Relazione Progetto Modelli e Tecniche Per Big Data
Progetto 11 - IMDB

Studenti: Marco Greco 242455 - Francesco Bonacci 242713
Struttura dati utilizzata: Resilient Distributed Dataset
Tecnologia utilizzata: Spark
Interfaccia grafica: Interfaccia web
Backend: Flask & Python
Frontend: Html & Css

Il Dataset

Il *dataset* contiene 50.000 istanze che descrivono le recensioni di film composta ciascuna da due *features*: la recensione stessa e il corrispondente sentiment (positivo/negativo).

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

Figura 1.1 - Dataset

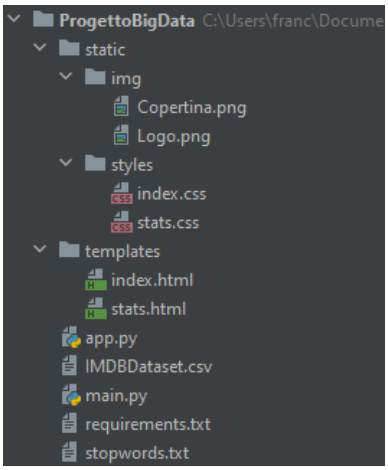
Come si può già intravedere dalla figura 1.1 nelle recensioni sono presenti dei *tag html* (

) e pertanto è stato necessario un breve *pre-processing* sui dati.

Struttura del progetto

Per realizzare il nostro progetto abbiamo utilizzato la tecnologia Spark per effettuare le interrogazioni sul *dataset*, la struttura dati RDD (Resilient Distributed Dataset) che ci permette di elaborare i dati in maniera distribuita.

Abbiamo deciso di realizzare una *web application* per permettere all'utente di interagire con i dati estratti dal *dataset*. Per fare ciò abbiamo utilizzato il *framework* Flask per il *backend* e HTML unito a CSS per l'interfaccia grafica (*frontend*).



Il progetto è composto da quattro file principali:

- **main.py** (in cui configuriamo l'ambiente Spark e definiamo le *query*);
- **app.py** (in cui configuriamo i *controller* per la comunicazione con il *frontend*, interagendo con il file main.py);
- **index.html** (in cui sviluppiamo la struttura dell'interfaccia grafica);
- **stats.html** (in cui sviluppiamo la pagina degli istogrammi);
- **index.css** (in cui definiamo gli stili dell'interfaccia grafica);
- **stats.css** (in cui definiamo gli stili della pagina "stats.html").

A seguire la schermata relativa alla *home page* della nostra *web application*, nella quale sono presenti le recensioni estratte dal dataset visualizzate sei per pagina.

Figura 1.2 – Schermata iniziale della *web application*

Configurazione

```
os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable
spark = SparkSession.builder.master("local[*]").appName("IMDB_Sentiment").getOrCreate()
conf_spark = SparkConf()
conf_spark.setAppName("IMDB_Sentiment").set("spark.executor.memory", "4g")
sc = spark.sparkContext
sc.setLogLevel("ERROR")
rdd_pyspark = sc.textFile("IMDBDataset.csv")
stopwords = sc.textFile("stopwords.txt").collect()
reviews = rdd_pyspark.map(lambda x: x.rsplit(',', 1)).cache()
```

Figura 1.3 - Codice Python configurazione

Inizialmente, abbiamo impostato l'ambiente per l'utilizzo della libreria PySpark. In particolare, le prime due righe di codice impostano le variabili d'ambiente PYSPARK_PYTHON e PYSPARK_DRIVER_PYTHON in modo tale che sia il *driver* che i nodi di elaborazione utilizzino la stessa versione di Python durante l'analisi dei dati con PySpark.

In seguito, abbiamo inizializzato la sessione Spark in modo da poter eseguire le *query* sul cluster e abbiamo configurato l'ambiente per effettuare l'analisi dei dati recuperati dal *dataset*. Questo è stato fatto utilizzando il metodo getOrCreate(), che crea la sessione Spark specificando (con local[*]) che *driver* e nodi verranno eseguiti sulla stessa macchina, e creando lo Spark Context a partire dalla sessione Spark, con un numero di *worker* pari al numero di core della macchina.

A questo punto abbiamo creato l'RDD (Resilient Distributed Dataset) a partire dal file csv fornito dai docenti ("IMDBDataset.csv") invocando il metodo textFile() sull'oggetto Spark Context.

Inoltre, ai fini dell'analisi è stato necessario importare un file di testo ("stopwords.txt") contenente le parole della lingua inglese considerate poco significative.

Infine, con l'ultima istruzione, otteniamo il dataset finale creando una lista per ogni istanza dell'RDD, ciascuna contenente due elementi: *review* e relativo *sentiment*. Il tutto viene fatto invocando la funzione map() sull'RDD originario: la funzione map() applicherà la funzione rsplit() su ciascuna delle istanze. Da notare l'utilizzo della funzione cache() al fine di migliorare le prestazioni mantenendo una copia dell'RDD in memoria.

Pre-processing dei dati

```python
def processWord(word: str):
    return re.sub("[^A-Za-z0-9]+", "", word.lower())


marcogreco
def splitAndProcess(words: str):
    return [processWord(word) for word in words.split()]


reviews = reviews.map(lambda x: [x[0].replace("<br /><br />", " "), x[1]])
namesOfColumns = reviews.first()
reviews = reviews.filter(lambda x: x != namesOfColumns)
```

Figura 1.4 – Codice Python per il pre-processing

In questa fase, sostituiamo i *tag* “

” con una stringa vuota invocando la funzione map(). Inoltre, invocando il metodo filter() eliminiamo la prima istanza dell’RDD che rappresenta il nome delle *features* e non un’effettiva recensione con il proprio *sentiment*.

Infine, abbiamo definito le funzioni splitAndProcess() e processWord(), le quali rispettivamente restituiscono la lista delle parole di una recensione e la ripuliscono dai vari caratteri speciali.

Query per l’analisi dei dati (file main.py)

Filtra per recensioni positive/negative

```
def filterByPositive(reviews):
    return reviews.filter(lambda x: x[1] == 'positive')

marcogreco
def filterByNegative(reviews):
    return reviews.filter(lambda x: x[1] == 'negative')
```

Figura 1.5 – Uso tecnologia Spark per filtrare recensioni positive/negative

Le due *query* in figura 1.5 filtrano rispettivamente le recensioni positive e quelle negative. La prima invoca sull’RDD contenente tutte le recensioni il metodo filter(), specificando che dovranno essere restituite solo le istanze tali che il valore del secondo elemento (quello che identifica il *sentiment*) è pari alla stringa ‘positive’; la seconda esegue la stessa operazione della prima ma specificando che il valore del *sentiment* debba essere ‘negative’.

Queste due query vengono poi richiamate all’interno del file app.py, in particolare nelle funzioni che definiscono i due *controller* relativi al filtro in base al tipo di *sentiment* (per la spiegazione dei *controller* si rimanda a pagina 22).

