|  |  |
| --- | --- |
| Immagine di alberi e di una strada tortuosa  RACCOMANDAZIONE AGRICOLA INTELLIGENTE  Corso di Ingegneria della Conoscenza A.A 2023/2024 | **Gruppo di lavoro:**  Antonio Campanozzi [matr.763605]  a.campanozzi@studenti.uniba.it  Mauro Vito Dimauro [matr.738719]  m.dimauro21@studenti.uniba.it  link al progetto: <https://github.com/AntonioCampanozzi/ICON_23-24.git> |

Indice

[**1. Introduzione** 2](#_heading=h.32hioqz)

[1.1 Strumenti 2](#_heading=h.1hmsyys)

[1.1.1 Librerie 2](#_heading=h.41mghml)

[1.1.2 Software esterni 2](#_heading=h.2grqrue)

[**2. Dataset** 2](#_heading=h.vx1227)

[2.1. Preprocessing dei dati 3](#_heading=h.3fwokq0)

[**3 Ontologia** 7](#_heading=h.1v1yuxt)

[3.1 Scelte Progettuali 7](#_heading=h.4f1mdlm)

[3.2 Classi e Data Property 7](#_heading=h.2u6wntf)

[3.3 Object Property 8](#_heading=h.19c6y18)

[3.4 Visualizzazione grafica dell’ontologia 8](#_heading=h.3tbugp1)

[3.5 Individui 9](#_heading=h.28h4qwu)

[3.6 Query 9](#_heading=h.nmf14n)

[3.7 Owlready 10](#_heading=h.37m2jsg)

[**4. Apprendimento supervisionato** 13](#_heading=h.1mrcu09)

[4.1 Scelte progettuali 13](#_heading=h.46r0co2)

[4.2 Tuning 14](#_heading=h.2lwamvv)

[4.2.1 Linear SVM 14](#_heading=h.111kx3o)

[4.2.2 K-Nearest Neighbor 15](#_heading=h.3l18frh)

[4.2.3 Support Vector Machine a kernel non lineare 15](#_heading=h.206ipza)

[4.2.4 Random Forest 16](#_heading=h.4k668n3)

[4.3 Metodi di valutazione 16](#_heading=h.2zbgiuw)

[4.4 Risultati ottenuti 17](#_heading=h.1egqt2p)

[4.4.1 Linear Support Vector Machine 17](#_heading=h.3ygebqi)

[4.4.2 K-Nearest Neighbor 19](#_heading=h.2dlolyb)

[4.4.3 Support Vector Machine non lineare 20](#_heading=h.sqyw64)

[4.4.4 Random Forest 22](#_heading=h.3cqmetx)

[4.5 Scelta del modello finale 24](#_heading=h.1rvwp1q)

[**5. Knowledge Base su Prolog** 24](#_heading=h.4bvk7pj)

[5.1 Scelte progettuali 25](#_heading=h.2r0uhxc)

[5.2 Principali funzionalità 25](#_heading=h.1664s55)

[5.3 Struttura della knowledge base 25](#_heading=h.3q5sasy)

[5.3.1 fatti 25](#_heading=h.25b2l0r)

[5.3.2 Regole 27](#_heading=h.kgcv8k)

[5.4 Esempi 29](#_heading=h.34g0dwd)

[**6.Conclusione e sviluppi futuri** 31](#_heading=h.1jlao46)

[**7. Riferimenti Bibliografici** 31](#_heading=h.43ky6rz)

# 1. Introduzione

L'obiettivo di questo progetto è quello di agevolare il lavoro agricolo, principalmente per chi si è appena approcciato al settore, andando ad aiutare dal punto di vista di scelta delle colture, in base alle loro caratteristiche.

Il caso di studio affronta 3 argomenti:

* Creazione di un’ontologia[[1]](#_heading=h.43ky6rz)
* Apprendimento supervisionato: Classificazione
* Base di conoscenza in Prolog[[2]](#_heading=h.43ky6rz) per ragionamento logico

Il tutto realizzato partendo da un dataset presente su Kaggle, adattato successivamente alle esigenze progettuali.

## 1.1 Strumenti

Il progetto è stato realizzato in Python, utilizzando come IDE Pycharm, e per il lavoro collaborativo abbiamo deciso di utilizzare GitHub come piattaforma.

### 1.1.1 Librerie

Per lo sviluppo di questo progetto sono state utilizzate le seguenti librerie:

* **Pandas[[3](#_heading=h.43ky6rz)]**: una libreria per la manipolazione e l’analisi dei dati.
* **Numpy[[4]](#_heading=h.43ky6rz)**: una libreria che permette la manipolazione degli array.
* **Sklearn[[5]](#_heading=h.43ky6rz)**: una libreria per l’apprendimento automatico.
* **Matplotlib[[6]](#_heading=h.43ky6rz)**: una libreria per la creazione di grafici.
* **Pyswip[[7]](#_heading=h.43ky6rz)**: una libreria che permette l’integrazione del linguaggio Prolog in ambiente Python.
* **Owlready2**[**[8]**](#_heading=h.43ky6rz): una libreria che permette di consultare ontologie in ambiente Python.

### 1.1.2 Software esterni

* **Protégé[[9]](#_heading=h.43ky6rz)**: software che permette la creazione e manipolazione delle ontologie, con possibilità di effettuare query, tramite l’utilizzo di Dl query.
* **SWIProlog[[10]](#_heading=h.43ky6rz)**: ambiente di programmazione per il linguaggio Prolog

# 2. Dataset

Il dataset[[11]](#_heading=h.43ky6rz) che abbiamo selezionato da **Kaggle** contiene Dati dell'Organizzazione delle Nazioni Unite per l'Alimentazione e l'Agricoltura (FAO).

Il dataset è composto dalle seguenti features:

|  |
| --- |
| * **Nitrogen**: rateo di nitrogeno contenuto nel terreno * **Phosphorus**: rateo di fosforo contenuto nel terreno * **Potassium**: rateo di potassio contenuto nel terreno * **Temperature**: temperatura del terreno in gradi Celsius * **Humidity**: umidità in percentuale * **pH\_Value**: valore di ph del terreno * **Rainfall**: livello di piogge in mm   E dalla label:   * **Crop**: le colture consigliate in base alle caratterisitche del terreno, vi sono 22 possibili valori. |

## 2.1. Preprocessing dei dati

il dataset è stato preprocessato seguendo suddetto iter:

1. Analisi Esplorativa dei Dati

Inizialmente sono state analizzate le distribuzioni delle features:

Immagine che contiene testo, schermata, diagramma, Diagramma

Descrizione generata automaticamenteImmagine che contiene testo, diagramma, schermata, Diagramma

Descrizione generata automaticamente

Immagine che contiene testo, schermata, diagramma, Diagramma

Descrizione generata automaticamenteImmagine che contiene testo, diagramma, schermata, Diagramma

Descrizione generata automaticamente

Immagine che contiene testo, diagramma, schermata, Diagramma

Descrizione generata automaticamenteImmagine che contiene testo, diagramma, schermata, Diagramma

Descrizione generata automaticamente

Immagine che contiene testo, diagramma, schermata, Diagramma

Descrizione generata automaticamente

Non ci aspettavamo distribuzioni di dati perfettamente gaussiane, ma le feature *Temperature* e *pH\_value* godono di questa proprietà, l’unica feature che potrebbe avere una distribuzione problematica è *Potassium,* data la netta divisione delle colonne dei valori. Abbiamo tracciato un *boxplot* per monitorare gli *outliers:*

Immagine che contiene schermata, diagramma, testo, Rettangolo

Descrizione generata automaticamente

Prima di andare ad agire su di essi abbiamo voluto controllare la distribuzione delle classi per essere sicuri di non andare ad alterarla rimuovendo righe dal dataset:  
Immagine che contiene Policromia, cerchio, creatività

Descrizione generata automaticamente

Il dataset è perfettamente bilanciato, sospettiamo che prima di essere stato pubblicato abbia subito dell’ *oversampling*, vale a dire la generazione di esempi sintetici per ottenere classi bilanciate. Visto questo enorme vantaggio, abbiamo deciso di rimandare l’eliminazione degli outlier all’eventualità che le predizioni tramite apprendimento automatico diano risultati indesiderati.

