



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
1.1 Strumenti	2
1.1.1 Librerie	2
1.1.2 Software esterni	2
2. Dataset.....	2
2.1. Preprocessing dei dati.....	3
3 Ontologia	7
3.1 Scelte Progettuali	7
3.2 Classi e Data Property.....	7
3.3 Object Property	8
3.4 Visualizzazione grafica dell'ontologia	9
3.5 Individui	9
3.6 Query	10
3.7 Owlready.....	10
4. Apprendimento supervisionato	14
4.1 Scelte progettuali	14
4.2 Tuning	14
4.2.1 Linear SVM.....	15
4.2.2 K-Nearest Neighbor	15
4.2.3 Support Vector Machine a kernel non lineare	16
4.2.4 Random Forest.....	16
4.3 Metodi di valutazione	17
4.4 Risultati ottenuti	18
4.4.1 Linear Support Vector Machine	18
4.4.2 K-Nearest Neighbor	20
4.4.3 Support Vector Machine non lineare	21
4.4.4 Random Forest.....	23
4.5 Scelta del modello finale	24
5. Knowledge Base su Prolog	25
5.1 Scelte progettuali	25
5.2 Principali funzionalità	25
5.3 Struttura della knowledge base	25
5.3.1 fatti.....	25
5.3.2 Regole.....	27
5.4 Esempi	29

6. Conclusione e sviluppi futuri	31
7. Riferimenti Bibliografici e sitografici	31

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]
- Apprendimento supervisionato: Classificazione
- Base di conoscenza in Prolog^[2] 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]: una libreria per la manipolazione e l'analisi dei dati.
- **Numpy**^[4]: una libreria che permette la manipolazione degli array.
- **Sklearn**^[5]: una libreria per l'apprendimento automatico.
- **Matplotlib**^[6]: una libreria per la creazione di grafici.
- **Pyswip**^[7]: una libreria che permette l'integrazione del linguaggio Prolog in ambiente Python.
- **Owlready2**^[8]: una libreria che permette di consultare ontologie in ambiente Python.

1.1.2 Software esterni

- **Protégé**^[9]: software che permette la creazione e manipolazione delle ontologie, con possibilità di effettuare query, tramite l'utilizzo di DL query.
- **SWIProlog**^[10]: ambiente di programmazione per il linguaggio Prolog

2. Dataset

Il dataset^[11] 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:

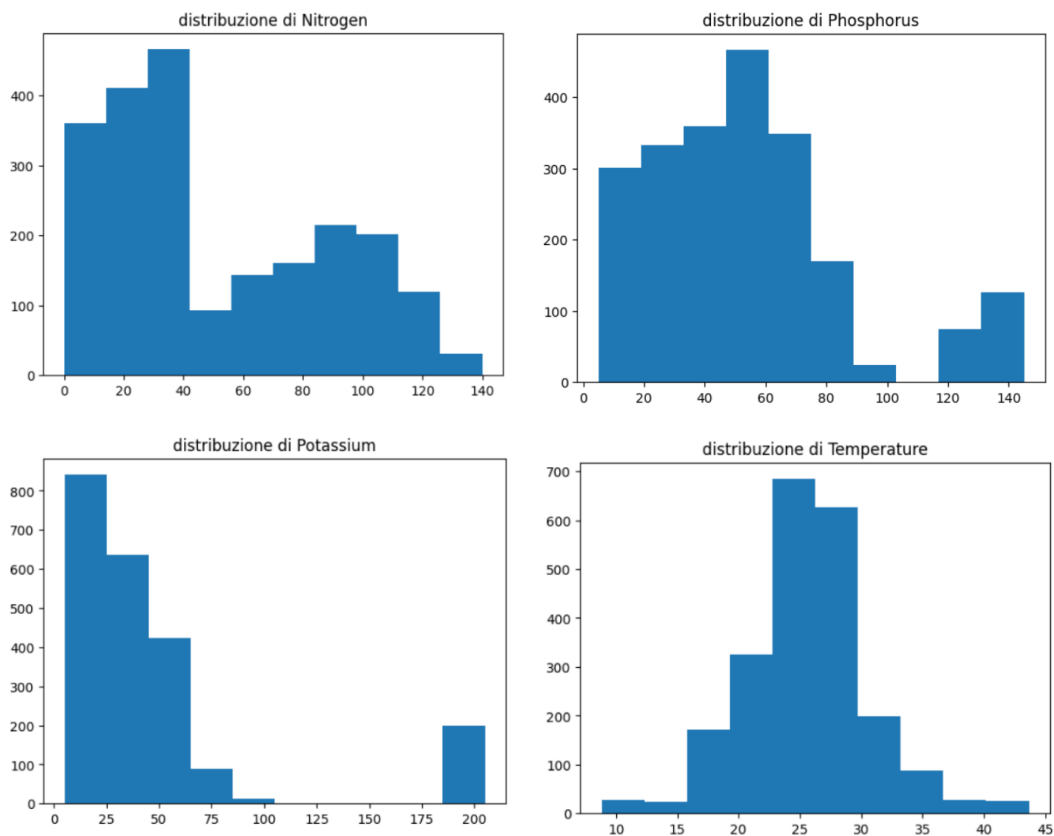
- **Crop**: le colture consigliate in base alle caratteristiche del terreno, vi sono 22 possibili valori.

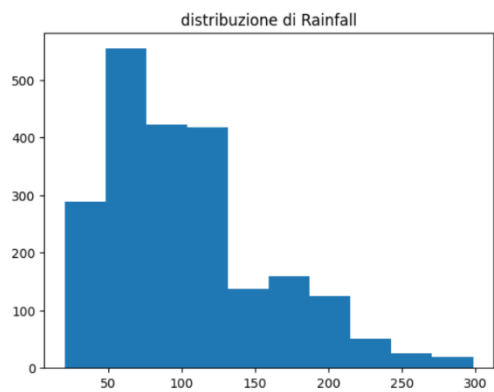
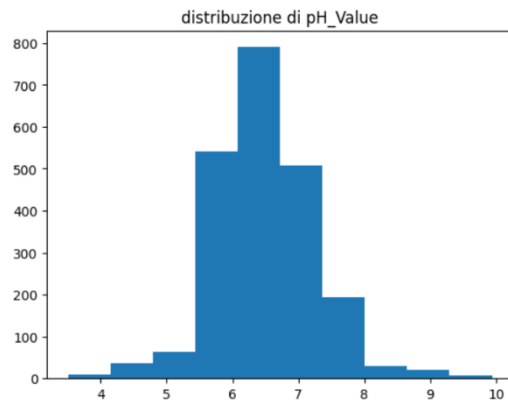
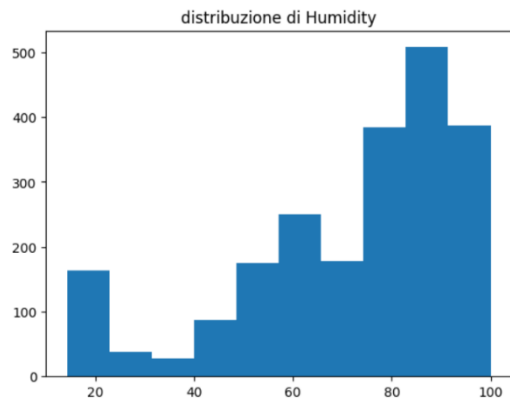
2.1. Preprocessing dei dati

il dataset è stato preprocessato seguendo suddetto iter:

