UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Università degli Studi di Napoli Federico II

Corso di Laurea Magistrale in Ingegneria Informatica

Elaborato in

BIG DATA ANALYTICS AND BUSINESS INTELIGENCE

e

BIG DATA ENGINEERING

Progetto finale

Candidati

Mattia Brescia M63001103

Luigi Garofalo M63001182

Prof: Giancarlo Sperlì

Anno accademico 2021/2022

Indice

Capitolo 1: Challenge description

1.1 Natural Language Processing & Sentiment Analysis

"The aim is to define algorithm for evaluating user's reviews (positive or negative) or for analyzing user's sentiment about a given business object. The objective is to use the Spark NLP library (JohnSnow library) in conjunction with MlLib to train a multi-class sentiment classifier. The proposed approach will be evaluated comparing the obtained result with review stars."

Conosciuta anche come opinion mining, la sentiment analysis è una tecnica che permette di estrapolare le opinioni degli utenti da un testo. Si utilizza in moltissimi settori, primariamente del marketing, ma la sentiment analysis è applicata di frequente anche nei mercati finanziari, nelle scienze mediche, nello sport.

Cosa si analizza? Prevalentemente, si prendono in considerazioni i post pubblicati dagli utenti sui diversi social media. È questo, infatti, il luogo virtuale in cui gli utenti danno ampio spazio a commenti e giudizi, nonché alla creazione autonoma di contenuti. Un brand può così capire cosa pensano i suoi clienti attuali e potenziali, quale percezione hanno della sua attività e dei suoi prodotti. Così come di quelli della concorrenza.

Uno degli aspetti più complessi riguarda la sfera dell'emotività. Il tipo di aggettivi e di avverbi che vengono scelti, l'utilizzo della punteggiatura e delle emoticon possono aiutare nella comprensione dello stato d'animo. Inoltre, l'analisi deve valutare la rilevanza del commento in questione. Non tutti i post, infatti, si equivalgono o hanno lo stesso valore. Alcuni sono fuori luogo, altri sono stati scritti solo per creare confusione o farsi pubblicità. Bisogna concentrarsi su quelli validi e significativi.

In questo tipo di analisi entra in gioco la **semantica**. Ovvero, i significati molteplici che lo stesso termine può assumere a seconda del contesto in cui è collocato. Ad esempio, affermare che l'appuntamento è verso le quattro, quella piccola parola, "verso", potrebbe essere interpretata dal software in un modo sbagliato. Essa può indicare infatti anche un a direzione o l'estratto di una frase. Così come la parola "tempo", che può indicare lo scorre re delle ore così come una variazione metereologica. Oggi, però, grazie agli sviluppi dell'intelligenza artificiale, i software di analisi del linguaggio naturale sono sempre più attenti ad individuare queste sottigliezze. E con ogni probabilità lo saranno sempre di più, in futuro.

Capitolo 2: Description of the proposed methodology and architecture

2.1 Descrizione del Problema

L'obiettivo principale di questo elaborato è quello di analizzare le recensioni degli utenti, provenienti dal dataset YELP, al fine di addestrare un classificatore multiclasse per suddividere le recensioni in 3 categorie:

- Positive
- Negative
- Neutrali

Il dataset di YELP è una collezione di attività, recensioni e dati degli utenti da utilizzare per scopi p ersonali, educativi ed accademici. Il dataset è disponibile come file JSON dove sono presenti:

• 6,990,280 reviews
• 150,346 attività (businesses)
• 200,100 immagini
• 11 aree metropolitane
• 908,915 consigli
• 1,987,897 utenti
• >1.2 mln di attributi per le attività (parcheggio, ore etc)

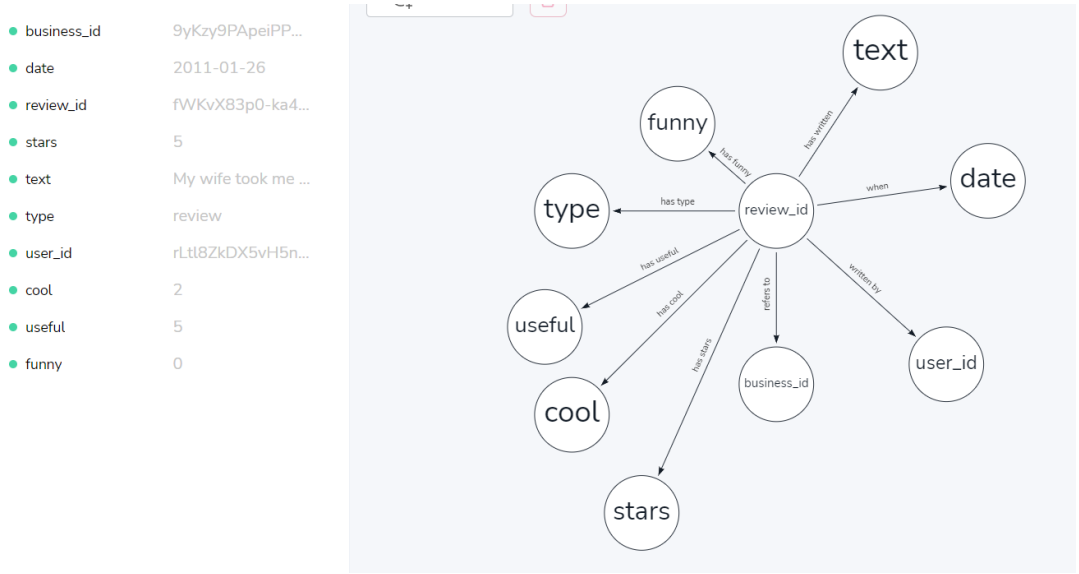
Il sottoinsieme di dati che è stato utilizzato è il file "review.json" il quale contiene informazioni relative alle recensioni lasciate dagli utenti tra cui la data, l'ID del locale, il numero di stelle ed il testo della recensione vera e propria. I vari attributi sono i seguenti:

