

FOOD MANAGER

Esame di Ingegneria della conoscenza

Anno Accademico 2023-2024

Gruppo di lavoro

- Angelo Pisciotta, 758430, a.pisciotta3@studenti.uniba.it
- Michele Amati, 716722, m.amati12@studenti.uniba.it

Materiale del progetto

<https://github.com/michele-amatizo/Icon.git>

Indice

1. Introduzione
2. Ontologia
3. Recommender system
4. CSP
5. Casi di test
6. Conclusioni

INTRODUZIONE

Il progetto si propone di facilitare la gestione dei prodotti alimentari presenti in casa e di aiutare l'utente nel loro utilizzo. In particolare, il nostro obiettivo è quello di consentire di conoscere i prodotti presenti in casa, quali tra questi sono scaduti o commestibili e di fornire ingredienti, dosi e metodo di cottura di ricette create sulla base degli alimenti preferiti da ciascun utente.

OBIETTIVI PROGETTUALI

- Creare un'ontologia che consenta di ottenere, tramite apposite query, le proprietà riguardanti un determinato alimento. Queste includono caratteristiche intrinseche dell'alimento, come tipologia e valori nutrizionali e caratteristiche estrinseche, come metodo di cottura e abbinamenti con altri alimenti.
- Strutturare un sistema di raccomandazione degli alimenti utilizzando l'algoritmo KD-Tree basandosi sulla somiglianza degli alimenti preferiti dall'utente con quelli disponibili nel dataset di input.
- Modellare un CSP (Constraint Satisfaction Problem) che decreti se ogni alimento delle ricette proposte è presente in casa e non è scaduto.

STRUMENTI ADOTTATI

- Linguaggio Python (ver. 3.9) data la presenza di librerie funzionali ai nostri obiettivi:
 - Pandas per la gestione dei dataset e per parte del relativo preprocessing
 - Scikit-learn per lo sviluppo dell'algoritmo KD-Tree e parte del preprocessing dei dataset
 - Constraint per modellare il CSP
 - Owlready2 per le query all'ontologia
- IDE PyCharm
- Protégé (ver. 5.5.0) per la creazione dell'ontologia

ONTOLOGIA

Un'ontologia è una specifica formale che documenta il significato dei simboli, ovvero associa simboli presenti nella macchina a concetti reali di un determinato dominio; essa identifica gli individui nel dominio e le relazioni che esistono tra questi.

Le ontologie possono essere implementate per migliorare la ricerca di informazioni, facilitare l'interoperabilità sintattica e semantica della conoscenza, ossia favorire la collaborazione tra più KB senza ambiguità.

Le ontologie distribuite sulla rete vengono descritte attraverso il Web Ontology Language (OWL). Esso permette di descrivere il dominio in termini di:

- Individui: entità del dominio
- Classi: insiemi di individui che condividono caratteristiche comuni
- Proprietà: relazioni che associano gli individui del dominio a
 - Altri individui del dominio (Object Property)
 - Dati o valori di tipi primitivi come interi o stringhe (DataType Property)

DESCRIZIONE DELL'ONTOLOGIA

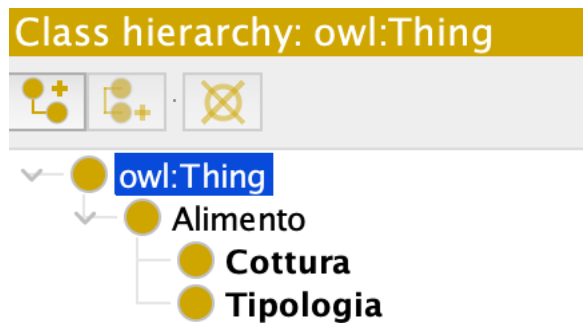
Per definire la nostra ontologia abbiamo utilizzato il software Protégé data la semplicità d'utilizzo e la presenza di estensioni utili ai fini progettuali.

L'ontologia si propone di modellare il dominio alimentare per poterlo sfruttare, tramite l'utilizzo di query, ai fini della creazione di ricette, meglio discussa nei paragrafi successivi.

