Università degli studi di
Milano-Bicocca

Text Mining and Search

Progetto finale

Amazon review classification and clustering

Autori:
Dario Carolla - 807547 - d.carolla@campus.unimib.it
Federico Manenti - 790032 - f.manenti3@campus.unimib.it
Matteo Gaverini - 808101 - m.gaverini1@campus.unimib.it

Luglio 2020

Sommario

Le recensioni degli utenti estratte da un sito di e-commerce possono essere utilizzate per svolgere diversi task di *Text Mining*, uno dei più comuni è la *Text Classification*. Esistono varie tipologie di classificazione testuali che, in ambito vendite online, possono essere impiegate per diversi fini. Per esempio applicando la *Sentiment Analysis*, si può decretare lo stato emozionale dell'autore della recensione oppure effettuando *Opinion Mining*, si può determinare se l'utente abbia espresso un giudizio positivo o negativo su un prodotto acquistato. Il progetto prevede inizialmente la classificazione di alcune recensioni Amazon in base alla loro categoria merceologica (Automotive, Baby, Pet Supplies etc.) definite nel catalogo americano. Una volta effettuata tale operazione, si effettua poi un'attività di *Text Clustering* su una specifica categoria per identificare o meno gruppi di recensioni che trattano prodotti con caratteristiche simili.

1 Introduzione

Amazon è un'azienda tecnologica americana con sede a Seattle il cui business principale è relativo all'e-commerce, ma negli ultimi tempi sta cercando sempre di più di aumentare la sua quota di mercato focalizzandosi in altri ambiti quali *cloud computing*, *digital streaming* e *artificial intelligence* [1]. Quest'ultimo settore è quello dove Amazon sta spendendo la maggior parte delle energie, prodotti come Alexa e servizi come AWS sono tutti esempi che dimostrano come l'azienda sfrutti sia tecniche di *NLP* che *Machine Learning* per raggiungere gli obiettivi preposti. Per quanto riguarda *NLP*, esistono diverse tecniche utilizzate da Amazon per aumentare i guadagni, una di queste è l'utilizzo di un *recommender system* che, in base alle preferenze e allo storico dei clienti, è in grado di consigliare prodotti interessanti per l'utente invogliandoli all'acquisto [2]. Oltre a questo l'azienda svolge anche altri task di *Text Mining* quali *Text Classification* e *Text Clustering*, ognuno per diverse finalità; la prima operazione può essere utilizzata per esempio per decretare in modo automatico dalle recensioni se la persona è soddisfatta o meno del prodotto acquistato, la seconda invece può essere effettuata per raggruppare i prodotti in base a delle caratteristiche comuni in modo tale che l'utente possa poi scegliere la sottocategoria più opportuna di un determinato item che sta cercando.

Il progetto si suddivide in due parti: nella prima parte si classificano le recensioni dei prodotti venduti su Amazon America in base alla loro categoria merceologica, nello specifico le label che possono essere associate per ogni review sono 'CDs and Vinyl', 'Grocery and Gourmet Food', 'Cell Phones and Accessories' e 'Sports and Outdoors'; la seconda parte invece, prevede un task di *Text Clustering* applicato solo sulle recensioni degli item appartenenti alla categoria 'CDs and Vinyl'. Questi due task sono stati svolti cercando un giusto trade-off tra efficacia ed efficienza delle soluzioni proposte, efficacia in termini di accuratezza dei risultati, efficienza rispetto alle risorse computazionali, questione particolarmente importante dato che, utilizzando le macchine virtuali messe a disposizione da Google con il servizio *Google Colab*, le risorse risultano essere limitate.

2 Dataset

Il dataset impiegato per svolgere i due task è stato creato *ad hoc* utilizzando le review di Amazon America del 2018. Per quanto riguarda le recensioni, si sono considerate solo quelle di prodotti che appartengono a 4 categorie definite nel catalogo americano che sono: 'CDs and Vinyl', 'Grocery and Gourmet Food', 'Cell Phones and Accessories' e 'Sports and Outdoors'. Tutte le recensioni sono salvate in file json separati per categoria. Ogni review è costituita da 12 feature così definite:

- *reviewerID*: ID utente che ha pubblicato la recensione;
- *asin*: ID prodotto;
- *reviewerName*: nome utente;
- *verified*: variabile booleana, indica se Amazon ha verificato o meno l'acquisto del prodotto;
- *reviewText*: testo della recensione;
- *overall*: valutazione 5 star del prodotto (da 1 a 5);
- *reviewTime*: data pubblicazione recensione (formato MM DD, YYYY);
- *summary*: sintesi recensione;
- *unixReviewTime*: data pubblicazione recensione (formato UNIX);
- *style*: dizionario che contiene descrizione prodotto (es. colore, formato, dimensione etc.);

- *vote*: numero di persone che hanno ritenuto utile la recensione;
- *image*: link ad eventuali immagini del prodotto pubblicate dall'utente.

2.1 Preprocessing generale

Una volta uniti i 4 file json corrispondenti a ciascuna delle categorie scelte, si è notato una disparità del numero di recensioni per item: alcuni prodotti avevano più di 5000 recensioni, altri invece ne avevano solo 1 (si veda Figura 1). Di conseguenza si è deciso di considerare un'unica recensione per prodotto, ovvero la più rappresentativa. Per far ciò si è ragionato in questo modo: dato che solitamente una recensione breve è poco informativa (es. 'This item works well'), si è deciso di considerare per ogni item solamente la più lunga, eliminando prima del conteggio eventuali URL che potessero influenzarne la lunghezza. In questo modo si rimuove da una parte il "rumore" provocato dalle review corte e quindi poco significative e dall'altra non si perde l'informatività in quanto, anche se tra n recensioni ne viene presa solo una, selezionando la più lunga probabilmente sarà quella che descriverà al meglio il prodotto corrispondente.

asin	
B00BUKL666	7387
B008QMX2SG	6228
B00D3M2QP4	6221
B0010O748Q	4163
B00R7PWK7W	3387
	...
B0013FBFFU	1
B0012IZ4JG	1
B0013LKXEI	1
B0012FY7TW	1
B000ZKAY8I	1

Figura 1: Distribuzione numero recensioni per item

Alla fine del preprocessing si ottengono 267906 recensioni univoche. Tra tutte queste però ne verranno prese solamente 41044 (10261 per categoria) poiché prendendone un numero maggiore, la sessione Colab sarebbe "crashata" nel momento in cui si effettuavano le operazioni di preprocessing sui testi delle recensioni. Selezionando solamente un sottoinsieme di dati, si ottiene così un

dataset bilanciato (*df_classification.csv*) rispetto alla variabile *target*, creata considerando l'appartenenza della categoria di ogni recensione.