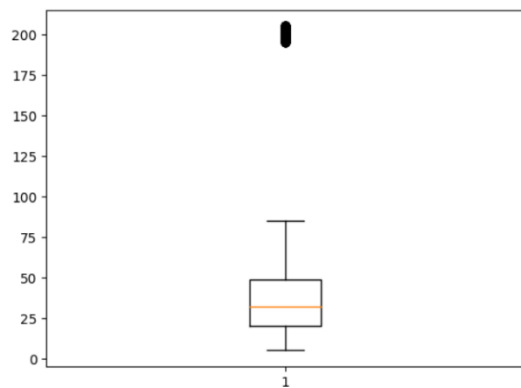
1. Analisi Esplorativa dei Dati

Inizialmente sono state analizzate le distribuzioni delle features:

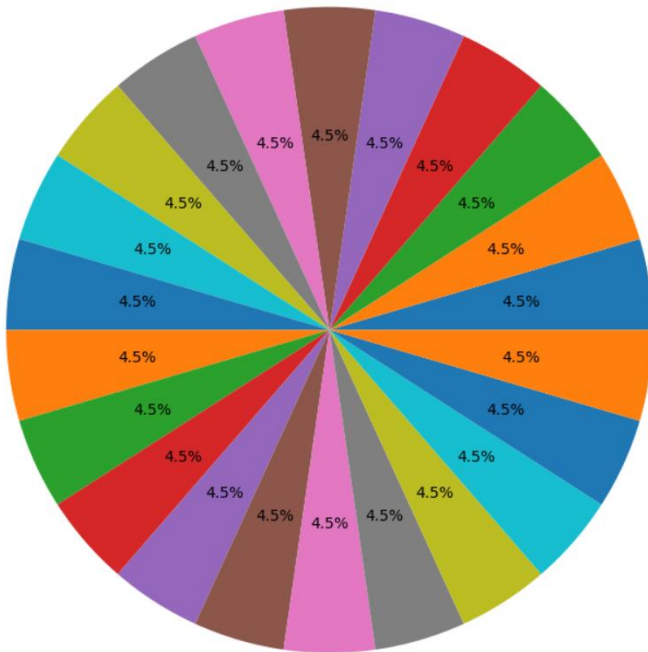




Non ci aspettavamo distribuzioni di dati perfettamente gaussiane, ma le feature *Temperature* e *pH_value* godono di questa proprietà, l'unica feature che potrebbe avere una distribuzione problematica è *Potassium*, data la netta divisione delle colonne dei valori. Abbiamo tracciato un *boxplot* per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

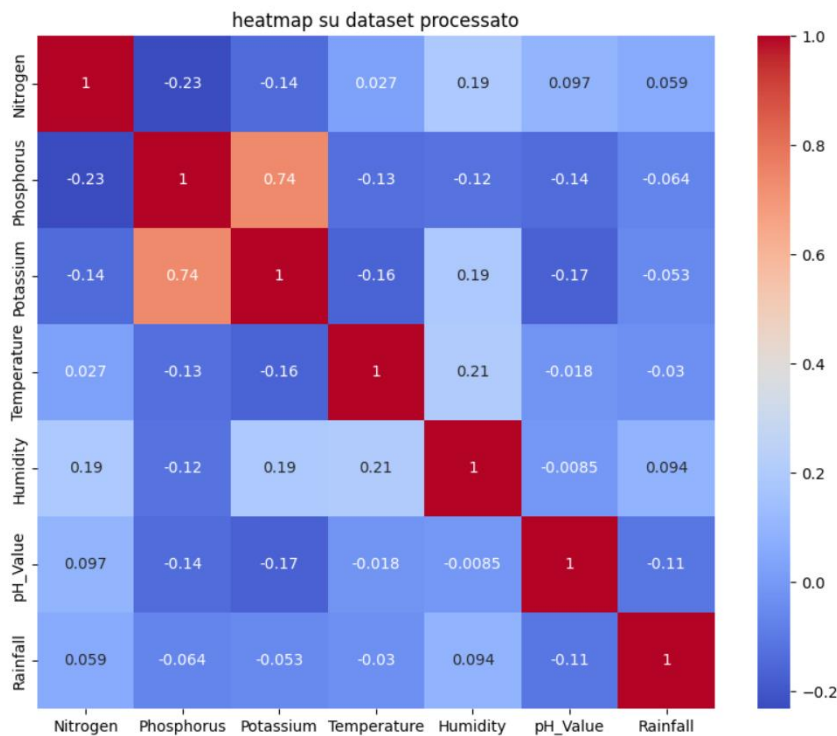


Prima di andare ad agire su di essi abbiamo voluto controllare la distribuzione delle classi per essere sicuri di non andare ad alterarla rimuovendo righe dal dataset:



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:



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

- *Phosphorus* 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.

2. 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.

$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

```
df=pd.read_csv('Crop_Recommendation.csv')

labelencoder=LabelEncoder()
stdscaler=MinMaxScaler(feature_range=(0,1))
columns_to_scale = ['Nitrogen', 'Phosphorus', 'Potassium', 'Temperature', 'Humidity', 'pH_Value', 'Rainfall']
df[columns_to_scale] = stdscaler.fit_transform(df[columns_to_scale])
df['Crop']=labelencoder.fit_transform(df['Crop'])
```

Figura 1 Codifica delle etichette e normalizzazione delle features

3. 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`.

```
#divisione dataset in feature e label

X=df.drop('Crop', axis=1)
Y=df['Crop']

#train-test split

X_train,X_test,Y_train,Y_test=train_test_split(X,Y, test_size=0.2, random_state=42)
```

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] (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.