Per la realizzazione della nostra ontologia abbiamo individuato le seguenti classi:

- “Alimento”, i cui individui sono alimenti della dieta italiana e occidentale.
- “Tipologia”, i cui individui sono macrocategorie di alimenti (carne, pesce, verdura...)
- “Cottura”, i cui individui sono metodi di cottura dei vari alimenti (fritto, bollito...). Alcuni individui di questa classe sono strutturati secondo la proprietà “same individual as”, la quale consente di raccogliere più individui e di trattarli come individuo unico, al fine di avere una grammatica più corretta.

Tali classi seguono la seguente gerarchia:



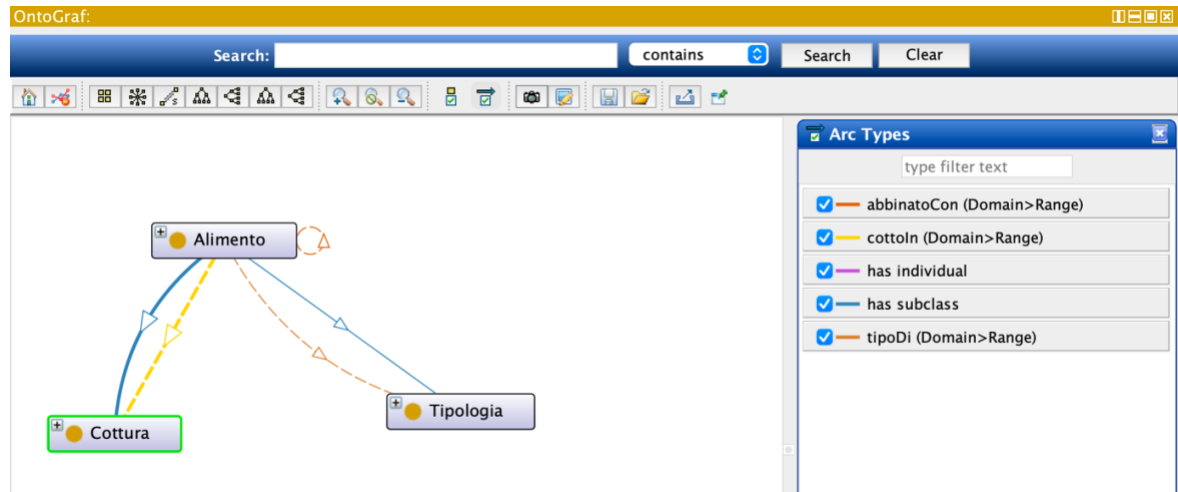
Le relazioni tra gli individui dell'ontologia sono rappresentate tramite le seguenti Object Properties:

- “tipoDi”, che ha come dominio gli individui della classe “Alimento” e come codominio gli individui della classe “Tipologia”, ovvero associa ogni alimento alla propria macrocategoria
- “cottoIn”, ha come dominio gli individui della classe “Alimento” e come codominio gli individui della classe “Cottura”, ovvero associa ad ogni alimento i suoi possibili metodi di cottura
- “abbinatoCon”, ha come dominio e codominio la classe “Alimento” e associa ad un alimento tutti quelli che, secondo ricette esistenti e gusto comune, si possono abbinare ad esso. Tale proprietà è simmetrica, quindi se A abbinatoCon B, allora B abbinatoCon A.

Le informazioni relative agli individui dell'ontologia sono rappresentate attraverso le seguenti Data Type Properties:

- “n_calorie”, che indica per ogni alimento il numero di Calorie per 100 grammi di prodotto crudo
- “n_carboidrati”, che indica per ogni alimento il peso in grammi dei carboidrati per 100 grammi di prodotto crudo
- “n_grassi”, che indica per ogni alimento il peso in grammi dei grassi per 100 grammi di prodotto crudo
- “n_proteine”, che indica per ogni alimento il peso in grammi delle proteine per 100 grammi di prodotto crudo

Utilizzando il plug-in di Protégé “OntoGraph”, di seguito visualizziamo graficamente le relazioni tra classi dell’ontologia.



Abbiamo popolato le classi citate con un sufficiente numero di individui, verificando, tramite plug-in “DL query” e Reasoner integrato di Protégé, l’effettivo funzionamento del sistema. Di seguito alcuni esempi di queries effettuate.

