No wAIste

Alessandro Satta, Roberto Piscopo

2023

Contents

Chapter 1

Introduzione

1.1 Contesto

Il progetto nasce dal problema che molte persone hanno, ovvero avere in dispensa/frigo degli ingredienti dimenticati oppure accumulati e non sanno cosa cucinare. Oppure banalmente ottenere nuove idee per delle ricette in base ad altre che ci piacciono.

1.2 Idea

Il nostro progetto ha lo scopo di creare un agente intelligente capace di consigliare una o più ricette in base a quello che l'utente ha a disposizione, o che gli piace

Chapter 2

PEAS

2.1 Specifiche

La prima cosa da fare quando si progetta un agente intelligente è la definizione delle Specifiche Peas

Performance: Migliore ricetta in base agli ingredienti forniti

Enviroment: L'ambiente nel quale il nostro agente opera è composto da tutte le ricette del dataset con i relativi: titolo, ingredienti, ratings, tempo di preparazione, istruzioni, tipologia L'ambiente risulta:

- Completamente osservabile: si ha accesso a tutte le informazioni di ogni ricetta
- Deterministico: L'ambiente deterministico significa che l'agente sa esattamente cosa aspettarsi in base alle informazioni fornite e agli input dell'utente. In altre parole, l'agente sa che dati gli stessi input, otterrà sempre lo stesso output, l'agente sa che se gli vengono forniti gli stessi ingredienti, troverà sempre la stessa ricetta migliore nell'ambiente statico
- Episodico: l'agente si trova in un singolo episodio in cui riceve gli ingredienti in input e restituisce la migliore ricetta come output, senza alcuna interazione ulteriore con il mondo esterno.
- Statico: L'insieme delle ricette e le relative informazioni non cambiano nel corso del tempo
- Discreto: L'agente ha a disposizione un numero limitato di azioni

Attuatori: L'algoritmo utilizzato per generare le possibili soluzioni e selezionare la migliore.

Sensori: Gli input dell'utente, ovvero la lista degli ingredienti.

Singolo Agente: è presente solo un agente che opera nell'ambiente

Chapter 3

Dataset

3.1 Creazione Dataset

Non esistendo dei dataset di ricette in italiano, abbiamo dovuto crearlo noi. Per farlo abbiamo utilizzato la tecnica del Web Scraping, ovvero una tecnica tramite la quale estrarre dati da un sito web, nel nostro caso da giallo zafferano. Abbiamo utilizzato una libreria di python chiamata recipe-scrapers insieme alla llibreria BeautifulSoup Installabile con il comando pip install recipe-scrapers. BeautifulSoup è stato utilizzata per prendere tutti i link delle ricette delle 426 pagine di GialloZafferano.

```
def searchRecipeLink():
    url_list = ["https://www.giallozafferano.it/ricette-cat/"]
    url_recipe_list = []

    count = 1
    while count < 425:
        url = "https://www.giallozafferano.it/ricette-cat/page" + str(count) + "/"
        url_list.append(url)
        count += 1

    for url in url_list:
        html = requests.get(url).content
        soup = BeautifulSoup(html, 'html.parser')
        links = soup.find_all('a', href=True)
        for link in links:
            recipe_url = link['href']
            if recipe_url.startswith("https://ricette.giallozafferano.it"):
                if ("#anchor=qz-comments-anchor" not in recipe_url):
                    if (recipe_url not in url_recipe_list):
                        url_recipe_list.append(recipe_url)
    return url_recipe_list
```

Dopodichè per ogni pagina di una ricetta sono state usate le funzioni di recipe scrapers

per ottenere le feature da inserire nel dataset quali: titolo ricetta, tempo preparazione, valutazione, categoria/tipologia, istruzioni.

```python
def createRecipesList(url_recipe_list):
    recipes = []
    count = 0
    for url in url_recipe_list:
        scraper = scrape_me(url)

        print(str(count) + " " + scraper.title())
        count += 1
        ingredients = cleanIngredients(scraper.ingredients())
        #ingredients = scraper.ingredients()

        recipe = {'title': scraper.title(), 'ingredients': ingredients, 'time': scraper.total_time(),
                  'ratings': scraper.ratings(), 'type': scraper.category(), 'instructions': scraper.instructions()}

        # Aggiungi le informazioni della ricetta alla lista
        recipes.append(recipe)
    return recipes
```