Successivamente abbiamo tracciato un’*heatmap,* rappresentazione grafica che mostra la correlazione tra le features del dataset:

Immagine che contiene schermata, Policromia, quadrato, diagramma

Descrizione generata automaticamente

I punti con colori più caldi rappresentano correlazioni forti, più il colore è freddo, maggiormente gli elementi sono debolmente correlati.

* *Phosporus* e *potassium* hanno una correlazione molto forte (0.74). Quando il fosforo aumenta, anche il potassio tende ad aumentare.
* *Nitrogen* e *Phosphorus* hanno una correlazione negativa debole (-0.231460). Quando l'azoto aumenta, il fosforo tende a diminuire leggermente.
* *Temperature* e *Humidity* hanno una correlazione positiva debole (0.205320). Quando la temperatura aumenta, l'umidità tende a aumentare leggermente
* *Rainfall* e *Potassium* hanno una correlazione molto bassa (-0.053461), indicando che non c'è una relazione lineare significativa tra la pioggia e il potassio.
* *pH\_Value* e *Temperature* hanno una Correlazione quasi inesistente (-0.017795), indicando che non c'è una relazione lineare significativa tra il valore del pH e la temperatura.

1. elaborazione dei Dati

Per preparare i dati per l'addestramento del modello, abbiamo effettuato le seguenti operazioni:

* Codifica delle Etichette: La label delle colture, contenente dati categorici, è stata codificata utilizzando il `LabelEncoder`, che assegna un numero univoco a ogni valore unico della label.
* Normalizzazione: Le features contengono modalità di misurazione differenti, sono state normalizzate a un’unica unità nell'intervallo 0-1 utilizzando “MinMaxScaler”, una delle forme più di normalizzazione più usate. opera sottraendo a ogni elemento di una data feature il valore minimo di essa e divide il risultato per la differenza tra massimo e minimo.

Immagine che contiene Carattere, linea, bianco, testo

Descrizione generata automaticamente

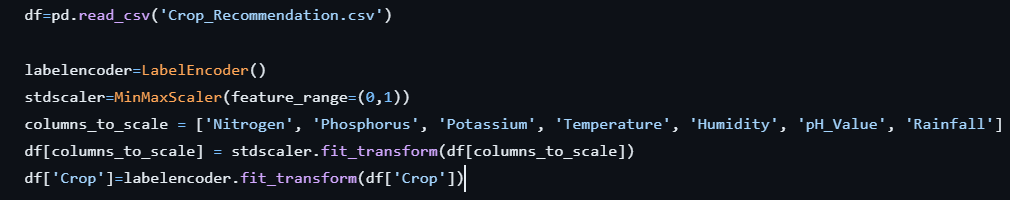


Figura 1 Codifica delle etichette e normalizzazione delle features

1. Suddivisione del Dataset

Il dataset è stato diviso in caratteristiche (features) e etichette (labels). Successivamente, è stato effettuato uno split dei dati in un set di addestramento e uno di test, con una proporzione del 20% per il test, utilizzando `train\_test\_split`.

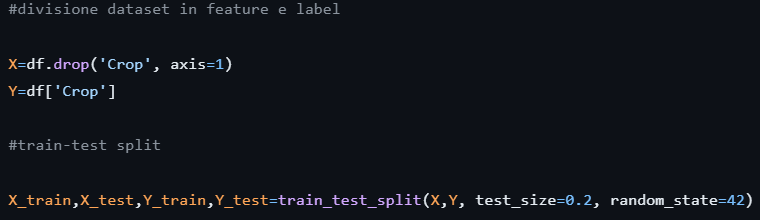


Figura 2 suddivisione del dataset in train e test

# 3 Ontologia

Un’ontologia è un modello di conoscenza che consente di rappresentare in modo univoco le informazioni relative ad un determinato dominio. Durante l’analisi del dominio abbiamo indentificato i concetti principali, le relazioni e le proprietà che caratterizzano il dominio. Successivamente questi concetti, relazioni e proprietà sono state formalizzate mediante il linguaggio di rappresentazione formale OWL[[12]](#_heading=h.43ky6rz) (Ontology Web Language).

## 3.1 Scelte Progettuali

Abbiamo deciso di utilizzare e creare una ontologia per rappresentare in modo formale il nostro dominio di conoscenza, per dare una visione chiara all’utente e interrogare agevolmente il sistema e carpire le informazioni sui dati utilizzati. Per la realizzazione dell’ontologia abbiamo deciso di utilizzare l’editor di ontologie Protégé.

## 3.2 Classi e Data Property

Analizzando il dominio abbiamo deciso di rappresentare come classi dell’ontologia:

* **Crop**: rappresenta l’insieme delle colture a cui abbiamo associato queste Data Property:
* **CropID**: rappresenta univocamente una determinata coltura.
* **CropName**: rappresenta il nome reale di una determinata coltura.

* **Fattori\_ambientali**: rappresenta tutte le caratteristiche ambientali in cui la coltura ha prosperato, a cui abbiamo associato queste Data Property:
* **Humidity\_percentage**: rappresenta la percentuale necessaria di umidità per la prosperità della coltura.
* **pH\_Value**: rappresenta il valore del ph del terreno necessaria per la coltura.
* **Rainfall\_mm**: rappresenta il livello di pioggia in cui ha prosperato una coltura.
* **Temperature\_celsius**: rappresenta i gradi in cui ha prosperato una coltura.

* **Fattori\_nutrizionali**: la classe rappresenta tutte le caratteristiche nutrizionali che aiutano lo sviluppo di una determina coltura a cui abbiamo associato queste Data Property:
* **Nitrogen**: rappresenta la quantità di azoto necessaria per la crescita vegetativa, ossia sviluppo delle foglie e degli steli.
* **Phosphorus**: rappresenta la quantità di fosforo necessaria per il trasferimento di energia nelle cellule delle piante.
* **Potassium**: rappresenta la quantità di potassio necessaria per la sintesi delle proteine, nella fotosintesi.

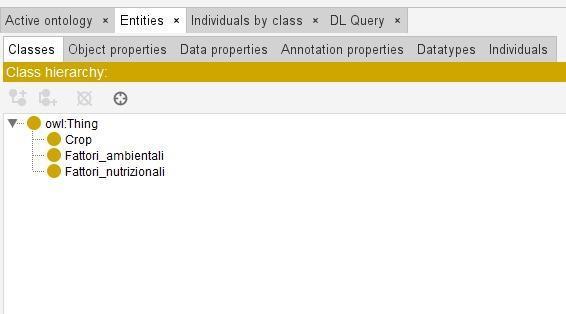


Figura 1 Classi definite

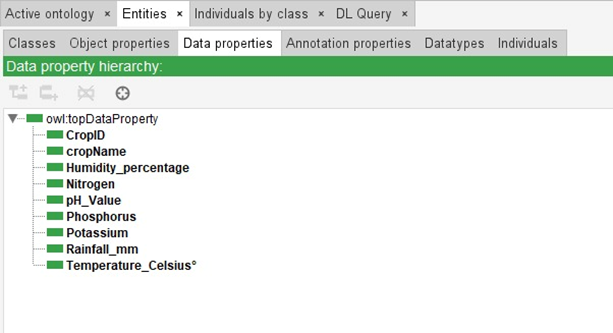


Figura 2 Data Property definite

## 3.3 Object Property

Una object property permette di mettere in relazione due individui, sia di classi distinte che della stessa classe.

