INTRODUCTION

Movie reviews are an important way to gauge the performance of a movie. While providing a numerical/stars rating to a movie tells us about the success or failure of a movie quantitatively, a collection of movie reviews is what gives us a deeper qualitative insight on different aspects of the movie. A textual movie review tells us about the strong and weak points of the movie and deeper analysis of a movie review can tell us if the movie in general meets the expectations of the reviewer.

Sentiment analysis refers to the use of natural language processing, text analysis and computational linguistics to extract and identify subjective information in source materials. It aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document. The attitude can be,

- His or her judgment or evaluation.
- Affective state (that is to say, the emotional state of the author when writing).
- The intended emotional communication (that is to say, the emotional effect the author wishes to have on the reader).

In last decade there is a rise of social media such as blogs and social networks, which has fueled the interest in sentiment analysis. Online opinion has turned into a kind of virtual currency with the proliferation of reviews, ratings, recommendations and other forms of online expression, for businesses that are looking to market their products, identify new opportunities and manage their reputations. In order to automate the process of filtering out the noise, understanding the conversations, identifying the relevant content and following appropriate actions, many are now looking to the field of sentiment analysis.

The problem of most sentiment analysis algorithms is that they use simple terms to express sentiment about a product or service. However, cultural factors, sentence negation, sarcasm, terseness, language ambiguity and differing contexts make it extremely difficult to turn a string of written text into a simple pro or con sentiment.

A fundamental task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature/aspect. It focuses on whether the expressed opinion in a document, a sentence or a feature/aspect is positive, negative, or neutral. Sometimes it goes beyond polarity and looks at emotional states such as "angry," "sad," and "happy.".

Another task in sentiment analysis is subjectivity/objectivity identification where it focuses on classifying a given text (usually a sentence) into one of the two classes (objective or subjective). As the subjectivity of words and phrases may depend on their context and an objective document may contain subjective sentences (a news article quoting people's opinions), this problem can sometimes be more difficult than polarity classification.

PROBLEM STATEMENT

The main objective is to predict the sentiment for several movie reviews obtained from the Internet Movie Database (IMDb). In essence, it is the process of determining the emotional tone behind a series of words, used to gain an understanding of the attitudes, opinions and emotions expressed within an online mention.

AIM

Sentiment Analysis is a major subject is machine learning which aims to extract subjective information from the textual reviews. The field of sentiment of analysis is closely tied to natural language processing and text mining. It can be used to determine the attitude of the reviewer with respect to various topics or the overall polarity of review. Using sentiment analysis, we can find the state of mind of the reviewer while providing the review and understand if the person was “happy”, “sad”, “angry” and so on. In this project we aim to use Sentiment Analysis on a set of movie reviews given by reviewers and try to understand what their overall reaction to the movie was, i.e., if they liked the movie or hated it. We aim to utilize the relationships of the words in the review to predict the overall polarity of the review.

In essence, it is the process of determining the emotional tone behind a series of words, used to gain an understanding of the attitudes, opinions and emotions expressed within an online mention.

APPLICATIONS

Let's look at the most popular applications of sentiment analysis in real life:

1. Social media monitoring
2. Customer support
3. Brand monitoring and reputation management
4. Customer feedback
5. Voice of customer (VoC)
6. Voice of employee
7. Product analysis
8. Market research and competitive research

1.Social media monitoring: Sentiment analysis of social data will keep an eye on customer opinion 24/7 and in real time. You'll be able to quickly respond when something negative starts circulating and boost your image when you receive positive mentions. And, you'll get regular, dependable insights about your customers, which you can use to monitor your progress from one quarter to the next.

2. Customer support: Customer support management presents many challenges due to the sheer number of requests, varied topics, and diverse branches within a company – not to mention the urgency of any given request. Sentiment analysis with natural language understanding (NLU) reads regular human language for meaning, emotion, tone, and more, to understand customer requests, just as a person would. You can automatically process customer support tickets, online chats, phone calls, and emails by sentiment, which might also indicate urgency, and route to the appropriate team.