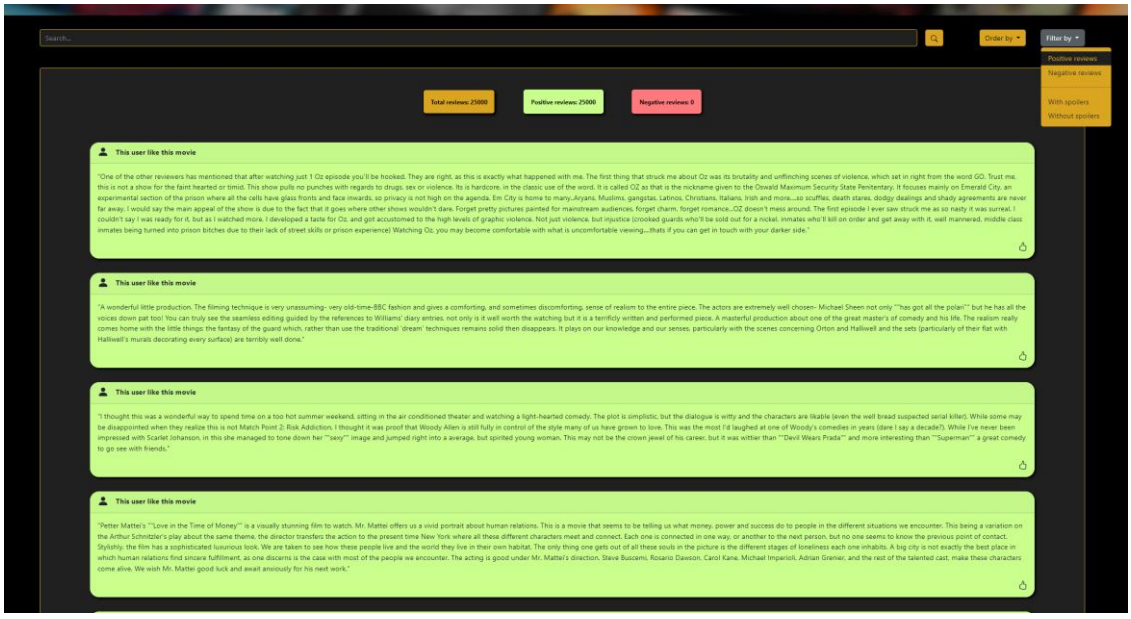


Figura 1.6 – Schermata recensioni filtrate per recensioni positive

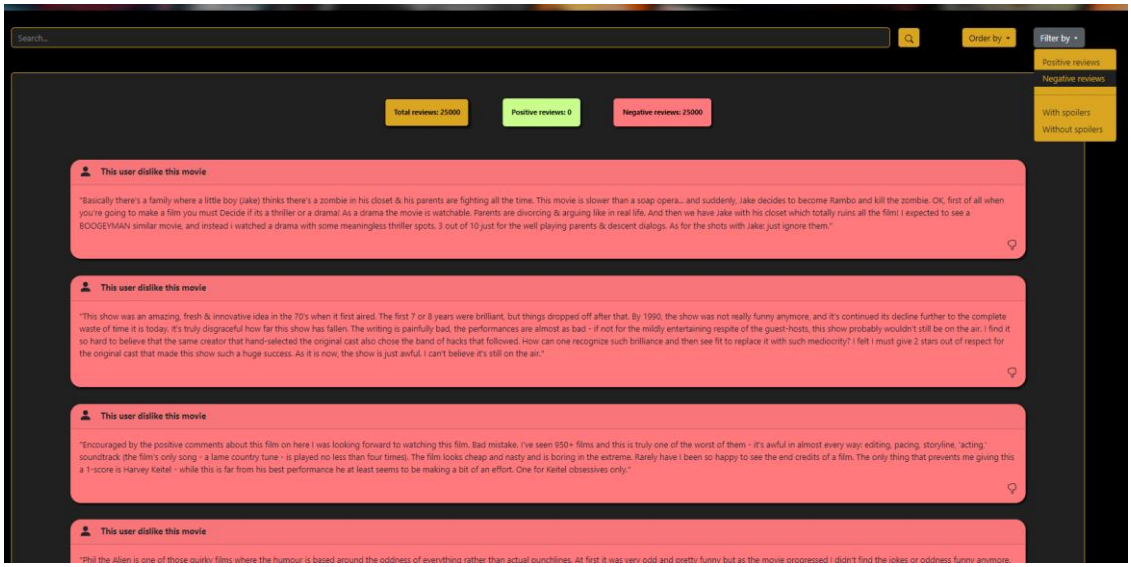


Figura 1.7 – Schermata recensioni filtrate per recensioni negative

Come si può vedere dalle figure 1.6 e 1.7 le recensioni vengono effettivamente filtrate per sentiment positivo. Questo si può facilmente vedere dal conteggio del numero di risultati totali che coincide con il conteggio delle recensioni positive in seguito all'esecuzione della query di filtraggio per recensioni positive e che coincide con il conteggio delle recensioni negative in seguito all'esecuzione della query di filtraggio per recensioni negative.

Di seguito il codice HTML per i corrispondenti componenti di filtraggio.

```
<div class="dropdown">
    <button class="btn btn-secondary dropdown-toggle" type="button" data-bs-toggle="dropdown" aria-expanded="false">
        Filter by
    </button>
    <ul class="dropdown-menu">
        <li><a class="dropdown-item {{'selected' if flags[0]}}" href="/positive">Positive reviews</a></li>
        <li><a class="dropdown-item {{'selected' if flags[1]}}" href="/negative">Negative reviews</a></li>
        <hr>
        <li><a class="dropdown-item {{'selected' if flags[2]}}" href="/withSpoilers">With spoilers</a></li>
        <li><a class="dropdown-item {{'selected' if flags[3]}}" href="/withoutSpoilers">Without spoilers</a></li>
    </ul>
</div>
```

Figura 1.8 – Codice HTML componente menu a tendina per i filtri

Filtra per recensioni con spoilers e senza spoilers

```
def filterBySpoilers(reviews):
    return reviews.filter(lambda x: (x[0].upper().count("SPOILER") - x[0].upper().count("NO SPOILER")) > 0)

francescobonacci +1
def filterByNoSpoilers(reviews):
    return reviews.filter(lambda x: (x[0].upper().count("SPOILER") - x[0].upper().count("NO SPOILER")) <= 0)
```

Figura 1.9 – Uso tecnologia Spark per filtrare recensioni con spoilers/senza spoilers

Le due *query* in figura 1.9 filtrano rispettivamente le recensioni che presentano degli spoilers e quelle che non ne presentano. In particolare, la prima invoca sull'RDD contenente tutte le recensioni il metodo filter(), specificando che dovranno essere restituite solo le istanze tali che la recensione (ovvero il primo elemento, x[0]) contiene la stringa 'Spoiler'; la seconda esegue la stessa operazione della prima ma specificando che la recensione non debba contenere la stringa 'Spoiler' al suo interno.

Bisogna precisare che non basta solo controllare la presenza o meno della stringa "Spoiler" all'interno della recensione in quanto essa è inglobata nella stringa "No Spoiler" che però esprime l'opposto significato. Per tale motivo è stato necessario contare il numero di occorrenze della parola "spoiler" e il numero di occorrenze della stringa "no spoiler": se la stringa "spoiler" è presente più volte rispetto alla stringa "no spoiler" allora significa che la recensione presenta degli spoiler, altrimenti no.

Queste due query vengono poi richiamate all'interno del file "app.py", in particolare nelle funzioni che definiscono i due *controller* relativi ai due filtri definiti sopra.

Il risultato dell'applicazione dei due differenti filtri si può vedere nelle due figure seguenti (2.1 e 2.2).

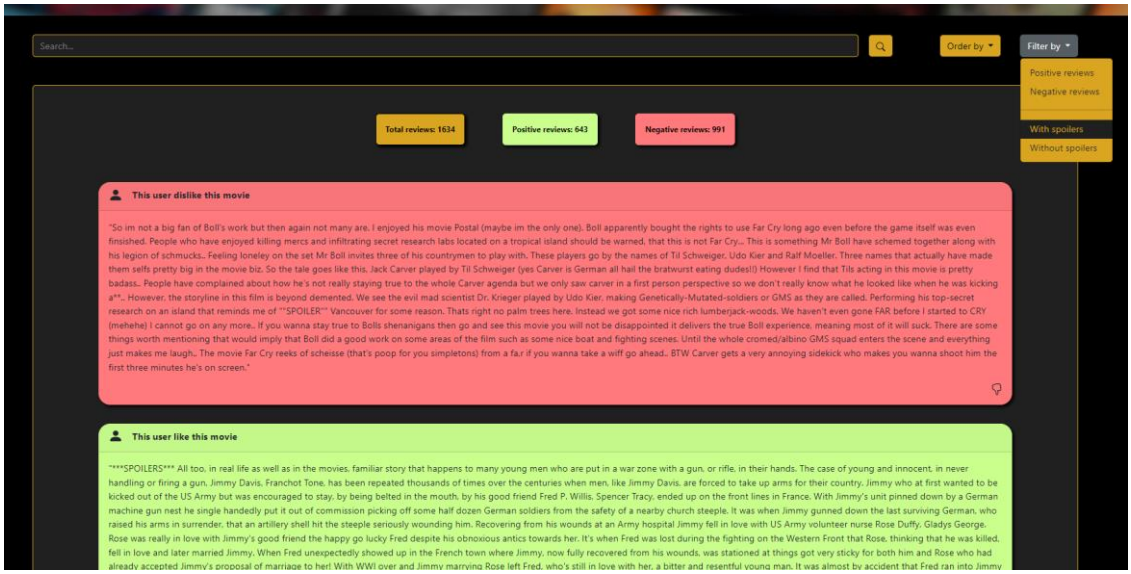


Figura 2.1 – Schermata recensioni filtrate per recensioni con spoiler

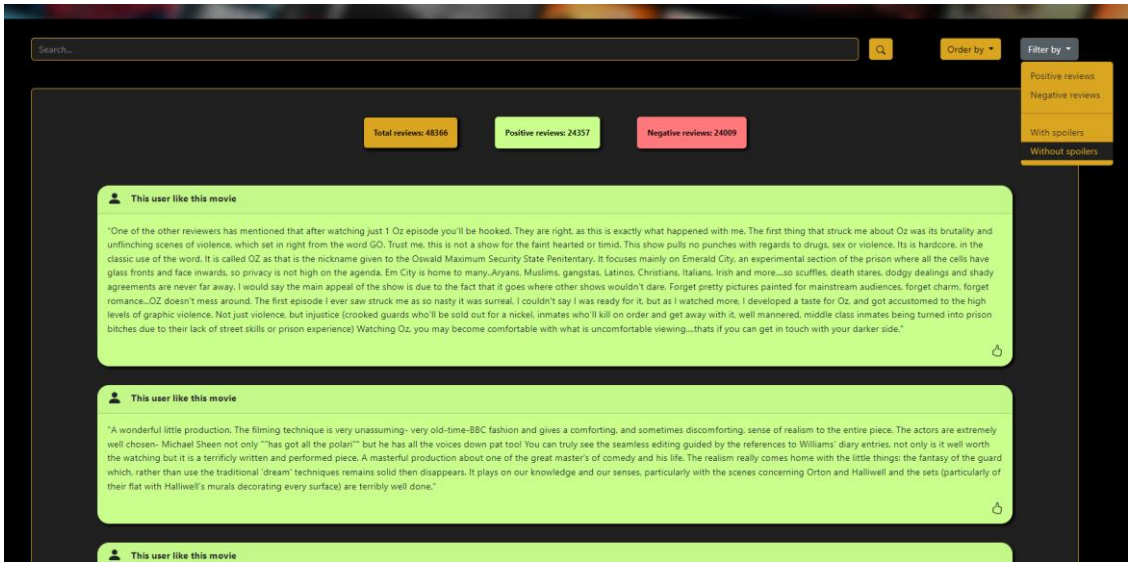


Figura 2.2 – Schermata recensioni filtrate per recensioni senza spoiler

Il codice HTML relativo a questi componenti di filtraggio è uguale a quello dei componenti di filtraggio relativi alle recensioni positive/negative, che si può vedere nella figura 1.8.

Filtra per recensioni che contengono una determinata stringa

```
def filterByWord(reviews,word):
    return reviews.filter(lambda x: re.search(r"\b"+word+r"\b", str(x[0]), re.IGNORECASE))
```

Figura 2.3 – Uso tecnologia Spark per filtrare recensioni che contengono una determinata stringa

La *query* in figura 2.3 filtra le recensioni considerando solamente quelle che al loro interno contengono la parola, o più in generale la stringa, che viene passata come parametro (*word*). In particolare, sull'RDD contenente tutte le recensioni viene invocato il metodo filter(), specificando che dovranno essere restituite solo le istanze tali che la recensione (ovvero il primo elemento, x[0]) contiene la stringa *word* passata come parametro.