1

DL query:

Query (class expression)
Alimento **that** cottoln **value** bolliti

Execute **Add to ontology**

Query results

Instances (4 of 4)

- patate
- polpo
- spinaci
- uova

2

DL query:

Query (class expression)
Alimento **that** cottoln **value** bollito

Execute **Add to ontology**

Query results

Instances (4 of 4)

- patate
- polpo
- spinaci
- uova

DL query:


Query (class expression)

Alimento **that** abbinatoCon **value** pasta **and** n_calorie **value** 17 **and** cottoln **value** padella **and** tipoDi **value** verdura

Execute Add to ontology

Query results

Instances (1 of 1)

 **zucchini**

Nelle queries 1 e 2 abbiamo dimostrato il funzionamento del metodo “same individual as”: gli individui “bolliti” e “bollito” appartenenti alla classe “Cottura” coincidono; infatti, le due query effettuate con i diversi termini restituiscono gli stessi risultati.

La query 3 mostra come ottenere un risultato utilizzando l’operatore AND.

UTILIZZI DELL’ONTOLOGIA

L’ontologia è stata codificata in python importando il file [food.owl](#) (presente nella cartella [onto](#) del progetto), creato con Protégé tramite libreria Owlready2.

Abbiamo codificato delle funzioni, ognuna delle quali effettua una diversa query all’ontologia prendendo in input il nome dell’alimento:

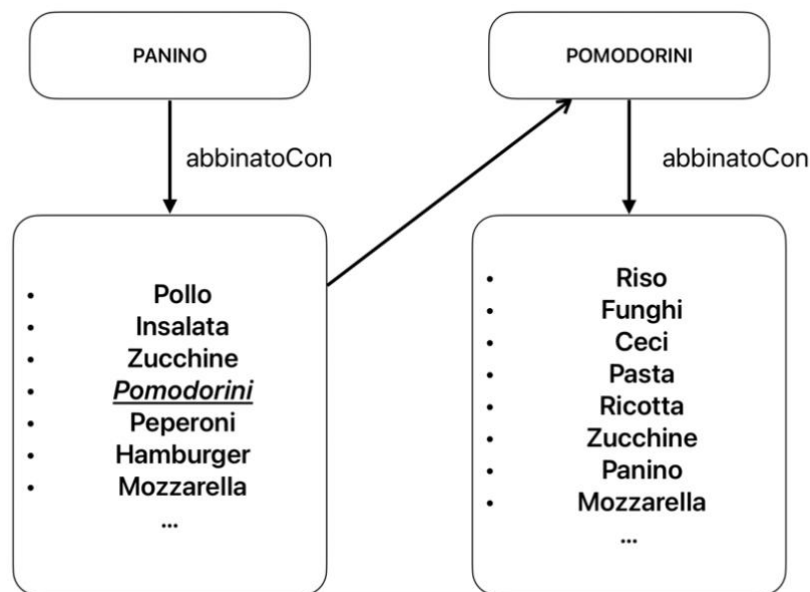
- Query per ottenere le calorie di un alimento
- Query per ottenere la tipologia di un alimento
- Query per ottenere i tipi di cottura di un alimento
- Query per ottenere gli alimenti abbinati ad un alimento

Creazione Ricetta

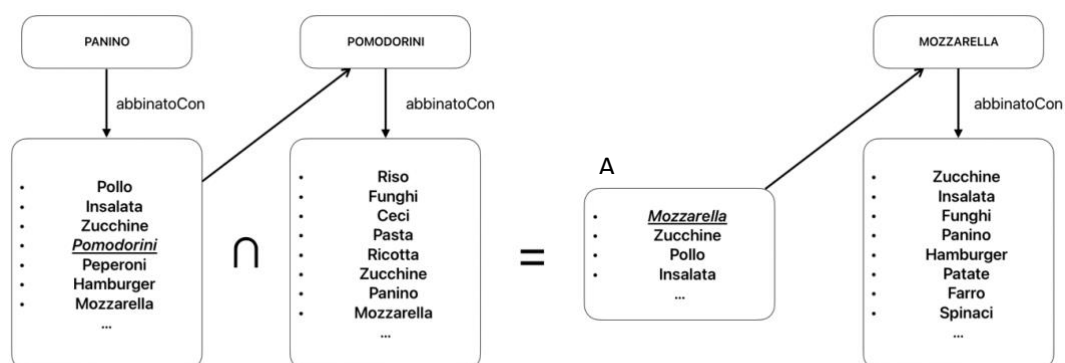
La creazione delle ricette utilizza le query all’ontologia partendo da una lista di alimenti ottenuta dal recommender system (discusso nel capitolo successivo) creando una ricetta distinta per ognuno degli alimenti presenti nella lista.