Categoria Amazon	**Target**
CDs and Vinyl	CD
Grocery and Gourmet Food	Food
Cell Phones and Accessories	Cell
Sports and Outdoors	Sport

Tabella 1: Definizione variabile target

3 Text Classification

La prima attività svolta è la *Text Classification*, ovvero un procedimento supervisionato che assegna ad ogni documento testuale (nel contesto di riferimento sono recensioni) una label estratta da un insieme predefinito di classi [3]. Tale task trova spazio in numerosissimi ambiti applicativi (marketing, e-commerce etc.) e oggi, grazie alla possibilità di estrarre dati da qualsiasi sorgente, si rivela fondamentale in seguito ad un aumento sempre più crescente di dati testuali che si possono trattare in modo automatico [4].
Esistono principalmente due tipologie di classificazione: *binary* e *multi-class*. Il primo prevede che per ogni item venga assegnata una label binaria, il secondo invece che le label possano far parte di un insieme finito che contenga più di due classi. L'assegnamento in questo caso dipende dalla tipologia di classificazione *multi-class* adottata: se è una *single-label* o *ordinal*, ogni item appartiene ad una sola etichetta, se invece è una *multi-label*, un oggetto può appartenere a 0/1 o più classi.
Oltre a questo la *Text Classification* può avvenire, in generale, per *topic* o per *genere*; il primo indica che la classificazione degli item avviene rispetto all'entità tematica, il secondo invece rispetto al genere [4] (es. autore del documento).
Per quanto riguarda il progetto, si è realizzata una classificazione per *topic single-label multi-class*: *multi-class* perchè, come si è già detto nel Capitolo 1, le classi che possono essere associate sono 4 ('CDs and Vinyl', 'Grocery and Gourmet Food', 'Cell Phones and Accessories' e 'Sports and Outdoors'),

single-label perché ad ogni recensione viene assegnata una ed una sola etichetta e per *topic* perchè si classificano le recensioni rispetto all'entità tematica che risulta essere la categoria merceologica del prodotto recensito. Oltre a questo la classificazione realizzata può essere anche catalogata come *hard classification* in quanto, per ogni recensione, si effettua un assegnamento pieno ad una sola classe.
Per risolvere il problema di classificazione si è proceduto nel seguente modo: prima di tutto si sono addestrati 3 modelli di *Machine Learning* (*SVM*, *Random Forest* e *Neural Network*), dopodiché si sono valutati i risultati ottenuti grazie alla tecnica *cross-validation* infine si è individuato il modello migliore utilizzando come metrica di performance l'*accuracy* dato che il dataset è bilanciato e nessuna etichetta risulta essere maggiormente importante di un'altra.

3.1 Preprocessing testuale

Le principali operazioni di preprocessing effettuate sono:

- tokenization;
- normalization;
- stemming;
- stop-words removal.

3.1.1 Tokenization

La prima operazione risulta essere la *tokenization*, fase principale del *text preprocessing*. Tale task prevede di suddividere un testo in token significativi dove un token è definito come un'istanza di una sequenza di caratteri. Considerando il progetto, i token sono stati interpretati come singole parole e la suddivisione è avvenuta considerando le strutture e le forme della lingua inglese visto che le recensioni provengono da Amazon America. Oltre a questo nella divisione delle parole si è considerata anche la presenza di eventuali numeri e simboli gestendo le due situazioni in maniera differente. Per quanto riguarda i numeri non sono stati considerati come token, di conseguenza qualsiasi valore numerico, compreso un numero all'interno di una parola, veniva eliminato; i simboli invece, sono stati rimossi solo nei casi in cui non si trovavano tra due sequenze finite di n caratteri alfabetici, in tutti gli altri casi

venivano sostituiti con lo spazio in modo da separare i due termini che comparivano prima e dopo il simbolo. Questa scelta è motivata da due aspetti: il primo è la presenza in molte recensioni di parole separate da simboli come 'fit=good' oppure 'quality+price' di conseguenza, se si dovesse eliminare il simbolo e unire le due parole, si perderebbe l'informatività dei due termini singoli; il secondo motivo è legato alla *stop list* utilizzata per rimuovere le *stop word* che contiene le forme contratte (es. 'll', 'd', 're') perciò quando si incontra una contrazione (es. ' I'll ') è necessario separare i due termini affinché la rimozione avvenga in modo corretto.

3.1.2 Normalization

La seconda operazione effettuata è la *normalization*, task la cui finalità è quella di evitare che uno stesso concetto venga presentato più volte in forma diversa (es. 'Price', 'price', 'PRICE'). Per normalizzare i termini si sono svolte due operazioni: *case folding* e gestione slang. Il primo consiste nel convertire tutte le parole in minuscolo, il secondo invece, prevede di riconoscere eventuali slang presenti nelle recensioni e sostituirli con il termine completo (si veda Tabella 2 per degli esempi).

Slang	**Termine completo**
yall	you all
cause	because
asap	as soon as possible
gonna	going to

Tabella 2: Esempi slang riconosciuti

3.1.3 Stemming

La terza operazione è lo *stemming*, task molto importante quando si effettua il *text preprocessing* poichè permette la riduzione delle parole che verranno poi utilizzate per rappresentare il corpus. Questa operazione prevede di eliminare le parti conclusive di una parola in modo da ottenere una radice comune (es. 'organization' diventa 'organ'). Per effettuare lo *stemming* si è utilizzato, tra tutti gli algoritmi disponibili, lo *Snowball Stemmer* [5] (implementato con la

libreria Python *nltk*) in quanto, secondo la letteratura, risulta essere uno dei migliori.

3.1.4 Stop-words removal

La quarta e ultima operazione svolta è la rimozione delle *stop word*, ovvero eliminare tutte quelle parole non utili per comprendere il contenuto di un testo. I termini rimossi sono stati estratti da una *stop list* (ottenuta dalla libreria Python *nltk*) che contiene 179 parole inglesi. Oltre a questo, nella *stop list* sono stati inseriti anche altri termini poco discriminanti nel contesto di riferimento poichè ricorrono con una frequenza molto alta nelle recensioni (si veda Tabella 3 per degli esempi di parole eliminate).

Termini
product, item, price, amazon, prime, buy etc.

Tabella 3: Stop word relative al dominio di riferimento

3.2 Text representation

Una volta effettuato il preprocessing e creato il vocabolario che contiene 114367 parole, si è proceduto ad individuare la rappresentazione formale migliore per le recensioni. Esistono diverse *text representation* che si possono utilizzare, le più comuni sono: *term-document matrix* e *Word Embedding*. La prima prevede la creazione di una matrice in cui ogni riga rappresenta un documento e ogni colonna una parola contenuta nel dizionario a cui è associato un peso. Le principali funzioni di *weighting* che si possono applicare sono così definite:

- *Binary*: 1 se la parola è contenuta nel documento, 0 altrimenti;
- *Raw Frequency*: frequenza parola nel documento;
- *TF-IDF*: combinazione metriche *term frequency* (TF) e *inverse document frequency* (IDF).

La seconda *text representation* invece, prevede la conversione di vettori sparsi che rappresentano una parola o un documento in vettori densi di numeri reali

[6]. Matematicamente si tratta di modelli che proiettano uno spazio a molte dimensioni per parola in uno spazio vettoriale a meno dimensioni. Esistono due tipologie principali di *Word embedding*: *count-based models* (effettuano una *dimensionality reduction* sulla *term-context matrix*) e *predictive models* (sfruttano reti neurali).
Per quanto riguarda il progetto, si è utilizzata come *text representation* sia una *term-document matrix* con funzione di *weighting TF-IDF* che una tecnica di *Word embedding* di tipo *predictive* che è *Doc2vec*.

