Ingegneria della Conoscenza

Event classifier:

Il seguente progetto è stato elaborato degli studenti:

- Congedo Alessandro (mat. 677315)
- Romano Andrea (mat. 735541)

questo logo è stato generato da un AI.

Il progetto è nato da un esigenza ovvero quella di arricchire il precedente lavoro dei **BroUsers** (gruppo di sviluppatori indipendenti di cui entrambi facciamo parte) chiamato **OneNight**, un **bot Telegram** che raccoglie informazioni sugli eventi tramite il crawling delle piattaforme social e suggerisce agli utenti eventi in una determinata zona geografica a seconda delle loro preferenze con un sistema di apprendimento automatico che in grado di estrapolare la semantica da descrizioni testuali e inserirla all'interno di una KB che tramite regole sia in grado di consigliare agli utenti e scavare il web per gli eventi che fanno al caso loro.

DATASET:

Il dataset è stato creato tramite un crawler che, automaticamente data una categoria di **Google My Business** e una specifica area geografica cerca i riferimenti alle pagine Facebook per ogni organizzatore che trova all'interno del database di Google My Business.

```
def AskChatGpt(evento):
    #open the file
    # Sostituisci "YOUR_PROMPT" con la tua domanda
    if (len(evento['descrizione'])>3900):
        evento['descrizione'] = evento['descrizione'][:3900]
    prompt = "Dimmi dalla descrizione di questo evento se è un e

    # Invia la richiesta di completamento del testo a me
    completion = openai.Completion.create(
        engine="text-davinci-003",
        prompt=prompt,
        max_tokens=1024,
        temperature=0.5,
    )

    charachterization = {
                'id': evento['id'],
                'isFood': 0,
                'isCulture': 0,
                'isFun': 0,
                'isOpen': -1,
                'isMusic': 0,
                'description': evento['descrizione']
            }
```

Tramite specifici moduli che passano da **G-maps**, a **Google** fino a **Facebook** abbiamo quindi creato un CSV con più di 6000 eventi da cui abbiamo estratto la descrizione.

Inizialmente abbiamo provato ad etichettare gli eventi stessi tramite **ChatGPT** di **OPEN AI**, tramite una query al modello ,la funzione invia una domanda a ChatGPT che descrive l'evento e aspetta una risposta in cui ChatGPT fornisce una classificazione dell'evento come **"APERTO", "MUSICA", "CULTURA", "DIVERTIMENTO" o "MANGIARE"**. La funzione elabora la risposta di ChatGPT e crea un dizionario chiamato che rappresenta la classificazione dell'evento. Il dizionario include i seguenti campi: `id` che rappresenta l'identificatore univoco dell'evento, `isFood`, `isCulture`, `isFun`, `isOpen`, `isMusic` e `description` che rappresenta la descrizione dell'evento. La funzione assegna valori 1 o 0 a `isFood`, `isCulture`, `isFun` e `isMusic` a seconda che la parola chiave corrispondente sia presente o meno nella risposta di ChatGPT.

Ci siamo tuttavia resi conto di quanto questo approccio fosse impreciso e introducesse molte inesattezze certando abbiamo quindi provveduto a correggere il dataset **a mano** , e tramite il supporto di un algoritmo di **clustering**.

Il clustering è un'utile tecnica di apprendimento **non supervisionato** che consente di raggruppare insiemi di dati simili in "**cluster**". Questa tecnica è particolarmente utile per etichettare un dataset precedentemente etichettato male in quanto può aiutare a identificare eventuali pattern o relazioni nascoste all'interno del dataset.

In molte situazioni, **i dati possono essere etichettati in modo errato a causa di diversi fattori come l'errore umano o la mancanza di informazioni complete.** In questi casi, il clustering può essere utilizzato per raggruppare i dati in modo più significativo e quindi per correggere eventuali etichette errate.