Poi tutte le ricette sono state salvate in un file csv

```python
def saveRecipe2CSV(recipes):
    import csv

    # Apri il file in modalità di scrittura
    with open('ricetteEsatte.csv', mode='w', newline='') as csv_file:
        writer = csv.writer(csv_file)
        writer.writerow(['title', "ingredients", 'time', 'ratings', 'type', 'instructions'])

        # Scrivi le informazioni delle ricette nel file
        for recipe in recipes:
            row = [
                recipe['title'],
                recipe['ingredients'],
                recipe['time'],
                recipe['ratings'],
                recipe['type'],
                recipe['instructions']
            ]

            writer.writerow(row)
        print("dataset cvs creato\n")
```

3.2 Data Preparation & Data Cleaning

Un primo problema si è presentato con la lista degli ingredienti. Se prediamo ad esempio la ricetta “Tiramisù” i suoi ingredienti sono:

Mascarpone 750 g
Uova(freschissime, circa 5 medie) 260 g
Savoiardi 250 g

Zucchero 120 g

Caffè(della moka, zuccherato a piacere) 300 g

A noi interessano solo i nomi degli ingredienti, in quanto saranno utilizzati per trovare ricette simili dati degli ingredienti in input. Quindi andavano cancellate tutte le grammature e altre informazioni non necessarie come "freschissime, circa 5 medie".

Per fare questo è stato creato un programma sempre in python che andava a rimuovere le grammature, i numeri, informazioni nelle parentesi ed informazioni di contorno in generale. Inoltre bisognava anche andare a sostituire alcuni ingredienti composti usando solo l'ingrediente principale ad esempio: Estratto di vaniglia -¿ Vaniglia.

Poi venivano rimossi tutti i caratteri non necessari come: [,](,) /, ecc

Andavano rimosse tutti quei dettagli come "piccola". E se ci sta piccolo o piccoli ? Abbiamo rimosso tutte le varianti manualmente, (solo dopo siamo venuti a conoscenza di funzioni automatiche di lemmatizzazione e stemming)

Tutti gli ingredienti sono stati portati in lower Case.

Fatto ciò la lista di ingredienti pulita veniva salvata come stringa ed inserita al posto della lista di ingredienti precedente alla pulizia. Alla fine del processo veniva creato un novo dataset.

La fase di pulizia del dataset però non finiva qui. Abbiamo creato un programma per l'esplorazione del dataset in modo tale da effettuare il conteggio delle ricette, del tempo medio per la preparazione , del numero di ricette per tipologia, del numero di ingredienti totali, del numero di ingredienti più usati, di quelli meno usati, del conteggio per ogni singolo ingrediente. Ed è proprio quest'ultimo dato che ci ha permesso di pulire ulteriormente il dataset da ricette superflue. Difatti abbiamo eliminato circa un centinaio di ricette che avevano ingredienti unici, troppo specifici che non ci avrebbero aiutato nell'identificazione di ricette simili, in quanto uniche nel loro genere. (Per il codice vedere removeRow.py in RecipeScraper)

```
# Carica il dataset
df = pd.read_csv('ricette.csv',encoding='iso-8859-1')

df.dropna(subset=['ingredients'], inplace=True)

print(df.shape)
print(df.head)
```

[6037 rows x 6 columns]

Il nostro dataset ha poco più di 6000 ricette ed è un file CSV.

```
(6037, 6)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6037 entries, 0 to 6038
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   title         6037 non-null   object
 1   ingredients   6037 non-null   object
 2   time          6037 non-null   int64
 3   ratings       6037 non-null   float64
 4   type          6037 non-null   object
 5   instructions  6036 non-null   object
dtypes: float64(1), int64(1), object(4)
memory usage: 330.1+ KB
None
```

Esempio prime righe...

Titolo	Ingredienti	tempo	Valutazione	Tipologia	Istruzioni
Tiramisù	mascarpone,uova, savoiardi,zucchero, caffè,cacao	46	4.2	Dolci	Per preparare il tiramisù preparate il caffé...
Spaghetti alla carbonara	spaghetti,guanciale, uova,pecorino,pepe	25	4.2	Primi piatti	Per preparare gli spaghetti alla carbonara...
ecc...					