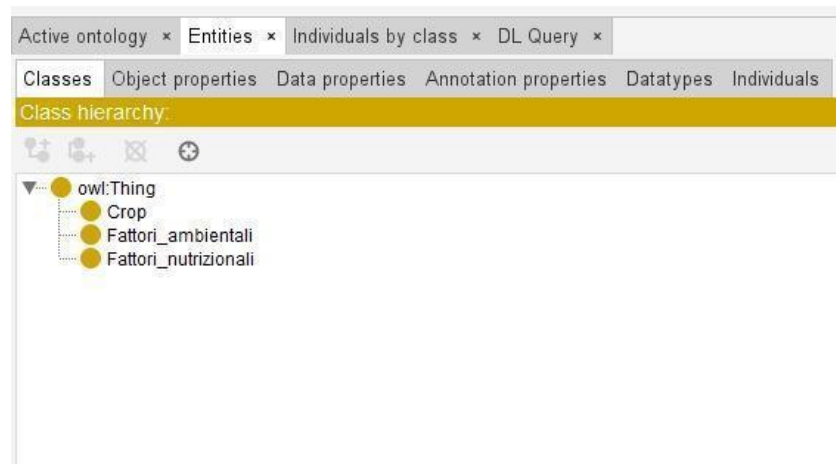


Figura 1 Classi definite

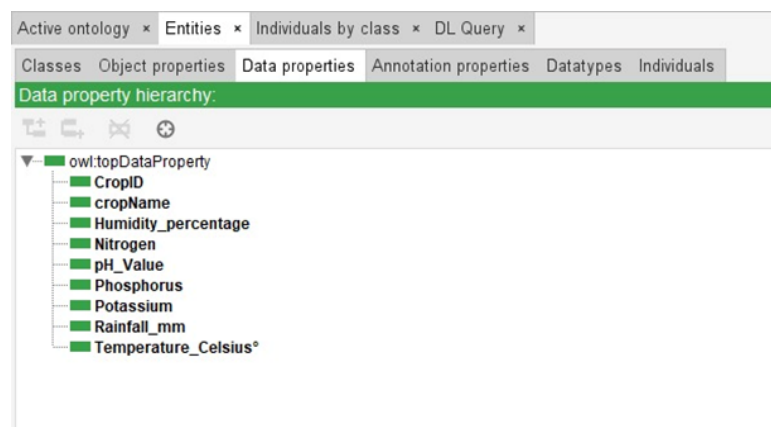


Figura 2 Data Property definite

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

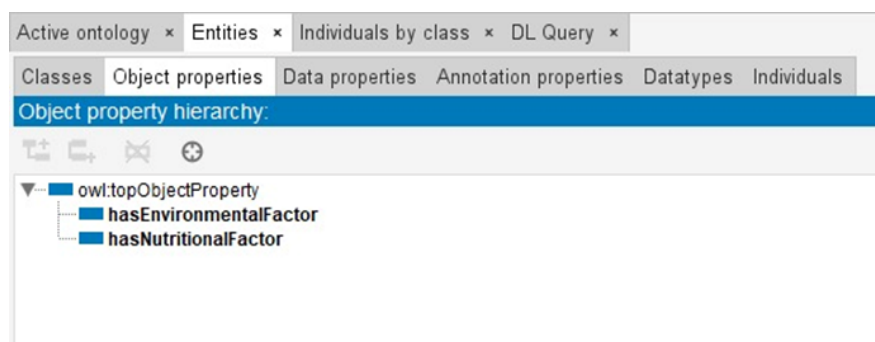
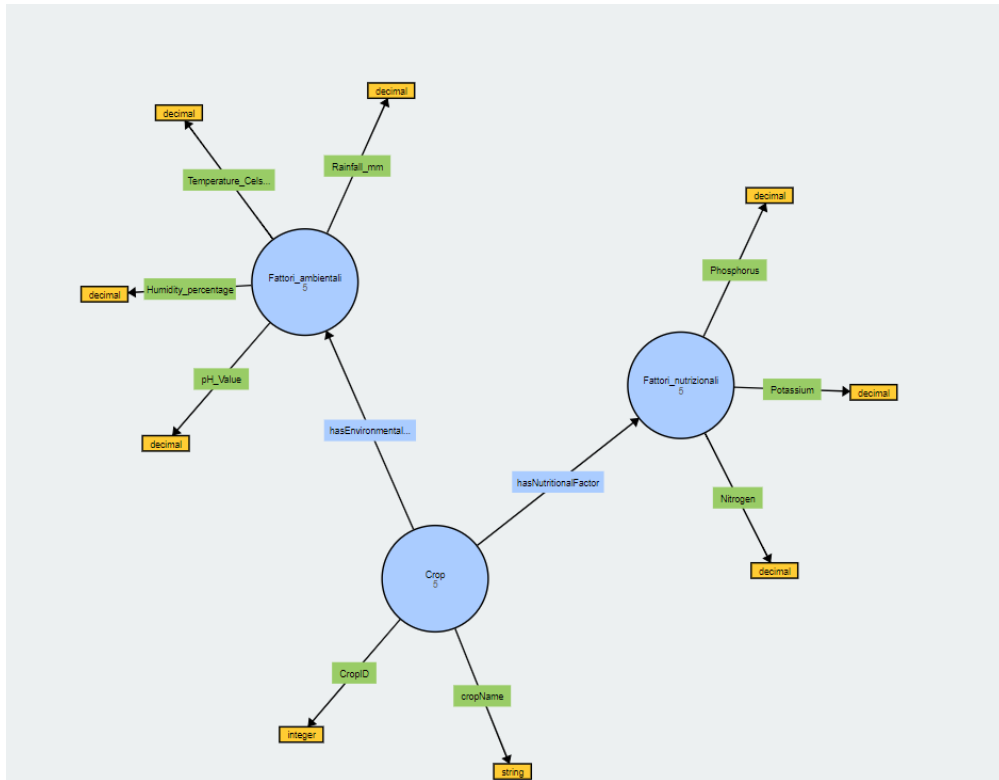


Figura 3 Object Property definite

3.4 Visualizzazione grafica dell'ontologia

Per visualizzare l'ontologia abbiamo utilizzato il tool online WebVowl^[13].



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.

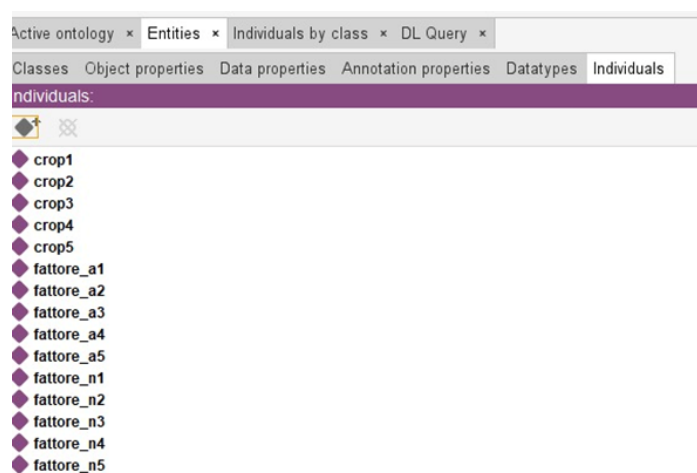
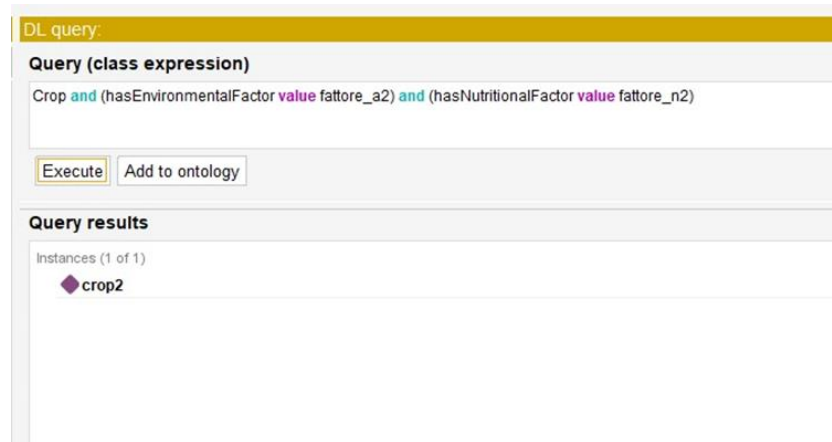


Figura 4 individui definiti

3.6 Query

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



The screenshot shows the DL Query interface in Protégé. It features a yellow header bar labeled "DL query:". Below this, a section titled "Query (class expression)" contains the query text: "Crop and (hasEnvironmentalFactor value fattore_a2) and (hasNutritionalFactor value fattore_n2)". The words "and" are highlighted in green, and the values "fattore_a2" and "fattore_n2" are highlighted in purple. Below the query text are two buttons: "Execute" and "Add to ontology". Underneath these buttons is a section titled "Query results". It shows "Instances (1 of 1)" and a single result: a purple diamond icon followed by the text "crop2".