Abbiamo creato un algoritmo che individui tutti gli ingredienti della ricetta, basandosi sugli abbinamenti di ogni alimento ottenuti con la query: partendo da un ingrediente (1) della lista citata sopra, vengono considerati tutti i suoi abbinamenti. Tra questi se ne sceglie uno casualmente (2) e si crea un insieme (A) risultante dall’intersezione degli abbinamenti dell’alimento 1 con l’insieme degli abbinamenti

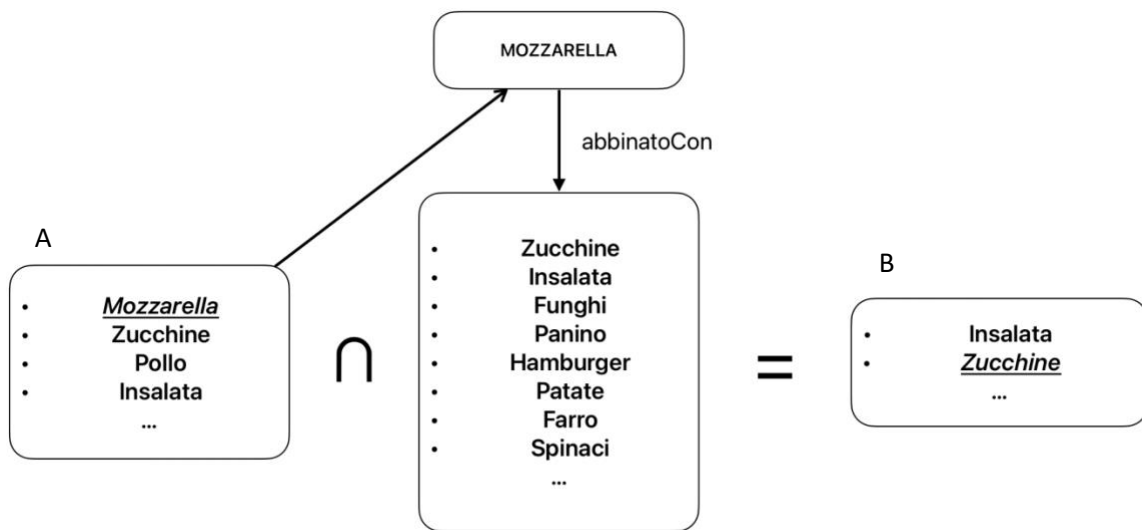
dell'alimento 2; si procede poi ciclicamente, scegliendo un alimento (3) dall'insieme A. Si crea quindi un nuovo insieme B che è l'intersezione di A con l'insieme degli abbinamenti dell'alimento 3. Il ciclo si ferma quando si raggiunge il numero di alimenti prefissato (scelto casualmente tra 4 e 5) o quando l'intersezione che si crea è vuota, ossia non è più possibile abbinare alcun alimento a tutti i precedenti. Tutti gli alimenti scelti vengono poi inseriti in una lista.



In questo esempio il panino è l'Alimento 1 e tra i suoi abbinamenti vengono scelti casualmente i pomodorini. La lista è ora [panino, pomodorini]



Mostriamo come si crea l'intersezione A tra gli abbinamenti del primo e secondo alimento dalla quale verrà scelto un terzo alimento che si aggiunge alla lista: [panino, pomodorini, mozzarella]



Viene poi intersecato l'insieme A con l'insieme degli abbinamenti del terzo alimento ottenendo l'insieme B, dal quale viene scelto il quarto alimento. Supponendo di voler creare una ricetta con quattro ingredienti, si ottiene la lista completa, composta da [panino, pomodorini, mozzarella, zucchini].

Una volta ottenuta la lista con tutti gli ingredienti, si procede al calcolo del peso e alla scelta della cottura di ognuno. Si eseguono le query per ottenere metodi di cottura, numero di Calorie e tipologia. Si costruisce la ricetta nel seguente modo:

Per ogni alimento nella lista

1. La query per i metodi di cottura ne restituisce uno casualmente
2. Si ottengono le Calorie con apposita query
3. Si ottiene la tipologia con apposita query
4. Per ogni tipologia presente nell'ontologia ci sono dei valori minimi e massimi prefissati che indicano il range di calorie assumibili dagli alimenti di quella tipologia (es. è possibile assumere un minimo di 250 Calorie dalla carne e un massimo di 280). Questo consente di calcolare il peso dell'ingrediente all'interno della ricetta. Si ottiene, in questo modo, che un alimento molto calorico sarà presente nella ricetta in quantità minore rispetto ad uno meno calorico.