Ad esempio, in una ricerca pubblicata su **"IEEE Transactions on Knowledge and Data Engineering"** (Liu et al., 2012), gli autori hanno utilizzato il clustering per correggere etichette errate in un dataset di immagini. Il loro approccio ha permesso di migliorare notevolmente l'accuratezza della classificazione rispetto ai metodi tradizionali di classificazione supervisionata.

Per fare questo abbiamo scritto la funzione "**predict_closest_clusters**" prende in input un data-frame, una descrizione, un modello di clustering K-Means "km", una matrice "X" e un oggetto di vectorizer . La descrizione viene elaborata per estrarre i sinonimi delle parole usando la funzione "get_syns" e convertita in una rappresentazione tf-idf utilizzando il vettorizzatore.

Successivamente, vengono calcolate le distanze tra la descrizione e ognuno dei 10 cluster utilizzando la funzione "**transform**" del modello

RAPPRESENTAZIONE:

Abbiamo utilizzato diverse maniere per rappresentare la **conoscenza** a seconda del livello di semantica che ci conveniva avere.

text	filtered_desc	wordnet_desc
Martedi 05 gennaio super tombolata al Barfly!\...	super tombolata barfly tante calze vincere for...	briefcase rule win bun superintendent booking...
Ingresso free senza obbligo di consumazione,se...	ingresso free obbligo consumazione serata disc...	entrance responsibility bar discus free eveni...
ovedi 8 Marzo, Napulitanata "celebra la donn...	napulitanata celebra donna serata eccezione ma...	level museum trick arcade evening result porc...
yn e lieta di presentare\n\nNapoolyn x Resilie...	yn lieta presentare napoolyn x resilienza reco...	propaganda allow single project producer refl...
Serata live \nVENERDI' 18 MARZO \nUn piacevole...	serata live piacevole ritorno palco biergrube ...	auction_block wallop evening passion group Lo...
...	...	...
Hot Club Roma Trio\n\nMoreno Viglione chitarra...	hot club trio moreno viglione chitarra solista...	project jazz trio double_bass group first whi...
Presentazione del libro:\n "Tasmania" (Einaudi...	presentazione libro tasmania einaudi editore p...	feeling novel contemporary leak check future ...
n occasione delle GIORNATE EUROPEE DEL PATRIM...	occasione giornate europee patrimonio ore muse...	park concert museum cello uracil ore backgrou...
Vieni a Danzare gratuitamente per un giorno co...	vieni danzare gratuitamente giorno amici tagga...	dance day acting morning headquarters contemp...
Il convegno, in linea col precedente Leducazio...	convegno linea precedente leducazione scoperta...	park test_suit discovery jab line project Jen...

La funzione ***reduced_description*** si occupa di ridurre una descrizione data in una rappresentazione più semplificata. Questa funzione effettua una tokenizzazione della descrizione in input utilizzando un tokenizzatore (tkn), quindi riduce le parole ai loro sinonimi più comuni utilizzando la libreria WordNet di NLTK.

La rappresentazione più semplificata viene ottenuta tramite l'utilizzo delle "**synset**", ossia insiemi di sinonimi, per ogni parola nella descrizione in input. La funzione cerca il sinonimo più comune per ogni parola utilizzando il modulo **lemma** e prendendo il sinonimo con il numero più elevato di occorrenze (determinato tramite la funzione count()).

In questo modo, la funzione fornisce una **rappresentazione più sintetica della descrizione** in input, che può essere utilizzata per comparare descrizioni più facilmente. Questa tecnica di riduzione delle parole ai loro sinonimi più

```
def reduced_description(description):
    # Tokenize the description
    tokens = tkn(description)
    # Reduce the words to their most common synset
    reduced_description = []
    for token in tokens:
      try:
          synset = wn.synsets(token, lang='ita',pos=wn.NOUN)
          if(synset == []):
            synset = wn.synsets(token, lang='ita',pos=wn.VERB)
            if(synset == []):
              synset = wn.synsets(token, lang='eng',pos=wn.NOUN)
              if(synset == []):
                synset = wn.synsets(token, lang='eng',pos=wn.VERB)
          lemma = max(synset[0].lemmas(), key=lambda x: x.count()).name()
          reduced_description.append(lemma)
      except:
          continue
    return list(set(reduced_description))
```

comuni è spesso utilizzata in ambito **NLP (Natural Language Processing)** per semplificare le rappresentazioni delle testi e rendere più facile la loro comparazione.