Tra le classi abbiamo definito delle relazioni che rappresentano come queste interagiscono tra di loro. Le relazioni definite sono:

* **hasEnvironmentalFactor**: che mette in relazione la classe Crop ai Fattori\_ambientali
* **hasNutritionalFactor**: che mette in relazione la classe Crop ai Fattori\_nutrizionali

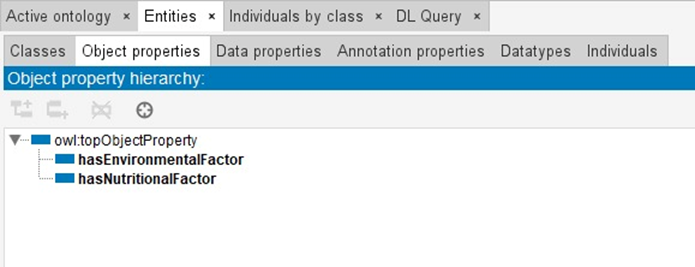


Figura 3 Object Property definite

## 3.4 Visualizzazione grafica dell’ontologia

Per visualizzare l’ontologia abbiamo utilizzato il tool online WebVowl[[13]](#_heading=h.43ky6rz)[:](#_heading=h.43ky6rz)

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

## 3.5 Individui

Infine, abbiamo definito quelli che sono gli individui, che rappresentano alcuni esempi di Crop, e individui che rappresentano i fattori ambientali, e individui che rappresentano i valori nutrizionali.

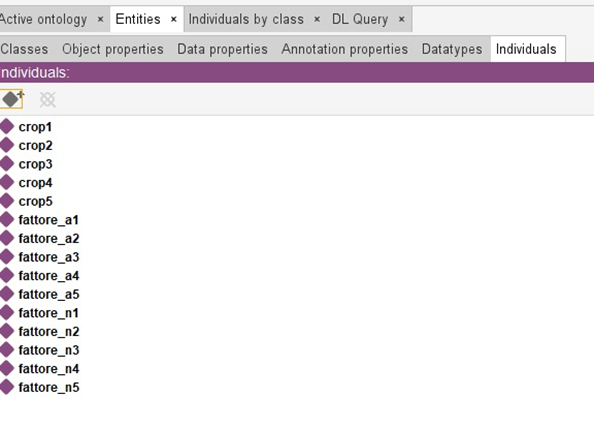


Figura 4 individui definiti

## 3.6 Query

Successivamente sono state formulate query semplici per interrogare l’ontologia, tramite DL Query e Owlready2.

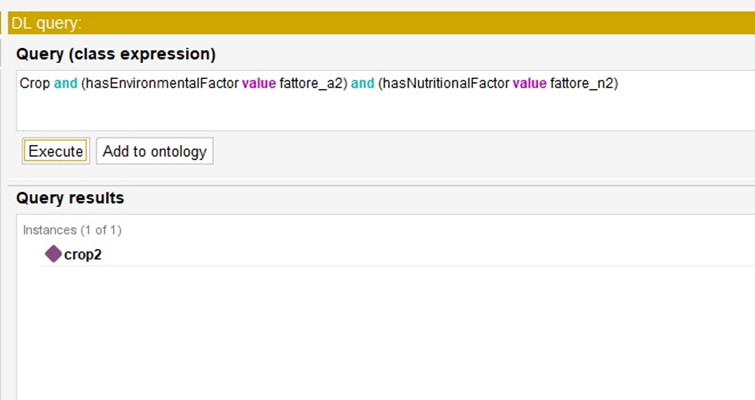
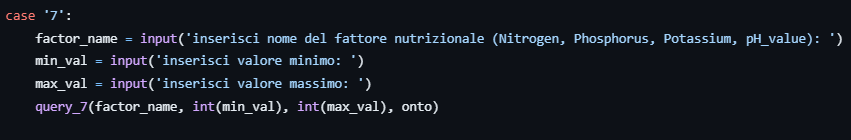
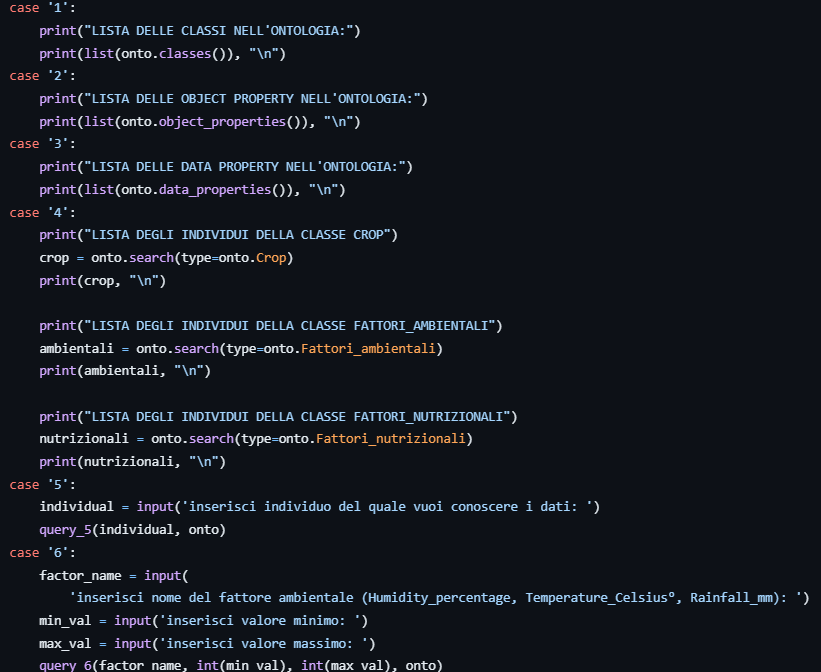


Figura 5 esempio di query su protégé

## 3.7 Owlready

Per la consultazione in Python dell’ontologia è stata utilizzata la libreria Owlready2, che permetta di consultare l’ontologia, e soprattutto interrogarla.

Tramite queste istruzioni è possibile ottenere tutte le informazioni necessarie alla consultazione dell’ontologia, ottenendo per l’appunto tutte le classi, Object property, Data property, tutti gli individuals delle 3 classi sopra citate.



ottenendo come input tutte le liste:



Figura 6 liste di classi e object property

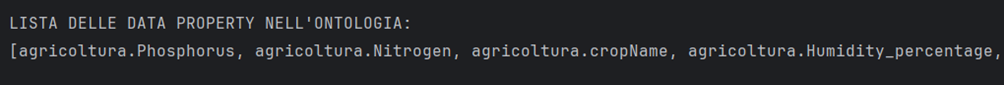


Figura 7 lista delle Data Property

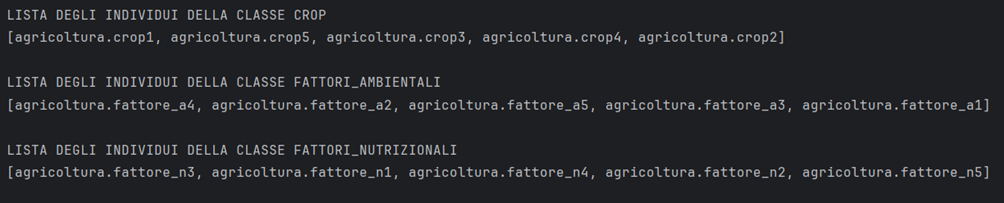
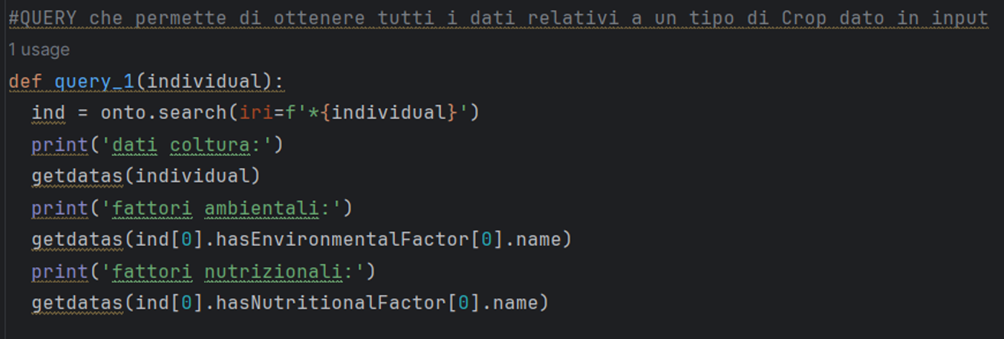


Figura 8 liste degli individui

Con il seguente codice abbiamo invece creato una query che permette di ottenere tutti i valori contenuti nelle Data Property per un singolo individuo Crop.