Anche qui, come nel caso dei filtri per recensioni con o senza spoiler, non basta controllare semplicemente se la stringa *word* è contenuta o meno all'interno della recensione, perché la stringa potrebbe essere presente in diverse forme. Per tale motivo viene effettuata una ricerca utilizzando una espressione regolare (*regex*): la funzione lambda re.search() specificata nel metodo filter() manterrà le recensioni che al loro interno contengono almeno una parola rappresentata dalla *regex* `r"\b"+word+r"\b"`. Tale espressione regolare garantisce che la parola cercata sia una parola completa (circondata dai cosiddetti "confini di parola") e non solo una sottostringa di un'altra parola.

Questa *query* viene poi richiamata all'interno del file "app.py", in particolare nella funzione che definisce il *controller* relativo al filtro appena descritto. Il risultato della sua applicazione può essere visibile nella figura 2.4.

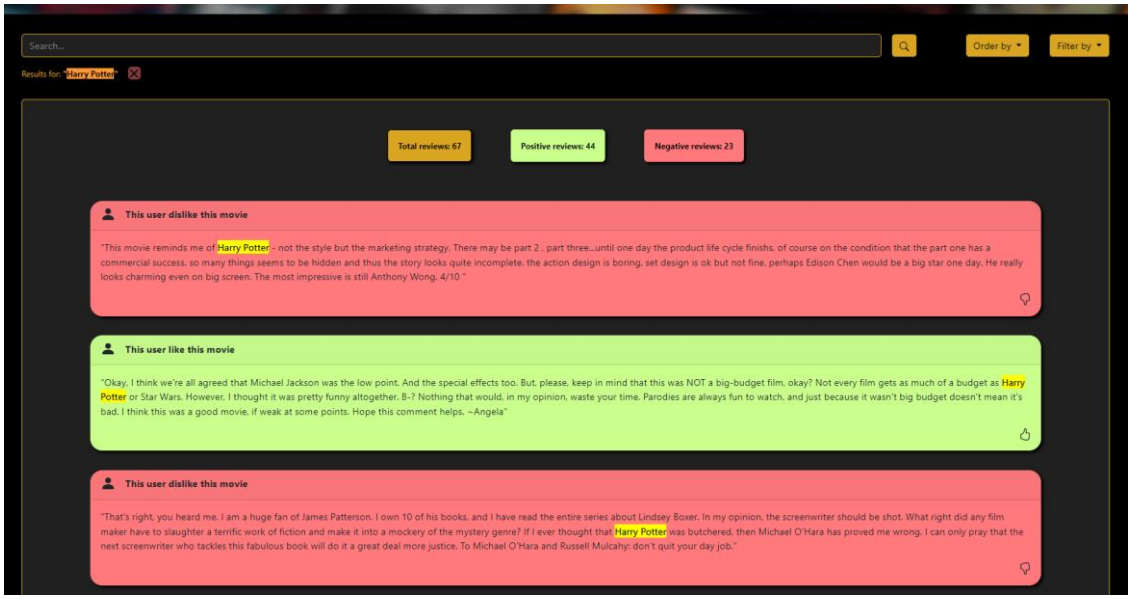


Figura 2.4 – Schermata recensioni filtrate per una determinata stringa ricercata dall'utente

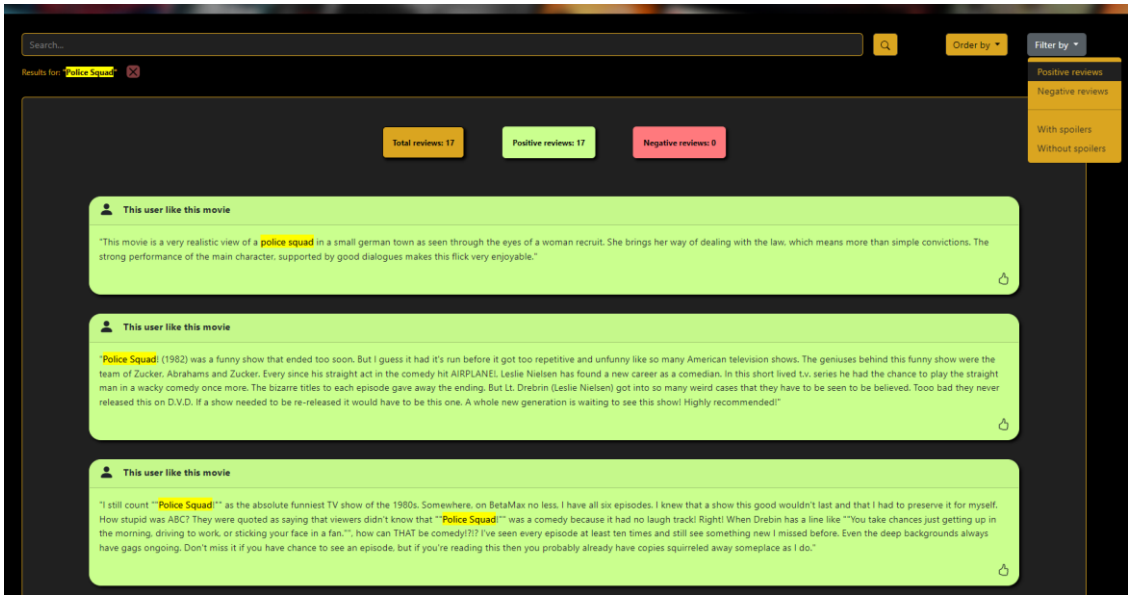


Figura 2.5 – Schermata recensioni filtrate per una determinata stringa ricercata dall'utente

Nella figura 2.5 possiamo notare come è possibile utilizzare contemporaneamente due o più criteri di filtraggio e/o ordinamento: in questo caso le recensioni sono state filtrate per quelle contenenti la stringa "Police Squad" e inoltre sono state filtrate per recensioni positive (come si può vedere dal numero di recensioni negative visualizzate che è pari a zero).

A seguire il codice HTML per tale componente di filtraggio.

```
<form class="row g-3" action="/search">
    <div class="col-auto input-search">
        <input type="text" class="form-control" name="searchString" placeholder="Search...">
    </div>
    <div class="col-auto">
        <button type="submit" class="btn btn-secondary mb-3">
            <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-search"
                viewBox="0 0 16 16">
                <path d="M11.742 10.344a6.5 6.5 0 1 0-1.397 1.398h-.001c.03.04.062.078.098.115l3.85 3.85a1 1 0 0 0 1
            </svg>
        </button>
    </div>
</form>
```

Figura 2.6 – Codice HTML componente barra di ricerca per stringa

Ordina per recensioni più corte/lunghe

```
francescobonacci
def orderByLongReviews(reviews):
    return reviews.sortBy(lambda x: len(x[0]), False)

francescobonacci +1
def orderByShortReviews(reviews):
    return reviews.sortBy(lambda x: len(x[0]), True)
```

Figura 2.7 – Uso tecnologia Spark per ordinare le recensioni in base alla loro lunghezza

Le due *query* in figura 2.7 ordinano le recensioni in base alla loro lunghezza. In particolare, viene invocato sull'RDD contenente tutte le recensioni il metodo sortBy(), specificando che il criterio di ordinamento dovrà essere la lunghezza della recensione (ovvero il primo elemento, x[0]) di ciascuna istanza dell'RDD. Il diverso tipo di ordinamento, poi, viene definito grazie al secondo parametro passato al metodo sortBy(): nella prima *query* viene passato il booleano *False* per ordinare in senso decrescente, mentre nella seconda *query* viene passato il booleano *True* per ordinare in senso crescente.

Anche queste due query vengono poi richiamate all'interno del file "app.py", in particolare nelle funzioni che definiscono i due *controller* relativi ai due tipi di ordinamento definiti.