3. Brand monitoring and reputation management: Brand monitoring is one of the most popular applications of sentiment analysis in business. Bad reviews can snowball online, and the longer you leave them the worse the situation will be. With Sentiment analysis tools, you will be notified about negative brand mentions immediately. Not only that, but you can also keep track of your brand's image and reputation over time or at any given moment, so you can monitor your progress. Whether monitoring news stories, blogs, forums, and social media for information about your brand, you can transform this data into usable information and statistics. You can also trust machine learning to follow trends and anticipate outcomes, to stay ahead and go from reactive to proactive.

4. Customer feedback: Sentiment analysis can also be used to gain insights from the troves of customer feedback available (online reviews, social media, surveys) and save hundreds of employee hours. Sentiment analysis can read beyond simple definition to detect sarcasm, read common chat acronyms (lol, rofl, etc.), and correct for common mistakes like misused and misspelled words.

Comment 1

"Love the user interface. Setup took five minutes, and we were ready to go."

Comment 2
"Took me 2 hours to set up, then I find out I have to update my OS. Love it!"

Sentiment analysis would classify the second comment as negative, even though they both use words that, without context, would be considered positive.

Keeping track of customer comments allows you to engage with individual customers in real time. And you can target read for new products or specific user issues.

5. Voice of customer (VoC): Combine and evaluate all of your customer feedback: from the web, customer surveys, chats, call centers, and emails. Sentiment analysis allows you to categorize and structure this data to identify patterns and discover recurring topics and concerns. You can understand your customer base collectively, then segment them to target directly. For example, using data from a customer survey, you might want to offer free services or promotions to entice unhappy customers. Or offer rewards to those that are extremely happy with your company, encouraging them to spread the word about your product or service. Listening to the voice of your customers and learning how to communicate with them – what works and what doesn't – will help you create a personalized customer experience.

6. Voice of employee: Engage your employees, reduce turnover, and increase productivity. Use AI to evaluate employee surveys or analyze Glassdoor reviews, emails, Slack messages, and more (without feeling like Big Brother). Sentiment analysis software allows you to analyze employee opinions subjectively, with no human input. Process unstructured data to go beyond *who* and *what* to uncover the *why*. Create analysis models for your specific needs. You'll discover the most common topics and concerns to keep your employees happy and productive.

7. Product analysis: Find out what the public is saying about a new product right after launch or analyze years of feedback you may have never seen. You can search keywords for a particular product feature (interface, UX, functionality) and train sentiment analysis models to find only the information you need. Discover how a product is perceived by your target audience? Which elements of your product need to be improved? Sentiment analysis provides better results than humans because AI doesn't alter its results and it's not subjective.

8. Market research and competitive research: Another use case of sentiment analysis is market and competitor research. Find out who's trending among your competitors and how your marketing efforts compare. Get a comprehensive view from the ground, from every aspect of your and your competition's customer base. Analyze your competitor's content to find out what works with the public that you may not have considered. You'll understand your strengths and weaknesses and how they relate to that of your competitors.

System Requirements

- Central Processing Unit (CPU) — Intel Core i5 6th Generation processor or higher. An AMD equivalent processor will also be optimal.
- Likely provision at least 8-16 cores per machine.
- RAM — 8 GB minimum, 16 GB or higher is recommended.
- Graphics Processing Unit (GPU) — GPU support requires a CUDA®-enabled card (Ubuntu and Windows).
- Operating System — ○ Ubuntu 16.04 or later (64-bit) ○ MacOS 10.12.6 (Sierra) or later (64-bit) (no GPU support) ○ Windows 7 or later (64-bit).
- Jupyter/ipython version 4.0 or later, with notebook support.
- Pip 19.0 or later (requires manylinux2010 support).
- Pandas version 0.23.2 or above.
- NumPy version 1.7 or above.
- Pyarrow version1.0.0 or later.
- Py4J version 0.10.9 or later
- Beautiful soup version 4.5.6 or above.
- NLTK version 3.5 or above.
- Pyspark version 3.1.2.
- Hadoop version 3.2.