Ottenendo i seguenti risultati:

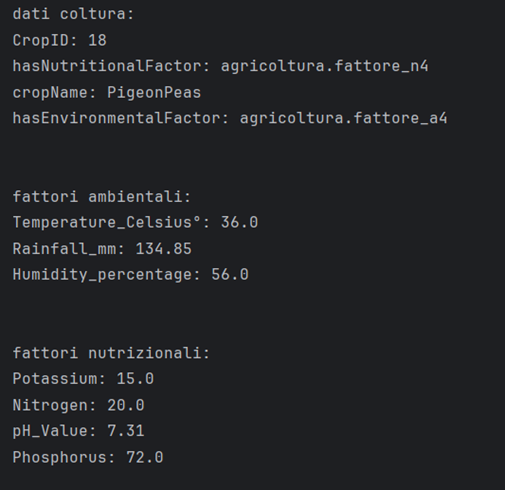
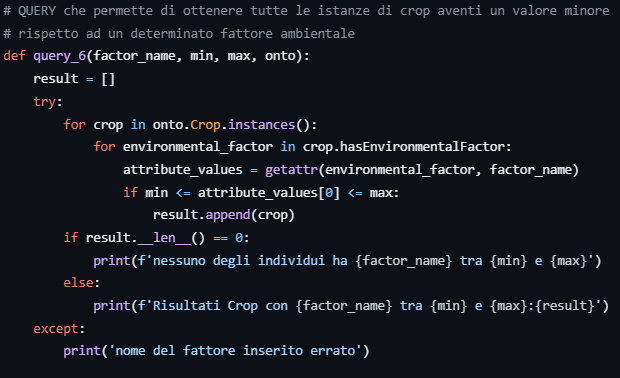
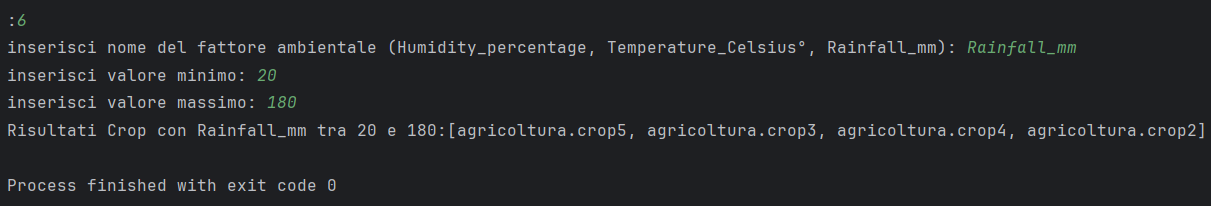


Figura 9 dati delle data Property relativi a un singolo Crop

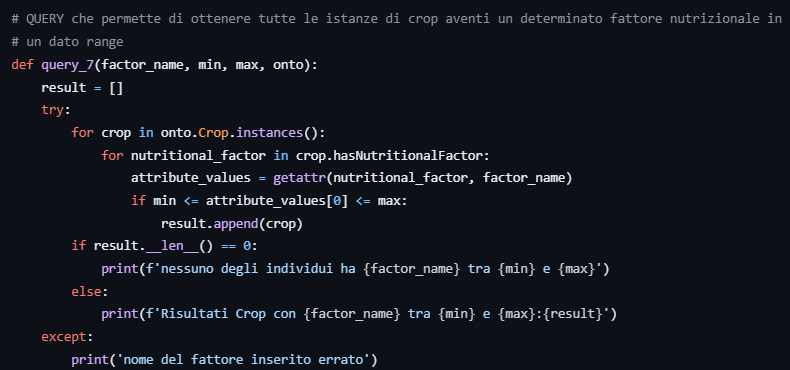
Con questo codice invece andiamo a creare una query che permette di ottenere tutti i crop che possiedono un fattore ambientale a scelta dell’utente, inferiore ad un determinato valore sempre a scelta dell’utente.



Ottenendo come risultato in questo esempio, ricercando tutte le colture con un valore di “Rainfall\_mm” compreso tra 20 e 180::



Con questo codice invece andiamo a creare una query che permette di ottenere tutti i crop che possiedono un fattore nutrizionale a scelta dell’utente, inferiore ad un determinato valore sempre a scelta dell’utente.



Ottenendo come risultato in questo esempio, ricercando tutte le colture con un valore di “Nitrogen” compreso tra 20 e 60:



# 4. Apprendimento supervisionato

L’apprendimento supervisionato[[14]](#_heading=h.43ky6rz) è un approccio dell’apprendimento automatico che si serve di dati etichettati in dati di input, detti **features**, e dati di output, detti **labels**. Il dataset viene diviso in due parti non uguali:

* Un set di **train** con cui la macchina apprende i pattern che legano l’input all’output
* Un set di **test**, su cui si misura la capacità della macchina di inferire su dati non visti

Tramite le feature la macchina predice i valori delle label, nel caso di valori continui si sta affrontando un problema di **regressione**, nel caso i possibili valori della label siano un numero finito, si parla di problema di **classificazione**.

## 4.1 Scelte progettuali

Abbiamo deciso di valutare le performance su 4 modelli diversi, l’iter ci è stato suggerito dalla linea guida ufficiale di *sklearn[[](#_heading=h.43ky6rz)*[15*]*:](#_heading=h.43ky6rz)

1. **Support Vector Machine a kernel lineare**[**[16]**](#_heading=h.43ky6rz): si tratta di un algoritmo che trova l’iperpiano che separa le classi con il massimo margine possibile. Per margine si intende la distanza tra l’iperpiano e i punti che più vicini all’iperpiano stesso, detti support vectors. Il kernel lineare suppone che le classi siano linearmente separabili. La sua scelta come modello di classificazione iniziale è utile per valutare una baseline di separabilità delle varie classi
2. **K-Nearest Neighbor[[17]](#_heading=h.43ky6rz)**: il Knn rientra nel case-based learnig, dove gli esempi di training non vengono usati per allenare il modello a trovare dei pattern, ma vengono conservati e, fornito un dato non visto, vengono trovati gli esempi più vicini ad esso, e, nel caso del knn per classificazione, il valore della label predetta sarà quella che, tra i k vicini individuati, risulta la più presente. Qui vi è una forte enfasi sulla località dei dati
3. **Support Vector Machine a kernel non lineare**: in questo caso non viene supposta la lineare separabilità dei dati. Il suo utilizzo è atto a catturare la complessità e la eventuale non linearità dei dati
4. **Random Forest[[18]](#_heading=h.43ky6rz)**: utilizza più alberi di decisione per effettuare le predizioni. Per assicurare la robustezza del modello, gli alberi effettuano le predizioni in maniera diversa. Si può scegliere di fargli predire su subset diversi di feature o di dati. La random forest si rivela molto robusta su dati complessi, non lineari e con outliers, la sua versatilità è una delle qualità principale per il quale viene scelto