3.2.1 TF-IDF matrix

TF-IDF matrix è una rappresentazione testuale che consiste in una matrice M *term-document* dove ogni riga rappresenta un documento, ogni colonna una parola e l'elemento m_{ij} indica il peso TF-IDF associato alla parola j nel documento i. La funzione di peso TF-IDF, rappresentata formalmente dall'equazione 1, è la combinazione di due metriche: *term frequency* (TF) e *inverse document frequency* (IDF). La *term frequency* calcola la frequenza del termine t in ogni documento d, la *inverse document frequency* invece è una misura inversa dell'informatività del termine t, nello specifico è il logaritmo del rapporto tra il totale di documenti nel corpus e il numero di documenti in cui compare il termine t.

$$TF - IDF_{t,d} = \frac{tf_{t,d}}{\max_{t_i \in d} tf_{t_i,d}} \cdot \log\left(\frac{N}{df_t}\right) \tag{1}$$

La TF-IDF matrix è stata implementata grazie alla libreria Python *sklearn* che permette di impostare diversi parametri, per la *Text classification* sono così definiti:

- max_df = 0.8
- min_df = 10
- ngram_range = (1,1)

I primi due parametri definiscono delle threshold che aiutano a diminuire la dimensionalità del vocabolario (si passa da 114367 a 22525 parole) e quindi a ridurre la matrice ottenuta, il terzo invece definisce la tipologia di n-grammi utilizzati che in questo caso risultano essere solo unigrammi.
Nel progetto è stata utilizzata la TF-IDF perchè, rispetto alle altre funzioni

di *weighting*, decreta l'importanza di un termine considerando contemporaneamente due aspetti che sono *corpus-wise* (considera quanti documenti contengono termine t rispetto al corpus) e *document-wise* (considera frequenza termine t in un doc) e quindi, grazie a questo, si può valutare in maniera efficace l'importanza di una parola all'interno di un corpus. Analizzando la TF-IDF matrix ottenuta, emerge un problema che solitamente compare quando si utilizza tale rappresentazione, ovvero la sparsità della matrice. A tal proposito, si è deciso di utilizzare come alternativa un'altra tecnica di *text representation*, ovvero il già citato *Word embedding* ed in particolare il *Doc2vec*.

3.2.2 Doc2vec

Doc2vec è una tecnica di *Word embedding* in cui si prevede che ogni documento venga rappresentato da un vettore denso di numeri reali di lunghezza fissa, indipendentemente dal numero di parole contenute in esso.
Come si può intuire dal nome Doc2vec è "un'evoluzione" del Word2vec, per poter comprendere il modello utilizzato nel progetto è necessario quindi spiegare il funzionamento del Word2vec. In questa tecnica per ogni parola contenuta nel corpus, in modo univoco, viene costruito un vettore in modo da rappresentare ogni termine come un punto nello spazio multidimensionale creato.

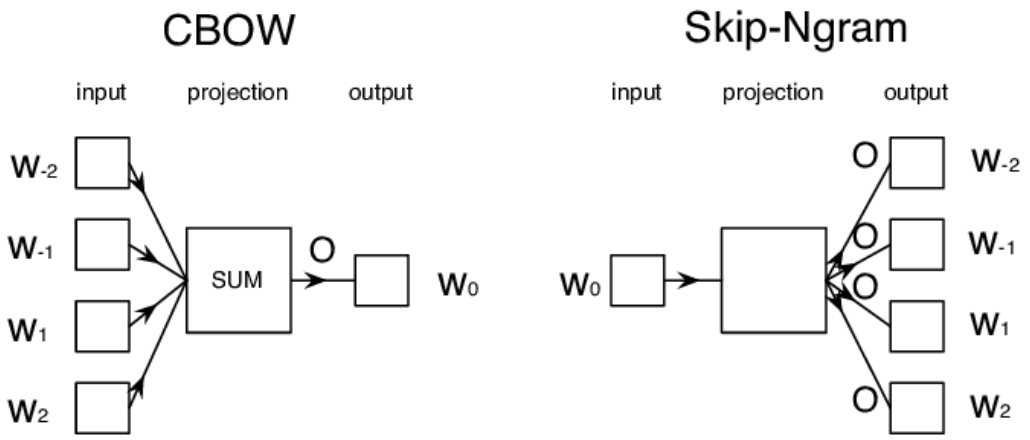


Figura 2: Architetture Word2vec

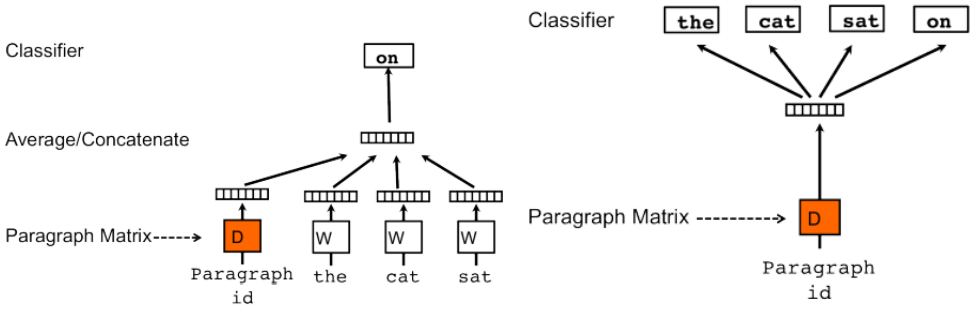
Esistono due tipologie di architetture che Word2vec può utilizzare per raggiungere il suo obiettivo:

- *Skip-grams*
- *Continuos bag of words* (CBOW)

La prima predice le *context word* in base alla *target word*, la seconda invece svolge il lavoro opposto. Nel Doc2vec si utilizzano le stesse due architetture del Word2vec, ma siccome lo scopo è rappresentare un documento e non più una singola parola, si aggiunge un altro vettore di feature relativo al documento stesso (nella Figura 3 è chiamato *Paragraph Matrix*) che viene addestrato insieme ai vettori delle parole. Queste due architetture sono:

- *Distributed memory*: derivante da CBOW
- *Distributed bag of words*: derivante da Skip-Grams

La prima (Figura 3a) funziona come CBOW ma con l'aggiunta di un vettore univoco per il documento. La seconda architettura (Figura 3b) invece, non salva i vettori delle parole e quindi, utilizzando meno memoria, risulta essere più veloce durante il training del modello.



(a) Distributed Memory (b) Distributed Bag of Words

Figura 3: Architetture Doc2vec

Per quanto riguarda il progetto, si è implementato un Doc2vec con architettura *Distributed bag of words* definendo una lunghezza fissa dei vettori pari a 300. Si è scelta questa architettura e non l'altra perchè, come si è detto precedentemente, utilizza meno memoria e quindi si evita il "crash" della sessione Colab.