Terminologies:

1. Text corpus consists of multiple text documents and each document can be as simple as a single sentence to a complete document with multiple paragraphs.

2. Sentiment analysis is also popularly known as opinion analysis or opinion mining. The key idea is to use techniques from text analytics, NLP, Machine Learning, and linguistics to extract important information or data points from unstructured text. This in turn can help us derive qualitative outputs like the overall sentiment being on a positive, neutral, or negative scale and quantitative outputs like the sentiment polarity, subjectivity, and objectivity proportions.

3. Sentiment polarity is typically a numeric score that's assigned to both the positive and negative aspects of a text document based on subjective parameters like specific words and phrases expressing feelings and emotion. Neutral sentiment typically has 0 polarity since it does not express and specific sentiment, positive sentiment will have polarity > 0, and negative < 0. Here, we will go through a step-by-step process to make a model which would perform Sentiment Analysis on a large Movie database. The data set is taken from Internet Movie Database (IMDb). We focus on trying to analyze a large corpus of movie reviews and derive the sentiment. We will perform our analysis using the Unsupervised lexicon-based model and the traditional supervised ML model.

4. Features are individual independent variables that act like a input in your system. Actually, while making the predictions, models use such features to make the predictions. And using the feature engineering process, new features can also be obtained from old features in machine learning.

5. Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes. In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).

Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

6. Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words "chocolates", "chocolatey", "choco" to the root word, "chocolate" and "retrieval", "retrieved", "retrieves" reduce to the stem "retrieve".

TOOLS AND TECHNOLOGY USED

The tools and frameworks used to set up the entire project are as follows:

1.Hadoop: The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

2.Apache spark: Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

3.Pyspark: PySpark is an interface for Apache Spark in Python. It not only allows you to write Spark applications using Python APIs, but also provides the PySpark shell for interactively analyzing your data in a distributed environment. PySpark supports most of Spark's features such as Spark SQL, DataFrame, Streaming, MLlib (Machine Learning) and Spark Core.

4.Spark SQL and DataFrame: Spark SQL is a Spark module for structured data processing. It provides a programming abstraction called DataFrame and can also act as distributed SQL query engine.

5.Streaming: Running on top of Spark, the streaming feature in Apache Spark enables powerful interactive and analytical applications across both streaming and historical data, while inheriting Spark's ease of use and fault tolerance characteristics.

6.MLlib: Built on top of Spark, MLlib is a scalable machine learning library that provides a uniform set of high-level APIs that help users create and tune practical machine learning pipelines.

7.Spark Core: Spark Core is the underlying general execution engine for the Spark platform that all other functionality is built on top of. It provides an RDD (Resilient Distributed Dataset) and in-memory computing capabilities.