RECOMMENDER SYSTEM

Abbiamo strutturato un sistema di raccomandazione degli alimenti basato su algoritmo KD Tree che considera le caratteristiche degli alimenti per consigliare cibi simili che siano nutrizionalmente e qualitativamente simili a quelli preferiti dagli utenti.

STRUMENTI UTILIZZATI

Abbiamo utilizzato la libreria Python Scikit-learn per l'implementazione dell'algoritmo KD-Tree. Tale algoritmo è stato scelto in quanto adeguato all'utilizzo di dataset di piccole dimensioni e con un basso numero di features. Un dataset utilizzato è stato ottenuto tramite l'esportazione di un file .csv mediante l'uso del plug-in "CSV export" di Protégé. Esso contiene gli alimenti e alcune delle relative informazioni presenti nell'ontologia.

PREPROCESSING DEI DATI

Per l'addestramento e l'utilizzo dell'algoritmo abbiamo utilizzato due dataset (presenti nella cartella dataset del progetto):

- food-dataset.csv, contenente i nomi di tutti gli alimenti, la tipologia, il numero di calorie, di carboidrati, di grassi e di proteine
- voti-dataset.csv, contenente i nomi degli stessi alimenti con i rispettivi voti assegnati da ogni utente

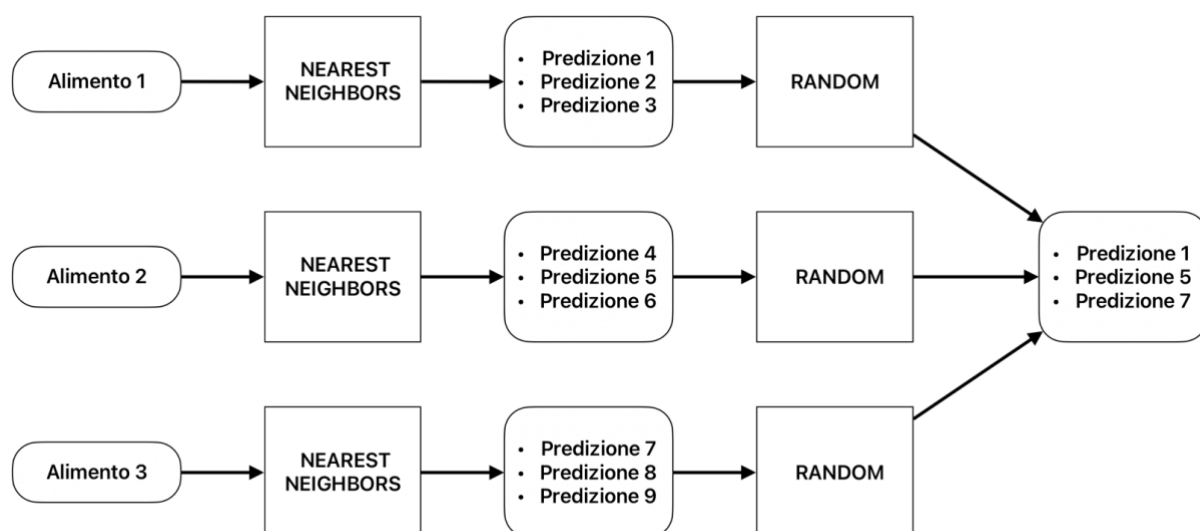
Il preprocessing dei due dataset si articola come segue:

1. Rimozione di caratteri superflui e dell'ultima colonna vuota del file
2. Merge dei due dataset basandosi sul nome dell'alimento
3. Conversione dei valori categorici del dataset tramite la creazione di variabili fittizie con valore booleano tramite funzione `get_dummies()` della libreria Scikit-learn
4. Normalizzazione dei dati trasformando i valori numerici in valori compresi tra 0 e 1 utilizzando la funzione `MinMaxScaler()` della libreria Scikit-learn

UTILIZZO DELL'ALGORITMO

Il sistema di raccomandazione utilizza la ricerca dei 3 vicini più vicini. Per farlo abbiamo creato una funzione che seleziona, all'interno del dataset, i tre alimenti con il voto massimo dell'utente selezionato. Successivamente utilizziamo l'algoritmo per la ricerca dei Nearest Neighbors per ottenere una lista dei tre alimenti più simili ad ognuno di quelli dati. Si ottengono così tre liste, da ognuna delle quali verrà estratto un alimento al fine di offrire varietà di scelta all'utente.