3.3 Modelli

Una volta individuate le due rappresentazioni testuali, si sono addestrati 3 modelli di *Machine Learning* che sono: *SVM*, *Random Forest* e *Neural Network*.

3.3.1 SVM

SVM è un algoritmo di *Machine Learning* supervisionato che rappresenta i record del dataset in uno spazio n-dimensionale [7]. Trovare la soluzione vuol dire individuare gli iperpiani che separano al meglio gli elementi che fanno parte di classi diverse. Il classificatore è stato implementato tramite la libreria Python *sklearn* e gli iperparametri principali settati sono il kernel di tipo *linear* e il regolarizzatore C impostato a 1.

3.3.2 Random Forest

Random Forest è un algoritmo di apprendimento basato su un insieme di *decision tree*. Si tratta, dunque, di un modello *ensemble*. Ciò significa che vengono combinati più modelli dove ciascun classificatore da un voto rispetto all'assegnamento di una label per una istanza e, in base alla modalità di *voting* definita, viene decretata l'etichetta finale. Nel caso del Random Forest la modalità di *voting* definita è *Max Voting*, ovvero la classe con il maggior numero di voti (assegnamenti) sarà quella predetta dall'algoritmo per l'istanza che si sta considerando [8]. Anche in questo caso è stata utilizzata la libreria *sklearn* per l'implementazione del modello il cui iperparametro principale, ovvero il numero di *decision tree* è stato impostato a 10.

3.3.3 Neural Network

Neural Network (NN) è un modello composto da neuroni artificiali, interconnessi tra loro, i quali elaborano i segnali ricevuti e trasmettono il risultato ai nodi successivi [9]. Per l'implementazione di questo modello è stato utilizzato il framework Python *keras* e sono stati effettuati numerosi test empirici per trovare la combinazione migliore degli iperparametri principali. Nello specifico, la rete implementata è composta solamente da un layer di input e uno di output; il primo strato è costituito da 64 neuroni con funzione di attivazione *relu*, il secondo invece da 4 neuroni con funzione di attivazione *softmax* dato che l'obiettivo è ottenere una distribuzione di probabilità sull'appartenenza

delle classi per ogni istanza. Come funzione di loss invece, trattandosi di una classificazione multiclasse, è stata utilizzata la *sparse categorical crossentropy* e l'ottimizzatore scelto risulta essere *adam*.
Per la fase di train del modello è stato utilizzato un *batch size* pari a 64, un *learning rate* settato a 0.001 e un numero massimo di epoche impostato a 10. Per evitare eventuali problemi di overfitting è stato inoltre utilizzato sia il *Dropout* (valore *dropout rate* pari a 0.3) che l'*Early stopping* impostando come numero massimo di epoche entro la quale la *validation loss* non peggiori un valore pari a 2.

3.4 Risultati

Una volta implementati e addestrati i modelli, si è proceduto a valutare le performance di ciascuno attraverso la tecnica *10-fold cross validation* utilizzando come metrica di performance l'*accuracy*. Tale metrica è così definita formalmente:

$$a = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

dove TP rappresenta i veri positivi, TN i veri negativi, FP i falsi positivi infine FN i falsi negativi.
Di seguito vengono riportate due tabelle, una relativa ai risultati ottenuti utilizzando come rappresentazione testuale *TF-IDF matrix* (si veda Tabella 4), l'altra invece utilizzando *Doc2Vec* (si veda Tabella 5).

Classificatore	**Tempo (s)**	**Cross Validation (Accuracy)**	**Intervallo di confidenza (Accuracy)**
SVM	18	98.0%	[97.9, 98.1]
Random Forest	301	94.7%	[94.5, 94.9]
Neural Network	315	97.5%	[97.3, 97.7]

Tabella 4: Risultati TF-IDF matrix

Classificatore	Tempo (s)	Cross Validation (Accuracy)	Intervallo di confidenza (Accuracy)
SVM	433	97.7%	[97.6, 97.8]
Random Forest	225	97.2%	[97.1, 97.3]
Neural Network	144	98.0%	[97.8, 98.1]

Tabella 5: Risultati Doc2Vec

Osservando le due tabelle, emergono diversi aspetti interessanti. Il primo è che non esistono differenze significative tra le performance ottenute con *TD-IF matrix* e con *Doc2Vec*; tutti i classificatori meno Random Forest che con la prima rappresentazione raggiunge *accuracy* 94%, ottengono performance elevate, i valori di accuratezza sono tra 97% e 98%. Entrando nel dettaglio, si osserva che il classificatore ottimo per la prima *text representation* risulta essere SVM mentre per la seconda rappresentazione è la Neural Network ed entrambi, oltre a raggiungere lo stesso valore *accuracy* 98%, risultano essere anche i più efficienti considerando le rispettive rappresentazioni. Esaminando invece globalmente l'efficienza dei modelli, si osserva che SVM risulta essere quello che richiede meno tempo per completare il processo di *cross validation*. Per questo motivo il classificatore migliore, sia considerando l'efficacia che l'efficienza, risulta essere SVM.
Dopo aver ottenuto i risultati precedentemente mostrati, si è cercato di apprendere in che modo i modelli effettuassero determinate classificazioni. Per far ciò si è utilizzata *Explainable AI* (XAI), ovvero un insieme di tecniche che permettono di comprendere e presentare una visione più chiara del funzionamento dei modelli di apprendimento automatico considerati meno "trasparenti" (*black box*). Le tecniche di XAI sono state applicate per osservare il comportamento della sola rete neurale implementata e non degli altri due classificatori perché tra tutti la NN risulta essere il modello *black box* per eccellenza. Per usufruire di queste tecniche è stata utilizzata la libreria Python *lime* che consente di svolgere una *Local Interpretation* [10], cioè osservare per ogni recensione le feature più significative in base alle quali il modello etichetta quel record con una certa label.
Di seguito viene riportata un'immagine di esempio per una review appartenente alla classe 'Cell' che è stata correttamente classificata; in questo caso si vede che le parole più significative che aiutano la rete neurale a classificare correttamente la recensione siano 'case', 'phone' e 'protect'.