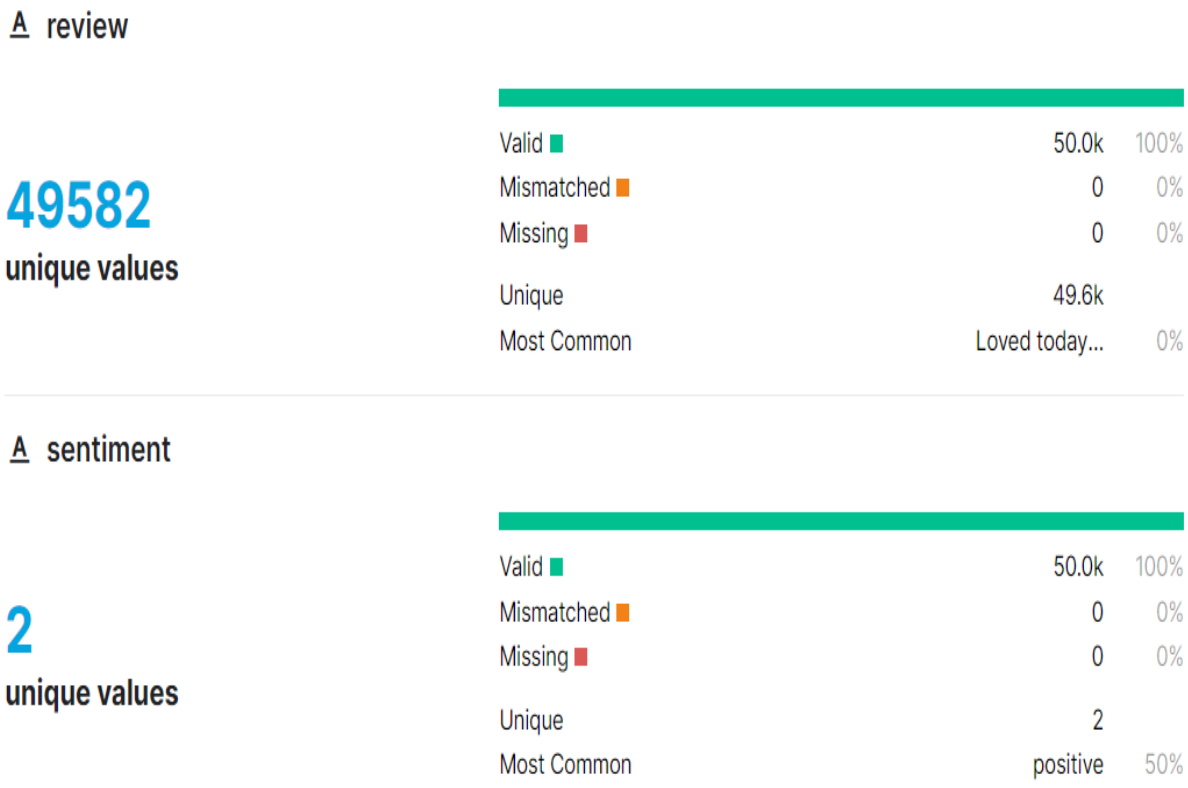
8.NLTK: NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

9.Beautiful Soup: It is a Python package for parsing HTML and XML documents (including having malformed markup, i.e. non-closed tags, so named after tag soup). It creates a parse tree for parsed pages that can be used to extract data from HTML,[2] which is useful for web scraping.