• business_id: string
• cool: long
• date: string
• funny: long
• review_id: string
• stars: double
• text: string
• useful: long
• user_id: string

L'architettura utilizzata per la realizzazione di questo elaborato è stata la seguente:

In primo luogo, il dataset è stato caricato all'interno di Neo4j.
Neo4j è un software per basi di dati a grafo open source sviluppato interamente in Java ed è un database totalmente transazionale, La struttura a grafo di Neo4j si mostra estremamente comoda ed efficiente nel trattare strutture come gli alberi estratte ad esempio da file XML, filesystem e reti, che ovviamente vengono rappresentate con naturalezza da un grafo poiché sono esse stesse dei grafi. L'esplorazione di queste strutture risulta in genere più veloce rispetto a un database a tabelle . Ogni nodo contiene l'indice delle relazioni entranti e uscenti da esso, quindi la velocità di attraversamento del grafo non risente delle dimensioni complessive ma solo della densità dei nodi attraversati. Esistono delle implementazioni già pronte per le operazioni più comuni sui grafi, come la ricerca del cammino minimo tra due nodi tramite l'algoritmo di Dijkstra, la ricerca di cicli e il calcolo del diametro della rete.

Lo schema utilizzato per caricare il dataset di YELP su Neo4j è il seguente:



Tramite il codice riportato di seguito è stato possibile creare una connessione col DB ed importare un sottoinsieme del dataset (il testo ed il numero di stelle) all'interno dell' ambiente di lavoro Kaggle.

```
1.  !wget http://setup.johnsnowlabs.com/kaggle.sh -O - | bash
2.  pip install neo4j
3.  from neo4j import __version__ as neo4j_version
4.  print(neo4j_version)
5.  from neo4j import GraphDatabase
6.  class Neo4jConnection:
7.
8.      def __init__(self, uri, user, pwd):
9.          self.__uri = uri
10.         self.__user = user
11.         self.__pwd = pwd
12.         self.__driver = None
13.         try:
14.             self.__driver = GraphDatabase.driver(self.__uri, auth=(self.__user, self.__pwd))
15.         except Exception as e:
16.             print("Failed to create the driver:", e)
17.
18.     def close(self):
19.         if self.__driver is not None:
20.             self.__driver.close()
21.
22.     def query(self, query, db=None):
23.         assert self.__driver is not None, "Driver not initialized!"
24.         session = None
25.         response = None
26.         try:
27.             session = self.__driver.session(database=db) if db is not None else
    self.__driver.session()
28.             response = list(session.run(query))
29.         except Exception as e:
30.             print("Query failed:", e)
31.         finally:
32.             if session is not None:
33.                 session.close()
34.         return response
35.
```

```
conn = Neo4jConnection(uri="neo4j+s://1450681e.databases.neo4j.io", user="neo4j", pwd="a9igi8HfgRgXv8VRRr2U91RsTol6cjS2ZO7ZUj6QtPc")
from neo4j import GraphDatabase
#establish connection
graphdp = GraphDatabase.driver(uri="neo4j+s://1450681e.databases.neo4j.io", auth=("neo4j","a9igi8HfgRgXv8VRRr2U91RsTol6cjS2ZO7ZUj6QtPc"))
session = graphdp.session()

```

```
query = ''' MATCH (t:text)-[g:`has written`]-(r:review_id)-[h:`has stars`]-(s:stars) RETURN t,s
'''
result = session.run(query).data()
df = pd.DataFrame(result)
a2 = spark.createDataFrame(df)
review=a2.select('t.text','s.stars')
review.show()

```

```
+--------------------+-----+
|                text|stars|
+--------------------+-----+
|Beautiful restaur...|    5|
|I find it hilario...|    5|
|We had brunch at ...|    5|
|Great food and se...|    5|
|As a bonafide rec...|    5|
|My favorite sushi...|    5|
|Let's see...what ...|    5|
|I have a fond pla...|    5|
|Went to Yogurt Ki...|    5|
|I'm not generally...|    5|
|Best Cakes around...|    5|
|Highly recommend....|    5|
|Why did I wait so...|    5|
|Loved it. Food wa...|    5|
|Headed back to Lo...|    5|
|I'm not normally ...|    5|
|Great place for S...|    5|
|My profile says.....|    5|
|This is one excel...|    5|
|I know Kerrie thr...|    5|
+--------------------+-----+
```

Successivamente è stato possibile svolgere le dovute analisi e portare a termine il task assegnato. Infine, i risultati ottenuti sono stati salvati all'interno di MongoDB.

MongoDB è un database di tipo NoSQL, sviluppato in C++, open-source, document-oriented e scalabile che si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico.