Conteggio ingredienti è stato fatto sia con la libreria nltk sia con la libreria csv leggendo le righe:

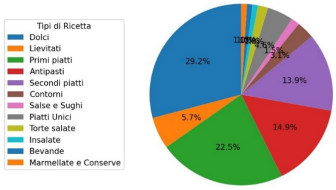
```
vocabulary = nltk.FreqDist()
for ingredients in df['ingredients']:
    ingredients = ingredients.split(',')
    vocabulary.update(ingredients)

for word, frequency in vocabulary.most_common(200):
    print(f'{word}: {frequency}')
    fdist = nltk.FreqDist(ingredients)

common_words = []
for word, _ in vocabulary.most_common(250):
    common_words.append(word)
print(common_words)
```

```
sale: 4385
olio extravergine d'oliva: 3043
pepe: 2954
farina: 2461
uova: 2432
zucchero: 2125
burro: 1929
acqua: 1531
latte: 1371
aglio: 1324
cipolle: 1182
limone: 1106
lievito: 1095
```

Conteggio ricette per Tipologia




```
Numero di ricette per tipo:
Dolci: 1761
Lievitati: 342
Primi piatti: 1356
Antipasti: 898
Secondi piatti: 840
Contorni: 189
Salse e Sughi: 90
Piatti Unici: 277
Torte salate: 109
Insalate: 58
Bevande: 58
Marmellate e Conserve: 61
```

3.3 Feature Scaling

Non c'erano dati numeri particolarmente discordanti e quindi non c'è stato bisogno di scalare e normalizzare.

3.4 Feature Selection

Come già detto , il dataset delle ricette è stato creato e quindi di seguito vengono spiegati i motivi di ogni caratteristica:

- Titolo: è utile per mostrare all'utente la ricetta consigliata
- Ingredienti: è la feature dominante in quanto
- Rating: utile per capire se una determinata ricetta piace agli utenti o meno (non verrà usata nella soluzione finale)
- Prep time:è utile per capire la difficoltà della preparazione
- Tipologia/Categoria: è utile per capire che tipologia di ricetta è

Abbiamo pensato di creare un agente intelligente capace di suggerire delle ricette in base ad una serie di ingredienti in input.

Abbiamo provato a realizzare ciò attraverso gli Algoritmi Genetici. (per il codice vedere cartella GA) (Dataset Vecchio utilizzato : id, title, ingredients, likes, prep_time, Solo 300 ricette)

Un GA è una proceduta ad alto livello (meta-euristica) ispirata alla genetica per definire un algoritmo di ricerca. Evolve una popolazione di individui (soluzioni candidate) producendo di volta in volta soluzioni sempre migliori rispetto ad una funzione obiettivo, fino a raggiungere l'ottimo o un'altra condizione di terminazione. La creazione di nuove generazioni di individui avviene applicando degli operatori genetici, precisamente selezione, crossover e mutazione.

Nel nostro caso le caratteristiche del GA erano le seguenti: Codifica individuo (ricetta) era formata da un array di 8 interi che potevano assumere il valore solo di 0 o 1.

L'intero individuo andava ad individuare l'indice di una riga del dataset e quindi di una ricetta

Size popolazione: 5

Selezione: abbiamo usato la tecnica dei Steady-State GA, invece di generare nuove generazioni si va a migliorare costantemente quella attuale. Si selezionano i due migliori individui come genitori

Crossover: il crossover (Single/Two-Point) avviene tra i due genitori selezionati al punto precedente. Il nuovo individuo andrà a sostituire il peggiore della popolazione. Nel caso in cui si va a creare un individuo già esistente nella popolazione allora si applica la mutazione (bit flip)

Stopping condition: terminiamo l'evoluzione quando si superano le 500 iterazioni o quando si è raggiunta una certa fitness.

La funzione di fitness è data dal numero di ingredienti in comune con quelli inseriti moltiplicati per i likes il tutto diviso per il tempo di preparazione.