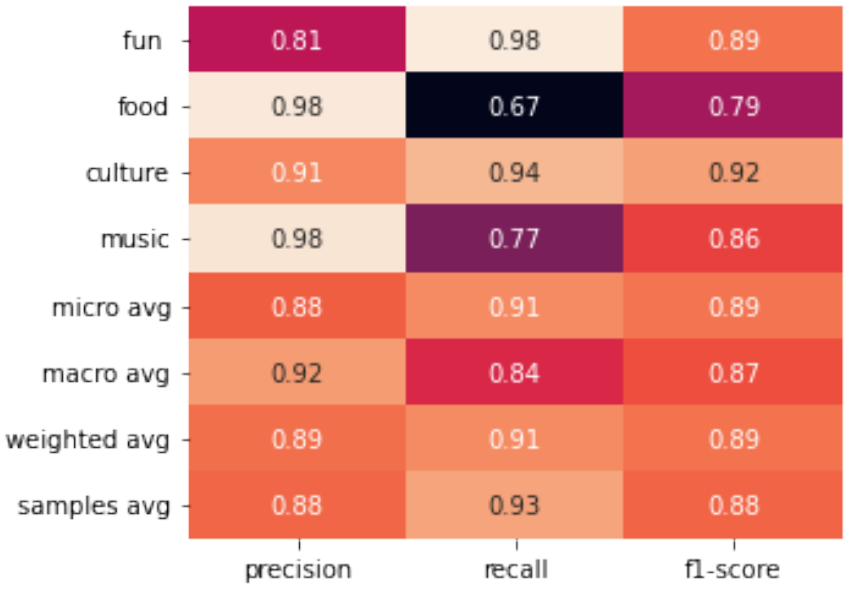
Le seguenti funzioni sono state inoltre utilizzate per estrarre la semantica del testo. La funzione "**remove_numbers**" rimuove tutti i numeri dalla stringa di input utilizzando la funzione "translate" con una tabella di traduzione creata che mappa tutti i numeri a None. La funzione "**remove_punctuation**" rimuove i simboli di punteggiatura dalla stringa di input sostituendoli con spazi. La funzione "**remove_non_N_V**" utilizza un pipeline di **Stanza** per identificare solo le parole che sono nomi o verbi, che sono considerati più importanti per la semantica del testo rispetto agli altri part of speech

APPRENDIMENTO:

La tecnica di **classificazione a multipli riferimenti (OneVsRest)** è una tecnica utilizzata per la classificazione multi-etichetta, ovvero quando un esempio può appartenere a più di una classe. Questa tecnica addestra un classificatore per ogni classe, con l'obiettivo di distinguere tra la classe specifica e tutte le altre. Questo approccio è utile quando il **dataset** è **sbilanciato**, ovvero quando le diverse classi hanno un numero di esempi diversi. In questo caso, la tecnica **OneVsRest** permette a ogni classificatore di concentrarsi sulle proprietà distintive della classe che deve identificare.

```
classificatori = [OneVsRestClassifier, MultiOutputClassifier,LabelPowerset]
estimatori =[DecisionTreeClassifier(),RandomForestClassifier(), SVC()]
param_grid = {
    'estimator__criterion': ['gini', 'entropy'],
    'estimator__max_depth': [None, 5, 10],
    'estimator__min_samples_split': [2, 10],
    'estimator__min_samples_leaf': [1, 4]
}
```

Il codice che segue utilizza tre tipi di classificatori (**OneVsRestClassifier**, **MultiOutputClassifier**, **LabelPowerset**) e tre estimatori (**DecisionTreeClassifier**(), **RandomForestClassifier**(), **SVC**()) per creare una **griglia di parametri** che vengono utilizzati per l'ottimizzazione dei modelli di classificazione. La griglia di parametri include il criterio di classificazione (**gini o entropia**), la profondità massima dell'albero di decisione (None, 5, 10), il numero minimo di esempi richiesti per dividere un nodo (2 o 10) e il numero minimo di esempi richiesti per essere considerato un foglia (1 o 4). Questa griglia di parametri verrà utilizzata per trovare i migliori parametri per i modelli di classificazione.