Per suddividere le recensioni in positive, negative e neutrali, è stato utilizzato il campo "review_stars" presente all'interno di ogni recensione (selezionando solo 5000 recensioni per classe) il quale ha permesso di ottenere una label per ogni recensione, seguendo il seguente schema:

- Se una recensione ha più di 3 stelle : Positiva
- Se una recensione ha esattamente 3 stelle: Neutrale
- Se una recensione ha meno di 3 stelle : Negativa

In parallelo a questa fase di etichettatura, le recensioni sono state processate al fine di rimuovere qualsiasi segno di punteggiatura, con lo scopo di rendere il lavoro del classificatore più semplice. Inoltre, poiché alcune feature vengono calcolate in base alla frequenza di occorrenza di una parola, è stato opportuno rimuovere questi segni di punteggiatura al fine di evitare di ottenere feature poco utili.

```
1.  def for_exist_column(df, col, pre):
2.      if col in df.columns:
3.          return pre(df[col])
4.      else:
5.          return F.lit(False)
6.  review_neutral=review.select('*').where(review['stars']==3)
7.  review=review.join(review_neutral,how='leftanti',on='text')
8.  review_neutral=review_neutral.withColumn('Target_sentiment',F.when(for_exist_column(review_neut
    ral, 'stars', lambda c: c==3), 'neutral'))
9.  review=review.withColumn("Target_sentiment",F.when(for_exist_column(review, 'stars', lambda c:
    c>3), 'positive').otherwise('negative'))
10. review=review.withColumn( 'text',regexp_replace('text',r',|\.|&|\\|\||-|_|!|.', ''))
11. review_positive=
    review.select('text','stars','Target_sentiment').where(review['Target_sentiment'] =='positive')
12. review_positive=review_positive.orderBy(rand()).limit(5000)
13. review_negative=
    review.select('text','stars','Target_sentiment').where(review['Target_sentiment'] =='negative')
14. review_negative=review_negative.orderBy(rand()).limit(5000)
15. review_neutral=review_neutral.orderBy(rand()).limit(5000)
16. reviewf=review_positive.union(review_negative)
17. reviewf=reviewf.union(review_neutral)
18.
19.
```

In questa fase , ci si è soffermati anche ad analizzare altre caratteristiche di queste recensioni come ad esempio :

- Lunghezza della recensione
- Numero di parole presenti all'interno della recensione
- Parole più frequenti all'interno della recensione

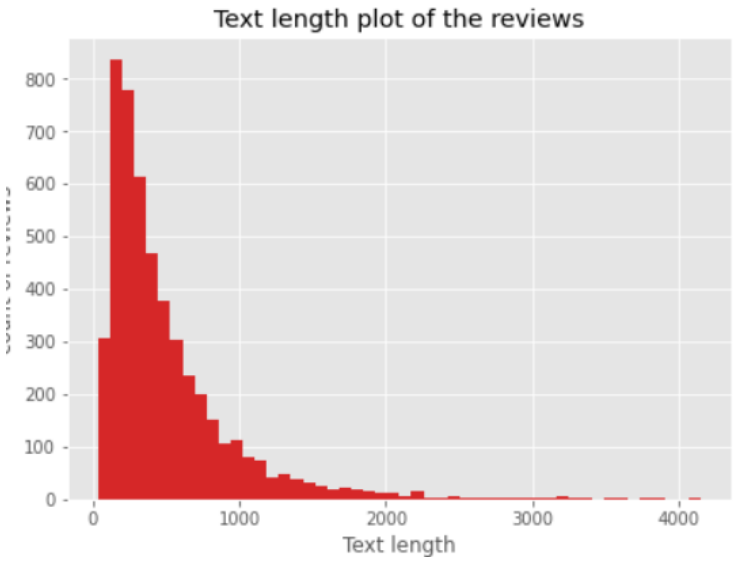


Figura 1: Positive

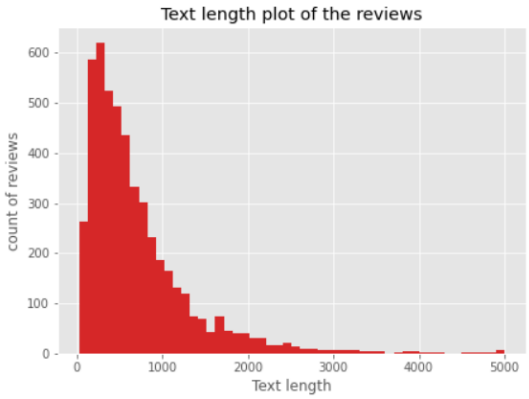


Figura 2:Negative

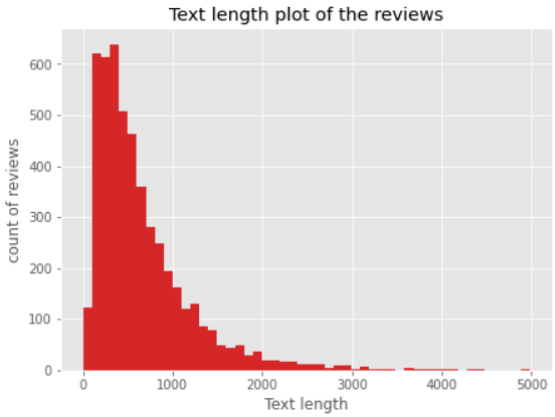


Figura 3: Neutrali

Analizzando questi istogrammi salta subito all'occhio che la lunghezza di una recensione positiva , in media , è più piccola rispetto a quella di una recensione neutrale o negativa. Gli stessi risultati sono visibili anche negli istogrammi seguenti che riportano il numero di parole presenti all'interno delle recensioni.