Dopo varie prove e modifiche agli operatori genetici ci siamo resi conto che l'algoritmo era un po' troppo casuale. Molte volte trovava le due migliori ricette in una 50ina di generazioni altre volte arrivava fino a 500 iterazioni trovando due ricette con fitness molto basse.

Così abbiamo pensato che la codifica non era adatta o probabilmente non era adatto al nostro problema un algoritmo Genetico.

Quindi abbiamo optato per un algoritmo di ricerca informata. Il più utilizzato tra gli algoritmi di ricerca informata è l'A*. (Dataset Vecchio utilizzato : id, title, ingredients, likes, prep_time, Solo 300 ricette)

Per il codice vedere la cartella AStar

L'algoritmo A* è un algoritmo di ricerca che utilizza una funzione di valutazione per determinare la distanza tra un nodo corrente e un nodo obiettivo. In questo caso, la funzione di valutazione è "evaluate_recipe" che valuta una singola ricetta sulla base di quanti ingredienti corrispondono a quelli forniti in input e sulla base del numero di likes della ricetta.

La funzione "evaluate_recipe" valuta una singola ricetta in base agli ingredienti forniti in input. La funzione utilizza un ciclo for per verificare se ciascun ingrediente della ricetta è presente negli ingredienti forniti in input. Se un ingrediente è presente, la valutazione della ricetta viene incrementata di 1. Inoltre, la valutazione della ricetta viene incrementata anche dal numero di "likes" della ricetta diviso 100,per normalizzare i "likes"

```
def evaluate_recipe(recipe, input_ingredients):
    score = 0
    for ingredient in recipe.ingredients:
        if ingredient in input_ingredients:
            score += 1
    score += recipe.likes / 100  # dividere per 100 per normalizzare i likes
    return score
```

La funzione "find_best_recipe" utilizza l'algoritmo A* per trovare la migliore ricetta possibile. Inizia creando due liste: "open_list" e "closed_list". "open_list" contiene tutte

le ricette del dataset all'inizio, mentre "closed_list" è vuota.

La funzione "find_best_recipe" utilizza l'algoritmo A* per trovare la migliore ricetta possibile dalle ricette date in input. Inizialmente, la lista "open_list" contiene tutte le ricette e la lista "closed_list" è vuota. Poi, all'interno del ciclo while, l'algoritmo seleziona la ricetta con la valutazione più alta dalla lista "open_list" e la sposta nella lista "closed_list". Successivamente, verifica se tutti gli ingredienti della ricetta corrente sono presenti negli ingredienti forniti in input. Se lo sono, la ricetta viene aggiunta alla lista "best_recipes".

L'algoritmo termina quando la lista "open_list" è vuota e restituisce la lista "best_recipes" contentenente le migliori ricette

```python
# funzione di ricerca A*
def find_best_recipe(recipes, input_ingredients):
    open_list = []
    closed_list = []
    best_recipes = []
    best_score = 0
    alternative_recipes = []
    missing_ingredient = []

    open_list = recipes

    while open_list:
        # scegli la ricetta con la valutazione più alta
        current_recipe = max(open_list, key=lambda x: evaluate_recipe(x, input_ingredients))
        open_list.remove(current_recipe)
        closed_list.append(current_recipe)

        score = evaluate_recipe(current_recipe, input_ingredients)
        if score!=0:
            if score == len(current_recipe.ingredients):
                best_recipes.append(current_recipe)  # aggiungo la ricetta corrente alla lista delle migliori ricette
            elif len(current_recipe.ingredients)-score==1:
                otherPossibleRecipe(alternative_recipes, missing_ingredient,current_recipe,input_ingredients)

    return best_recipes, alternative_recipes, missing_ingredient
```

Il programma funzionava abbastanza bene riusciva a dare in output le ricette che ottimizzavano gli ingredienti forniti in input ed anche le ricette a cui mancava un solo ingredienti di quelli a disposizione. Il problema era troppo troppo lento, già con un dataset piccolo.

Facendo un paio di ricerche sul web abbiamo compreso che il nostro era un problema facente parte del NLP, una branca dell'AI. Nello specifico un Probelma di Text Classification, ovvero identificare gli argomenti di un testo nel nostro caso gli ingredienti delle ricette e la loro correlazione.