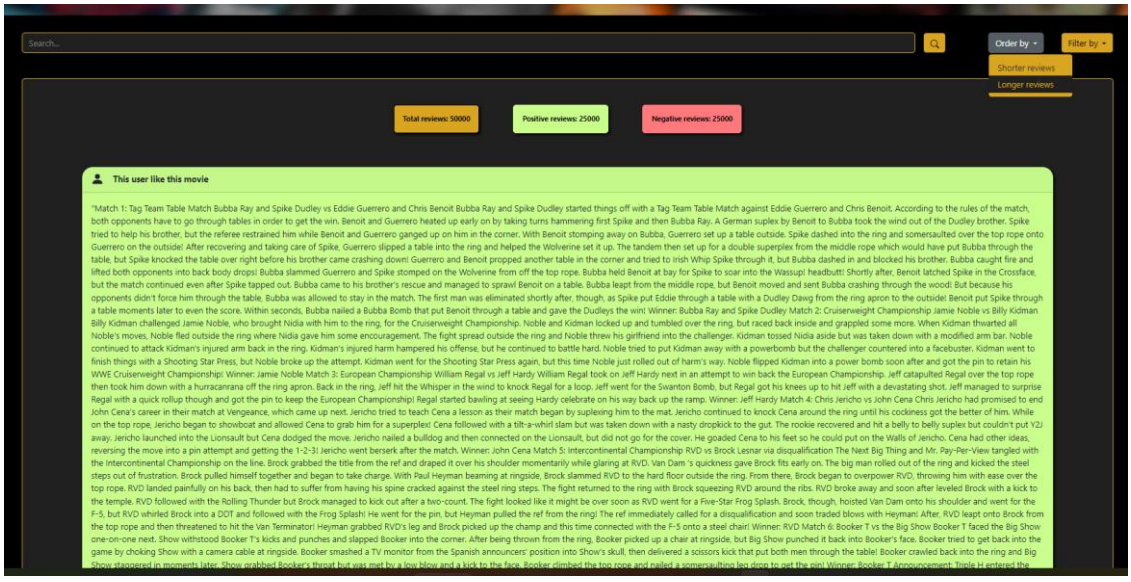


Figura 2.8.1 – Schermata recensioni ordinate per lunghezza decrescente

Figura 2.8.2 – Schermata recensioni ordinate per lunghezza decrescente

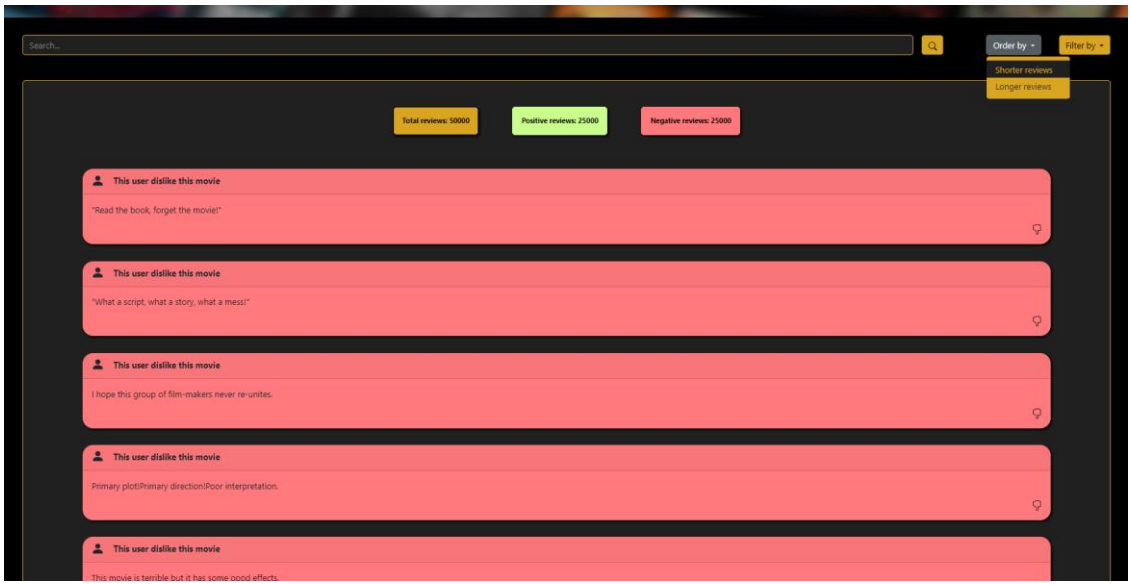


Figura 2.9 – Schermata recensioni ordinate per lunghezza crescente

Di seguito il codice HTML per i corrispondenti componenti di ordinamento.

```
<div class="dropdown dd-order">
    <button class="btn btn-secondary dropdown-toggle" type="button" data-bs-toggle="dropdown" aria-expanded="false">
        Order by
    </button>
    <ul class="dropdown-menu">
        <li><a class="dropdown-item {{'selected' if flags[4]}}" href="/shorter">Shorter reviews</a></li>
        <li><a class="dropdown-item {{'selected' if flags[5]}}" href="/longer">Longer reviews</a></li>
    </ul>
</div>
```

Figura 3.1 – Codice HTML componente menu a tendina per gli ordinamenti

Machine Learning con pyspak.ml (facoltativo)

```python
def predict_sentiment(review):
    global reviews

    regex_tokenizer = RegexTokenizer(inputCol="review", outputCol="words", pattern="\\W")
    reviews_df = reviews.toDF(namesOfColumns).cache()

    raw_words = regex_tokenizer.transform(reviews_df)

    remover = StopWordsRemover(inputCol="words", outputCol="filtered")
    words_df = remover.transform(raw_words)

    cv = CountVectorizer(inputCol="filtered", outputCol="features")
    model = cv.fit(words_df)
    countVectorizer_train = model.transform(words_df)
    countVectorizer_train = countVectorizer_train.withColumn("label", when(col('sentiment') == "positive", 1).otherwise(0))

    (reviews_train, reviews_test) = countVectorizer_train.randomSplit([0.7, 0.3], seed=0)

    nb = NaiveBayes(modelType="multinomial", labelCol="label", featuresCol="features")
    nbModel = nb.fit(reviews_train)
    nb_predictions = nbModel.transform(reviews_test)

    # tokenizzazione della recensione
    tokenized_review = regex_tokenizer.transform(spark.createDataFrame([(review,)], ["review"]))
    # rimozione stopwords
    filtered_review = remover.transform(tokenized_review)
    # creazione features usando il modello allenato a partire dal CountVectorizer
    features = model.transform(filtered_review)
    # predizione sentiment usando il modello NaiveBayes
    prediction = nbModel.transform(features).select("prediction").collect()[0]["prediction"]
    return prediction
```

Figura 3.2 – Uso libreria pyspark.ml ai fini della predizione del sentiment

Abbiamo deciso di arricchire il nostro progetto introducendo un algoritmo di apprendimento automatico ai fini della predizione del *sentiment* data una nuova recensione.

Per fare ciò abbiamo approfondito la documentazione pyspark.ml in particolare nel contesto del *Natural Language Processing* (NLP) e ci è venuto in aiuto il fatto che uno dei due membri del gruppo ha seguito il corso *Data Mining* all'UNICAL.

Alcune funzioni della libreria pyspark.ml richiedono in input un dataframe e non un *resilient distributed dataset* e per tale motivo per semplicità abbiamo scelto di utilizzare tale struttura.

Su tale *dataframe* inizialmente abbiamo eseguito un pre-processing dei dati al fine di allenare il modello su un dataset ottimale.

Innanzitutto, confermiamo che non ci siano valori nulli nel dataset con la seguente istruzione

```python
sum([reviews_df.filter(isnull(c)).count() for c in reviews_df.columns])
```

che appunto ritorna 0.

Verifichiamo inoltre che la distribuzione dei valori del sentiment sia più o meno equamente distribuita andando a contare il numero di occorrenze di recensioni positive e negative.

```
reviews_df.filter(reviews_df[1]=='positive').count()#25000
reviews_df.filter(reviews_df[1]=='negative').count()#25000
```

Abbiamo appena verificato che il dataset non presenta un problema di *class unbalance*.

Con l'utilizzo del **RegexTokenizer** della libreria pyspark.ml si dividono (tokenizzano) le parole delle recensioni e si rimuovono i caratteri speciali.

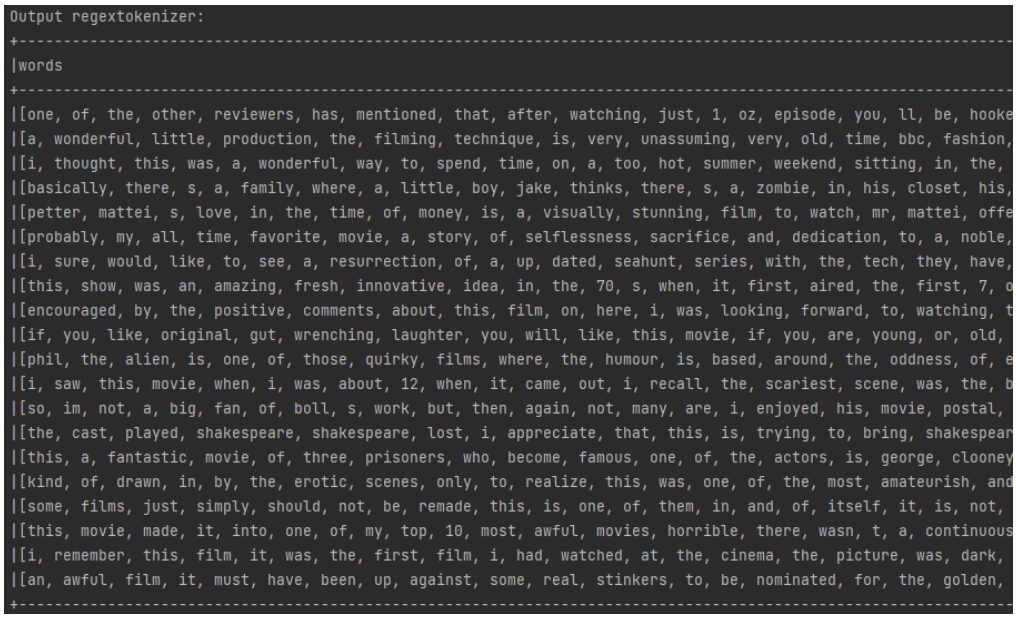


Figura 3.3 – Output regextokenizer

Con **StopWordsRemover** è un metodo alternativo usato per rimuovere le parole inglesi ritenute non importanti ai fini della predizione che ci fornisce la libreria pyspark.ml senza utilizzare il file "stopwords.txt".