Figura 4: Positive

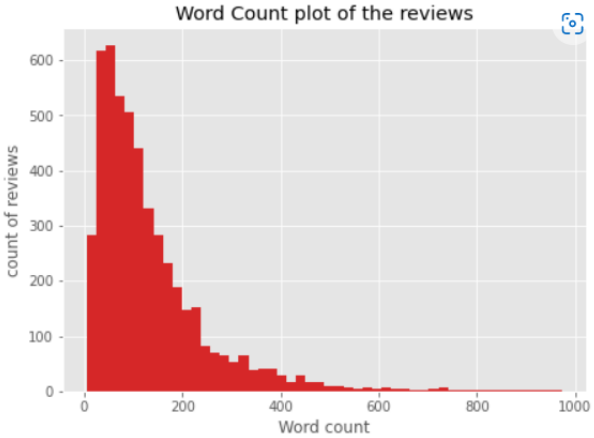


Figura 5:Negative

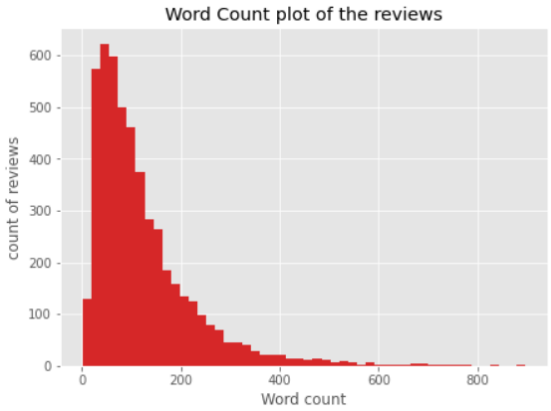


Figura 6: Neutrali

Come detto precedentemente, oltre ad analizzare la lunghezza delle recensioni è opportuno analizzarne il contenuto, di conseguenza sono state estratte (a seguito di un processo di lemmatizzazione) le parole più frequenti all'interno delle recensioni, ottenendo la seguente tabella:

POSITIVE		NEUTRALI		NEGATIVE	
place	2503	good	3711	order	2824
good	2219	food	3235	time	2703
food	2197	place	2871	food	2609
great	2155	order	2511	place	2266
time	1653	time	2235	back	2032
make	1509	service	1589	service	2008
service	1323	great	1465	tell	1528
back	1260	back	1426	good	1527
order	1254	nice	1190	call	1491
love	1034	drink	1064	wait	1429
delicious	875	pretty	1050	bad	1187
firendly	858	restaurant	1037	minute	1170
nice	831	price	989	leave	1119
recommend	825	wait	955	ask	1109

Una delle prime cose che si può notare è ovviamente la correlazione con i grafici precedenti: la frequenza di occorrenza delle parole è direttamente proporzionale alla lunghezza dei testi, di conseguenza è ovvio aspettarsi una frequenza minore per quanto riguarda le recensioni positive rispetto a quelle neutrali e negative. Entrando un po' più nel dettaglio, all'interno delle recensioni positive troviamo frequentemente parole come " place" , ovviamente "good" e "great", "food" , " delicious" , " recommend" etc.

Queste parole sono utili ad un'utente esterno per capire che qualcun altro ha apprezzato il locale, ma forniscono poca semantica in generale , come se l'utente che recensisce positivamente si limitasse solo ad apprezzare il locale senza specificare molto il perché. Per quanto riguarda quelle neutrali invece, abbiamo parole più eterogenee, si inizia a parlare di "drink" , di "restaurant" , di prezzo e di attesa. Queste parole potrebbero indurre a pensare che le recensioni neutrali sono più utili rispetto alle altre, perché magari essendo neutrali mettono a paragone i pro ed i contro del locale, senza limitarsi ad elogiarlo o criticarlo,
fornendo un set di informazioni più completo al lettore della recensione. Infine, come era possibile prevedere, all'interno delle recensioni negative si parla soprattutto di "ordini", "tempo" ed iniziano a comparire parole come "ask", "tell", "call" le quali potrebbero significare che l'utente abbia chiesto o chiamato un responsabile del locale (magari) per velocizzare il servizio , oppure che riferirà ai propri amici di non andare in quel locale.

In seguito, è riportato il codice utilizzato per questa analisi.

```
1.  def plot_histogram(data,x,y,t,u):
2.      ## Plot Distribution of the reviews
3.      data=data.select(u).toPandas().to_numpy().ravel()
4.
5.      plt.style.use('ggplot')
6.      fig, ax = plt.subplots(figsize=(7,5))
7.      plt.hist(data, bins=50, color = "tab:red")
8.      ax.set_title(t)
9.      ax.set_xlabel(x)
10.     ax.set_ylabel(y)
11.
12.
13.
14.
15.
16. a1=review_neutral.rdd.map(lambda x: [x['text'],len(x['text'])])
17. schema = StructType([StructField('text', StringType(), True),StructField('Text_Length',
    IntegerType(), True)])
18. a1 = spark.createDataFrame(a1,schema)
19. a2=review_neutral.rdd.map(lambda x: [x['text'],len(x['text'].split())])
20. schema = StructType([StructField('text', StringType(), True),StructField('Word_Count',
    IntegerType(), True)])
21. a2 = spark.createDataFrame(a2,schema)
22.
23. a1=a1.join(a2,on='text',how='left')
24. nlp_pipeline=Pipeline(
25. stages=[document,
26. sentence,
27. token,
28. stop_words,
29. lemmatizer,
```

```
30. finisher])
31. model=nlp_pipeline.fit(a1)
32. a=model.transform(a1)
33.
34. plot_histogram(a, x = "Text length", y = "count of reviews", t = 'Text length plot of the
    reviews',u='Text_Length')
35. plot_histogram(a, x = "Word count", y = "count of reviews", t = 'Word Count plot of the
    reviews',u='Word_Count')
```

2.2 Soluzioni proposte

Per effettuare questa classificazione Multiclasse utilizzando tecniche di Sentiment Analysis si è fatto un uso congiunto di due librerie principali : SparkNLP , Mllib

Spark NLP (Spark Natural Language Processing) è una libreria di elaborazione del testo open source per l'elaborazione avanzata del linguaggio naturale la quale supporta i linguaggi di programmazione Python, Java e Scala. La libreria è basata su Apache Spark e sulla sua libreria Spark ML.