Su ognuno di questi modelli è stata eseguita una fase di *tuning* volta a trovare i migliori iperparametri per il caso specifico del dataset usato, successivamente sono state eseguite sia valutazioni su singolo sample, utilizzando come misure *matrici di confusioni*  e *classification reports*. Queste sono state affiancate a valutazioni incrociate (*cross validation*) dove è stata presa in considerazione media deviazione standard e varianza, per avere un’overview che non sia dipendente da un singolo run e da una singola divisione in train e test set. non da meno è l’ottenimento di dati più attendibili su cui diagnosticare la presenza o meno di *overfitting.*

## 4.2 Tuning

Per tuning si intende il processo di ottimizzazione dei parametri di un modello o algoritmo di machine learning al fine di migliorarne le prestazioni predittive su nuovi dati. Abbiamo deciso di usare la *Grid Search Cross-Validation*[[19]](#_heading=h.43ky6rz)*:* quello implementato da scikit-learn prende in input un estimatore, una griglia di iperparametri con i rispettivi valori su cui fare tuning, e per ognuno di loro restituisce quello migliore

### 4.2.1 Linear SVM

Parametri presi in cosiderazione:

**dual**: impostato a ‘False’, perché il problema di classificazione affrontato nel nostro caso non è su classe binaria, ma n-aria

**C**: parametro di regolarizzazione, controlla quanto si penalizzano le violazioni dei margini. Uno C più elevato può portare a una maggiore aderenza sui dati di train, al contrario, uno C molto piccolo invece è più bonario sulle violazioni di margine. I valori scelti sono [0.01, 0.1, 1, 10, 100],

**'loss'**: definisce la funzione di loss, che può essere standard (hinge) o quadrata (squared\_hinge).

**'tol'**: criterio di tolleranza per lo stop nella fase di training, i valori scelti sono [1e-4, 1e-3, 1e-2, 1e-1, 1],

**'max\_iter'**: numero massimo di iterazioni da eseguire nella fase di training, i valori scelti sono [3000, 4000, 5000, 10000, 20000, 50000]

Risultati ottenuti:

MIGLIORI IPERPARAMETRI PER LINEARSVC:

{

  'C': 100,

  'loss': 'squared\_hinge',

  'tol': 0.01,

  'max\_iter': 3000}

### 4.2.2 K-Nearest Neighbor

Parametri presi in considerazione:

**"n\_neighbors":** numero di vicini da prendere in considerazione per la predizione, i valori tra cui scegliere sono [1, 3, 5, 7, 9, 12, 15],

**"weights":** weight function usata nella predizione, se *uniform* i punti del vicinato hanno peso equo, se *distance* i vicini più prossimi al punto da predire avranno un'influenza maggiore.

**"metric":** metriche usate per misurare la distanza tra i punti, abbiamo scelto tra minkowski(la distanza euclidea) e *manhattan,*(la distanza di manhattan), non abbiamo scomodato la distanza del coseno, i dati non sono né sparsi né hanno una dimensionalità così alta.

**"algorithm":** algoritmo usato per computare i nearest neighbor, i valori tra cui scegliere sono ["ball\_tree", "kd\_tree", "brute"],

**"leaf\_size":** dimensioni delle foglie nel caso l’algoritmo scelto sia *ball\_tree* o *kd\_tree.* I valori tra cui scegliere sono [20, 30, 40]

Risultati ottenuti:

MIGLIORI IPERPARAMETRI PER KNN:

{'algorithm': 'ball\_tree',

'leaf\_size': 20,

'metric': 'manhattan',

'n\_neighbors': 3,

'weights': 'distance'}

### 4.2.3 Support Vector Machine a kernel non lineare

Parametri presi in considerazione (per questione di semplicità non viene ripetuta la spiegazione dei parametri ustilizzati anche nella SVM lineare):

**'C'**: valori presi in considerazione [0.1, 1, 10, 100, 1000],

**'kernel'**: tipologia di funzione kernel utilizzata per l’algoritmo, non abbiamo inserito pre-computed tra i valori perché necessita in input una matrice quadrata di esempi, nel nostro caso il set di train dato in input è una matrice rettangolare. Ne consegue che i valori presi in esame sono ['linear', 'rbf', 'poly', 'sigmoid'].

**'gamma'**: coefficiente kernel per le funzioni *rbf, poly, sigmoid*. Se pari a *scale* gamma sarà *1 / (n\_features \* X.var()),* , se pari ad *auto* gamma sarà *1 / n\_features*.

**'degree'**: grado della funzione kernel *poly*, ignorato da tutti gli altri kernel, i valori presi in considerazione sono [0, 1, 2, 3, 4, 5, 6]

Risultati ottenuti:

MIGLIORI IPERPARAETRI PER SVC:

{

  'C': 1,

  'kernels': 'poly',

  'gamma': 'scale',

  'degree': 2}.

Abbiamo anche la prova del 9 che la funzione kernel più ottimale non sia quella lineare, quindi è maggiore la probabilità che i dati non siano linearmente separabili

### 4.2.4 Random Forest

Parametri presi in considerazione:

**'criterion'**: funzione utilizzata per misurare la qualità dello split, i valori presi in considerazione sono ['gini', 'entropy', 'log\_loss'],

**'n\_estimators'**: numero di alberi di decisione da usare nella random forest, i valori presi in considerazione sono [50,100, 200],

**'max\_depth'**: profondità massima degli alberi di decisione, i valori presi in considerazione sono [None, 5, 10], se *max\_depth*=*None* i nodi vengono espansi fino a quando tutte le foglie sono pure o fino a quando tutte le foglie contengono meno di *min\_samples\_split* campioni.

**'min\_samples\_split'**: il minimo numero di sample necessari allo split di un nodo interno, i valori presi in considerazione sono [2, 5, 10, 20],

**'min\_samples\_leaf'**: il minimo numero di sample necessari per rendere tale un nodo foglia, i valori presi in considerazione sono [1, 2, 5, 10, 20]

Risultati ottenuti:

MIGLIORI IPERPARAMETRI PER LA RANDOM FOREST

{'criterion': 'gini',

'n\_estimators': 100,

'max\_depth': 10,

'min\_samples\_split': 10,

'min\_samples\_leaf': 1}

## 4.3 Metodi di valutazione

Una volta trovati gli iperparametri per ognuno dei modelli, essi sono stati allenati tramite il metodo *fit*(X\_train,Y\_train). Segue poi la validazione del modello su dati non visti. In questo paragrafo varranno elencate le metodologie utilizzate.

**Classification report:**

è la metrica di valutazione più comunemente utilizzata. Essa calcola la performance ottenuta su una singola divisione di train e test per ogni run. I parametri che vengono riportati sono

* Accuracy
* Precision
* Recall
* F1-score

È buona per avere una prima visione di come il modello ha predetto, ma va tenuto presente che il risultato ottenuto è fortemente dipendente dalla divisione ottenuta

**Matrice di confusione:**

La matrice di confusione è un’altra metrica di valutazione utilizzata per valutare le prestazioni del modello su una split alla volta. È una tabella a doppia entrata le cui:

* Righe rappresentano le etichette reali delle classi
* Colonne rappresentano le etichette predette delle classi

Un indicatore caratterizzante di una buona matrice di confusione è la diagonale principale molto popolata. Abbiamo scelto questa metrica per avere una visione del numero preciso di predizioni corrette per ogni classe e dove andassero le predizioni sbagliate. Resta il problema presente nel classification report, di fatti, per valutare la performance globale di ogni singolo modello abbiamo utilizzato altri criteri

**Cross validation**[[20]](#_heading=h.43ky6rz)

La cross validation valuta il modello eseguendo più iterazioni della pipeline split train\_test, prediction su test set. Il dataset è diviso in k sottogruppi, detti *fold*, k-1 fold sono usati come set di train, 1 come set di test.