Che cos'è il Word Embedding?

La semplice definizione di word embedding è la conversione di testo in numeri. Per far sì che il computer capisca il linguaggio, convertiamo il testo in forma vettoriale in modo che i computer possano sviluppare connessioni tra i vettori e le parole e capire ciò che diciamo. Con il word embedding, risolviamo i problemi relativi al Natural Language Processing.

Le rappresentazioni vettoriali dei tokens, chiamate Word Embeddings, sono apprese da modelli NLP, definiti come modelli spaziali vettoriali

Le singole parole sono analizzate e rappresentate atomicamente come unità singole in **Bag-of-Words** (BOW): questo 'bagaglio' di parole, o 'set', sono una rappresentazione

sparsa delle parole e della loro presenza, indipendentemente dall'ordine sintattico in cui appaiono in una data frase.

L'idea è molto semplice: ogni documento del nostro corpus viene rappresentato contando quante volte ogni parola appare in esso. Ad esempio se prendiamo due ricette e quindi due liste di ingredienti(documenti)

R1 Tiramisù : "mascarpone,uova,savoiardi,zucchero,caffè,cacao"

R2 Spaghetti alla Carbonara :"spaghetti,guanciale,uova,pecorino,pepe"

	mascarpone	uova	savoiardi	zucchero	caffè	cacao	spaghetti	guanciale	pecorino	pepe
R1	1	1	1	1	1	1	0	0	0	0
R2	0	1	0	0	0	0	1	1	1	1

Questa viene chiamata matrice documento-caratteristica (document-feature matrix): ogni riga rappresenta un diverso documento (ricetta) e ogni colonna definisce la caratteristica usata per rappresentare il documento

Nel nostro dataset abbiamo circa 560 ingredienti diversi, quindi il modello bag of word crea un'enorme matrice sparsa che memorizza i conteggi di tutte le parole nel nostro corpus (tutti i documenti, ovvero tutti gli ingredienti per ogni ricetta). Per farlo praticamente si puo utilizzare CountVectorizer della libreria scikit-learn (questa libreria la useremo spesso nel nostro discorso)

Bag of Words crea solo un insieme di vettori contenente il conteggio delle occorrenze di parole nelle frasi/corpus, ma non contiene informazioni su parole importanti.

Continuando nelle ricerche abbiamo trovato un altro metodo chiamato TF-IDF (term frequencies- inverse document frequency). Che ha lo scopo di riflettere quanto sia rilevante un termine in un dato documento.

TF-IDF è una tecnica nell'elaborazione del linguaggio naturale per convertire le parole in vettori e con alcune informazioni semantiche e dà peso a parole non comuni, utilizzate in varie applicazioni di NLP.

Ad esempio sale , olio sono presenti in più del 60% delle ricette quindi non sono molto rilevanti al fine di determinare la similitudine tra due ricette. TF-IDF assegnerà un valore più basso al sale. Per una spiegazione più precisa del funzionamento di TF-IDF vedere il seguente link medium.datadriveninvestor.

Per usare TF-IDF in python si utilizza TfidfVectorizer della libreria sk-learn

La libreria scikit-learn TfidfVectorizer esegue un processo di tokenizzazione dei testi per separare le parole individuali (o le frasi) nei documenti di testo e quindi costruisce una matrice di term frequency-inverse document frequency (TF-IDF) come descritto sopra

La matrice TF-IDF è utilizzata per rappresentare i documenti (ricette) come un insieme di features (ingredienti) in un formato che può essere utilizzato come input per i modelli di machine learning.

```
# Carica il dataset
df = pd.read_csv('ricette6k.csv',encoding='iso-8859-1')

df.dropna(subset=['ingredients'], inplace=True)

df['ingredients'] = df.ingredients.values.astype('U')

tfidf = TfidfVectorizer(min_df=5, max_df=0.60,encoding='iso-8859-1',token_pattern = r'[^,]+')
tfidf.fit(df['ingredients'])
tfidf_recipe = tfidf.transform(df['ingredients'])
```

max_df non considera le parole che sono presenti più del 60%

token_pattern = r'[;]+: Questa regex matcha una qualsiasi serie di caratteri che non sia una virgola

Il modello TF-IDF viene addestrato sull'intero dataset di ingredienti utilizzando il metodo fit().