Il suo scopo è fornire un'API per pipeline di elaborazione del linguaggio naturale che implementi i recenti risultati della ricerca accademica come software di livello produttivo, scalabile e addestrabile. La libreria offre modelli di rete neurale pre-addestrati, pipeline e supporto per l'addestramento di modelli personalizzati.

Il design della libreria fa uso del concetto di pipeline che è un insieme ordinato di annotatori di testo. Gli annotatori pronti all'uso includono:

- Tokenizer
- Normalizer
- Stemming
- Lemmatizer
- Regular expression
- TextMatcher
- Chunker

- DateMatcher
- SentenceDetector
- …

La libreria inoltre include anche risorse e modelli pre-addestrati per più di duecento lingue come inglese, francese, cinese, turco etc..

MLlib è la libreria di apprendimento automatico (ML) di Spark. Il suo obiettivo è rendere scalabile e facile l'apprendimento automatico. Ad alto livello, fornisce strumenti come:

- Algoritmi ML: algoritmi di apprendimento comuni come classificazione, regressione, clustering e filtraggio collaborativo.
- Feature: estrazione delle caratteristiche, trasformazione, riduzione della dimensionalità e selezione.
- Pipeline: strumenti per costruire, valutare e ottimizzare le pipeline ML.
- Persistenza: salvataggio e caricamento di algoritmi, modelli e pipeline.
- Utilità: algebra lineare, statistica, trattamento dati, ecc.

Grazie al supporto di queste librerie sono stati valutati 5 approcci differenti per risolvere il problema:

- Logistic Regression
- Random Forest
- Naive Bayes
- Universal Sentence Encoder
- BERT

2.2.1 Regressione Logistica

In seguito alla fase di etichettatura delle recensioni, il primo approccio che è stato valutato è quello della regressione logistica.
In primo luogo, si è suddiviso il dataset in training set (80%) e test set (20%) tramite uno split stratificato per preservare la distribuzione delle classi.

```
1.  train_set, test_set = reviewf.randomSplit([0.8,0.2],seed=100)
2.
```

Train

```
+----------------+-----+
|Target_sentiment|count|
+----------------+-----+
|        positive| 4021|
|         neutral| 4029|
|        negative| 3987|
+----------------+-----+
```

Test

```
+----------------+-----+
|Target_sentiment|count|
+----------------+-----+
|        positive|  979|
|         neutral|  971|
|        negative| 1013|
+----------------+-----+
```

General

```
+----------------+-----+
|Target_sentiment|count|
+----------------+-----+
|        positive| 5000|
|         neutral| 5000|
|        negative| 5000|
+----------------+-----+
```

Vengono quindi richiamati il Document Assembler, il Sentence Detector, il Tokenizer , lo StopWords Cleaner , il Lemmatizer ed il Finisher.

```
document= DocumentAssembler()\
.setInputCol("text")\
.setOutputCol("document")
sentence=SentenceDetector()\
.setInputCols(['document'])\
.setOutputCol('sentence')
token=Tokenizer()\
.setInputCols(['sentence'])\
.setOutputCol('token')
stop_words=StopWordsCleaner.pretrained('stopwords_en','en')\
.setInputCols(['token'])\
.setOutputCol('cleanTokens')\
.setCaseSensitive(False)
lemmatizer=LemmatizerModel.pretrained()\
.setInputCols(['cleanTokens'])\
.setOutputCol('lemma')
finisher=Finisher()\
.setInputCols(['lemma'])\
.setOutputCols(['token_features'])\
.setOutputAsArray(True)\
.setCleanAnnotations(False)

```

- Document Assembler:

  Ha il ruolo di trasformare il testo in un formato processabile da sparkNLP ed è il punto di partenza per ogni Pipeline di sparkNLP

- Sentence Detector:

  E' un annotatore che rileva i limiti delle frasi utilizzando espressioni regolari, ovvero numeri, abbreviazioni, punteggiature, periodi multipli.

  Esempio:

  Input: Questa è la prima frase, questa è la seconda.

  Output: [Questa è la prima frase],[questa è la seconda]

- Tokenizer:

  "Tokenizza" il testo non elaborato, identificando i token con standard di Tokenizzazione.

  Esempio:

  Input: [Questa sera vado all'aperto]->[Questa,sera,vado,all',aperto]

- StopWordsCleaner:

  Questo annotatore prende una sequenza di stringhe e rimuove tutte le "stop words" dalle sequenze di input, ovvero elimina delle parole che forniscono poca semantica come per esempio congiunzioni, pronomi etc..

  Esempio:

  [This is my first home] -> [first,home]

- Lemmatizer:

  Identifica i lemmi dalle parole con l'obiettivo di restituire una parola del dizionario di base. Recupera la parte significativa di una parola.

  Esempio:

  [mangiato]->[mangiare]

- Finisher:

  Converte i risultati delle annotazioni in un formato più facile da usare ed è utile per estrarre i risultati dalle pipelines Spark NLP.

A questo punto vengono richiamati dei metodi di Mllib per processare i testi ed estrarre delle features , come per esempio:

- HashingTF:

  Mappa una sequenza di termini alle loro frequenze utilizzando il trucco dell'hashing.