Di seguito uno schema riassuntivo del funzionamento della procedura appena spiegata.



La lista ottenuta verrà fornita in input alla funzione utilizzata per la creazione delle ricette.

SCELTA DELLE FEATURES

La ricerca dei Nearest Neighbors si basa sulla somiglianza degli alimenti secondo i seguenti criteri:

- Tipologia dell'alimento: gli alimenti appartenenti alla stessa tipologia (carne, pesce...) sono chiaramente più simili tra di loro rispetto a quelli di altre tipologie
- Valori nutrizionali: suddivisi come spiegato precedentemente in calorie, carboidrati, grassi e proteine per 100 grammi; consentono di trovare alimenti nutrizionalmente simili tra loro
- Voto dell'utente: consente, data la scelta degli alimenti in input tra quelli con voto più alto, di ritrovare altri alimenti con voto maggiore rispetto ad altri.

MODIFICA DEI VOTI

Per affinare il sistema di raccomandazione, è data all'utente la possibilità di assegnare dei voti ad una delle ricette proposte dal sistema. il voto, unico per la ricetta e compreso tra 1 e 10, verrà utilizzato per il calcolo del nuovo voto di ogni ingrediente. Il nuovo voto viene calcolato per ogni alimento della ricetta come la media tra il nuovo voto e il voto precedente dell'alimento presente nel dataset.

CONSTRAINT SATISFACTION PROBLEM

Poiché il progetto è mirato ad aiutare l'utente nel rapporto con il suo stile alimentare, abbiamo ritenuto opportuno fornirgli alcuni strumenti che gli consentano una migliore gestione dei prodotti presenti in casa. Tra questi la possibilità di visualizzare, per ogni ricetta consigliata, la disponibilità degli ingredienti, ossia se gli stessi sono presenti in casa e non sono scaduti. Per fare ciò ci siamo avvalsi del CSP, implementato tramite libreria Python Constraint.

Per CSP si intende un problema di soddisfacimento dei vincoli ed è definito da un insieme di variabili (X_1, X_2, \dots, X_n) i cui valori appartengono a domini (D_1, \dots, D_n) e su un insieme di vincoli (C_1, \dots, C_n). Un vincolo su un insieme di variabili è una restrizione di valori che le variabili possono assumere simultaneamente, ovvero, un vincolo tra k variabili $C(X_{i1}, X_{i2}, \dots, X_{ik})$ è un sottoinsieme del prodotto cartesiano delle variabili coinvolte (D_{i1}, \dots, D_{ik}) che specifica quali valori delle variabili sono compatibili con le altre.

Il nostro problema si articola come segue:

VARIABILI

La scelta delle variabili si basa sull'obiettivo che vogliamo raggiungere sfruttando il CSP, cioè essere a conoscenza della presenza di un alimento in casa e della sua fruibilità dipendentemente dalla data di scadenza; pertanto, le due variabili sono:

- “alimento”
- “data di scadenza”

DOMINI

Per l'individuazione dei due domini sono state fatte alcune considerazioni sulle variabili:

- Dominio della variabile “alimento”, il quale prende in considerazione la totalità degli alimenti rappresentati nell'ontologia. Per il suo ottenimento viene effettuata una query all'ontologia che restituisce una lista i cui elementi sono i nomi degli alimenti.
- Dominio della variabile “data di scadenza”, il quale, per motivi computazionali e progettuali, è stato ristretto ad un insieme di date che partono da due anni prima della data corrente a quattro anni successivi alla stessa.

