInCaserma(cas3,9).
agentiInCaserma(cas3,10).
agentiInCaserma(cas3,11).
agentiInCaserma(cas3,12).
agentiInCaserma(cas3,13).
agentiInCaserma(cas3,14).
agentiInCaserma(cas3,15).
agentiInCaserma(cas3,16).
agentiInCaserma(cas3,17).
agentiInCaserma(cas3,18).
agentiInCaserma(cas3,19).
agentiInCaserma(cas3,20).


veicoliInCaserma(cas1,1).
veicoliInCaserma(cas1,2).
veicoliInCaserma(cas1,3).
veicoliInCaserma(cas2,1).
veicoliInCaserma(cas2,2).
veicoliInCaserma(cas2,3).
veicoliInCaserma(cas2,4).
veicoliInCaserma(cas2,5).
veicoliInCaserma(cas2,6).
veicoliInCaserma(cas2,7).
veicoliInCaserma(cas3,1).
veicoliInCaserma(cas3,2).
veicoliInCaserma(cas3,3).
veicoliInCaserma(cas3,4).
veicoliInCaserma(cas3,5).
veicoliInCaserma(cas3,6).
veicoliInCaserma(cas3,7).
veicoliInCaserma(cas3,8).
veicoliInCaserma(cas3,9).
veicoliInCaserma(cas3,10).


veicoliSPInCaserma(cas1,0).
veicoliSPInCaserma(cas2,0).
veicoliSPInCaserma(cas2,1).
veicoliSPInCaserma(cas2,2).
veicoliSPInCaserma(cas3,0).
veicoliSPInCaserma(cas3,1).
veicoliSPInCaserma(cas3,2).
veicoliSPInCaserma(cas3,3).
veicoliSPInCaserma(cas3,4).
veicoliSPInCaserma(cas3,5).



agentiNecessariCas1(X):-agentiInCaserma(cas1,X).
veicoliNecessariCas1(X):-veicoliInCaserma(cas1,X).
veicoliSPNecessariCas1(X):-veicoliSPInCaserma(cas1,X).
```

```
agentiNecessariCas2(X):-agentiInCaserma(cas2,X).
veicoliNecessariCas2(X):-veicoliInCaserma(cas2,X).
veicoliSPNecessariCas2(X):-veicoliSPInCaserma(cas2,X).

agentiNecessariCas3(X):-agentiInCaserma(cas3,X).
veicoliNecessariCas3(X):-veicoliInCaserma(cas3,X).
veicoliSPNecessariCas3(X):-veicoliSPInCaserma(cas3,X).