La funzione **atlas** invece prende come input addestramento x_train, un set di dati di test x_test, i rispettivi etichette di addestramento y_train e di test y_test, e un insieme di parametri opzionali.

```python
def atlas (estimatori, classificatori, x_train, x_test, y_train, y_test, parametri = None):
  results = []
  for ml in classificatori:
    for each in estimatori:
      clf = ml(each)
      if (parametri is None):
          clf.fit(x_train, y_train)
          y_pred = clf.predict(x_test)
      else:
        try:
          grid_search = GridSearchCV(clf, parametri, cv=5, scoring='accuracy')
          grid_search.fit(x_train, y_train)
          best_clf = grid_search.best_estimator_
          y_pred = best_clf.predict(x_test)
        except:
          clf.fit(x_train, y_train)
          y_pred = clf.predict(x_test)
```

Per ogni combinazione di classificatore e di estimatore, la funzione addestra il modello utilizzando i dati di addestramento e i parametri forniti, quindi fa le previsioni sui dati di test.

In particolare, se i parametri sono forniti, la funzione **cerca il modello migliore** utilizzando la procedura di ricerca sulla griglia di valutazione (GridSearchCV). Altrimenti, addestra direttamente il modello con i dati di addestramento.

La funzione calcola e stampa le metriche di **accuracy**, **loss di hamming** e **Label ranking average precision** per ogni modello addestrato. Inoltre, la funzione restituisce un elenco di risultati che contiene informazioni sul classificatore, sull'estimatore e sulle previsioni.

```python
target_variables = ["culture", "music", "food", "fun"]

# Fit a logistic regression model for each target variable
models = []
for target in target_variables:
    clf = LogisticRegression(random_state=0)
    clf.fit(X, tot[target])
    models.append(clf)

# Generate probability estimates for each target variable
probabilities = np.zeros((tot.shape[0], len(target_variables)))
for i, clf in enumerate(models):
    probabilities[:, i] = clf.predict_proba(vectorizer.fit_tran

# Normalize the probabilities so that they sum up to 1 for each
probabilities /= probabilities.sum(axis=1, keepdims=True)

# Creare un nuovo dataset con le probabilità
new_df = pd.DataFrame(probabilities, columns=target_variables)

# Aggiungere le descrizioni come nuova colonna
new_df["filtered_desc"] = tot["filtered_desc"]
new_df["wordnet_desc"] = tot["wordnet_desc"]
new_df["text"] = tot["text"]
new_df["id"]=tot["id"].astype(str)
# Salvare il nuovo dataset
#new_df.to_csv("probabilities.csv", index=False)
```

In sintesi, la funzione "**atlas**" utilizza queste tre metriche per valutare la precisione dei modelli di classificazione utilizzando diverse combinazioni di algoritmi di apprendimento automatico (**estimatori**) e di tecniche di classificazione a più riferimenti (**classificatori**). La funzione stampa i risultati di ogni combinazione e restituisce una lista con i risultati.

PROBABILITA’ E INCERTEZZA:

Inoltre abbiamo utilizzato la **regressione logistica** per la classificazione di variabili target discrete. In particolare, l'obiettivo è quello di stimare la probabilità che una data descrizione faccia riferimento a una o più categorie specifiche tra quelle individuate dalle variabili target (in questo caso: "culture", "music", "food", “fun").

La regressione logistica è una tecnica di **modellizzazione statistica** che permette di analizzare la relazione tra una variabile dipendente binaria e una o più variabili indipendenti continue o discrete. Il suo utilizzo è molto diffuso in ambito di machine learning, in particolare in applicazioni di classificazione, grazie alla sua semplicità e alla capacità di fornire stime di probabilità.