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.

```

case '1':
    print("LISTA DELLE CLASSI NELL'ONTOLOGIA:")
    print(list(onto.classes()), "\n")
case '2':
    print("LISTA DELLE OBJECT PROPERTY NELL'ONTOLOGIA:")
    print(list(onto.object_properties()), "\n")
case '3':
    print("LISTA DELLE DATA PROPERTY NELL'ONTOLOGIA:")
    print(list(onto.data_properties()), "\n")
case '4':
    print("LISTA DEGLI INDIVIDUI DELLA CLASSE CROP")
    crop = onto.search(type=onto.Crop)
    print(crop, "\n")

    print("LISTA DEGLI INDIVIDUI DELLA CLASSE FATTORI_AMBIENTALI")
    ambientali = onto.search(type=onto.Fattori_ambientali)
    print(ambientali, "\n")

    print("LISTA DEGLI INDIVIDUI DELLA CLASSE FATTORI_NUTRIZIONALI")
    nutrizionali = onto.search(type=onto.Fattori_nutrizionali)
    print(nutrizionali, "\n")
case '5':
    individual = input('inserisci individuo del quale vuoi conoscere i dati: ')
    query_5(individual, onto)
case '6':
    factor_name = input(
        'inserisci nome del fattore ambientale (Humidity_percentage, Temperature_Celsius°, Rainfall_mm): ')
    min_val = input('inserisci valore minimo: ')
    max_val = input('inserisci valore massimo: ')
    query_6(factor_name, int(min_val), int(max_val), onto)
case '7':
    factor_name = input('inserisci nome del fattore nutrizionale (Nitrogen, Phosphorus, Potassium, pH_value): ')
    min_val = input('inserisci valore minimo: ')
    max_val = input('inserisci valore massimo: ')
    query_7(factor_name, int(min_val), int(max_val), onto)

```

ottenendo come input tutte le liste:

```

ONTOLOGIA

LISTA DELLE CLASSI NELL'ONTOLOGIA:
[agricoltura.Crop, agricoltura.Fattori_ambientali, agricoltura.Fattori_nutrizionali]

LISTA DELLE OBJECT PROPERTY NELL'ONTOLOGIA:
[agricoltura.hasEnvironmentalFactor, agricoltura.hasNutritionalFactor]

```

Figura 6 liste di classi e object property

```

LISTA DELLE DATA PROPERTY NELL'ONTOLOGIA:
[agricoltura.Potassium, agricoltura.Nitrogen, agricoltura.cropName, agricoltura.Humidity_percentage,
agricoltura.Temperature_Celsius°, agricoltura.pH_Value, agricoltura.Potassium, agricoltura.CropID, agricoltura.Rainfall_mm]

```

Figura 7 lista delle Data Property

```

LISTA DEGLI INDIVIDUI DELLA CLASSE CROP
[agricoltura.crop1, agricoltura.crop5, agricoltura.crop3, agricoltura.crop4, agricoltura.crop2]

LISTA DEGLI INDIVIDUI DELLA CLASSE FATTORI_AMBIENTALI
[agricoltura.fattore_a4, agricoltura.fattore_a2, agricoltura.fattore_a5, agricoltura.fattore_a3, agricoltura.fattore_a1]

LISTA DEGLI INDIVIDUI DELLA CLASSE FATTORI_NUTRIZIONALI
[agricoltura.fattore_n3, agricoltura.fattore_n1, agricoltura.fattore_n4, agricoltura.fattore_n2, agricoltura.fattore_n5]

```

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.

```

# funzione per recuperare e stampare le proprietà di un determinato individuo in un'ontologia
def getdatas(individual, onto):
    # ricerca dell'individuo
    individual = onto.search(iri=f'#{individual}')
    # Inizializzazione del dizionario delle proprietà
    data_properties = {}
    # Raccolta delle proprietà dell'individuo
    for prop in individual[0].get_properties():
        data_properties[prop.name] = getattr(individual[0], prop.name)
    # Stampa delle proprietà
    for prop_name, prop_values in data_properties.items():
        print(f"{prop_name}: {prop_values[0]}")
    print('\n')
    return data_properties

```

```

#QUERY che permette di ottenere tutti i dati relativi a un tipo di Crop dato in input
1 usage
def query_1(individual):
    ind = onto.search(iri=f'#{individual}')
    print('dati coltura:')
    getdatas(individual)
    print('fattori ambientali:')
    getdatas(ind[0].hasEnvironmentalFactor[0].name)
    print('fattori nutrizionali:')
    getdatas(ind[0].hasNutritionalFactor[0].name)

```

Ottenendo i seguenti risultati:

```

dati coltura:
CropID: 18
hasNutritionalFactor: agricoltura.fattore_n4
cropName: PigeonPeas
hasEnvironmentalFactor: agricoltura.fattore_a4

fattori ambientali:
Temperature_Celsius°: 36.0
Rainfall_mm: 134.85
Humidity_percentage: 56.0

fattori nutrizionali:
Potassium: 15.0
Nitrogen: 20.0
pH_Value: 7.31
Phosphorus: 72.0

```

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.

```
# QUERY che permette di ottenere tutte le istanze di crop aventi un valore minore
# rispetto ad un determinato fattore ambientale
def query_6(factor_name, min, max, onto):
    result = []
    try:
        for crop in onto.Crop.instances():
            for environmental_factor in crop.hasEnvironmentalFactor:
                attribute_values = getattr(environmental_factor, factor_name)
                if min <= attribute_values[0] <= max:
                    result.append(crop)
        if result.__len__() == 0:
            print(f'nessuno degli individui ha {factor_name} tra {min} e {max}')
        else:
            print(f'Risultati Crop con {factor_name} tra {min} e {max}:{result}')
    except:
        print('nome del fattore inserito errato')
```

Ottenendo come risultato in questo esempio, ricercando tutte le colture con un valore di "Rainfall_mm" compreso tra 20 e 180::

```
:6
inserisci nome del fattore ambientale (Humidity_percentage, Temperature_Celsius°, Rainfall_mm): Rainfall_mm
inserisci valore minimo: 20
inserisci valore massimo: 180
Risultati Crop con Rainfall_mm tra 20 e 180:[agricoltura.crop5, agricoltura.crop3, agricoltura.crop4, agricoltura.crop2]

Process finished with exit code 0
```

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.

```
# QUERY che permette di ottenere tutte le istanze di crop aventi un determinato fattore nutrizionale in
# un dato range
def query_7(factor_name, min, max, onto):
    result = []
    try:
        for crop in onto.Crop.instances():
            for nutritional_factor in crop.hasNutritionalFactor:
                attribute_values = getattr(nutritional_factor, factor_name)
                if min <= attribute_values[0] <= max:
                    result.append(crop)
        if result.__len__() == 0:
            print(f'nessuno degli individui ha {factor_name} tra {min} e {max}')
        else:
            print(f'Risultati Crop con {factor_name} tra {min} e {max}:{result}')
    except:
        print('nome del fattore inserito errato')
```

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

```
:7
inserisci nome del fattore nutrizionale (Nitrogen, Phosphorus, Potassium, pH_value): Nitrogen
inserisci valore minimo: 20
inserisci valore massimo: 60
Risultati Crop con Nitrogen tra 20 e 60:[agricoltura.crop3, agricoltura.crop4]

Process finished with exit code 0
```

4. Apprendimento supervisionato

L'apprendimento supervisionato^[14] 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*](#)^[15].