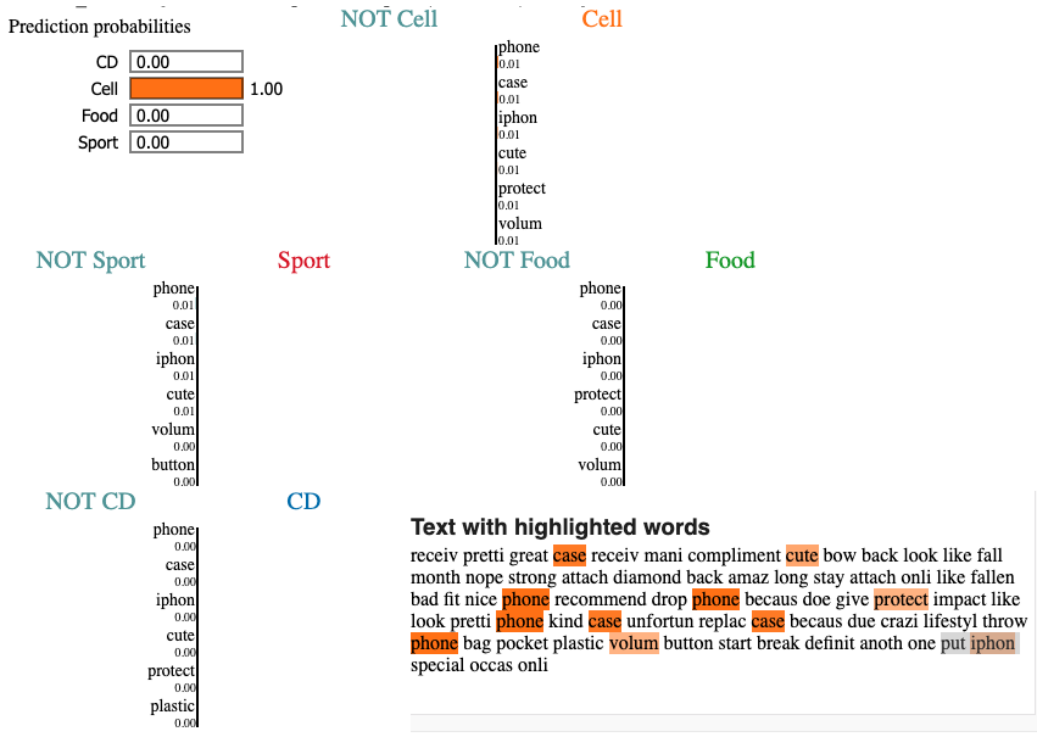


Figura 4: Esempio XAI classe Cell

Nell'Appendice è riportato un caso di previsione corretto per ogni altra classe.

4 Text Clustering

La seconda attività svolta è la *Text Clustering* che, rispetto alla *Text Classification*, si rivela essere un procedimento non supervisionato in quanto si estraggono dei pattern nascosti, utili per creare i cluster, dai "dati grezzi" senza conoscere le label a priori che decretano la corrispondenza reale istanza-gruppo. Questo task, come si intuisce dal nome, prevede appunto di raggruppare documenti simili in gruppi o cluster basandosi su una *proximity score* che può essere una misura di similarità o una funzione di distanza [11]. L'attività di *Text Clustering* trova applicazione in numerosissimi contesti: dal marketing fino all'e-commerce, in particolare quest'ultimo campo risulta essere quello da cui si possono trarre maggiori vantaggi. Per esempio con la

Text Clustering si potrebbero individuare sottoclassi di item che appartengono ad una stessa categoria in modo tale che l'utente possa poi scegliere il prodotto specifico in base a quella specializzazione individuata; così facendo, il cliente, soddisfatto dell'efficacia del sito, continuerà ad acquistare su quella piattaforma.
Esistono principalmente due tipologie di clustering che si possono realizzare: *flat* e *hierarchical*. Il primo consiste nella creazione di cluster "piatti", cioè gruppi statici dove ad ogni item viene assegnato un solo gruppo, il secondo invece, prevede la creazione di cluster organizzati in modo gerarchico secondo una struttura chiamata dendrogramma (albero) che, in base al livello di specificità richiesto, decreta il numero totale di cluster ottenuti e i componenti di ciascuno.
Per quanto riguarda il progetto, il task di *Text Clustering* è stato effettuato solamente sulle recensioni relative alla categoria 'CD'; il motivo di tale scelta è che si era interessati a verificare l'ipotesi che i cluster venissero raggruppati secondo uno specifico criterio di raggruppamento quale il genere musicale del CD o vinile recensito (es. pop, rock). Per svolgere tale task si è utilizzato un solo algoritmo di *Text Clustering* che risulta essere *K-Means*; il motivo di tale scelta è che provando altri algoritmi come *DBSCAN* e *Ward*, la sessione Colab "crashava" quindi, al posto che ridurre il dataset, si è preferito usare un algoritmo semplice ed efficiente come *K-Means* che potesse restituire una soluzione di clustering completa rispetto ad una parziale.
Una volta individuati i cluster dopo aver decretato il numero ottimo k, mediante *elbow rule*, si è proceduto a valutare la soluzione di clustering ottenuta sia da un punto di vista quantitativo che qualitativo per decretare se i gruppi rilevati rappresentassero o meno un possibile scenario reale.

4.1 Preprocessing testuale

Per quanto riguarda il preprocessing, alcune operazioni effettuate sono identiche a quelle per il task di *Text Classification* come la *normalization*, altre invece come la *lemmatization* sono state realizzate esclusivamente per questo task perché contribuivano ad un miglioramento della soluzione di clustering ottenuta. Nello specifico le operazioni effettuate sono:

- riconoscimento di entità (NER)
- tokenization
- normalization

- lemmatization
- stop-words removal

N.B. La *normalization* non verrà spiegata perchè prevede le stesse operazioni effettuate per il task di *Text Classification*.

4.1.1 NER

La prima operazione effettuata è il riconoscimento di entità, in particolare di persone/gruppi musicali (*PERSON*) e titoli di canzoni/album discografici (*WORK_OF_ART*). Questo passaggio risulta essere molto importante in quanto può contribuire ad una miglior definizione dei cluster che raggruppano le recensioni in base ad un determinato criterio come ad esempio il genere musicale. Basti pensare al caso in cui due recensioni parlano di due CD di due artisti diversi con un nome comune (ad esempio Michael Jackson e Michael Bublé). Non eseguendo la NER, quindi conseguentemente non sistemando la *tokenization*, aumenterebbe la similarità tra le due recensioni a causa del token ('Michael'), nonostante quest'ultimo si riferisca a due artisti differenti. Per effettuare la NER è stato utilizzato *SpaCy*, un framework *NLP* per Python che contiene un modello NER *Rule-based* allenato su OntoNotes. Una volta individuate le entità di tipo *PERSON* e *WORK_OF_ART*, si è deciso di filtrare i risultati ottenuti dato che il modello NER restituiva molti falsi positivi (sequenze di parole individuate che nella realtà non appartengono ad alcuna entità); a tal proposito si è optato quindi di considerare valide solamente le entità costituite da una sequenza massima di 3 parole per limitare il rumore ottenuto.

4.1.2 Tokenization

La seconda operazione è la *tokenization* che, a differenza di quella effettuata per la *Text Classification*, prevede questa volta una suddivisione del testo considerando congiuntamente eventuali entità individuate nel task precedente. Questo vuol dire che se in una recensione si dovesse trovare la parola 'Michael Jackson' correttamente riconosciuta come entità di tipo *PERSON*, la *tokenization* non separerà la sequenza in due token singoli ma si considererà un unico token.

4.1.3 Lemmatization

La terza operazione effettuata, non contando la *normalization*, è la *lemmatization*, tecnica utile per ridurre la dimensionalità del vocabolario in quanto riporta le forme flesse e varianti di una parola alla sua forma base (es. 'am', 'are', 'is' → 'be'). Si è preferito utilizzare questa tecnica perché in questo modo non si riscontrano problemi di ambiguità che si potrebbero avere usando lo *stemming* in seguito ad un eventuale *overstemming* ottenuto nei testi (per esempio 'organization', 'organ', 'organic' la radice comune per tutte è 'organ'). Per il progetto è stato utilizzato *WordNet lemmatizer*, ovvero un lemmatizzatore contenuto nella libreria Python *nltk* che utilizza *WordNet* [12] come risorsa di riferimento. Inoltre, per migliorare i risultati, è stato utilizzato congiuntamente anche un *POS tagger*, ossia un tool che etichetta ogni parola all'interno di un testo secondo la sua categoria grammaticale di appartenenza. Anche in questo caso il *POS tagger* è stato implementato grazie alla libreria *nltk*.