Il modello viene quindi utilizzato per trasformare questi ingredienti in una rappresentazione numerica utilizzando il metodo transform(). Questa rappresentazione viene chiamata "codifica TF-IDF".

Per Misurare la somiglianza tra le liste degli ingredienti e gli ingredienti dati in input abbiamo utilizzato la Similarità del coseno, funzione di Sklearn con le stesse somiglianze e per i dati testuali viene utilizzata per trovare la somiglianza di testi nel documento. Viene usata la funzione ottieni consigli per classificare i punteggi e generare un dataframe, tramite la libreria pandas, contente il titolo, gli ingredienti e il punteggio delle ricette consigliate.

I risultati ottenuti sono accettabili ma molte volte consiglia delle ricette per niente simili.

Per migliorarlo avevamo bisogno di tenere traccia degli ingredienti comunemente usati insieme.

IL nostro problema era: Come facciamo a far comprendere all'agente intelligente che alcuni ingredienti stanno bene insieme dato che sono presenti delle ricette nel dataset in cui compaio gruppi di ingredienti insieme? Per fare questo abbiamo addrestrato un modello chiamtao Word2vec

Per fare ciò abbiamo utilizzato Word2Vec in quanto è altamente efficiente nel raggruppare insieme parole simili.

Per esempio i pomodori sono simili ad altri ingredienti usati spesso insieme.

```
33      print(model_cbow.wv.most_similar(u'pomodori'))
34
35      '''
36      model_cbow.save('model_cbow.bin')
37      print("Word2Vec model successfully trained")
38      '''
```

Shell

```
>>> %Run wordVec.py
 [('melanzane', 0.9986651539802551), ('spaghetti', 0.9984390139579773), ('origano', 0.99
 83479976654053), ('aglio', 0.9978357553482056), ('passata di pomodoro', 0.9977803230285
 645), ('prezzemolo', 0.9969820976257324), ("olio extravergine d'oliva", 0.9968529939651
 489), ('mozzarella', 0.9964960217475891), ('capperi', 0.9962614178657532), ('basilico',
 0.9959782958030701)]

>>>
```

Word2vec è una semplice rete neurale artificiale a due strati progettata per elaborare il linguaggio naturale, l'algoritmo richiede in ingresso un corpus e restituisce un insieme di vettori che rappresentano la distribuzione semantica delle parole nel testo.

Per addestrare il modello, lo strato di input includerà il numero di neuroni pari alle parole del vocabolario. La dimensione dello strato di uscita e dello strato di ingresso rimane la stessa. Tuttavia, la dimensione dello strato nascosto è impostata in base ai vettori delle dimensioni delle parole risultanti. È possibile eseguire l'incorporazione di parole con Word2Vec attraverso due metodi. In entrambi i metodi sono necessarie reti neurali artificiali. Questi metodi sono:

CBOW o Common Bag of Words e Skim Gram

Nel nostro caso abbiamo usato CBOW

Ci sono due modi in cui Word2Vec può essere addestrato: quello esposto — dove un vettore di parole è usato per predire un contesto target — che è detto metodo skip-gram; il secondo invece è denominato metodo "Continuous Bag of Words" (CBOW), il quale utilizza il contesto per predire una parola target basata su una sequenza contigua di n-grammi.

Per realizzarlo in python abbiamo usato la libreria gensim e seguito alcune guide online come questa: guida.

Dovevamo rappresentare ogni documento corpus (lista di ingredienti per ricetta) come un singolo incorporamento in modo tale da calcolare le somiglianze.

Di seguito il codice per allenare il modello Word2Vec

```
df = pd.read_csv('ricette6k.csv',encoding='iso-8859-1')

df.dropna(subset=['ingredients'], inplace=True)
# ottieni il corpus
corpus = get_and_sort_corpus(df)

# allena e salva il modello  Word2Vec CBOW
model_cbow = Word2Vec(corpus, sg=0, workers=8, window=get_window(corpus), min_count=1, vector_size=100 )
```

- Vector_size= n, permette di decidere la dimensione dei vettori che l'algoritmo andrà a creare.