Nel codice presentato, si utilizzano diversi modelli di regressione logistica, uno per ciascuna variabile **target**. Per ciascun modello, si addestra il classificatore sulla base delle feature estratte dalle descrizioni **presenti nel dataset.** Successivamente, si utilizza il modello addestrato per generare le probabilità che ogni descrizione faccia riferimento alla categoria specifica della variabile target. Infine, si **normalizzano** le probabilità in modo che sommino a 1 per ciascuna descrizione e si creano un nuovo dataset contenente queste probabilità e le altre informazioni relative alle descrizioni.

```
def is_night_event(event_vector, night_vector, threshold=25):
  similarity = fuzz.token_set_ratio(event_vector, night_vector)
  if similarity >= threshold:
    return True
  else:
    return False
```

Deduco la costruzione della regola da delle stime probabilistiche effettuate sul c

```
def categorize_text(df, string):
  words = string.split()
  result = [0, 0, 0, 0]
  count = 0
  for index, row in tot.iterrows():
    for word in words:
      if word in row['text']:
        result[0] = result[0] + row['fun']
        result[1] = result[1] + row['food']
        result[2] = result[2] + row['culture']
        result[3] = result[3] + row['music']
        count = count + 1
        break
  result[0] = result[0] / count
  result[1] = result[1] / count
  result[2] = result[2] / count
  result[3] = result[3] / count
  return result
#questi rappresentano i valori medi su cui definiremo la regola
valoriNotte=categorize_text(tot,'notte')
valoriNotte
```

In generale, è possibile fare riferimento a numerosi studi che hanno dimostrato l'efficacia della regressione logistica in ambito di classificazione di variabili discrete, ad esempio il lavoro di **Hosmer e Lemeshow (2000) intitolato "Applied Logistic Regression"**.

FUZZY LOGIC:

Questo codice utilizza la **logica fuzzy** per determinare se un evento è di notte o meno. La logica fuzzy è un approccio… basato sulla probabilità per la risoluzione di problemi di decisione e di classe di appartenenza. La **logica** fuzzy utilizza le proprietà linguistiche, come le parole imprecise, per descrivere la relazione tra le variabili.

Il primo passo è la definizione del vettore notte, che rappresenta le caratteristiche medie degli eventi di notte. Questo vettore viene calcolato tramite la funzione **categorize_text**, che analizza il testo dei singoli eventi per identificare le parole associate a ciascuna caratteristica e calcola la **media** delle **somme delle caratteristiche associate** a quelle parole.

```
def is_night_event(event_vector, night_vector, threshold=25):
  similarity = fuzz.token_set_ratio(event_vector, night_vector)
  if similarity >= threshold:
    return True
  else:
    return False
```

Il secondo passo è quello di estrarre il vettore delle caratteristiche per ciascun evento che si vuole verificare. Ciò viene fatto tramite la funzione get_event_vector, che estrae le caratteristiche di ogni evento dal dataset e le organizza in un vettore.

Infine, la funzione **is_night_event** utilizza il modulo fuzzy di Python per calcolare la **somiglianza** tra il vettore notte e il vettore dell'evento. La somiglianza viene calcolata utilizzando il rapporto tra token (un metodo di somiglianza basato sulle parole), che utilizza una scorciatoia per la somiglianza di **Jaccard**. Se la somiglianza supera una **soglia definita** (25 in questo caso), la funzione restituisce True, indicando che l'evento è di notte. Altrimenti, la funzione restituisce False, indicando che l'evento non è di notte.

Questo codice utilizza un **approccio simile alle clausole di Horn**, che sono un tipo di regola basate sulla logica fuzzy. Tuttavia, mentre le clausole di Horn sono più strutturate e formalizzate, questo codice utilizza un approccio più semplice basato sulla somiglianza per la classificazione degli eventi.