4.1.4 Stop-words removal

Come ultima operazione di preprocessing si è proceduto a rimuovere le *stop word* con la differenza che, rispetto a quello effettuato per la *Text Classification*, la *stop list* usata contiene oltre alle parole rimosse per il primo task, anche termini relativi al contesto musicale (si veda Tabella 6 per degli esempi). Si è effettuata questa operazione perché, come si è detto nel Capitolo 4, le recensioni utilizzate per il task di *Text Clustering* sono solo quelle relative alla categoria 'CD' e quindi si trovano molte parole che occorrono parecchie volte solo perché sono delle *stop word* rispetto al dominio di riferimento.

Termini
cd, music, song, album, track, listen etc.

Tabella 6: Stop word relative al dominio musicale

4.2 Text representation

Una volta completate tutte le operazioni di preprocessing, si è proceduto ad individuare la rappresentazione testuale più adeguata per il task corrispon-

dente considerando anche le performance ottenute nel primo task. Dopo aver osservato i risultati ottenuti nel Capitolo 3.4, si è deciso di utilizzare una *TF-IDF matrix* perché a parità di prestazioni risulta essere più veloce di un approccio *Word Embedding*. Questa rappresentazione però è diversa da quella ottenuta per la *Text Classification* in quanto i token non sono più solo unigrammi ma anche bigrammi e trigrammi in seguito alla NER effettuata in fase di preprocessing. I parametri per realizzare la *TF-IDF matrix* sono così definiti:

- max_df = 15000
- min_df = 10
- max_features = 4096

4.3 K-Means clustering

K-Means è un algoritmo di apprendimento non supervisionato che trova un numero fisso di cluster in un insieme di dati [13]. Tale algoritmo, basandosi sul concetto di centroide (elemento rappresentante di un determinato cluster), assegna in modo esclusivo le osservazioni ad un cluster. L'idea alla base è quella di definire inizialmente a priori k centroidi, uno per ogni cluster, dopodiché si procede ad assegnare ogni item al centroide più vicino (decretato con una misura di similarità o distanza), infine dopo aver assegnato ogni osservazione ad un gruppo e successivamente ricalcolato i centroidi, si reitera il processo descritto fino a quando non esistono più cambiamenti riguardo gli assegnamenti dei cluster [14].
Per quanto riguarda il progetto, l'algoritmo è stato implementato con la libreria Python *kmeanstf* e la funzione utilizzata per calcolare la distanza tra le osservazioni e i centroidi risulta essere *euclidean distance*. La scelta del numero dei k centroidi, naturalmente, è fondamentale durante l'implementazione dell'algoritmo K-Means. Per questo motivo, prima di generare la soluzione di clustering definitiva, si è individuato il numero ottimo di cluster utilizzando il metodo *elbow*. Tale tecnica prevede di eseguire iterativamente il K-Means per un range di valori k e, per ciascuna iterazione, si calcola la *distortion* definita come la somma delle distanze al quadrato tra ogni istanza e il centroide del cluster a cui appartiene [15]; questi valori poi vengono tutti rappresentati graficamente da un linechart e il punto in cui in cui emerge la massima flessione della curva (a formare un "braccio"), identifica il numero di cluster ideale. Il range di valori definito per eseguire iterativamente il

K-Means è $[3, 50]$ e, analizzando graficamente i valori di *distortion* riportati dal metodo (si veda Figura 5), si osserva che il numero ottimo k di cluster risulta essere pari a 20.

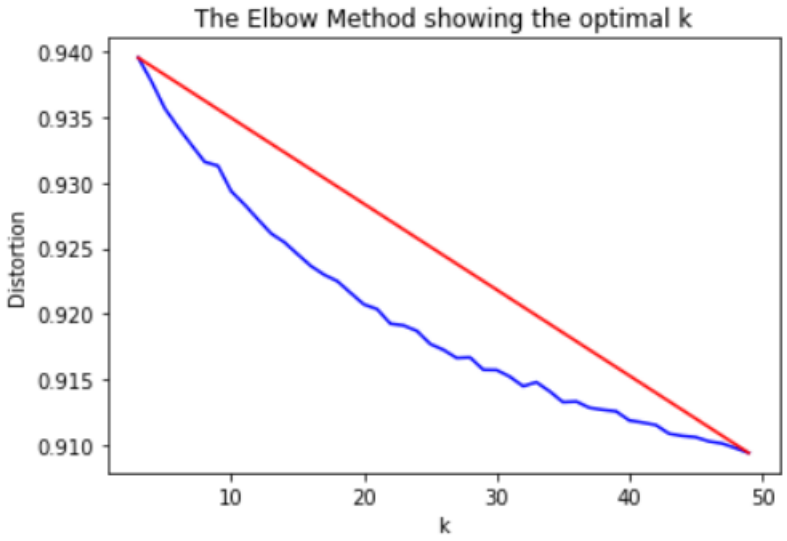


Figura 5: Risultato metodo elbow

4.4 Risultati

Una volta trovato il valore ottimo k di cluster pari a 20, si è proceduto a valutare i gruppi individuati. La prima valutazione che si è effettuata è di tipo quantitativo e prevede di utilizzare il coefficiente di *silhouette*; si è considerata una misura interna e non una esterna perchè nello scenario in questione non si ha una *ground truth* di conseguenza non si conosce il cluster reale a cui appartiene ogni recensione. Formalmente il coefficiente di *silhouette* è definito come:

$$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{3}$$

dove i indica l'istanza, $b(i)$ la distanza media inter-cluster e $a(i)$ la distanza media intra-cluster.
La seconda valutazione invece, è stata effettuata da un punto di vista qualitativo realizzando delle wordcloud per ogni cluster in modo da individuare

eventuali entità tematiche (es. genere musicale) contenute in ciascun gruppo, utilizzate per raggruppare le recensioni.