- IDF:

  Viene utilizzata la formulazione standard: idf = $\log\left(\frac{(m+1)}{(d(t)+1}\right)$, dove m è il numero totale di documenti e d(t) è il numero di documenti che contengono il termine t.

- StringIndexer:

  Un indicizzatore di etichette che esegue il mapping di una colonna di stringhe di etichette ad una colonna di ML di indici di etichette.

- IndexToString:

  Fa l'opposto di StringIndexer.

```
1.  from pyspark.ml.feature import HashingTF, IDF, StringIndexer, IndexToString
2.  from pyspark.ml.classification import LogisticRegression, NaiveBayes
3.  from pyspark.ml.evaluation import MulticlassClassificationEvaluator
4.  hashTF=HashingTF(inputCol="token_features",outputCol="raw_features")
```

```
idf=IDF(inputCol="raw_features",outputCol="features",minDocFreq=5)
label_strIdx=StringIndexer(inputCol='Target_sentiment',outputCol="label")
LogReg=LogisticRegression(maxIter=10)
label_idxStr=IndexToString(inputCol="label",outputCol="class")
nlp_pipeline=Pipeline(
stages=[document,
        sentence,
        token,
        stop_words,
        lemmatizer,
        finisher,
        hashTF,
        idf,
         label_strIdx,
        LogReg,
        label_idxStr])
clas_model=nlp_pipeline.fit(train_set)

```

Arrivati a questo punto, la pipeline è stata istanziata ed il modello viene addestrato sul training set. Successivamente si passa alla fase di predizione sul test set e la relativa analisi dei risultati.

```
pred=clas_model.transform(test_set)
evaluator=MulticlassClassificationEvaluator(
labelCol='label',predictionCol='prediction',metricName='accuracy')
accuracy=evaluator.evaluate(pred)
print('Accuracy=%g' %(accuracy))
print('Test Error =%g' %(1.0-accuracy))

```

```
[Stage 237:=======================================>
Accuracy=0.880189
Test Error =0.119811
```

Sono state valutate quindi le prestazioni sul test set, analizzando anche altre statistiche oltre all'accuracy come la precisione la recall:

```
from sklearn.metrics import classification_report,accuracy_score
df_lr=clas_model\
.transform(test_set)\
.select('Target_sentiment','label','prediction')\
.toPandas()
df_lr.head()
print(classification_report(df_lr.label,df_lr.prediction))

```

	precision	recall	f1-score	support
0.0	0.87	0.91	0.89	971
1.0	0.88	0.87	0.87	979
2.0	0.89	0.86	0.88	1013
accuracy			0.88	2963
macro avg	0.88	0.88	0.88	2963
weighted avg	0.88	0.88	0.88	2963

Per concludere viene riportata di seguito la matrice di confusione.

	0	1	2
0	885.0	64.0	65.0
1	47.0	849.0	74.0
2	39.0	66.0	874.0

2.2.2 Naive Bayes

Utilizzando la stessa pipeline dell'approccio visto precedentemente e cambiando unicamente il classificatore utilizzato (in questo caso Naive Bayes) sono stati ottenuti i seguenti risultati:

```
hashTF=HashingTF(inputCol="token_features",outputCol="raw_features")
idf=IDF(inputCol="raw_features",outputCol="features",minDocFreq=5)
label_strIdx=StringIndexer(inputCol='Target_sentiment',outputCol="label")
bayes_class=NaiveBayes(smoothing=111)
label_idxStr=IndexToString(inputCol="label",outputCol="class")
nlp_pipeline_b=Pipeline(
stages=[document,
        sentence,
        token,
        stop_words,
        lemmatizer,
        finisher,
        hashTF,
        idf,
         label_strIdx,
        bayes_class,
        label_idxStr])
clas_model_naive=nlp_pipeline_b.fit(train_set)

```

```
pred_2=clas_model_naive.transform(test_set)
evaluator=MulticlassClassificationEvaluator(
labelCol='label',predictionCol='prediction',metricName='accuracy')
accuracy_2=evaluator.evaluate(pred_2)
print('Accuracy=%g' %(accuracy_2))
print('Test Error =%g' %(1.0-accuracy_2))
from sklearn.metrics import classification_report,accuracy_score
```

```
df_lr_2=clas_model_naive\
.transform(test_set)\
.select('Target_sentiment','label','prediction')\
.toPandas()

```

	precision	recall	f1-score	support
0.0	0.59	0.99	0.74	971
1.0	0.99	0.26	0.41	979
2.0	0.77	0.81	0.79	1013
accuracy			0.69	2963
macro avg	0.78	0.69	0.65	2963
weighted avg	0.78	0.69	0.65	2963

Come è possibile notare , questo modello performa in maniera peggiore rispetto a quello precedentemente illustrato. Questo andamento si riflette anche nella matrice di confusione:

]:

	0	1	2
0	966.0	483.0	192.0
1	0.0	252.0	2.0
2	5.0	244.0	819.0

È possibile notare che il classificatore sbaglia poco per quanto riguarda la classe 1 (neutrale), mentre per le altre 2 classi gli errori sono significativi.