Abbiamo optato per una k-fold stratificata, differente da quella standard perché tende a mantenere un certo equilibrio tra le classi in ogni fold, un’ulteriore misura precauzionale per un dataset già comunque eccellentemente bilanciato. Il numero di k è stato lasciato al valore di default (5) per mantenere la divisione tradizionale 80% train e 20% test.

Per la cross-validation abbiamo riportato le medie, le deviazioni standard, le varianze per ogni fold e sia per train che per test delle seguenti misure:

* Accuracy: misura l’accuratezza delle predizioni del modello, restituisce un valore compreso tra 0 e 1 (1 rappresenta predizioni accurate al 100%)
* Precision: la proporzione di esempi correttamente classificati come positivi tra tutti gli esempi che sono stati classificati come positivi.
* Recall: la proporzione di esempi correttamente classificati come positivi tra tutti gli esempi effettivamente positivi.
* F1-score: media armonica di precision e recall

Per tutte le misure escluse l’accuracy è stata considerato l’approccio weighted e macro:

* Macro-averaging, ogni classe viene considerata di equo peso
* Weighted-averaging: le classi hanno un peso proporzionale alla loro frequenza nel set

Volevamo vedere come cambiasse la performance nel caso fossero presenti nei set classi che, per natura della divisione, apparissero con meno forza

**Learning curve**[[21]](#_heading=h.43ky6rz)

Grafico che mostra come cambia la performance di train e test(o validazione, ma non è questo il caso) all’aumentare degli esempi. Molto utile per diagnosticare overfitting o underfitting. Ci serviva un metodo per essere maggiormente sicuri che i modelli non fossero vittima di overfitting, questo perché nonostante i risultati fornissero dalla cross-validation siano abbastanza solidi, non è in assoluto certo dire che una differenza tra deviazione standard, o varianza tra train e test sia piccola o grande, l’ordine di grandezza dei valori potrebbe trarre in inganno

## 4.4 Risultati ottenuti

Di seguito sono riportati i risultati ottenuti su ognuno dei modelli presi in esame e la loro interpretazione.

### 4.4.1 Linear Support Vector Machine

Classification Report e Matrice di confusione:

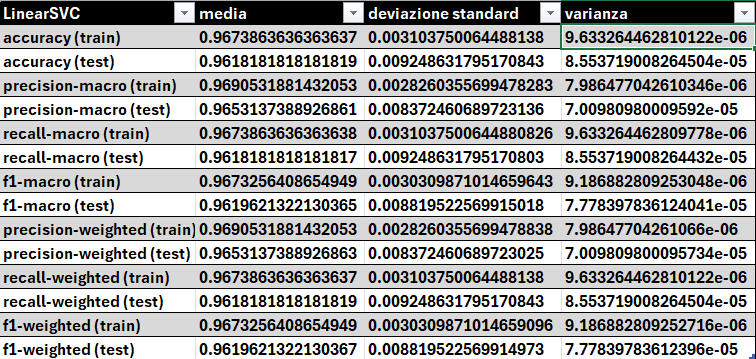
Immagine che contiene testo, schermata

Descrizione generata automaticamenteImmagine che contiene schermata, testo, Policromia, linea

Descrizione generata automaticamente

La performance su questo test set è molto buona, abbiamo un’accuracy del 96%, la matrice di confusione ha pochi valori sparsi e un ottimo raggruppamento sulla diagonale principale, la maggior parte delle classi ha predizioni perfette

Cross validation



Tra train e test c’è poca differenza sulle medie, come ci si poteva aspettare, i valori sono leggermente più alti sul set di allenamento, vi sono differenze leggermente più marcate sulla deviazione standard, leggero il delta tra le varianze, che varia solo di un ordine di grandezza (da 10^-6 a 10^-5). Le performance macro e weighted sono praticamente identiche, i vari fold sono bilanciati

Curve di apprendimento

Immagine che contiene testo, diagramma, linea, Diagramma

Descrizione generata automaticamenteImmagine che contiene testo, linea, Diagramma, diagramma

Descrizione generata automaticamente

Le curve sono abbastanza distanti fino ai 400 esempi e si uniscono, ma lentamente verso la fine, nonostante le performance di partenza media dei test set non siano affatto negative, c’è un divario non indifferente, potrebbe essere un indicatore di overfitting. Supporre una lineare separabilità dei dati per questo dataset non sembra essere la mossa migliore

### 4.4.2 K-Nearest Neighbor

Prima di mostrare i risultati è doveroso fare un appunto: il knn, come detto in precedenza, è un modello di case learning, esso non si allena su dati di train, ma li va ad usare per trovare i neighbor, di conseguenza non ci è sembrato sensato tracciare curve di apprendimento, poiché la parte di train sarebbe stata una retta parallela, né riportare i valori di train nella cross validation (sono tutti uno con deviazione standard e varianza 0). Ciò che ci è interessato vedere è quanto i vicini trovati portassero a predizioni corrette.

Classification report e matrice di confusione:

Immagine che contiene testo, schermata, numero

Descrizione generata automaticamenteImmagine che contiene schermata, testo, Policromia, modello

Descrizione generata automaticamente

La preformance è leggermente migliore, nel classification report la classe 10, presente in numero evidentemente minore rispetto alle altre, ha predizioni più deboli. La matrice di confusione è ancora più concentrata sulla diagonale principale

Cross Validation

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Anche in questo caso i fold sembrano decisamente bilanciati, le performance medie sul set di test migliorano rispetto al modello analizzato in precedenza, come riportato in precedenza, data la natura del modello stesso, tutto ciò che possiamo limitarci a dire è che il modello è molto solido nelle predizioni, sinonimo di una forte località dei dati. Ogni classe ha un’identità molto forte e la metrica utilizzata si è rivelata molto efficace

### 4.4.3 Support Vector Machine non lineare

Classification Report e matrice di confusione:

Immagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene schermata, testo, Policromia, modello

Descrizione generata automaticamente

Su singolo run sembra non ci siano evidenti miglioramenti. Nella matrice di confusione abbiamo una performance un po' debole per la classe 20, ma resta l’unico aspetto che risalta, nel resto dei casi non vi sono più di 2 predizioni errate

Cross Validation

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Performance sicuramente migliorata rispetto ai modelli precedenti , anche se lievemente, sulle medie, diminuisce il gap train-test sulle deviazioni standard e le varianze, si nota inoltre che i valori sono decisamente inferiori rispetto alla linear SVM. Ci sembra improbabile la presenza di overfitting.

Curve di apprendimento

Immagine che contiene testo, linea, Diagramma, diagramma

Descrizione generata automaticamente

Immagine che contiene testo, linea, Diagramma, diagramma

Descrizione generata automaticamente

Le curve di errore e accuratezza mostrano una forte vicinanza fin da subito tra train e test, nel caso degli errori in alcuni punti le curve coincidono perfettamente, questo rafforza la nostra tesi di assenza di overfitting. È evidente che i dati all’inizio non sono perfettamente separabili linearmente

### 4.4.4 Random Forest

Classification report e matrice di confusione:

Immagine che contiene testo, menu, schermata

Descrizione generata automaticamenteImmagine che contiene schermata, testo, Policromia, modello

Descrizione generata automaticamente

Su questo specifico test set i valori rasentano la performance massima ottenibile, la solidità della random forest ignora eventuali classi sbilanciate. La matrice di confusione contiene quasi tutti i valori sulla diagonale principale.