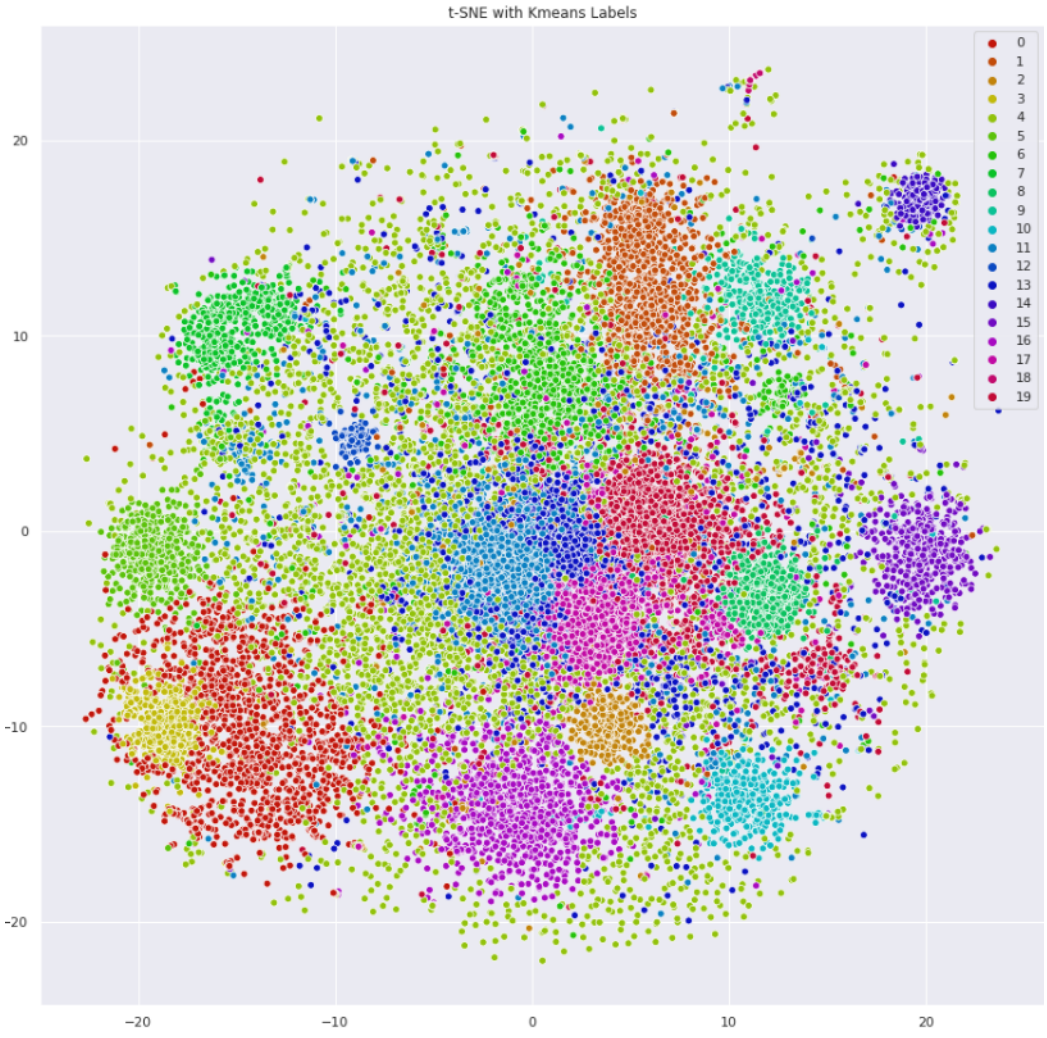


Figura 6: Risultato grafico K-Means

```
0    -0.004469
1     0.011141
2     0.021791
3     0.053661
4    -0.018193
5     0.040789
6     0.003244
7     0.039772
8     0.045528
9     0.041767
10    0.040905
11   -0.000724
12    0.074798
13   -0.007178
14    0.132936
15    0.021715
16    0.009342
17    0.013417
18    0.368565
19    0.003549
dtype: float64
```

Figura 7: Coefficienti silhouette per ogni cluster

Osservando i coefficienti di *silhouette* media per ogni cluster, emergono diversi aspetti interessanti. La prima cosa che si evidenzia è che tutti i cluster, meno 14 e 18, assumono valori prossimi a 0. Questo implica che molti gruppi si sovrappongono (*overlapping cluster*) sicché due o più cluster, probabilmente, possono essere rappresentati secondo un unico raggruppamento. La presenza di *overlapping cluster* comunque era già evidente osservando il grafico che mostra la soluzione di clustering ottenuta (si veda Figura 6): come si può vedere i gruppi non sono ben definiti, distinti tra di loro di conseguenza è normale trovare un'istanza che risulta essere più vicina ad un'altra anche se non fa parte del cluster a cui appartiene.
Inoltre, un altro aspetto che emerge osservando i coefficienti di *silhouette* è che i due cluster 14 e 18, assumono valori completamente differenti rispetto alla media, il primo ha un valore di *silhouette* pari a 0.13, l'altro invece 0.36. Ciò implica una buona coesione all'interno di ciascun gruppo, maggiore nel secondo rispetto al primo.

Figura 8: Wordcloud più interessanti

Analizzando i wordcloud più interessanti ottenuti dai vari cluster, si possono trarre diverse considerazioni. La prima, sorprendente, è che nel cluster 18 si visualizzano solo parole spagnole; ciò implica che all'interno del dataset, non sono presenti solo recensioni inglesi, situazione che ci si aspetta visto che i dati provengono da Amazon America, ma si trovano anche delle recensioni scritte in spagnolo. Per questo motivo tale cluster, come si è visto precedentemente, assume il valore più alto di *silhouette* proprio perché la similarità tra gli item è molto alta visto che tutti i testi sono in spagnolo, a differenza delle altre recensioni che sono scritte in inglese.
Un'altra considerazione che si può fare osservando i wordcloud è che alcuni cluster raggruppano le recensioni rispetto al genere musicale del CD o vinile recensito. Per esempio osservando il wordcloud del cluster 8, si notano molte parole riconducibili al genere musicale *metal* come 'heavy', 'riff', 'guitar' e 'metal' mentre considerando il wordcloud del cluster 15, si evidenziano molti termini riconducibili questa volta al genere *rap* come 'beat', 'hip hop', 'rapper' e 'flow'. Oltre a questo un altro aspetto interessante che traspare è che nel cluster 14 il raggruppamento avviene rispetto alla tipologia di canzoni natalizie presenti nel CD o vinile recensito infatti, osservando il wordcloud, si notano molte parole riconducibili al Natale come 'christmas', 'holiday' e

'carol'. Dall'analisi dei grafici, si può dire quindi che la soluzione di clustering ottenuta è avvenuta considerando diversi criteri di raggruppamento.

5 Conclusioni

Il progetto si è incentrato nella prima parte alla risoluzione di un problema di *Text Classification*, nella seconda invece ad un task di *Text Clustering*. Per quanto riguarda il primo problema, i risultati ottenuti sono molto buoni, tutti i modelli implementati, ad eccezione del *Random Forest* che risulta essere il classificatore peggiore, raggiungono valori di *accuracy* superiori al 97%. Considerando la *Text Clustering* invece, dal punto di vista quantitativo i risultati ottenuti non sono eccellenti come nel precedente task: si può osservare infatti che l'indice di *silhouette* media nella maggior parte dei cluster si attesta intorno a 0. Tuttavia, da un punto di vista qualitativo, la situazione migliora parzialmente: alcuni cluster sono generati rispetto al criterio di raggruppamento ipotizzato, altri invece secondo criteri non ben identificabili; inoltre si è individuato un cluster che ha raggruppato gli item rispetto alla lingua del testo delle recensioni. Per quanto riguarda i possibili sviluppi futuri, se ne registra solamente uno per la *Text Clustering*, ovvero testare altri algoritmi che adottano una strategia diversa rispetto al *K-Means* e verificare se le soluzioni ottenute risultino essere migliori rispetto a quelle presentate nel progetto. Considerando il task di *Text Classification* invece, non si registra alcuna operazione di *refine* in quanto si ritiene che le performance ottenute siano già eccellenti.