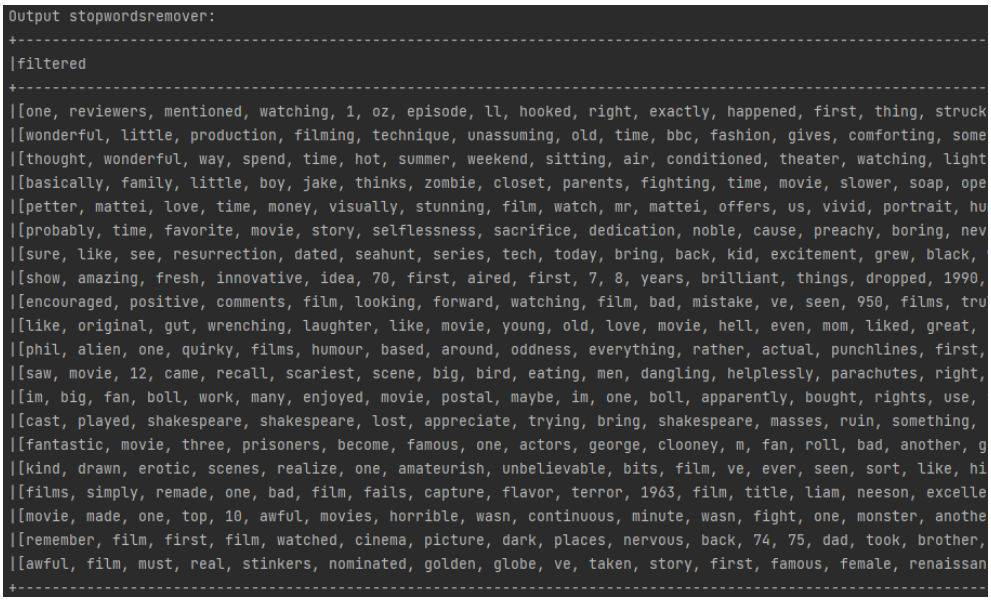


Figura 3.4 – Output stopwordsremover

CountVectorizer è molto importante ai fini dell'apprendimento in quanto un modello necessita di non lavorare con stringhe e per tale motivo trasforma le recensioni testuali in vettori di conteggio delle parole e infine sostituiamo la stringa 'positive' in 1 e la stringa 'negative' in 0.

```
Output countvectorizer:
+--------------------------------------------------------------------------------------------------------------------
|features
+--------------------------------------------------------------------------------------------------------------------
|(101359,[2,10,13,17,28,32,35,39,45,46,50,53,54,57,83,85,91,93,97,101,108,121,128,138,160,161,169,174,184,191,195,
|(101359,[2,5,8,9,10,15,33,38,46,54,56,63,75,80,91,98,118,146,165,171,176,192,234,244,250,260,282,295,300,301,330,
|(101359,[2,5,6,9,10,15,20,22,28,30,31,32,44,45,47,50,54,60,79,87,92,97,98,101,105,116,117,127,143,242,282,285,299
|(101359,[0,1,3,5,9,10,17,19,33,38,52,55,64,72,100,124,184,194,224,284,289,317,334,344,385,454,518,564,576,618,647
|(101359,[0,1,2,4,5,9,14,19,20,22,24,30,34,36,37,53,58,65,66,70,71,73,77,78,79,86,89,102,113,115,116,167,172,179,1
|(101359,[0,2,3,5,7,24,29,32,60,63,69,77,105,107,115,133,135,136,141,240,241,260,283,298,353,400,403,426,440,458,6
|(101359,[3,9,20,21,28,45,47,50,51,64,66,73,75,77,91,99,100,134,145,155,168,206,221,230,253,297,311,337,379,407,44
|(101359,[2,5,8,9,12,16,17,18,35,44,60,61,71,78,80,96,100,109,114,121,127,135,147,155,213,258,260,263,291,316,325,
|(101359,[1,2,9,12,20,26,29,34,36,42,47,53,54,75,86,107,114,120,121,122,130,136,144,149,151,181,195,231,241,260,26
|(101359,[0,3,6,15,30,63,87,109,315,465,1078,1355,2021,5214,5721],[2.0,2.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0
|(101359,[0,1,2,17,22,26,32,41,48,59,61,65,71,83,85,117,146,159,222,245,249,314,329,336,466,493,495,618,693,697,73
|(101359,[0,1,7,10,11,19,21,26,30,42,43,44,59,63,72,84,87,89,101,108,116,129,130,132,143,187,198,204,214,229,272,2
|(101359,[0,1,2,3,4,6,8,9,14,17,18,28,34,37,43,46,48,50,66,67,69,71,80,83,88,89,90,91,95,101,108,115,119,121,125,1
|(101359,[0,4,32,37,45,48,72,78,120,149,157,161,179,280,297,314,403,475,488,504,550,589,614,618,641,642,768,864,98
|(101359,[0,2,4,12,40,49,53,56,65,170,218,317,602,612,677,690,1145,1640,1749,3630,5528,7741,8386,15806,54286],[3.0
|(101359,[1,2,3,12,13,22,29,39,46,47,82,83,96,137,147,183,191,255,284,298,319,332,381,396,417,450,458,511,523,617,
|(101359,[1,2,7,11,12,25,26,45,78,101,110,121,126,180,200,209,211,223,247,306,314,531,614,825,837,874,886,1350,145
|(101359,[0,2,3,5,7,11,12,18,21,25,26,42,43,45,46,52,63,64,65,89,119,120,175,183,185,214,233,237,239,263,267,298,3
|(101359,[1,2,3,16,17,23,29,30,37,51,71,89,115,138,142,156,168,173,184,210,270,288,320,321,341,389,432,466,472,535
|(101359,[1,4,7,14,17,18,33,41,42,47,52,94,100,105,113,121,123,182,209,223,259,261,263,371,486,505,528,543,634,690
+--------------------------------------------------------------------------------------------------------------------
```

Figura 3.5 – Output countvectorizer

Il modello di apprendimento che abbiamo scelto di utilizzare è il classificatore Naive Bayes in quanto abbiamo un *dataset* comunque semplice (a parte il *sentiment* è composto da una sola features) ed evita lo svantaggio del modello che non considera la correlazione tra le *features*.

Quindi semplicità del *dataset* e semplicità del modello ci fanno ottenere una buona *performance*.

Ovviamente prima di addestrare il modello andiamo a suddividere il dataset *in training set* e *test set* con una corrispondente suddivisione pari al 70% e 30%.

Una volta allenato il modello andiamo ad effettuare le stesse operazioni di pre-processing sulla recensione passata dall'utente per poi andare a ritornare in *output* la predizione del *sentiment*.

Ora, andiamo a valutare quanto è performante il nostro modello Naive Bayes andando ad analizzare la sua accuratezza.

```
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
nb_accuracy = evaluator.evaluate(nb_predictions)
print("Accuratezza NaiveBayes = %g"% (nb_accuracy))
```

Figura 3.6 – Valutazione delle prestazioni del modello

Il valore di accuratezza ottenuto invocando il metodo evaluate() sull'oggetto evaluator creato è pari a 0,859548. Naturalmente, quella ottenuta è un'accuratezza sicuramente migliorabile, ad esempio andando a modificare gli "iperparametri" del modello oppure andando a fare un ulteriore analisi dei dati ed eventualmente fare altro pre-processing.

Inoltre andrebbero utilizzati anche altri criteri di valutazione delle performance e non solamente basarsi sull'*accuracy* del modello. Questo però va oltre lo scopo di questo progetto, in quanto tali tecniche di verranno approfondite in corsi più avanzati del nostro percorso di studi.

Sulla *web application* questa funzionalità è stata implementata utilizzando un componente detto "modale" in cui è presente un'area di testo nella quale l'utente scrive la propria recensione e un pulsante "Predict" che, una volta premuto, avvia il processo di allenamento del modello e di predizione del *sentiment* descritti in precedenza.