1. **Support Vector Machine a kernel lineare**^[16]: 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]: 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]: 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]: 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 considerazione:

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 utilizzati 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 IPERPARAMETRI PER SVC:
{
  'C': 1,
  'kernel': '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 verranno 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 la 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]

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]

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

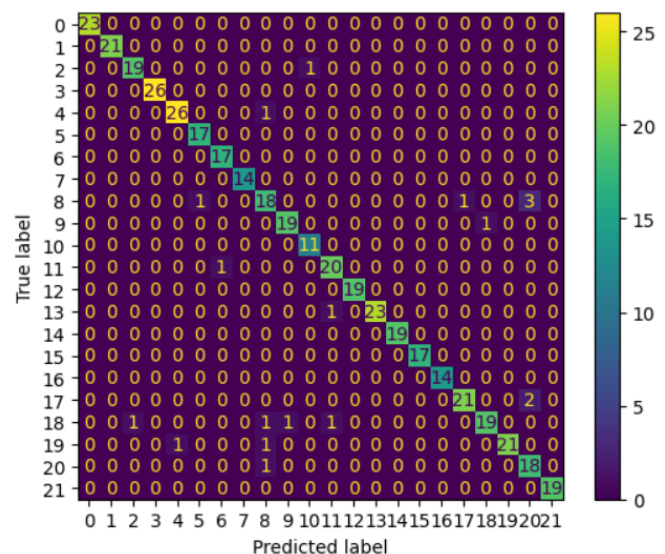
4.4 Risultati ottenuti

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:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	23
1	1.00	1.00	1.00	21
2	0.95	0.95	0.95	20
3	1.00	1.00	1.00	26
4	0.96	0.96	0.96	27
5	0.94	1.00	0.97	17
6	0.94	1.00	0.97	17
7	1.00	1.00	1.00	14
8	0.82	0.78	0.80	23
9	0.95	0.95	0.95	20
10	0.92	1.00	0.96	11
11	0.91	0.95	0.93	21
12	1.00	1.00	1.00	19
13	1.00	0.96	0.98	24
14	1.00	1.00	1.00	19
15	1.00	1.00	1.00	17
16	1.00	1.00	1.00	14
17	0.95	0.91	0.93	23
18	0.95	0.83	0.88	23
19	1.00	0.91	0.95	23
20	0.78	0.95	0.86	19
21	1.00	1.00	1.00	19
accuracy			0.96	440
macro avg	0.96	0.96	0.96	440
weighted avg	0.96	0.96	0.96	440



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

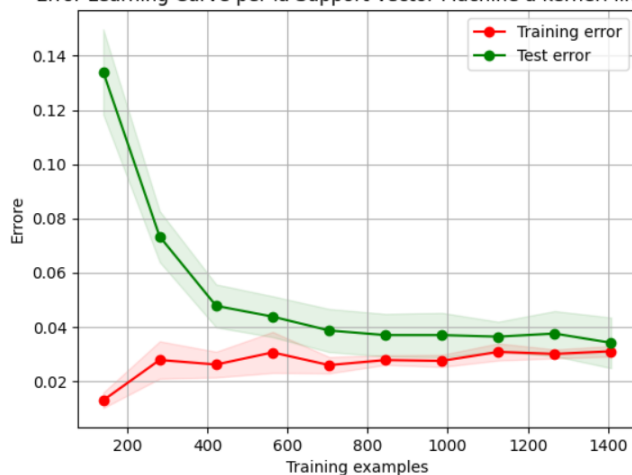
Cross validation

LinearSVC	media	deviazione standard	varianza
accuracy (train)	0.9673863636363637	0.003103750064488138	9.633264462810122e-06
accuracy (test)	0.9618181818181819	0.009248631795170843	8.553719008264504e-05
precision-macro (train)	0.9690531881432053	0.0028260355699478283	7.986477042610346e-06
precision-macro (test)	0.9653137388926861	0.008372460689723136	7.00980980009592e-05
recall-macro (train)	0.9673863636363638	0.0031037500644880826	9.633264462809778e-06
recall-macro (test)	0.9618181818181817	0.009248631795170803	8.553719008264432e-05
f1-macro (train)	0.9673256408654949	0.0030309871014659643	9.186882809253048e-06
f1-macro (test)	0.9619621322130365	0.008819522569915018	7.778397836124041e-05
precision-weighted (train)	0.9690531881432053	0.0028260355699478838	7.98647704261066e-06
precision-weighted (test)	0.9653137388926863	0.008372460689723025	7.009809800095734e-05
recall-weighted (train)	0.9673863636363637	0.003103750064488138	9.633264462810122e-06
recall-weighted (test)	0.9618181818181819	0.009248631795170843	8.553719008264504e-05
f1-weighted (train)	0.9673256408654949	0.0030309871014659096	9.186882809252716e-06
f1-weighted (test)	0.9619621322130367	0.008819522569914973	7.77839783612396e-05

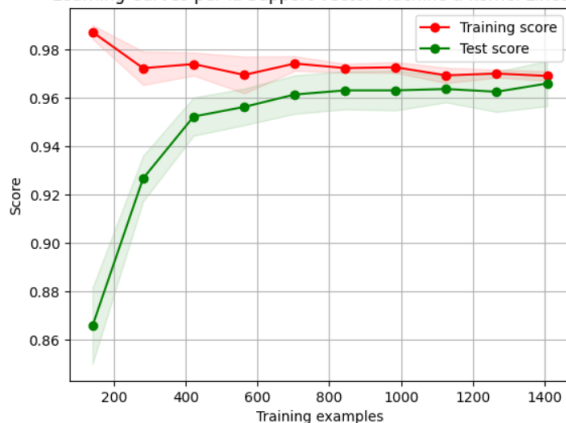
Tra train e test c'è poca differenza sulle medie, come ci si poteva aspettare, i valori sono leggermente più alti sul set di allenamento, vi sono differenze leggermente più marcate sulla deviazione standard, leggero il delta tra le varianze, che varia solo di un ordine di grandezza (da 10^{-6} a 10^{-5}).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

Curve di apprendimento

Error Learning Curve per la Support Vector Machine a kernel lineare



Learning Curves per la Support Vector Machine a kernel Lineare



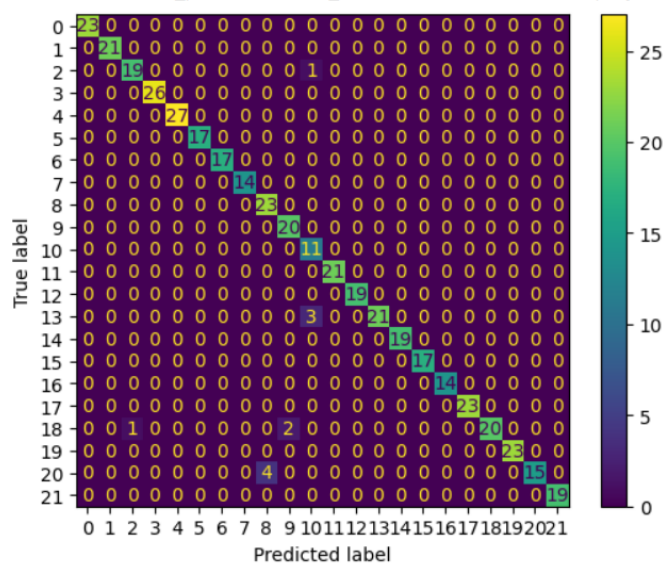
Le curve sono abbastanza distanti fino ai 400 esempi e si uniscono, ma lentamente verso la fine, nonostante le performance di partenza media dei test set non siano affatto negative, c'è un divario non indifferente, potrebbe essere un indicatore di overfitting. Supporre una linear separabilità dei dati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

	precision	recall	f1-score	support
0	1.00	1.00	1.00	23
1	1.00	1.00	1.00	21
2	0.95	0.95	0.95	20
3	1.00	1.00	1.00	26
4	1.00	1.00	1.00	27
5	1.00	1.00	1.00	17
6	1.00	1.00	1.00	17
7	1.00	1.00	1.00	14
8	0.85	1.00	0.92	23
9	0.91	1.00	0.95	20
10	0.73	1.00	0.85	11
11	1.00	1.00	1.00	21
12	1.00	1.00	1.00	19
13	1.00	0.88	0.93	24
14	1.00	1.00	1.00	19
15	1.00	1.00	1.00	17
16	1.00	1.00	1.00	14
17	1.00	1.00	1.00	23
18	1.00	0.87	0.93	23
19	1.00	1.00	1.00	23
20	1.00	0.79	0.88	19
21	1.00	1.00	1.00	19
accuracy			0.97	440
macro avg	0.97	0.98	0.97	440
weighted avg	0.98	0.97	0.98	440