Riferimenti bibliografici

[1] Wikipedia, "Amazon (company)." [Online]. Available: https://en.wikipedia.org/wiki/Amazon_(company)

[2] B. Morgan, "How amazon has reorganized around artificial intelligence and machine learning." [Online]. Available: https://www.forbes.com/sites/blakemorgan/2018/07/16/how-amazon-has-re-organized-around-artificial-intelligence-and-machine-learning/#3a7427d37361

[3] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39.

[4] M. Ikonomakis, S. Kotsiantis, and V. Tampakas, "Text classification using machine learning techniques." *WSEAS transactions on computers*, vol. 4, no. 8, pp. 966–974, 2005.

[5] M. Porter, "Snowball stemmer," 2001. [Online]. Available: https://snowballstem.org/

[6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[7] S. Ray, "Understanding support vector machine(svm) algorithm from examples (along with code)." [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[8] V. F. Rodriguez-Galiano, B. Ghimire, J. Rogan, M. Chica-Olmo, and J. P. Rigol-Sanchez, "An assessment of the effectiveness of a random forest classifier for land-cover classification," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 67, pp. 93–104, 2012.

[9] M. H. Hassoun *et al.*, *Fundamentals of artificial neural networks*. MIT press, 1995.

[10] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 1135–1144.

[11] C. C. Aggarwal and C. Zhai, "A survey of text clustering algorithms," in *Mining text data*. Springer, 2012, pp. 77–128.

[12] C. Fellbaum, *WordNet*. American Cancer Society, 2012. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781405198431.wbeal1285

[13] L. Govoni, "Algoritmo k-means: cos'è e come funziona?" [Online]. Available: https://lorenzogovoni.com/algoritmo-k-means-cose-e-come-funziona/

[14] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl *et al.*, "Constrained k-means clustering with background knowledge," in *Icml*, vol. 1, 2001, pp. 577–584.

[15] "Elbow method." [Online]. Available: https://www.scikit-yb.org/en/latest/api/cluster/elbow.html#module-yellowbrick.cluster.elbow

Appendice

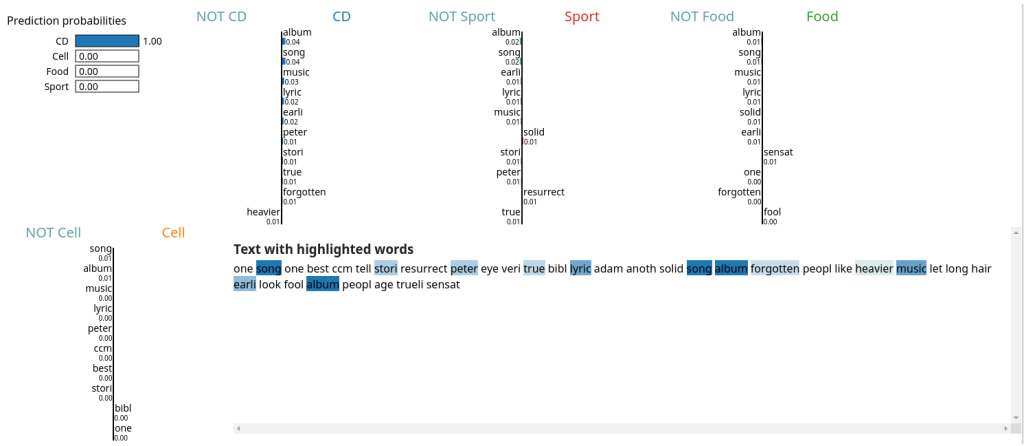


Figura 9: Esempio XAI classe CD

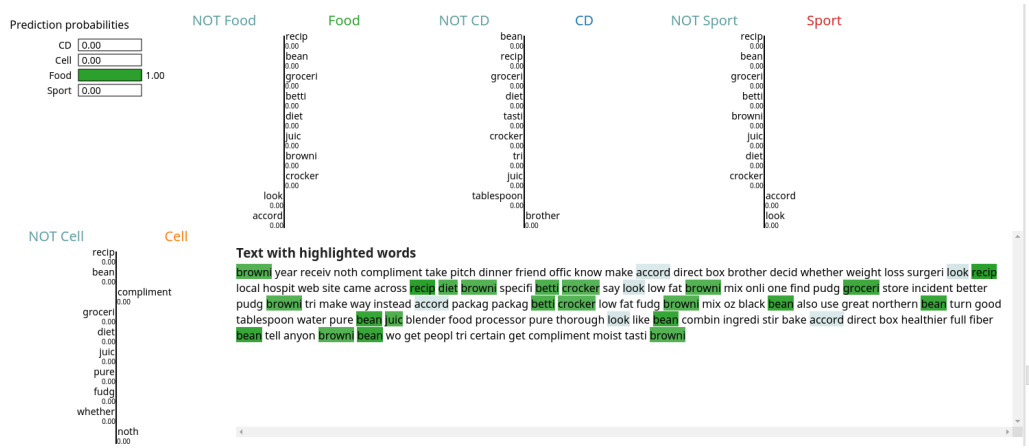


Figura 10: Esempio XAI classe Food

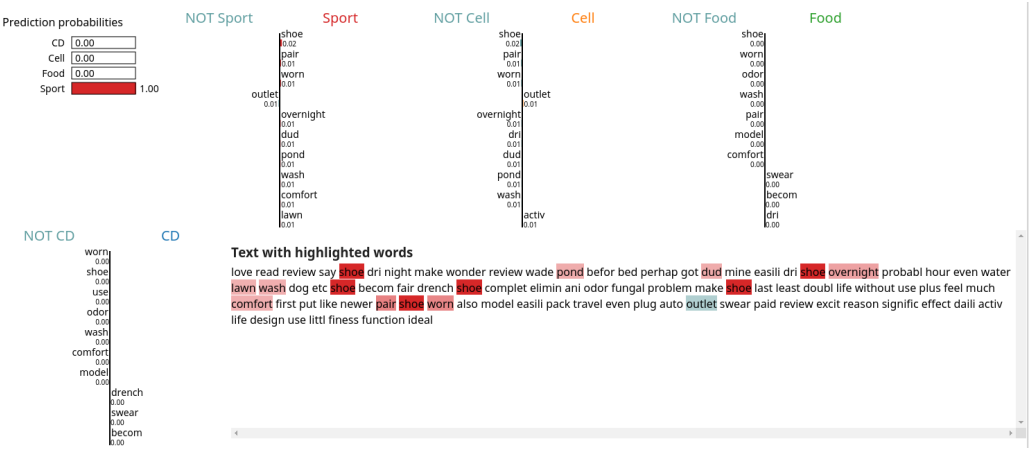


Figura 11: Esempio XAI classe Sport