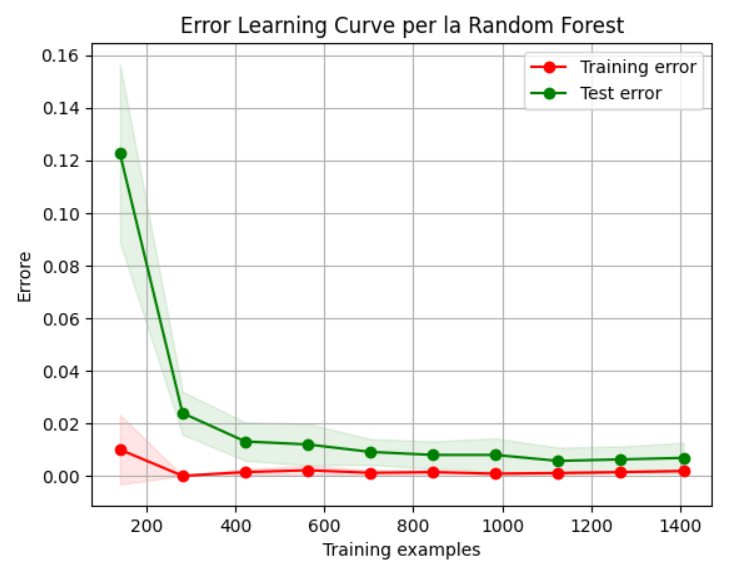
Cross Validation

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Decisamente migliori le medie, le deviazioni standard e le varianze delle misure calcolate sul set di train diminuiscono notevolmente rispetto ai modelli precedenti, ma sul test vi è poco cambiamento, questo ci porta a gap non effettivamente maggiori, ma misurati su ordini di grandezza differenti. Nonostante i discostamenti siano molto piccoli, l’oscillazione soprattutto sulla varianza ci ha portato a pensare a un possibile overfitting.

Curve di apprendimento





La conformazione delle curve non suggerisce overfitting. Rispetto all’SVM abbiamo notato una distanza leggermente superiore tra train e test nelle curve di accuratezza.

## 4.5 Scelta del modello finale

In ultima analisi abbiamo scelto la random forest. Nonostante le curve siano leggermente più vicine nella SVM, queste sono entrambe su medie leggermente più basse, inoltre abbiamo optato per un modello che sia più resistente agli outliers, questo perché nel caso si voglia predire su un punto mai visto, ci si può aspettare che i valori del terreno non siano necessariamente circoscritti a quelli presenti nel dataset, e un modello con medie molto alte e ,tutto sommato, un’ottima generalizzazione sul test set si dimostrerebbe decisamente molto solido. Ne consegue che sul sistema, per effettuare predizioni, viene usata la random forest pre addestrata

# 5. Knowledge Base su Prolog

La knowledge base in Prolog rappresenta un dominio di conoscenza e risolve problemi su di esso tramite l’*inferenza logica*[[22]](#_heading=h.43ky6rz). Si compone di **fatti** e **regole:**

* Fatti: affermazioni sempre vere, utilizzate per costruire la base di inferenza della KB.
* Regole: affermazioni vere se e soltanto se tutte le affermazioni e/o sotto-regole che la compongono risultano essere vere

Una KB prolog funge da sistema esperto, simula quindi il dialogo e il confronto con un profondo conoscitore di un dato dominio.

## 5.1 Scelte progettuali

Abbiamo optato per una knowledge base sviluppata su prolog principalmente per la potenza che questo linguaggio di programmazione ha. Su Prolog, infatti, è possibile condurre molto facilmente inferenze logiche abbastanza complesse ed elaborate. Nel nostro caso questa knowledge base va vista come successiva al processo di predizione tramite apprendimento automatico, è una integrazione che è possibile fare per ottenere più informazioni riguardo le colture da seminare, valutate anche su diversi parametri. Per questi ultimi, è stata decisa una forma più testuale e comprensibile che fosse maggiormente user-friendly.

## 5.2 Principali funzionalità

All’utente viene presentata:

* la possibilità di ottenere un set di colture consigliate in base a scelte di carattere economico
* la possibilità di ottenere un set di colture in base alla conformazione del luogo dove la semina avverrà.
* la possibilità di trovare la coltura più simile a una data in input, utile nel caso si cerchi una coltura da affiancare a una già in coltivazione che non richieda fattori ambientali e nutrizionali troppo diversi

## 5.3 Struttura della knowledge base

### 5.3.1 fatti

La kb creata si serve di 5 tipologie di fatti:

* Fatto ‘crop’: rappresenta la coltura. Ha i seguenti parametri:
  + ‘Nome’: nome della coltura
  + ‘Azoto’: livello medio di azoto individuato nei terreni dove la coltura è prosperata
  + ‘fosforo’: livello medio di azoto individuato nei terreni dove la coltura è prosperata
  + ‘potassio’: livello medio di potassio individuato nei terreni dove la coltura è prosperata
  + ‘Temperatura\_max’: temperatura massima mai sopportata dalla coltura
  + ‘Temperatura\_min’: temperatura minima mai sopportata dalla coltura
  + ‘Ph’: livello di ph che in media è stato trovato nei terreni dove è stata seminata la coltura
  + ‘Umidità’: percentuale di umidità che in media è stata trovata nelle zone dove è stata seminata la coltura
  + ‘Piogge’: livello di piogge, misurato in mm mensili, che in media è stato trovato nelle zone dove è stata seminata la coltura
  + ‘Prezzo’: costo di semina per ettaro della coltura



* Fatto ‘month’: descrive il mese dal punto di vista più utile all’obbiettivo agricolo. Ha i seguenti parametri:
* ‘Nome’: nome del mese di riferimento.
* ‘Temperatura\_max’: temperatura massima registrata nei campi in quel dato mese negli ultimi 5 anni
* ‘Temperature\_min’: temperatura minima registrata nei campi in quel dato mese negli ultimi 5 anni.



* Fatto ‘ph’: descrive di che tipo può essere il ph del terreno, ha 3 possibili valori:
* acido
* neutro
* basico

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

* Fatto ‘humidity’: descrive il livello di umidità che caratterizza la zona, assume valori:
  + Alta
  + Media
  + Bassa

Immagine che contiene testo, Carattere, bianco, schermata

Descrizione generata automaticamente

* Fatto ‘rainfall’: descrive il livello mensile in mm di piogge, la zona può essere:
  + Arida
  + Standard
  + Piovosa

Immagine che contiene testo, Carattere, bianco

Descrizione generata automaticamente

I dati relativi alle temperature dei mesi e i prezzi per ettaro sono stati ottenuti tramite un intervista a esperti che si occupano del settore agricolo.

### 5.3.2 Regole

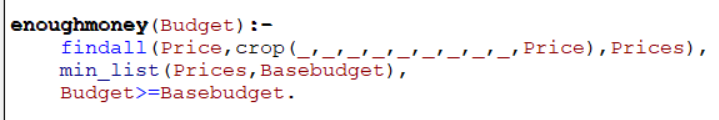
Di seguito vengono riportate tutte le regole che compongono la Knowledge base:

1. **sowingmonth** (Month,Crop): consiglia quali colture seminare in base al mese dato in input:

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

2. **enoughmoney** (Budget) : regola di controllo, si assicura che il budget inserito sia almeno pari o superiore a quello necessario per seminare la coltura più economica del dataset in un ettaro di terreno

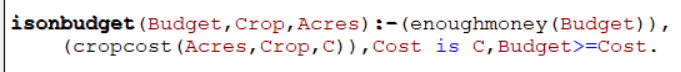


3. **cropcost** (Acres, Crop, Result): calcola il costo necessario a seminare una data coltura in un numero di ettari pari ad Acres, avvalora Result con il risultato del calcolo