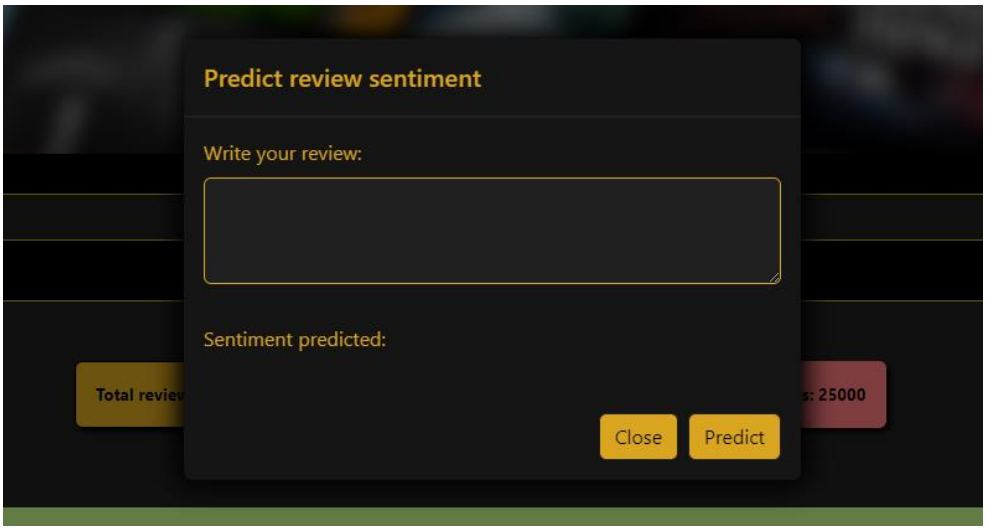


Figura 3.7 – Modale per la predizione

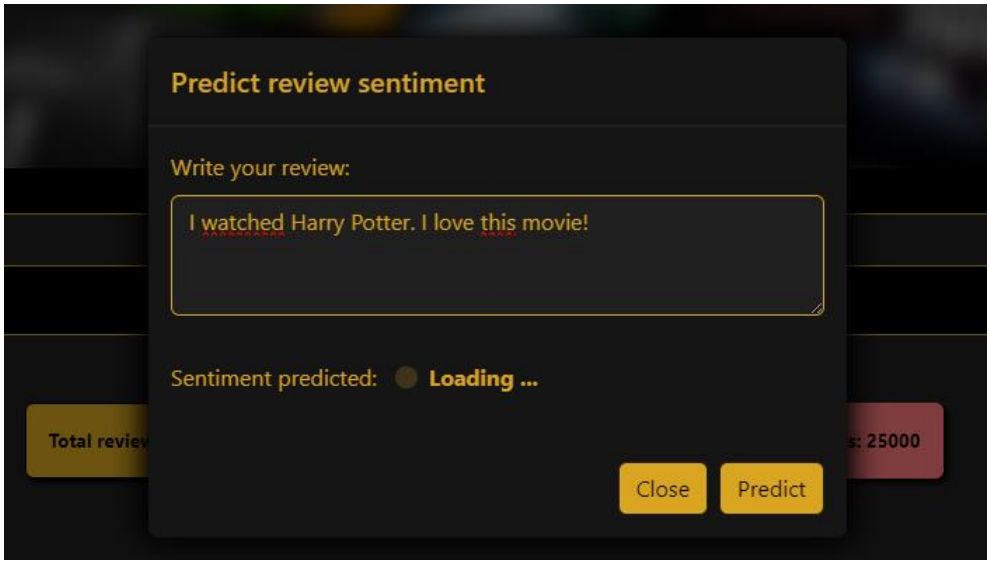


Figura 3.8 – Modale per la predizione in lavorazione

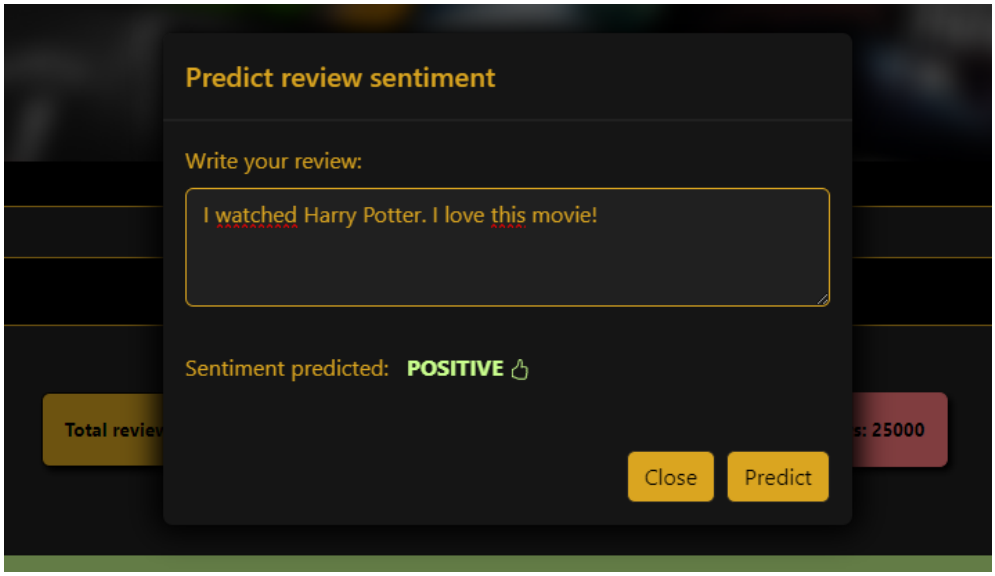


Figura 3.9 – Modale che restituisce come predizione il sentiment “POSITIVE”

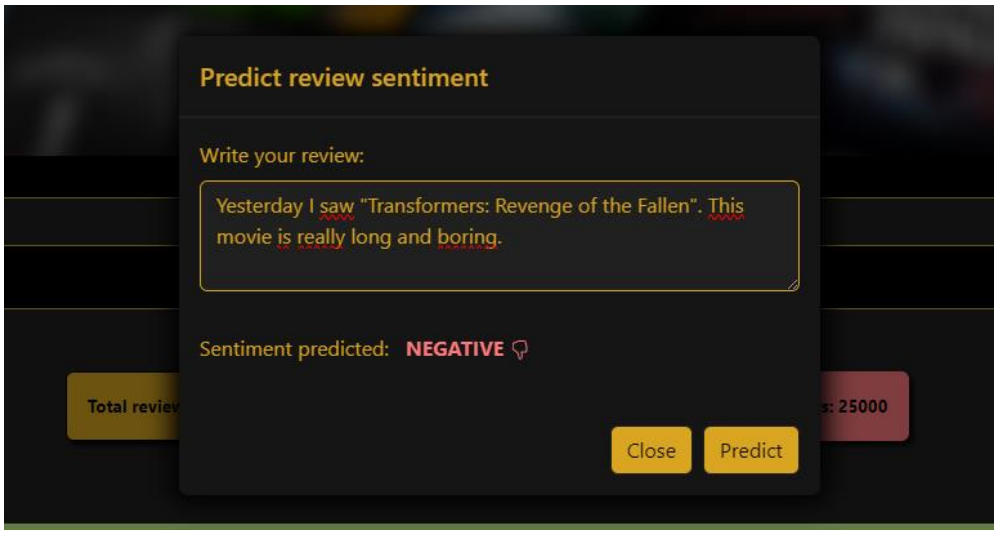


Figura 3.10 – Modale che restituisce come predizione il sentiment “NEGATIVE”

Di seguito il codice HTML per la realizzazione del componente “modale”.

```
<div class="modal fade" id="predictModal" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-labelledby="exampleModalLabel"
     aria-hidden="true">
    <div class="modal-dialog modal-dialog-centered">
        <div class="modal-content">
            <div class="modal-header">
                <h1 class="modal-title fs-5" id="exampleModalLabel"> Predict review sentiment </h1>
            </div>
            <div class="modal-body">
                <div class="mb-3">
                    <label for="userReviewTextarea" class="form-label"> Write your review: </label>
                    <textarea name="userReview" class="form-control" id="userReviewTextarea" rows="3"></textarea>
                </div>
                <div class="sentiment-result mb-3">
                    <span> Sentiment predicted: </span>

                    <span id="sentiment" class="fw-bold"></span>
                </div>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-dismiss="modal"> Close </button>
                <button type="button" onclick="sendReview()" class="btn btn-secondary"> Predict </button>
            </div>
        </div>
    </div>
</div>
```

Analisi esplorativa dei dati (EDA)

Parole più frequenti (tutte le recensioni)

Oltre ad interfacciarsi direttamente con il dataset, l'utente ha la possibilità di visualizzare varie statistiche riguardanti la totalità delle recensioni presenti nella *web application*.

Questo è stato fatto effettuando delle interrogazioni sul dataset utilizzando la tecnologia Spark e successivamente visualizzando i risultati su dei grafici (istogrammi) utilizzando una serie di librerie python dedicate all'esplorazione dei dati: la libreria "pandas" e la libreria "plotly.express". Il codice python per la realizzazione degli istogrammi è stato implementato prendendo spunto dal sito Kaggle (figura 4.1).

```
def showFrequentlyWordsHistogram(words_count):
    dict_words_count = dict(words_count.take(10))
    temp = pd.DataFrame(columns=["Common Words", 'Count'])
    temp["Common Words"] = list(dict_words_count.keys())
    temp["Count"] = list(dict_words_count.values())
    fig = px.bar(temp, x="Count", y="Common Words", title='Common Words in Reviews', orientation='h',
                 width=700, height=700, color='Common Words')
    fig.show()
```

Figura 4.1 – Uso librerie *pandas* e *plotly.express* per la creazione dell'istogramma

Il metodo in figura 4.1 riceve l'RDD che viene ottenuto dalla *query* da noi realizzata usando la tecnologia Spark e mostrata di seguito (figura 4.2).

```
def mostFrequentlyWords(reviews):
    countsRDD = reviews.flatMap(lambda x: splitAndProcess(x[0])).map(lambda word:(word, 1)).reduceByKey(lambda a,b: a+b)
    orderedRDD = countsRDD.sortBy(lambda x: x[1], False)
    return orderedRDD.filter(lambda x: x[0] not in stopwords)
```

Figura 4.2 – Uso tecnologia Spark per recuperare le parole più frequenti in tutte le recensioni

Questa funzione riprende la metodologia attuata dal metodo Word Count visto a lezione. Di fatti, con la prima istruzione invochiamo il metodo flatMap() sull'RDD contenente tutte le recensioni, applicando ad ogni istanza la funzione di pre-processing descritta precedentemente splitAndProcess(), la quale divide una recensione nelle parole che la compongono, ognuna ripulita dai caratteri speciali. Il risultato ottenuto è un RDD contenente l'insieme delle parole che sono presenti in tutte le recensioni del dataset. Su tale RDD viene poi invocata la funzione map() che crea le coppie (parola, 1), su cui viene infine richiamata la funzione reduceByKey() che restituirà le coppie (parola, occorrenze).

Ottenuto l'RDD delle occorrenze *countsRDD*, su di esso viene invocata la funzione sortBy() che ordinerà le istanze in base al numero di occorrenze di ogni parola, in ordine

decrescente (come si può vedere dal booleano *False*). Per concludere, il risultato viene filtrato dalle parole non significative (*stopwords*).

Il risultato dell'invocazione della funzione showFrequentlyWordsHistogram() (figura 4.1) è mostrato a seguire (figura 4.3).

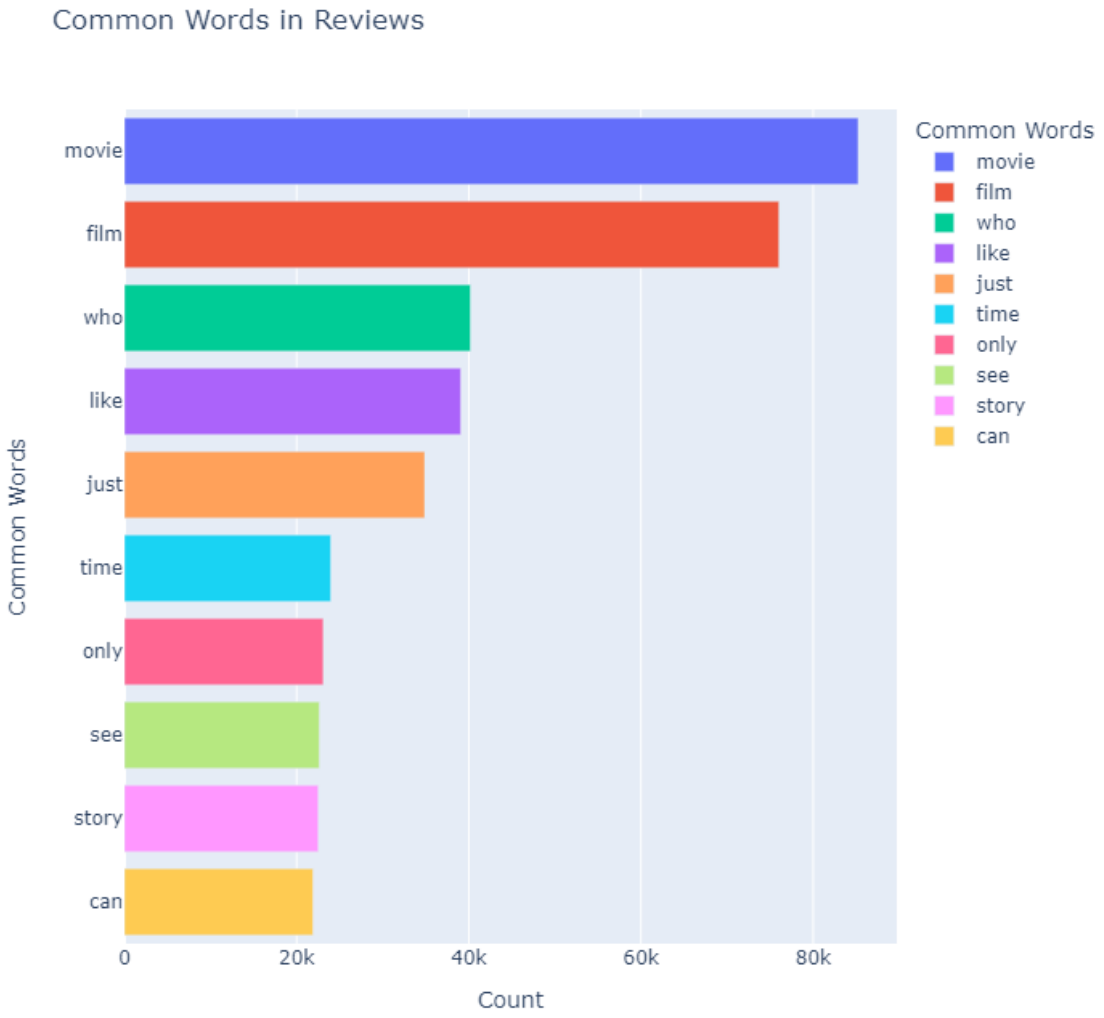


Figura 4.3 – Istogramma delle parole più frequenti in tutte le recensioni

Parole più frequenti (recensioni positive / negative)