VINCOLI

I vincoli considerati sono utili per far sì che dall'insieme di soluzioni trovate vengano eliminate quelle che effettivamente non corrispondono ai criteri di ricerca (prodotto presente in casa e non scaduto). Ci siamo avvalsi di un dizionario salvato nel file list.npy della cartella data. Questo dizionario, che è modificabile dall'utente con apposite funzioni, contiene gli alimenti presenti in casa e la rispettiva scadenza per la realizzazione dei seguenti vincoli:

- Il valore della variabile “alimento” dev'essere presente nella lista, scartando automaticamente gli alimenti che non sono in casa
- Il valore della variabile “data di scadenza” dev'essere in un giorno successivo o uguale al giorno corrente, in modo da eliminare dalla soluzione gli alimenti presenti in casa ma scaduti

Il solver utilizzato per la risoluzione del problema è il Backtracking, in quanto adeguato alla risoluzione del nostro problema.

CASI DI TEST

ONTOLOGIA

Mostriamo di seguito il caso di test di una query all'ontologia eseguita con python.

Consideriamo un alimento e stampiamo le sue informazioni relative a:

- Numero di calorie
- Tipologia
- Cotture
- Abbinamenti

Nel secondo esempio mostriamo il caso in cui un alimento non abbia nessun metodo di cottura.

```
Alimento considerato: calamari
Calorie: 79
Tipo: food.pesce
Cotture: ['forno', 'fritti']
Abbinamenti: ['insalata', 'pane', 'patate', 'pomodorini', 'risotto']

Process finished with exit code 0
```

```
Alimento considerato: panna
Calorie: 343
Tipo: food.derivati_animali
Cotture: []
Abbinamenti: ['zucchine', 'pasta', 'prosciutto_cotto', 'salmone_affumicato', 'speck']

Process finished with exit code 0
```

RECOMMENDER SYSTEM

In questo caso di test mostriamo il risultato ottenuto dall'algoritmo per la ricerca dei Nearest Neighbors: i tre alimenti individuati sono quelli con il voto più alto secondo l'utente 1 (il valore di VotoID2 non influisce sul risultato); per ognuno di questi vengono trovati i tre più simili, stampati in modo che i tre cibi più simili al primo alimento siano quelli all'interno della prima lista, e così via per i successivi. Al termine del procedimento vengono scelti tre di questi secondo i criteri indicati nel

paragrafo sul Recommender System che serviranno da input per la creazione delle tre ricette.

```
Alimenti input
      Entity  VotoID1  VotoID2
2      burrata      9.0      8.0
16 filetti_di_tonno      9.0      9.0
25      pane      9.0      6.0
[['cheddar' 'derivati_animali' '400' '2.4' '33' '25']
 ['mozzarella' 'derivati_animali' '300' '2.2' '22' '22']
 ['panna' 'derivati_animali' '343' '4.7' '36' '2']]

[['gamberi' 'pesce' '106' '2.2' '1.7' '20']
 ['cozze' 'pesce' '84' '2.2' '2' '18']
 ['branzino' 'pesce' '97' '0.8' '2.3' '18']]

[['cous_cous' 'carboidrati' '370' '70' '0.2' '3']
 ['farro' 'carboidrati' '323' '59' '2.5' '14']
 ['pasta' 'carboidrati' '374' '72' '1.4' '12']]

predizioni ['panna', 'branzino', 'farro']

Process finished with exit code 0
```

CSP

Nei seguenti casi di test, abbiamo dimostrato il corretto funzionamento del modulo CSP ottenendo i seguenti risultati:

- Spinaci (non presenti in casa): restituiscono esito negativo
- Fave (presenti in casa ma scadute): restituiscono esito negativo
- Pasta (presente in casa e non scaduta): restituisce esito positivo

```
alimento input: spinaci
non presente in casa o scaduto

Process finished with exit code 0
```

```
alimento input: fave
non presente in casa o scaduto

Process finished with exit code 0
```

```
alimento input: pasta
disponibile in casa

Process finished with exit code 0
```

Di seguito mostriamo la lista degli alimenti presenti in casa al momento del test con le rispettive date di scadenza.

Lista alimenti presenti in casa:		
Nome Alimento	Data di scadenza	Calorie
-----	-----	-----
philadelphia	02/06/2025	340
fave	23/02/2023	341
riso	20/12/2023	330
merluzzo	24/12/2023	82
pasta	10/02/2024	374

CASI D'USO

L'unico caso d'uso di questa sezione è quello relativo alla creazione delle ricette. Prendendo una ricetta qualunque come esempio (in questo caso quella creata a partire dal terzo alimento consigliato, dimostriamo, tramite DL Query all'interno di Protégé, come ogni alimento sia abbinato a tutti gli altri della ricetta, a prova del funzionamento della procedura spiegata nel paragrafo sugli utilizzi dell'ontologia

```
In base alle tue preferenze, ti consiglio: cheddar, cozze, pasta
```