- window= n, permette di decidere la massima distanza tra la parola corrente e quella predetta all'interno di una frase. Nel nostro caso è dato dalla media di ingredienti in una ricetta (9)
- min_count=n, utile per capire se una determinata ricetta piace agli utenti o meno (non verrà usata nella soluzione finale)
- workers= n, permette di decidere quanti thread del processore usare per addestrare il modello.
- sg= o, questo crea un modello CBOW

Usiamo tf-idf per aggregare gli accoparment

Calcoliamo la media ponderata di tutti gli incorporamenti di parole di ogni documento(lista ingredienti) e ridimensioniamo ogni vettore usando la sua frequenza inversa del documento (IDF).

IDF appesantirà i termini molto comuni in un corpus (nel nostro caso parole come olio extravergine d'oliva o sale e soppeserà i termini rari.

Questo sarà vantaggioso per noi perché ci conferirà una maggiore capacità di separare le ricette, poiché gli ingredienti che verranno modificati saranno quelli che l'utente tenderà a non fornire come input al sistema di consiglio.

Word2Vec cerca di prevedere le parole in base all'ambiente circostante, quindi era fondamentale ordinare gli ingredienti in ordine alfabetico

```
#ottieni il corpus con gli ingredienti in oridne alfabetico
def get_and_sort_corpus(data):
    corpus_sorted = []
    for doc in data.ingredients.values:
        doc=doc.split(",")
        doc.sort()
        corpus_sorted.append(doc)
    return corpus_sorted

#calcola la lunghezza media di ogni doc (lista ingredienti ricetta)
def get_mean(corpus):
    lengths = [len(doc) for doc in corpus]
    avg_len = float(sum(lengths)) / len(lengths)
    return round(avg_len)
```

Abbiamo poi utilizzato sia la similarità del coseno si a knn per trovare le ricette simili.

La somiglianza del coseno :

```
# usa TF-IDF come pesi per ogni incorporamento di parole
tfidf_vec_tr = TfidfEmbeddingVectorizer(model)
tfidf_vec_tr.fit(corpus)
doc_vec = tfidf_vec_tr.transform(corpus)
doc_vec = [doc.reshape(1, -1) for doc in doc_vec]
assert len(doc_vec) == len(corpus)

print('Shape of word-mean doc2vec...')
print(tfidf_vec_tr.transform(corpus).shape)


input = ingredients
input = input.split(",")



input_embedding = tfidf_vec_tr.transform([input])[0].reshape(1, -1)

# ottieni la smilarità del coseno e tutti gli embeddings nel documento
cos_sim = map(lambda x: cosine_similarity(input_embedding, x)[0][0], doc_vec)
scores = list(cos_sim)
# prendi le prime N recommendations
recommendations = get_recommendations(N, scores)
return recommendations
```

Abbiamo testato il funziomento anche con KNN. Il K-Nearest Neighbors (KNN) è un algoritmo di classificazione o regressione che prende in considerazione i K oggetti più vicini in un insieme di dati per prevedere la classe o la quantità prevista

In questo caso abbiamo usato KNN per trovare le ricette più vicine rispetto a agli ingredienti che l'utente ha fornito. Per fare questo, prima abbiamo convertito i nomi degli ingredienti in vettori utilizzando un modello di embedding. Questi vettori hanno una rappresentazione in uno spazio a più dimensioni che rappresenta la somiglianza semantica degli ingredienti.

Successivamente, abbiamo calcolato la distanza tra il vettore di embedding della ricetta dell'utente e il vettore di embedding di ogni ricetta nella nostra base di dati utilizzando la distanza Euclidea. Infine, abbiamo selezionato le K ricette più vicine (dove K è un parametro che possiamo impostare) e abbiamo restituito i nomi di queste ricette come raccomandazioni, usando la funzione get_recommendations.

In sintesi, KNN è stato utilizzato per trovare le ricette più simili a quella fornita dall'utente utilizzando la distanza Euclidea tra i vettori di embedding degli ingredienti.