Di seguito sono riportate le due *query* che restituiscono le parole più presenti nelle recensioni positive e in quelle negative rispettivamente (figura 4.4). Da notare che in entrambe viene richiamata la funzione mostFrequentlyWord() descritta in precedenza, passando come parametro l'RDD contenente tutte le recensioni dopo averlo filtrato rispettivamente per recensioni positive e per recensioni negative.

```
def mostFrequentlyPositiveWords(reviews):
    positiveReviews = filterByPositive(reviews)
    return mostFrequentlyWords(positiveReviews)

 marcogreco *
def mostFrequentlyNegativeWords(reviews):
    negativeReviews = filterByNegative(reviews)
    return mostFrequentlyWords(negativeReviews)
```

Figura 4.4 – Uso tecnologia Spark per recuperare le parole più frequenti (recensioni positive/negative)

Il risultato dell'invocazione della prima funzione è mostrato a seguire (figura 4.5).

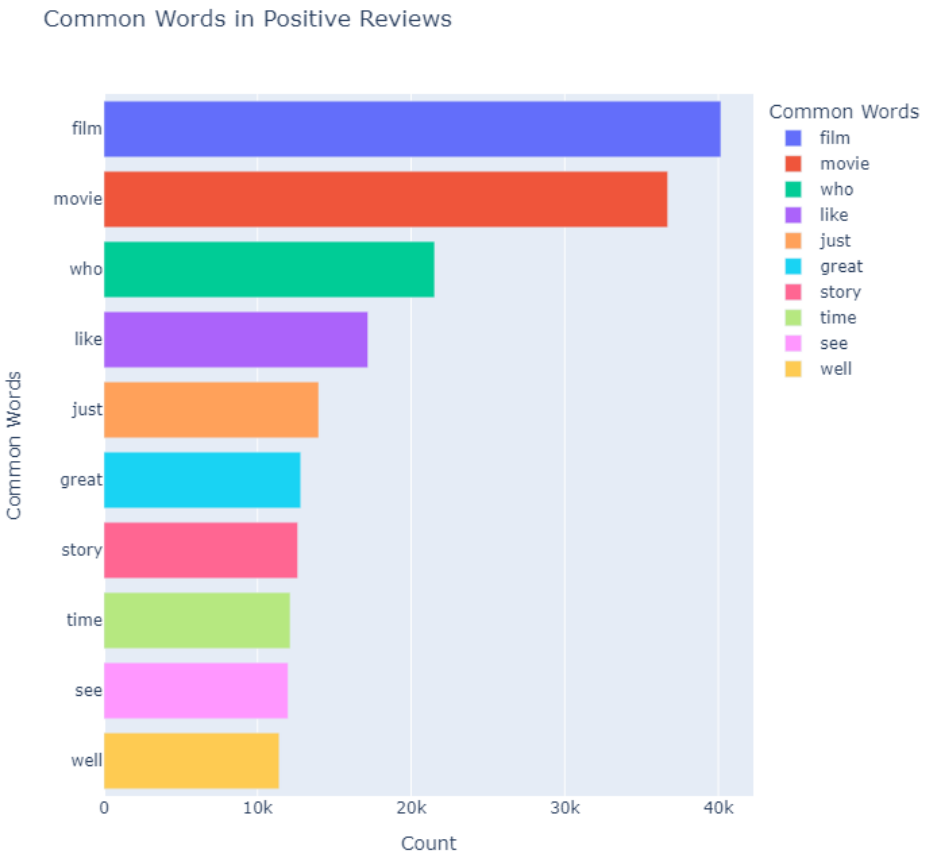


Figura 4.5 – Istogramma delle parole più frequenti nelle recensioni positive

Il risultato dell'invocazione della seconda funzione è mostrato di seguito (figura 4.6).

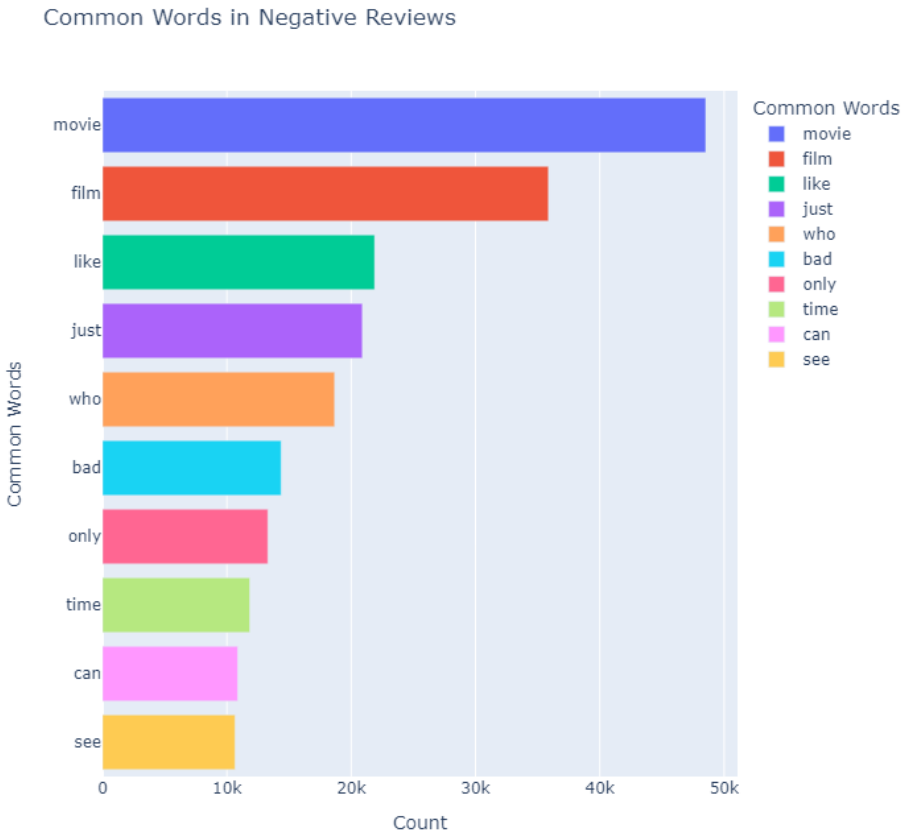


Figura 4.6 – Istogramma delle parole più frequenti nelle recensioni negative

Controller Flask (file app.py)

In questa sezione verranno mostrati i vari controller che abbiamo implementato per realizzare il *backend* della nostra *web application*, con i quali andranno ad interfacciarsi le pagine web realizzate nel *frontend* utilizzando l'*app.route()* definito in ogni controller.

Il file "app.py" che contiene i controller comunica con il file "main.py", ovvero il file in cui abbiamo configurato l'ambiente Spark e realizzato le nostre query. Tutti i controller invocano una o più interrogazioni definite nel file "main.py" per effettuare i filtraggi e gli ordinamenti descritti nelle pagine precedenti, in maniera distribuita.

```python
@app.route('/')
def index():
    global currentReviews, filteredByPositive, filteredByNegative, orderedByShorter,\
        orderedByLonger, filteredWithoutSpoilers, filteredWithSpoilers, searchString
    filteredByPositive = False
    filteredByNegative = False
    orderedByShorter = False
    orderedByLonger = False
    filteredWithoutSpoilers = False
    filteredWithSpoilers = False
    searchString = ""
    currentReviews = main.reviews
    total_pages = (currentReviews.count() + 6 - 1) // 6
    paginated_reviews = paginated_data(currentReviews.collect(), 1, 6)
    return render_template('index.html', reviews=paginated_reviews, flags=getFlags(), page=1, total_pages=total_pages)
```

Figura 5.1 – Controller relativo alla *home page*

```python
@app.route('/positive')
def getPositive():
    global currentReviews, filteredByPositive, filteredByNegative, orderedByShorter,\
        orderedByLonger, filteredWithoutSpoilers, filteredWithSpoilers, searchString
    if (not filteredByPositive):
        currentReviews = main.filterByPositive(main.reviews)
    else:
        currentReviews = main.reviews
    filteredByPositive = not filteredByPositive
    filteredByNegative = False
    # controlliamo se era stato effettuato l'ordinamento precedentemente
    if (orderedByShorter):
        currentReviews = main.orderByShortReviews(currentReviews)
    elif (orderedByLonger):
        currentReviews = main.orderByLongReviews(currentReviews)
    # controlliamo se era stato effettuato il filtro degli spoilers precedentemente
    if (filteredWithoutSpoilers):
        currentReviews = main.filterByNoSpoilers(currentReviews)
    elif (filteredWithSpoilers):
        currentReviews = main.filterBySpoilers(currentReviews)
    # controlliamo se era stato effettuata la ricerca per una determinata parola/frase
    if (searchString != ""):
        currentReviews = main.filterByWord(currentReviews, searchString)
    total_pages = (currentReviews.count() + 6 - 1) // 6
    paginated_reviews = paginated_data(currentReviews.collect(), 1, 6)
    return render_template('index.html', reviews=paginated_reviews, flags=getFlags(), page=1, total_pages=total_pages)
```