La performance 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

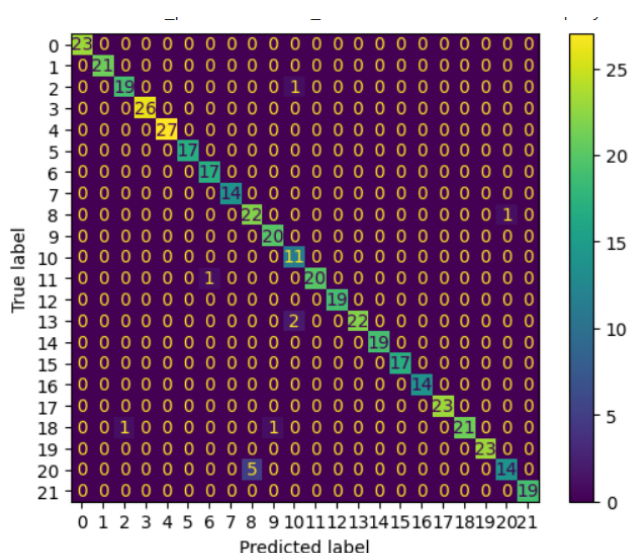
KNN	media	deviazione standard	varianza
accuracy (train)	/	/	/
accuracy (test)	0.9818181818181818	0.003803000120609439	1.4462809917355408e-05
precision-macro (train)	/	/	/
precision-macro (test)	0.9829693191395489	0.0038356847148612577	1.4712477231820287e-05
recall-macro (train)	/	/	/
recall-macro (test)	0.9818181818181817	0.0038030001206094256	1.4462809917355304e-05
f1-macro (train)	/	/	/
f1-macro (test)	0.9818670652877921	0.003869836193452913	1.4975632164158133e-05
precision-weighted (train)	/	/	/
precision-weighted (test)	0.9829693191395489	0.0038356847148611948	1.4712477231819804e-05
recall-weighted (train)	/	/	/
recall-weighted (test)	0.9818181818181818	0.003803000120609439	1.4462809917355408e-05
f1-weighted (train)	/	/	/
f1-weighted (test)	0.9818670652877923	0.003869836193452926	1.4975632164158235e-05

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:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	23
1	1.00	1.00	1.00	21
2	0.95	0.95	0.95	20
3	1.00	1.00	1.00	26
4	1.00	1.00	1.00	27
5	1.00	1.00	1.00	17
6	0.94	1.00	0.97	17
7	1.00	1.00	1.00	14
8	0.81	0.96	0.88	23
9	0.95	1.00	0.98	20
10	0.79	1.00	0.88	11
11	1.00	0.95	0.98	21
12	1.00	1.00	1.00	19
13	1.00	0.92	0.96	24
14	1.00	1.00	1.00	19
15	1.00	1.00	1.00	17
16	1.00	1.00	1.00	14
17	1.00	1.00	1.00	23
18	1.00	0.91	0.95	23
19	1.00	1.00	1.00	23
20	0.93	0.74	0.82	19
21	1.00	1.00	1.00	19
accuracy			0.97	440
macro avg	0.97	0.97	0.97	440
weighted avg	0.98	0.97	0.97	440



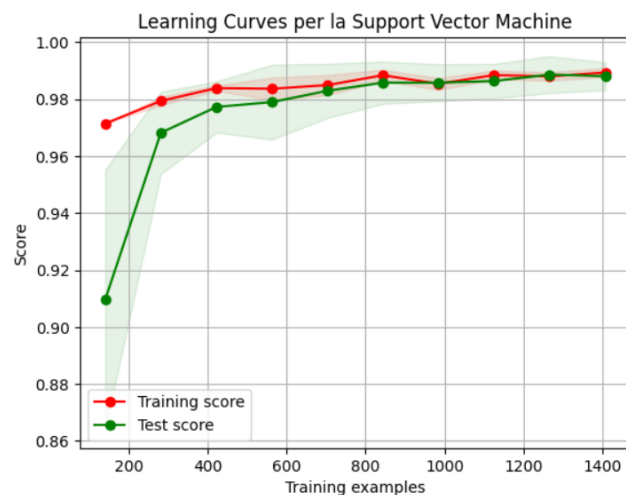
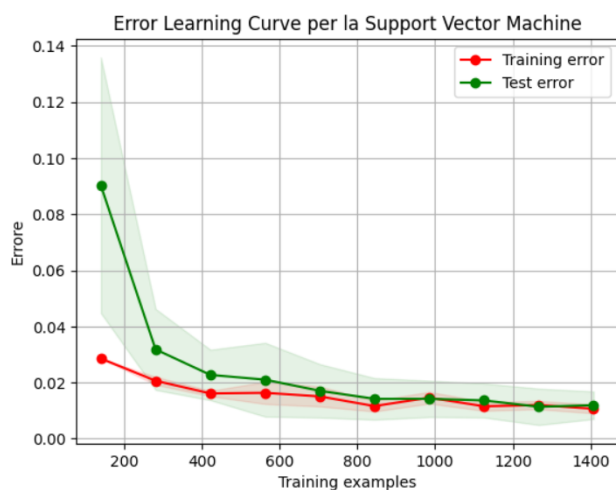
Su singolo run sembra non ci siano evidenti miglioramenti. Nella matrice di confusione abbiamo una performance un po' debole per la classe 20, ma resta l'unico aspetto che risalta, nel resto dei casi non vi sono più di 2 predizioni errate

Cross Validation

SVC	media	deviazione standard	varianza
accuracy (train)	0.9885227272727273	0.001315436011680729	1.7303719008265029e-06
accuracy (test)	0.9813636363636362	0.004406981688560299	1.942148760330578e-05
precision-macro (train)	0.9889684860988783	0.0011913740762904794	1.4193721896569932e-06
precision-macro (test)	0.9833980972913778	0.004819630193702447	2.322883520404829e-05
recall-macro (train)	0.9885227272727273	0.0013154360116807407	1.7303719008265338e-06
recall-macro (test)	0.9813636363636362	0.004406981688560299	1.942148760330578e-05
f1-macro (train)	0.9885078875719172	0.0013279253913918653	1.7633858451032388e-06
f1-macro (test)	0.9811512089318437	0.004509942118731736	2.03395779143105e-05
precision-weighted (train)	0.9889684860988783	0.001191374076290428	1.4193721896568704e-06
precision-weighted (test)	0.9833980972913778	0.004819630193702425	2.3228835204048075e-05
recall-weighted (train)	0.9885227272727273	0.001315436011680729	1.7303719008265029e-06
recall-weighted (test)	0.9813636363636362	0.004406981688560299	1.942148760330578e-05
f1-weighted (train)	0.9885078875719172	0.0013279253913917935	1.7633858451030478e-06
f1-weighted (test)	0.9811512089318439	0.004509942118731709	2.0339577914310258e-05

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.