2.2.3 Random Forest

Random Forest è un classificatore d'insieme ottenuto dall'aggregazione tramite bagging di alberi di decisione. L'algoritmo si pone come soluzione che minimizza l'overfitting del training set rispetto agli alberi di decisione. In questo caso poiché il classificatore necessitava in input una colonna di feature e non poteva essere inserito all'interno della pipeline, è stata creata una pipeline senza classificatore per estrarre le fatures dalle recensioni.

```
from pyspark.ml.classification import RandomForestClassifier
hashTF=HashingTF(inputCol="token_features",outputCol="raw_features")
idf=IDF(inputCol="raw_features",outputCol="features",minDocFreq=5)
label_strIdx=StringIndexer(inputCol='Target_sentiment',outputCol="label")
label_idxStr=IndexToString(inputCol="label",outputCol="class")
```

```
nlp_pipeline_b=Pipeline(
stages=[document,
        sentence,
        token,
        stop_words,
        lemmatizer,
        finisher,
        hashTF,
        idf,
         label_strIdx,
        label_idxStr])
clas_modell=nlp_pipeline_b.fit(train_set)
utile_test=clas_modell.transform(test_set)

```

L'output è stato dato in pasto al classificatore per addestrarsi, ottenendo i seguenti risultati:

```
utile_train=clas_modell.transform(train_set)
algo = RandomForestClassifier(featuresCol='features', labelCol='label')

model = algo.fit(utile_train)
predictions = model.transform(utile_test)

```

```
evaluator=MulticlassClassificationEvaluator(
labelCol='label',predictionCol='prediction',metricName='accuracy')
accuracy_2=evaluator.evaluate(predictions)
print('Accuracy=%g' %(accuracy_2))
print('Test Error =%g' %(1.0-accuracy_2))
from sklearn.metrics import classification_report,accuracy_score
df_lr_2=clas_modell\
.transform(test_set)\
.select('Target_sentiment','label','prediction')\
.toPandas()

```

```
Accuracy=0.776578
Test Error =0.223422
```

Il modello raggiunge quindi le seguenti prestazioni in termini di statistiche:

	precision	recall	f1-score	support
0.0	0.99	0.97	0.98	971
1.0	0.63	0.84	0.72	979
2.0	0.76	0.53	0.62	1013
accuracy			0.78	2963
macro avg	0.79	0.78	0.77	2963
weighted avg	0.79	0.78	0.77	2963

	0	1	2
0	944.0	0.0	6.0
1	19.0	822.0	472.0
2	8.0	157.0	535.0

In questo caso il modello , a differenza di Naive Bayes predice meglio la classe '0' (negative) , commette degli errori sulle altre classi ma in numero minore rispetto al modello precedente.

2.2.4 Universal Sentence Encoder

In questo quarto approccio è stata utilizzato l'Universal Sentence Encoder. L'Universal Sentence Encoder codifica il testo in vettori di grandi dimensioni che possono essere utilizzati per la classificazione del testo, la somiglianza semantica, il clustering e altre attività di linguaggio naturale.
Il modello è addestrato e ottimizzato per testi di lunghezza superiore a una parola, come frasi o brevi paragrafi. È addestrato su una varietà di fonti di dati con l'obiettivo di comprendere dinamicamente il linguaggio naturale, tramite un codificatore DAN (Deep averaging Network).
È un modello efficiente e garantisce prestazioni accurate su diverse attività di trasferimento. L'input è un testo inglese di lunghezza variabile e l'output è un vettore a 512 dimensioni.

```
document = DocumentAssembler()\
    .setInputCol("text")\
    .setOutputCol("document")

use = UniversalSentenceEncoder.pretrained()\
 .setInputCols(["document"])\
 .setOutputCol("sentence_embeddings")
classsifierdl = ClassifierDLApproach()\
  .setInputCols(["sentence_embeddings"])\
  .setOutputCol("class")\
  .setLabelColumn("Target_sentiment")\
  .setMaxEpochs(10)\
  .setEnableOutputLogs(True)
use_clf_pipeline = Pipeline(
    stages = [
        document,
        use,
```

```
        classsifierdl
    ])

```

Per la classificazione è stato utilizzato “ClassifierDLApproach()”, ovvero viene addestrato un ClassifierDL per la classificazione di testo multiclasse generica.
ClassifierDL utilizza Universal Sentence Encoder come input per le classificazioni del testo. L'annotatore ClassifierDL utilizza un modello di apprendimento profondo (DNN) creato all'interno di TensorFlow e che supporta fino a 100 classi.

```
use_pipelineModel = use_clf_pipeline.fit(train_set)
preds=use_pipelineModel.transform(test_set)
from sklearn.metrics import classification_report, accuracy_score
df=preds.select('text','Target_sentiment','class.result').toPandas()
df['result']=df['result'].apply(lambda x: x[0])
print(classification_report(df.Target_sentiment,df.result))
print(accuracy_score(df.Target_sentiment,df.result))

```

I risultati ottenuti sono i seguenti:

	precision	recall	f1-score	support
negative	0.78	0.80	0.79	1013
neutral	0.66	0.71	0.69	971
positive	0.84	0.75	0.79	979
accuracy			0.76	2963
macro avg	0.76	0.76	0.76	2963
weighted avg	0.76	0.76	0.76	2963

2.2.5 BERT

BERT sta per Rappresentazioni di encoder bidirezionali di Transformers .
È un'architettura di rete neurale profonda costruita in base agli ultimi progressi del deep learning. È stato rilasciato nel 2018 da Google e ha ottenuto prestazioni all'avanguardia (SOTA) in più benchmark di comprensione del linguaggio naturale (NLU). Oggi, altri modelli basati su trasformatore, come GPT-3 o PaLM , hanno superato BERT. Tuttavia, BERT rappresenta una svolta fondamentale nell'adozione e nel consolidamento di modelli basati su trasformatori in applicazioni NLP avanzate.
In questa sezione, verrà presentato l'ultimo approccio considerato che fa uso del modello di BERT: sent_small_bert_L8_512 .