RECOMMENDER SYSTEM:

Abbiamo infine implementato un un **algoritmo di raccomandazione** basato sul **contenuto** utilizzando la similarità di coseno. La funzione

```
def cosine_similarity(a, b):
    dot_product = np.dot(a, b)
    norm_a = np.linalg.norm(a)
    norm_b = np.linalg.norm(b)
    return dot_product / (norm_a * norm_b)

def content_based_recommendation(event_index, content_matrix, num_recommendations):
    similarity = np.zeros(content_matrix.shape[0])
    for i in range(len(similarity)):
        similarity[i] = cosine_similarity(content_matrix[event_index], content_matrix[i])

    recommendation_indices = np.argsort(similarity)[::-1][1:num_recommendations+1]
    return recommendation_indices
```

cosine_similarity calcola la similarità tra due vettori basandosi sul prodotto scalare tra di loro e la loro norma. La funzione **content_based_recommendation** utilizza la funzione **cosine_similarity** per calcolare la **similarità** tra un evento specifico e tutti gli altri eventi, restituendo poi gli indici degli eventi più simili.

```
num_events = new_df.shape[0]
num_recommendations = 5
recommendation_matrix = np.zeros((num_events, num_recommendations), dtype=int)
content_columns = ["culture", "music", "food", "fun"]
content_matrix = new_df[content_columns].values
for i in range(num_events):
    recommended_indices = content_based_recommendation(i, content_matrix, num_recommendations)
    recommendation_matrix[i, :] = recommended_indices

#np.savetxt("recommendation_matrix.csv", recommendation_matrix, delimiter=",", fmt='%d')
recommendation_matrix
```

La funzione content_based_recommendation viene poi utilizzata all'interno di un ciclo che itera su tutti gli eventi nella **matrice** di contenuti content_matrix e genera una matrice di raccomandazioni **recommendation_matrix** in cui ogni riga rappresenta un evento e ogni colonna rappresenta una raccomandazione per quell'**evento**.Questo tipo di algoritmo di raccomandazione basato sul contenuto è un esempio di **filtraggio** basato sugli **item**, dove la similarità tra gli item viene utilizzata per raccomandare elementi simili a quelli che l'utente ha mostrato interesse in precedenza (ad esempio, acquisto, visualizzazione, etc.). Questo approccio è descritto in dettaglio in molti articoli scientifici, come ad esempio "**Item-Based Top-N Recommendation Algorithms**" di J.Bobadilla et al. (2013).

CONCLUSIONI:

In conclusione, l'integrazione **dell'AI** nei **software** e nell'infrastruttura internet può migliorare notevolmente l'esperienza dell'utente fornendo soluzioni personalizzate e specifiche alle sue esigenze.

La **ricerca** e la **classificazione** degli eventi è solo uno dei molti esempi in cui **l'AI** può essere utilizzata per creare una migliore esperienza per gli utenti. L'uso di tecniche di apprendimento automatico come la **regressione logistica** e la classificazione a **multipli riferimenti** hanno dimostrato di essere altamente efficaci nel classificare eventi in base a diversi attributi. Inoltre, l'utilizzo dei sistemi di **raccomandazione** basati sul **contenuto**, come dimostrato nel nostro progetto, può fornire consigli personalizzati e pertinenti agli utenti in base alle loro preferenze. L'adozione di queste tecniche nei software e nell'infrastruttura internet può portare ad una maggiore soddisfazione degli **utenti** e ad un aumento del **coinvolgimento** degli utenti in queste piattaforme. Numerosi studi hanno dimostrato l'efficacia delle tecniche di **apprendimento automatico** e dei sistemi di raccomandazione basati sul contenuto in vari contesti, come la pubblicità, la raccomandazione di film e la prevenzione delle frodi, sottolineando così l'importanza di considerare l'AI nella progettazione e nello sviluppo di **software.**

CONTATTI:

- **a.romano76@studenti.uniba.it**
- **a.congedo7@studenti.uniba.it**
- **https://github.com/AleCongi/Esame-ICON-**