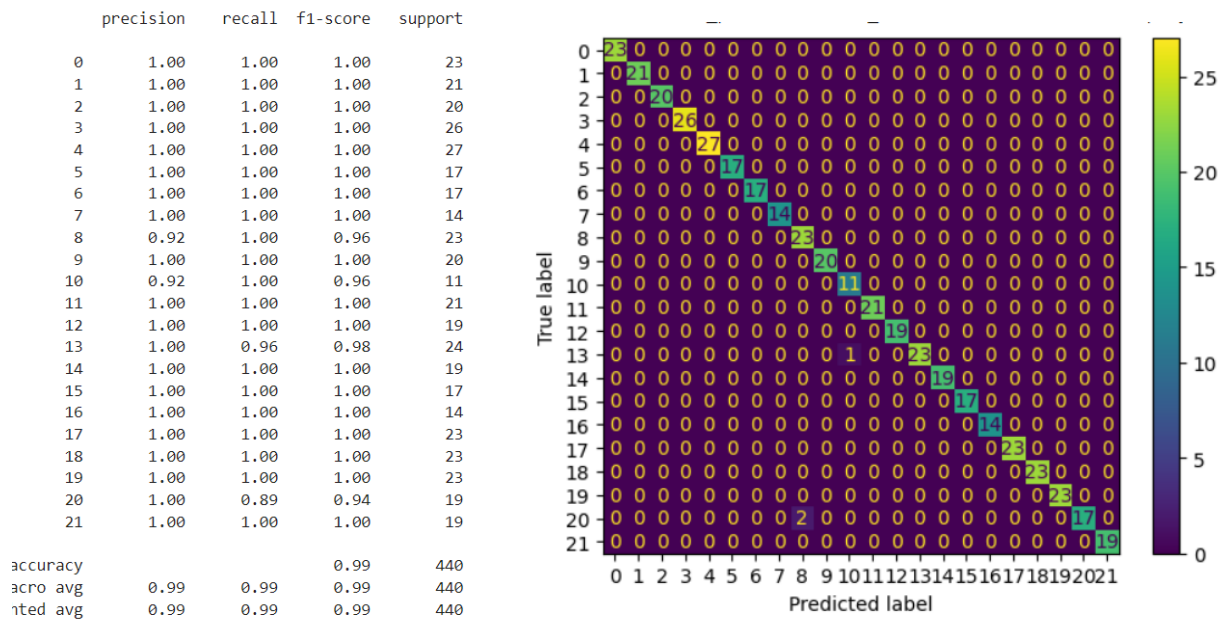
Curve di apprendimento



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:



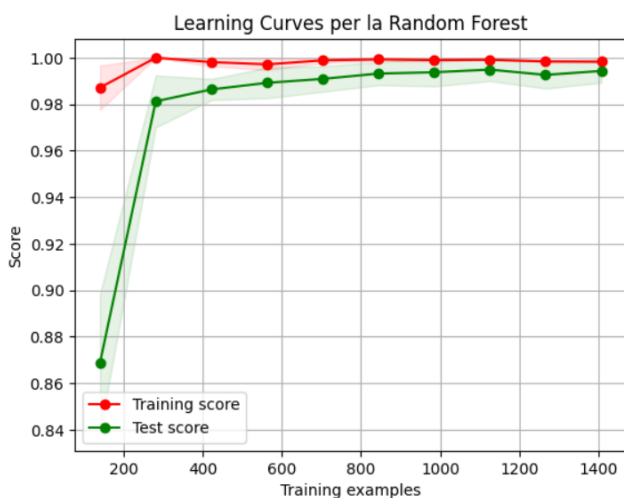
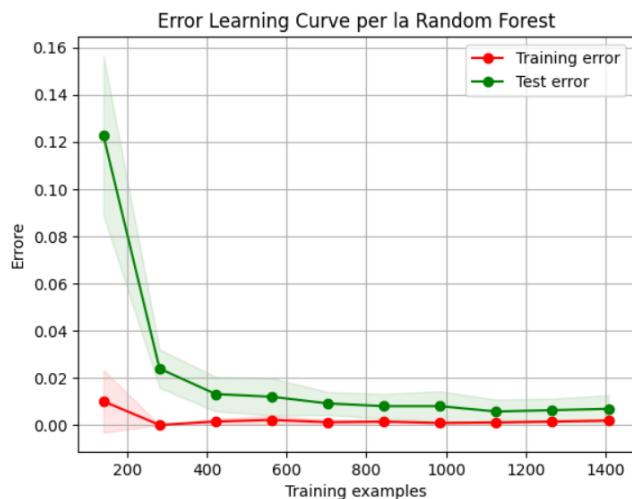
Su questo specifico test set i valori rasentano la performance massima ottenibile, la solidità della random forest ignora eventuali classi sbilanciate. La matrice di confusione contiene quasi tutti i valori sulla diagonale principale.

Cross Validation

Random Forest	media	deviazione standard	varianza
accuracy (train)	0.9977272727272727	0.0003593497341100322	1.2913223140495087e-07
accuracy (test)	0.9936363636363638	0.0033402132856134087	1.1157024793388322e-05
precision-macro (train)	0.9977336389101096	0.0002037178507766857	4.150096272507198e-08
precision-macro (test)	0.994529712711531	0.0025456059234626204	6.48010951756798e-06
recall-macro (train)	0.9977272727272727	0.00035934973410999715	1.2913223140492565e-07
recall-macro (test)	0.9945454545454545	0.002317736142542192	5.3719008264463605e-06
f1-macro (train)	0.997839465360818	0.0005574913244913708	3.107965768831428e-07
f1-macro (test)	0.9936203931785628	0.003352113076859884	1.1236662080055037e-05
precision-weighted (train)	0.9975322898715511	0.00040329680217862234	1.6264831064750282e-07
precision-weighted (test)	0.9938410074773711	0.0035351284014631854	1.2497132814831655e-05
recall-weighted (train)	0.9978409090909091	0.0002272727272727426	5.165289256199044e-08
recall-weighted (test)	0.9927272727272728	0.003910147848655738	1.5289256198347092e-05
f1-weighted (train)	0.9976118371185866	0.0004260316651271445	1.815029796910074e-07
f1-weighted (test)	0.9940769327878621	0.0030954393031721607	9.581744479622952e-06

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]. 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

```
%crop(nome, azoto, fosforo, potassio, Temperatura_max, Temperatura_min,  
% ph, umidita, piogge, prezzo)
```

- Fatto 'month': descrive il mese dal punto di vista più utile all'obiettivo 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.

```
%month(nome, temperatura_max, temperatura_min)
```

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

```
ph(acido) .  
ph(neutro) .  
ph(basico) .
```

- Fatto 'humidity': descrive il livello di umidità che caratterizza la zona, assume valori:
 - Alta
 - Media
 - Bassa

```
humidity(alta) .
humidity(media) .
humidity(bassa) .
```

- Fatto 'rainfall': descrive il livello mensile in mm di piogge, la zona può essere:
 - Arida
 - Standard
 - Piovosa

```
rainfall(arida) .
rainfall(standard) .
rainfall(piovosa) .
```

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:

```
sowingmonth(Month, Crop) :- (month(Month, TMmin, TMmax)) ,
    (crop(Crop, _, _, TCmin, TCmax, _, _, _)) ,
    TCmin >= TMmin, TCmax <= TMmax.
```

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

```
enoughmoney(Budget) :-
    findall(Price, crop(_, _, _, _, _, Price), Prices) ,
    min_list(Prices, Basebudget) ,
    Budget >= Basebudget.
```

3. **croppcost** (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

```

cropcost (Acres, Crop, Result) :- Acres > 0,
    (crop(Crop, _, _, _, _, _, _, Price)),
    Result is Acres * Price.