Questo è uno dei modelli BERT più piccoli e semplici. I modelli BERT più piccoli sono destinati ad ambienti con risorse computazionali limitate. È possibile fare il tuning allo stesso modo dei modelli BERT originali, tuttavia, sono più efficaci nel contesto della distillazione della conoscenza.

Come visto negli approcci precedenti, viene creata una pipeline :

```
document = DocumentAssembler()\
      .setInputCol("text")\
      .setOutputCol("document")

bert_sent = BertSentenceEmbeddings.pretrained('sent_small_bert_L8_512')\
      .setInputCols(["document"])\
      .setOutputCol("sentence_embeddings")


classsifierdl = ClassifierDLApproach()\
      .setInputCols(["sentence_embeddings"])\
      .setOutputCol("class")\
      .setLabelColumn("Target_sentiment")\
      .setMaxEpochs(10)\
      .setEnableOutputLogs(True)\
      .setLr(0.001)

bert_sent_clf_pipeline = Pipeline(
    stages = [
        document,
        bert_sent,
        classsifierdl
    ])

```

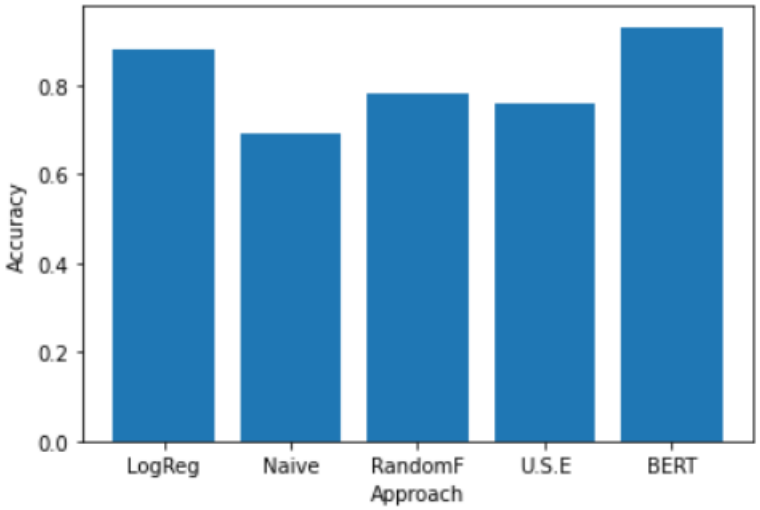
Successivamente dopo aver richiamato il metodo .fit() sul training set , vengono analizzate le prestazioni del modello.

```
1.  bert_sent_pipelineModel = bert_sent_clf_pipeline.fit(train_set)
2.  preds = bert_sent_pipelineModel.transform(test_set)
3.
4.  preds_df = preds.select('Target_sentiment','text',"class.result").toPandas()
5.
6.  preds_df['result'] = preds_df['result'].apply(lambda x : x[0])
7.
8.  print (classification_report(preds_df['result'], preds_df['Target_sentiment']))
9.
```

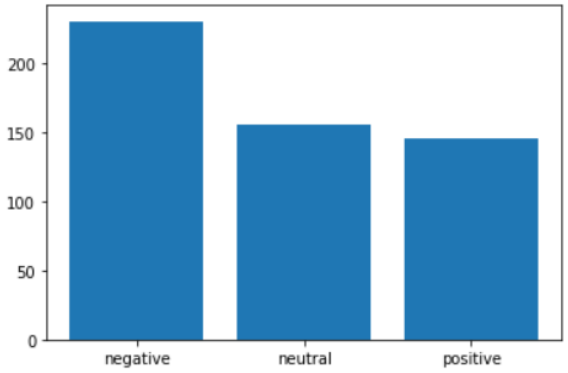
Ottenendo i seguenti risultati:

	precision	recall	f1-score	support
negative	0.90	0.90	0.90	1013
neutral	0.99	1.00	0.99	963
positive	0.90	0.89	0.90	987
accuracy			0.93	2963
macro avg	0.93	0.93	0.93	2963
weighted avg	0.93	0.93	0.93	2963

Capitolo 3: Conclusioni



In questo grafico a barre vengono riportati i risultati sull'accuracy ottenuti utilizzando i 5 approcci. Si è notato che il primo e l'ultimo approccio (Regressione logistica e BERT) hanno delle prestazioni migliori, riuscendo ad ottenere un accuracy intorno al 90% , Naive Bayes invece è quello che ha ottenuto il valore di accuracy più piccolo (69%). Gli altri due approcci presentano prestazioni simili tra di loro , con un accuracy del 78% circa.



Nel grafico mostrato sopra, viene rappresentato il numero di errori medio commesso dai classificatori per ogni classe. Si nota che la classe "negative" risulta quella maggiormente

classificata in maniera errata, mentre per quanto riguarda quelle positive e quelle neutrali abbiamo pressocché lo stesso numero di errori commesso dai classificatori.

Infine, i risultati ottenuti di queste classificazioni (il testo, la label e la predizione) sono stati salvati su MongoDB tramite il codice riportato di seguito.

```
1.  import pandas as pd
2.  from pymongo import MongoClient
3.  # Load csv dataset
4.  data =pred.select('text','label','prediction').toPandas()
5.  # Connect to MongoDB
6.  client =
    MongoClient("mongodb+srv://luigigarofalo:<PASSWORD><@cluster0.sy8pl.mongodb.net/?retryWrites=tr
    ue&w=majority")
7.  db = client['results']
8.  collection = db['logreg']
9.  data.reset_index(inplace=True)
10. data_dict = data.to_dict("records")
11. # Insert collection
12. collection.insert_many(data_dict)
13.
```