Metriche valutazione KNN:

	precision	recall	f1-score	support
0	0.43	0.58	0.50	162
1	0.57	0.40	0.47	10
2	0.36	0.35	0.35	40
3	0.83	0.96	0.89	368
4	0.25	0.08	0.12	12
5	0.57	0.51	0.54	68
6	1.00	0.11	0.20	18
7	0.29	0.16	0.21	43
8	0.75	0.72	0.73	291
9	0.21	0.16	0.18	19
10	0.66	0.56	0.60	158
11	0.00	0.00	0.00	19
accuracy			0.67	1208
macro avg	0.49	0.38	0.40	1208
weighted avg	0.66	0.67	0.65	1208

```
tfidf_vec_tr = TfidfEmbeddingVectorizer(model)
tfidf_vec_tr.fit(corpus)
doc_vec = tfidf_vec_tr.transform(corpus)

input = ingredients
input = input.split(",")

input_vec = tfidf_vec_tr.transform([input])

# ottieni KNN tra input embedding ae tutti gli embeddings nel documento
knn = NearestNeighbors(n_neighbors=N, algorithm='ball_tree')
knn.fit(doc_vec)
distances, indices = knn.kneighbors(input_vec)
recommendations = get_recommendations(N, indices[0])

return recommendations
```

KNN: input: ”farina,uova,grana padano,pangrattato”

```
---Gnocchi alla parigina
latte,uova,farina,parmigiano reggiano,burro,noce moscata,sale,groviera,pepe

---Bombe di patate veloci
patate,farina,lievito,uova,sale,prosciutto cotto,scamorza,origano,semi

---Crepe alla crema di asparagi
latte,farina,uova,sale,burro,noce moscata,asparagi,groviera,pepe,parmigiano reggiano

---Sformati di broccoli con salsa al caprino
broccoli,pepe,sale,uova,grana padano,burro,latte,farina,noce moscata,caprino

---Funghi fritti con maionese di barbabietola
funghi,pangrattato,farina,timo,uova,latte,sale,barbabietole,semi,limone,zenzero
```

COS_SIM: input ”farina,uova,grana padano,pangrattato”

```
---Passatelli in brodo
uova,pangrattato,parmigiano reggiano,limone,noce moscata,sale,brodo,farina

---Uova gratinate
uova,pancetta,burro,farina,latte,parmigiano reggiano,noce moscata,sale,pepe,prezzemolo

---Patate duchessa
patate,burro,grana padano,uova,sale,pepe

---Asparagi alla milanese
uova,grana padano,pepe,asparagi,burro

---Gnoccone agli spinaci
patate,farina,uova,sale,pepe,spinaci,prosciutto cotto,grana padano,aglio,burro
```

Chapter 4

GUI

Per creare l'interfaccia grafica dell'applicazione abbiamo usato la libreria customtkinter di python chepermette di crare una gui semplice e minimale. Inoltre è possibile non solo inserire una serie di ingredienti che abbiamo , ma anche selezionare a video delle ricette che ci piacciono per farci consigliare altre ricette simili.

4.1 Input Vocale

```
def get_audio_input():

    r = sr.Recognizer()

    with sr.Microphone() as source:
        print("Dimmi gli ingredienti che hai ")
        audio = r.listen(source)
    ingredients = r.recognize_google(audio, language ="it-IT")
    ingredients_list = ingredients.split()
    ingredients_string=",".join(ingredients_list)
    input_text.delete(0,tk.END)
    input_text.insert(0,ingredients_string)
```

Questo codice è una funzione chiamata "get_audio_input" che utilizza la libreria "SpeechRecognition" (sr) per riconoscere la voce dell'utente e tradurla in testo. La funzione utilizza un microfono come fonte di input audio e chiede all'utente di elencare gli ingredienti che ha a disposizione. Quando l'utente inizia a parlare, il codice registra l'audio e lo passa al metodo "recognize_google" dell'oggetto "Recognizer" per ottenere la traduzione in testo. Il risultato viene salvato come stringa in "ingredients". Successivamente, la stringa viene divisa in una lista di parole utilizzando il metodo "split" e la lista viene successivamente trasformata in una stringa separata da virgole utilizzando il metodo "join". Infine, il testo viene inserito nella casella di testo visualizzata sull'interfaccia utente.

Riferimenti Word2Vec : link.
NLP: link , link , link.
Web Scraping : link.
Libreria python : link.