```

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

```

isonbudget (Budget, Crop, Acres) :- (enoughmoney (Budget)),
    (cropcost (Acres, Crop, C)), Cost is C, Budget >= Cost.

```

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

```

sowing_economical_info (Acres, Month, Budget, Crop) :- (sowingmonth (Month, Crop)),
    (isonbudget (Budget, Crop, Acres)).

```

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

```

min_cost_crop (Acres, Month, Budget, MinCostCrop) :-
    %cerca tutti i crop che soddisfano i parametri economici
    findall(Crop, sowing_economical_info(Acres, Month, Budget, Crop), Crops),
    % estrai i costi di tutti i crop prima radunati
    findall(Cost, (member(Crop, Crops), cropcost(Acres, Crop, Cost)), Costs),
    %dei costi estra il minimo
    min_list(Costs, MinCost),
    %salva l'indice della lista dove si trova il costo minimo
    nth0(Index, Costs, MinCost),
    %dato che le liste hanno corrispondenza 1 a 1,
    %estrai il crop corrispondente a Index estratto in precedenza
    nth0(Index, Crops, MinCostCrop).

```

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

```

checkPH_level (PH_type, Crop) :-
    ph(PH_type),
    PH_type=acido, crop(Crop, _, _, _, PH_level, _, _), PH_level < 7;
    PH_type=neutro, crop(Crop, _, _, _, PH_level, _, _), PH_level = 7;
    PH_type=basico, crop(Crop, _, _, _, PH_level, _, _), PH_level > 7.

```

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

```

checkhumidity (Humiditytype, Crop) :-
    humidity(Humiditytype),
    Humiditytype=bassa, crop(Crop, _, _, _, Humiditylevel, _, _), Humiditylevel >= 0, Humiditylevel < 30;
    Humiditytype=media, crop(Crop, _, _, _, Humiditylevel, _, _), Humiditylevel >= 30, Humiditylevel < 60;
    Humiditytype=alta, crop(Crop, _, _, _, Humiditylevel, _, _), Humiditylevel >= 60.

```

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

```
checkrainfall(Rainfalltype,Crop):-
    rainfall(Rainfalltype),
    Rainfalltype=arida,crop(Crop,_,_,_,_,_,Rainfall_level,_),Rainfall_level>=0,Rainfall_level<55;
    Rainfalltype=standard,crop(Crop,_,_,_,_,_,Rainfall_level,_),Rainfall_level>=55,Rainfall_level<150;
    Rainfalltype=piovosa,crop(Crop,_,_,_,_,_,Rainfall_level,_),Rainfall_level>=150.
```

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 regola richiama i 3 micro-controlli presentati in precedenza

```
sowing_zone_info(PH, Humidity, Rainfall, Crop):-checkPH_level(PH,Crop),
    checkhumidity(Humidity,Crop),
    checkrainfall(Rainfall,Crop).
```

11. **crop_eucl_dist** (Crop1, Crop2, Dist): calcola la distanza euclidea tra due colture, intese come punti n-dimensionali 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'

```
crop_eucl_dist(Crop1,Crop2,D):-crop(Crop1,N1,F1,K1,TMX1,TMN1,PH1,H1,R1,P1),crop(Crop2,N2,F2,K2,TMX2,TMN2,PH2,H2,R2,P2),
    %distanza euclidea: sqrt((X1-X2)^2+(Y1-Y2)^2) per punti del tipo P1(X1,Y1), P2(X2,Y2)
    D is
    sqrt((N1-N2)**2+
        (F1-F2)**2+
        (K1-K2)**2+
        (TMX1-TMX2)**2 +
        (TMN1-TMN2)**2 +
        (PH1-PH2)**2 +
        (H1-H2)**2 +
        (R1-R2)**2 +
        (P1-P2)**2).
```

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

```
most_sim(Cropchosen,SimCrop):-
    crop(Cropchosen,_,_,_,_,_,_,_),
    %estrae tutti i crop escluso quello dato in input
    findall(Crop,(crop(Crop,_,_,_,_,_,_,_),Crop \= Cropchosen),Crops),
    %estrae tutte le distanze euclidee tra il crop in input e quelli estratti
    findall(Dist,(member(Crop,Crops),crop_eucl_dist(Cropchosen,Crop,Dist)),Distances),
    %trova il la distanza minima
    min_list(Distances,Nearest),
    %salva l'indice dove è presente la distanza minima
    nth0(Index,Distances,Nearest),
    %restituisce il crop presente nell'indice Index della lista dei crop
    nth0(Index,Crops,SimCrop).
```

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:

Scegli una delle seguenti opzioni per interrogare la base di conoscenza(inserisci il numero corrispondente)

1. coltura consigliata in base a ettari di terreno disponibili, mese di semina, budget
 2. coltura consigliata in base al mese di semina scelto
 3. budget minimo necessario data la coltura e la superficie in ettari
 4. coltura più economica dati gli ettari, il mese sdi semina e il budget
 5. coltura consigliata in base ai dettagli della zona di coltivazione
 6. coltura più simile a quella fornita in input
- :

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

```
inserisci ettari a disposizione: 5
inserisci mese di semina: agosto
inserisci il tuo budget: 40000
colture consigliate:
mothbeans
mungbean
blackgram
banana
mango
watermelon
muskmelon
papaya
coconut
cotton
jute
coffee

Process finished with exit code 0
```

2.coltura consigliata in base al mese di semina scelto

```
:2
inserisci mese di semina: maggio
colture consigliate: per il mese di maggio:
chickpea
apple

Process finished with exit code 0
```

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

```
:3
inserisci ettari a disposizione: 4
inserisci coltura: rice
budget minimo richiesto:4000

Process finished with exit code 0
```

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

```
:4
inserisci ettari a disposizione: 5
inserisci mese di semina: maggio
inserisci il tuo budget: 20000
la coltura più economica è chickpea con un costo di semina pari a: 4500

Process finished with exit code 0
```

5. coltura consigliata in base ai dettagli della zona di coltivazione

```
:5
inserisci il livello di ph del terreno(basico,neutro,acido): acido
inserisci il livello di umidità della zona(alta, media, bassa): media
inserisci atmosfere della zona(arida, standard, piovosa): piovosa
colture consigliate per un terreno acido, in una zona con media umidità e piovosa:
coffee

Process finished with exit code 0
```

6. coltura più simile a quella fornita in input

```
:6
inserisci coltura: maize
la coltura più simile a maize è: kidneybeans

Process finished with exit code 0
```

6. Conclusione e sviluppi futuri

Le diverse forme di rappresentazione di conoscenza e apprendimento si sono rivelate decisamente adatte ai dati forniti e agli obiettivi 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 e sitografici

[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_\(machine_learning\)](https://en.wikipedia.org/wiki/Learning_curve_(machine_learning))
- [22] https://www.okpedia.it/inferenza_logica