Ricetta 3:		
Alimento e Cottura	Quantità	Disponibilità
-----	-----	-----
pasta	110g	disponibile in casa
ceci in padella	40g	non presente in casa o scaduto
pomodorini	60g	non presente in casa o scaduto

DL query:

Query (class expression)

Alimento **that** abbinatoCon **value** pasta

Execute Add to ontology

Query results

Instances (20 of 20)

- carote
- ceci
- cicerchie
- cicorie
- cozze
- fagioli
- fave
- filetti_di_tonno
- funghi
- gamberi
- lenticchie
- panna
- peperoni
- philadelphia
- piselli
- pomodorini
- salmone_affumicato
- speck
- zucca
- zucchine

DL query:

Query (class expression)

Alimento **that** abbinatoCon **value** ceci

Execute Add to ontology

Query results

Instances (7 of 7)

- crostini
- farro
- pane
- pasta
- pomodorini
- speck
- zucca

DL query:

Query (class expression)

Alimento **that** abbinatoCon **value** pomodorini

Execute Add to ontology

Query results

- calamari
- ceci
- cheddar
- cicorie
- costine_di_maiale
- cous_cous
- cozze
- crostini
- fagioli
- farro
- filetti_di_tonno
- filetto_di_manzo
- funghi
- gamberi
- hamburger
- insalata
- lenticchie
- merluzzo
- mozzarella
- pane
- panino
- pasta
- natate

CONCLUSIONI

SVILUPPI FUTURI

Il nostro programma, con adeguati perfezionamenti, sarebbe fruibile come applicazione mobile da un'ampia parte di popolazione, in particolare da chi necessita di idee in cucina o da chi ha bisogno di un sistema di gestione dei cibi in casa propria.

Di seguito un elenco con i limiti del nostro progetto con spunti utili per i relativi perfezionamenti:

- Dimensioni dell'ontologia

La nostra ontologia è una rappresentazione del dominio alimentare.

L'obiettivo sarebbe, teoricamente, quello di rappresentare quanti più alimenti possibile, includendo eventualmente un numero maggiore di macrocategorie (es. aggiungere Bevande, Spezie, Frutta ecc.)

- Sistema di raccomandazione

Il limite in questo caso è rappresentato dalle dimensioni del dataset. Ciò non ci consente di sfruttare al massimo le potenzialità dell'algoritmo e complica l'interpretazione dei valori ottenuti dalle metriche di valutazione.

Aumentando il numero di alimenti considerati si otterrebbe maggiore precisione e alimenti sempre più simili tra di loro

- Creazione delle ricette

Gran parte della creazione delle ricette avviene con algoritmi che sfruttano funzioni random o che sono creati ad-hoc per il nostro progetto.

L'idea è quella di implementare ulteriori algoritmi di apprendimento automatico da utilizzare, nello specifico, per decretare il metodo di cottura di ogni alimento all'interno delle ricette e le quantità, che, aggiungendo le funzionalità adeguate, si baserebbe sul fabbisogno calorico giornaliero dell'utente.

- Gestione degli utenti

Volendo rendere più completo e usabile il nostro programma, sarebbe appropriato implementare un sistema di gestione degli utenti, poiché per ora il loro numero è fissato a due senza possibilità di modifica.

Idealmente il nostro programma potrebbe essere usato tra persone che vivono nella stessa casa, il cui numero potrebbe variare. Il sistema fornirebbe la possibilità di modificare i parametri degli utenti esistenti, di aggiungerne di nuovi o eliminare quelli esistenti. Inoltre, tramite un miglioramento del sistema di aggiornamento dei voti, ad esempio facendo una media tra tutti i voti assegnati allo stesso alimento in ogni utilizzo del programma, si renderebbe più preciso e affidabile il sistema di raccomandazione.

CONSIDERAZIONI FINALI

Nel complesso, ci riteniamo soddisfatti del lavoro svolto in quanto non ci siamo discostati dalle idee iniziali e i limiti del progetto, per quanto esso sia un prototipo, non hanno influito negativamente sul funzionamento del programma. Inoltre, riteniamo che la nostra idea, per quanto originale, si adatti perfettamente alle esigenze degli utenti a cui essa si rivolge.