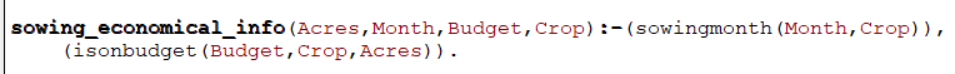
Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

4. **isonbudget** (Budget,Crop,Acres): regola di controllo, verifica che il budget dato in input sia sufficiente per seminare la coltura data in input negli ettari di terreno dati in input



5. **sowing\_economical\_info** (Acres, Month, Budget, Crop): restituisce in output uno o più colture consilgiate per il budget disponibile, il mese di semina scelto, e gli ettari di terreno a disposizione



6. **min\_cost\_crop** (Acres,Month,Budget,MinCostCrop) : delle colture estratte tramite ‘sowing\_economical\_info’, restituisce quella che richiede i costi minori, assegnandola alla variabile MinCostCrop

Immagine che contiene testo, schermata, Carattere

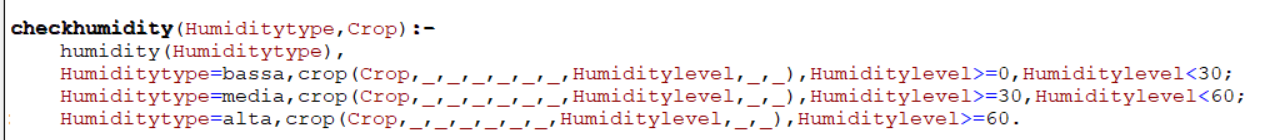
Descrizione generata automaticamente

7. **checkPHlevel** (PH\_type,Crop): restituisce tutte le colture che prosperano in un livello di ph classificabile in quello fornito in input

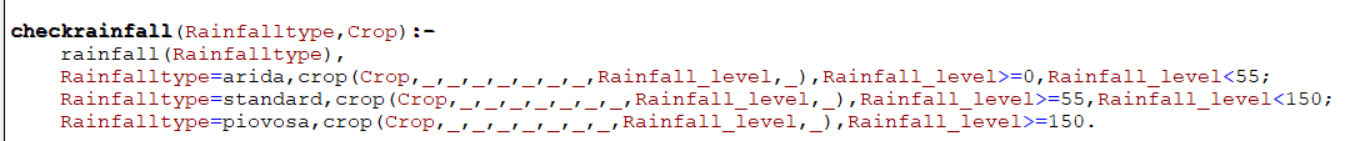
Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

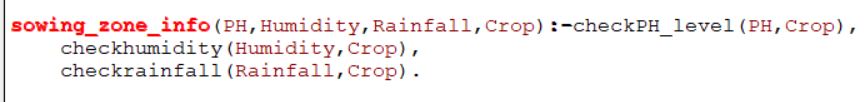
8. **checkhumidity** (Humiditytype, Crop): restituisce tutte le colture che prosperano in un livello di umidità classificabile in quella fornita in input



9. **checkrainfall** (Rainfalltype, Crop): restituisce tutte le colture che prosperano con un livello di piogge classificabile in quello fornito in input



10. **sowing\_zone\_info** (Ph, Humidity, Rainfall, Crop): consiglia tutte le colture adatte alle caratteristiche del luogo fornite in input. Il luogo è classificato in base al ph del terreno, all’umidità in percentuale, al livello di piogge mensili in mm. La regole richiama i 3 micro-controlli presentati in precedenza

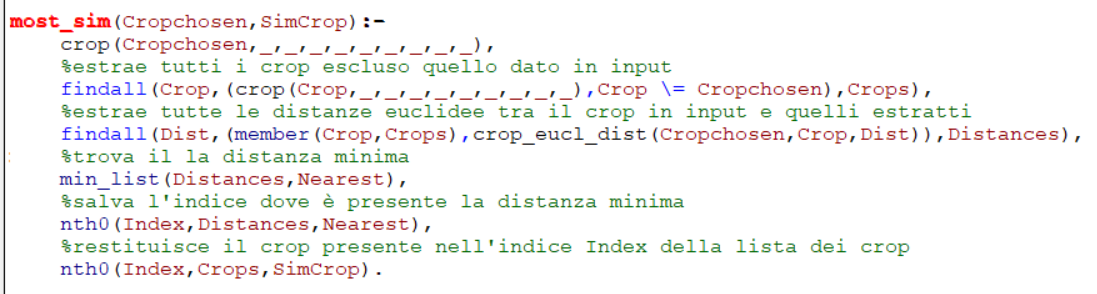


11. **crop\_eucl\_dist** (Crop1, Crop2, Dist): calcola la distanza euclidea tra due colture, intese come punti n-dimesionali dove n è il numero di parametri numerici del fatto ‘crop’. Il risultato viene assegnato come valore a ‘Dist’. Questa regola è principalmente applicata in ‘most\_sim’

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

12. **most\_sim** (Cropchosen, Simcrop): data una coltura in input, restituisce la più simile per nutrienti richiesti (azoto, fosforo, potassio), per caratteristiche favorevoli del luogo (ph, umidità, piogge) e il prezzo per ettaro. La misura di similarità usata è la distanza euclidea. Il crop più simile è quello con la più piccola distanza euclidea dalla coltura data in input. Dai possibili candidati viene chiaramente escluso Cropchosen, perché è chiaramente uguale a se stesso



## 5.4 Esempi

Di seguito vengono riportate dimostrazioni del funzionamento della base di conoscenza su python, le funzioni tra le quali è possibile scegliere dal main sono le seguenti:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

1.coltura consigliata in base a ettari di terreno disponibili, mese di semina, budget

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

2.coltura consigliata in base al mese di semina scelto

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

3. budget minimo necessario data la coltura e la superficie in ettari

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

4. coltura più economica dati gli ettari, il mese di semina e il budget

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

5. coltura consigliata in base ai dettagli della zona di coltivazioneImmagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

6. coltura più simile a quella fornita in input

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

# 6.Conclusione e sviluppi futuri

Le diverse forme di rappresentazione di conoscenza e apprendimento si sono rivelate decisamente adatte ai dati forniti e agli obbiettivi prefissati. Contiamo di espandere il prodotto affiancando nuove colture e altri dati che permettano la fruizione di servizi aggiuntivi, come informazioni sulle modalità di irrigazione, concimazione e altri aspetti tecnici del mondo agricolo

# 7. Riferimenti Bibliografici

[1] <https://artint.info/2e/html2e/ArtInt2e.Ch14.S3.html>

[2] <https://it.wikipedia.org/wiki/Prolog>

[3] <https://pandas.pydata.org/>

[4] <https://numpy.org/>

[5] <https://scikit-learn.org/stable/>

[6] <https://matplotlib.org/>

[7] <https://pypi.org/project/pyswip/0.2.2/>

[8] <https://pypi.org/project/Owlready/>

[9] <https://protege.stanford.edu/>

[10] <https://www.swi-prolog.org/>

[11] <https://www.kaggle.com/datasets/varshitanalluri/crop-recommendation-dataset>

[12] <https://it.wikipedia.org/wiki/Web_Ontology_Language>

[13] <http://vowl.visualdataweb.org/webvowl.html>

[14] <https://artint.info/2e/html2e/ArtInt2e.Ch7.S2.html>

[15] <https://scikit-learn.org/stable/tutorial/machine_learning_map/>

[16] <https://it.wikipedia.org/wiki/Macchine_a_vettori_di_supporto>

[17] <https://www.ibm.com/it-it/topics/knn>

[18] <https://www.ibm.com/it-it/topics/random-forest>

[19] <https://towardsdatascience.com/cross-validation-and-grid-search-efa64b127c1b>

[20] <https://artint.info/2e/html2e/ArtInt2e.Ch7.S4.SS3.html>

[21] <https://en.wikipedia.org/wiki/Learning_curve_(machine_learning)>

[22] <https://www.okpedia.it/inferenza_logica>