Figura 5.2 – Controller relativo al filtraggio per recensioni positive

```python
@app.route('/negative')
def getNegative():
    global currentReviews, filteredByNegative, filteredByPositive, orderedByShorter,\
        orderedByLonger, filteredWithoutSpoilers, filteredWithSpoilers, searchString
    if (not filteredByNegative):
        currentReviews = main.filterByNegative(main.reviews)
    else:
        currentReviews = main.reviews
    filteredByNegative = not filteredByNegative
    filteredByPositive = False
    # controlliamo se era stato effettuato l'ordinamento precedentemente
    if (orderedByShorter):
        currentReviews = main.orderByShortReviews(currentReviews)
    elif (orderedByLonger):
        currentReviews = main.orderByLongReviews(currentReviews)
    # controlliamo se era stato effettuato il filtro degli spoilers precedentemente
    if (filteredWithoutSpoilers):
        currentReviews = main.filterByNoSpoilers(currentReviews)
    elif (filteredWithSpoilers):
        currentReviews = main.filterBySpoilers(currentReviews)
    # controlliamo se era stato effettuata la ricerca per una determinata parola/frase
    if (searchString != ""):
        currentReviews = main.filterByWord(currentReviews, searchString)
    total_pages = (currentReviews.count() + 6 - 1) // 6
    paginated_reviews = paginated_data(currentReviews.collect(), 1, 6)
    return render_template('index.html', reviews=paginated_reviews, flags=getFlags(), page=1, total_pages=total_pages)
```

Figura 5.3 – Controller relativo al filtraggio per recensioni negative

```python
@app.route('/withSpoilers')
def getReviewsWithSpoilers():
    global currentReviews, filteredByPositive, filteredByNegative, orderedByShorter,\
        orderedByLonger, filteredWithoutSpoilers, filteredWithSpoilers, searchString
    if (not filteredWithSpoilers):
        currentReviews = main.filterBySpoilers(main.reviews)
    else:
        currentReviews = main.reviews
    filteredWithSpoilers = not filteredWithSpoilers
    filteredWithoutSpoilers = False
    # controlliamo se era stato effettuato l'ordinamento precedentemente
    if (orderedByShorter):
        currentReviews = main.orderByShortReviews(currentReviews)
    elif (orderedByLonger):
        currentReviews = main.orderByLongReviews(currentReviews)
    # controlliamo se era stato effettuato il filtro delle positive/negative precedentemente
    if (filteredByPositive):
        currentReviews = main.filterByPositive(currentReviews)
    elif (filteredByNegative):
        currentReviews = main.filterByNegative(currentReviews)
    # controlliamo se era stato effettuata la ricerca per una determinata parola/frase
    if (searchString != ""):
        currentReviews = main.filterByWord(currentReviews, searchString)
    total_pages = (currentReviews.count() + 6 - 1) // 6
    paginated_reviews = paginated_data(currentReviews.collect(), 1, 6)
    return render_template('index.html', reviews=paginated_reviews, flags=getFlags(), page=1, total_pages=total_pages)
```

Figura 5.4 – Controller relativo al filtraggio per recensioni con spoiler

```python
@app.route('/withoutSpoilers')
def getReviewsWithoutSpoilers():
    global currentReviews, filteredByPositive, filteredByNegative, orderedByShorter,\
        orderedByLonger, filteredWithoutSpoilers, filteredWithSpoilers, searchString
    if (not filteredWithoutSpoilers):
        currentReviews = main.filterByNoSpoilers(main.reviews)
    else:
        currentReviews = main.reviews
    filteredWithoutSpoilers = not filteredWithoutSpoilers
    filteredWithSpoilers = False
    # controlliamo se era stato effettuato l'ordinamento precedentemente
    if (orderedByShorter):
        currentReviews = main.orderByShortReviews(currentReviews)
    elif (orderedByLonger):
        currentReviews = main.orderByLongReviews(currentReviews)
    # controlliamo se era stato effettuato il filtro delle positive/negative precedentemente
    if (filteredByPositive):
        currentReviews = main.filterByPositive(currentReviews)
    elif (filteredByNegative):
        currentReviews = main.filterByNegative(currentReviews)
    # controlliamo se era stato effettuata la ricerca per una determinata parola/frase
    if (searchString != ""):
        currentReviews = main.filterByWord(currentReviews, searchString)
    total_pages = (currentReviews.count() + 6 - 1) // 6
    paginated_reviews = paginated_data(currentReviews.collect(), 1, 6)
    return render_template('index.html', reviews=paginated_reviews, flags=getFlags(), page=1, total_pages=total_pages)
```

Figura 5.5 – Controller relativo al filtraggio per recensioni senza spoiler

```python
@app.route('/search')
def search():
    global currentReviews, filteredByPositive, filteredByNegative, orderedByShorter,\
        orderedByLonger, filteredWithoutSpoilers, filteredWithSpoilers, searchString
    searchString = request.args.get('searchString')
    currentReviews = main.filterByWord(currentReviews, searchString)
    total_pages = (currentReviews.count() + 6 - 1) // 6
    paginated_reviews = paginated_data(currentReviews.collect(), 1, 6)
    return render_template('index.html', reviews=paginated_reviews, flags=getFlags(), page=1, total_pages=total_pages)
```

Figura 5.6 – Controller relativo al filtraggio per ricerca di una stringa nelle recensioni

```python
@app.route('/cancelResearch')
def cancelResearch():
    global currentReviews, filteredByPositive, filteredByNegative, orderedByShorter,\
        orderedByLonger, filteredWithoutSpoilers, filteredWithSpoilers, searchString
    searchString = ""
    currentReviews = main.reviews
    # controlliamo tutti i filtri e gli ordinamenti
    if (filteredByPositive): currentReviews = main.filterByPositive(currentReviews)
    elif (filteredByNegative): currentReviews = main.filterByNegative(currentReviews)
    if (filteredWithoutSpoilers): currentReviews = main.filterByNoSpoilers(currentReviews)
    elif (filteredWithSpoilers): currentReviews = main.filterBySpoilers(currentReviews)
    if (orderedByShorter): currentReviews = main.orderByShortReviews(currentReviews)
    elif (orderedByLonger): currentReviews = main.orderByLongReviews(currentReviews)
    total_pages = (currentReviews.count() + 6 - 1) // 6
    paginated_reviews = paginated_data(currentReviews.collect(), 1, 6)
    return render_template('index.html', reviews=paginated_reviews, flags=getFlags(), page=1, total_pages=total_pages)
```

Figura 5.7 – Controller relativo all’annullamento della ricerca per una determinata stringa

```python
@app.route('/longer')
def orderByLongerReviews():
    global currentReviews, orderedByLonger, orderedByShorter, filteredByPositive,\
        filteredByNegative, filteredWithoutSpoilers, filteredWithSpoilers, searchString
    if (not orderedByLonger):
        currentReviews = main.orderByLongReviews(currentReviews)
    else:
        currentReviews = main.reviews
        # applichiamo gli eventuali filtri che c'erano prima
        if (filteredByPositive): currentReviews = main.filterByPositive(currentReviews)
        elif (filteredByNegative): currentReviews = main.filterByNegative(currentReviews)
        if (filteredWithoutSpoilers): currentReviews = main.filterByNoSpoilers(currentReviews)
        elif (filteredWithSpoilers): currentReviews = main.filterBySpoilers(currentReviews)
        if (searchString != ""): currentReviews = main.filterByWord(currentReviews, searchString)
    orderedByLonger = not orderedByLonger
    orderedByShorter = False
    total_pages = (currentReviews.count() + 6 - 1) // 6
    paginated_reviews = paginated_data(currentReviews.collect(), 1, 6)
    return render_template('index.html', reviews=paginated_reviews, flags=getFlags(), page=1, total_pages=total_pages)
```

Figura 5.8 – Controller relativo all'ordinamento in senso decrescente delle recensioni

```python
@app.route('/shorter')
def orderByShorterReviews():
    global currentReviews, orderedByShorter, orderedByLonger, filteredByPositive,\
        filteredByNegative, filteredWithoutSpoilers, filteredWithSpoilers, searchString
    if (not orderedByShorter):
        currentReviews = main.orderByShortReviews(currentReviews)
    else:
        currentReviews = main.reviews
        # applichiamo gli eventuali filtri che c'erano prima
        if (filteredByPositive): currentReviews = main.filterByPositive(currentReviews)
        elif (filteredByNegative): currentReviews = main.filterByNegative(currentReviews)
        if (filteredWithoutSpoilers): currentReviews = main.filterByNoSpoilers(currentReviews)
        elif (filteredWithSpoilers): currentReviews = main.filterBySpoilers(currentReviews)
        if (searchString != ""): currentReviews = main.filterByWord(currentReviews, searchString)
    orderedByShorter = not orderedByShorter
    orderedByLonger = False
    total_pages = (currentReviews.count() + 6 - 1) // 6
    paginated_reviews = paginated_data(currentReviews.collect(), 1, 6)
    return render_template('index.html', reviews=paginated_reviews, flags=getFlags(), page=1, total_pages=total_pages)
```

Figura 5.9 – Controller relativo all'ordinamento in senso crescente delle recensioni

```python
@app.route('/predict')
def predictSentiment():
    userReview = request.args.get('userReview')
    sentiment = main.predict_sentiment(userReview)
    return str(sentiment)
```

Figura 5.10 – Controller relativo alla predizione del sentiment di una recensione scritta dall'utente

```python
def paginated_data(data, page, per_page):
    start = (page - 1) * per_page
    end = start + per_page
    paginated_data = data[start:end]
    return paginated_data
```

Figura 5.11 – Metodo di ausilio per realizzare la paginazione nella *web application*