caserma1Giusta(X,Y,Z):-
agentiNecessariCas1(X),veicoliNecessariCas1(Y),veicoliSPNecessariCas1(Z).
caserma2Giusta(X,Y,Z):-
agentiNecessariCas2(X),veicoliNecessariCas2(Y),veicoliSPNecessariCas2(Z).
caserma3Giusta(X,Y,Z):-
agentiNecessariCas3(X),veicoliNecessariCas3(Y),veicoliSPNecessariCas3(Z).
```

Ontologia:

Costanti:

atomi:

cas1,cas2,cas3: indicano la caserma a cui ci stiamo riferendo, cas1 equivale alla caserma 1, cas2 equivale alla caserma 2 e cas3 equivale alla caserma 3.

interi: sono stati utilizzati per indicare per ogni truppa il numero di truppe che la caserma può inviare per risolvere l'intervento.

Strutture:

agentiInCaserma: utilizzato per ogni caserma per definire gli agenti presenti in una caserma.

veicoliInCaserma: utilizzato per ogni caserma per definire i veicoli presenti in una caserma.

veicoliSPInCaserma: utilizzato per ogni caserma per definire i veicoli speciali presenti in una caserma.

agentiNecessariCas1, agentiNecessariCas2, agentiNecessariCas3: queste strutture sono utilizzate per verificare se dato un intero che rappresenta gli agenti necessari la caserma dispone di tale numero.

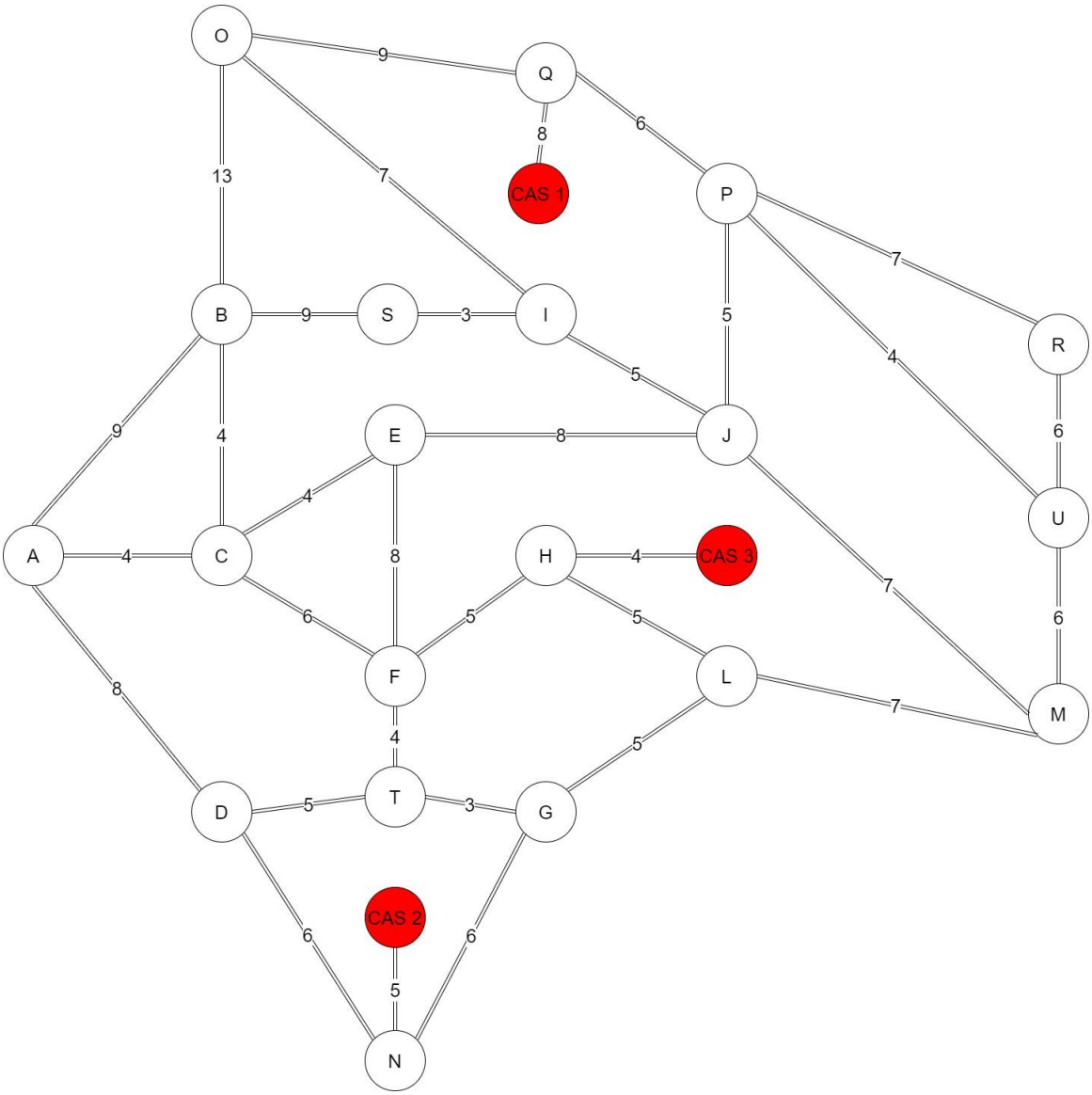
veicoliNecessariCas1, veicoliNecessariCas2, veicoliNecessariCas3: queste strutture sono utilizzate per verificare se dato un intero che rappresenta i veicoli necessari la caserma dispone di tale numero.

veicoliSPNecessariCas1, veicoliSPNecessariCas2, veicoliSPNecessariCas3: queste strutture sono utilizzate per verificare se dato un intero che rappresenta i veicoli speciali necessari la caserma dispone di tale numero.

caserma1giusta, caserma2Giusta, caserma3Giusta: queste strutture sono utlizzate per verificare se dati 3 interi che rappresentano rispettivamente agenti, veicoli e veicoli speciali necessari se la caserma dispone delle truppe necessarie.

Ricerca del percorso

Una volta trovate, le caserme che hanno le truppe necessarie all'esecuzione dell'intervento, dobbiamo trovare quali di queste sono le più vicine al luogo dell'incidente, in modo che possono intervenire nel minor tempo possibile. Perciò abbiamo deciso di trasformare tale problema in un problema di ricerca in un grafo. Poiché è importante il tempo impiegato dalle truppe di una caserma ad arrivare sul luogo dell'incidente usiamo un grafo pesato dove **ogni peso di un arco<X,Y> rappresenta i minuti necessari per arrivare dal nodo X al nodo Y**.



Utilizziamo l'algoritmo dijkstra per calcolare il percorso più breve dal nodo iniziale a ogni altro nodo del grafo e l'algoritmo uniform cost search (UCS) per ottenere il percorso più breve da un nodo iniziale a uno finale. L'algoritmo UCS prende in input il grafo, un nodo di partenza (**start**) e un nodo obiettivo (**goal**), dove il nodo di partenza è la caserma che soddisfa i requisiti per l'intervento, invece il nodo obiettivo è il nodo in cui è avvenuto l'incidente. Abbiamo deciso di utilizzare questo l'algoritmo perché volevamo calcolare il percorso a costo più basso nel grafo da un nodo iniziale a un nodo obiettivo.