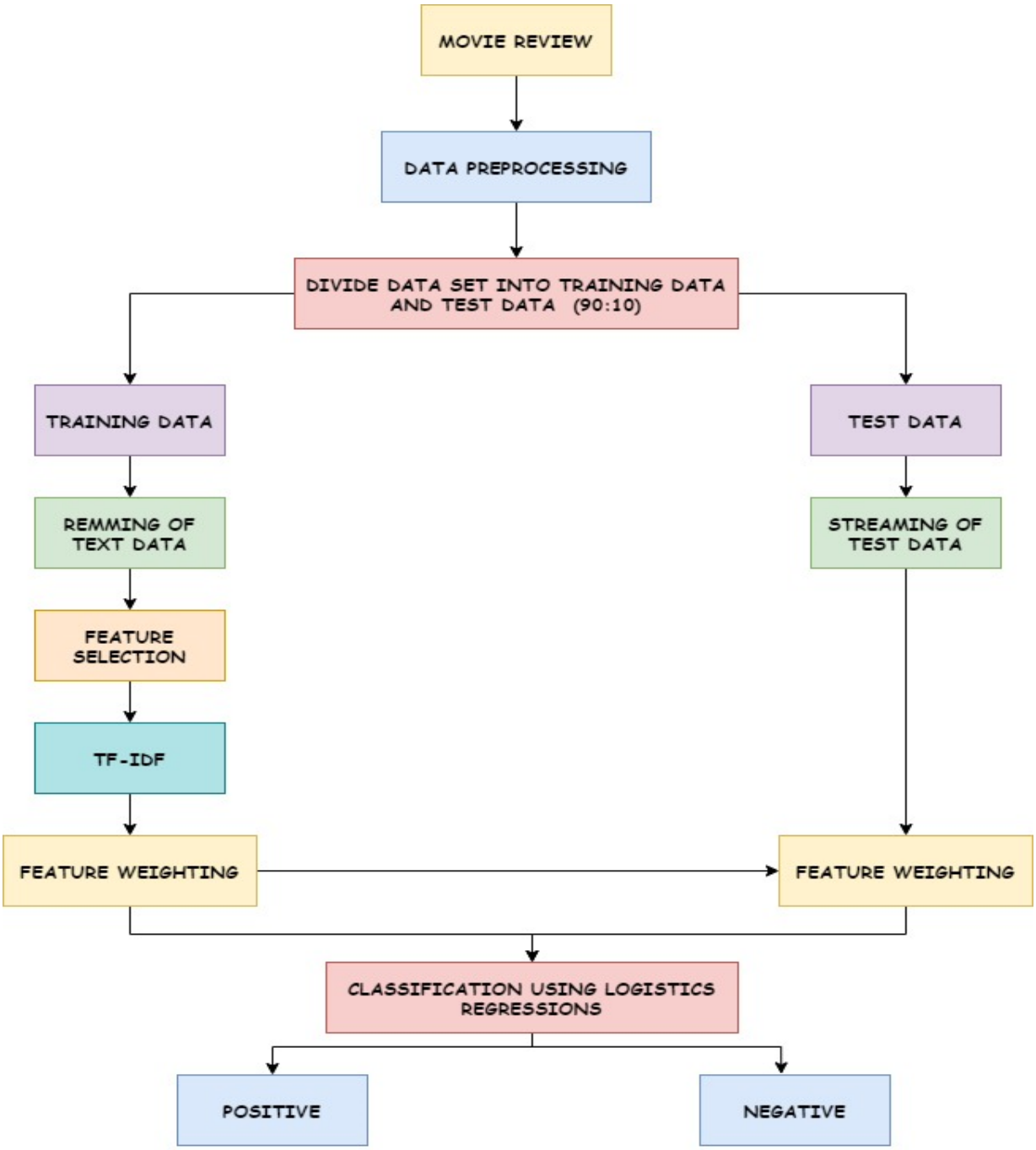
System Implementation

Dataset

IMDB dataset having 50K movie reviews for natural language processing or Text analytics. This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training and 25,000 for testing. So, predict the number of positive and negative reviews using either classification or deep learning algorithms. For more dataset information, please go through the following link, http://ai.stanford.edu/~amaas/data/sentiment/



System Architecture



ARCHITECTURE FOR SENTIMENT CLASSIFICATION

Data Preprocessing

In [3]:
```
#Summary of the dataset
imdb_data.describe()
```

Out[3]:

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

The review field in the file contains data like HTML tags, garbage values, and other symbols. This data affects the accuracy of the model, thus needs to be cleaned first. For cleaning the data, we have used 'beautifulsoup4', an open source Python data analysis library. We had to decide whether to include the syllables like punctuation marks and smiles, which may hold meaning to the reviews. But for the simplicity, we discarded such entities.
Training data is a .csv file with three columns. The first column is the unique ID, then its corresponding sentiment value and the last is the review. Sentiment 1 implies that the review is positive while 0 shows a negative review

Spark provides a handful of these algorithms such as TF- IDF (Term frequency and Inverse Document Frequency), count vectorization, Word2Vec, N-grams and few more. For the project, we used two algorithms, TF-IDF and Word2Vec and after the features are created, we used each algorithm of ML to above two feature creation methods and compared the evaluation metric results for both of these methods.

Import Required Libraries

In [1]:
```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem import WordNetLemmatizer
from bs4 import BeautifulSoup
import re,string,unicodedata
```

Import Dataset

In [2]:
```
imdb_data=pd.read_csv('Data/IMDB_Dataset.csv')
```

Initialise tockeniser and stopwords

In [3]:
```
tokenizer=ToktokTokenizer()
stopword_list=stopwords.words('english')
stop=set(stopword_list)
```

Define cleaner function

The cleaner function use Beautiful Soup html parser and removes html tags in the review. Then we use refular expression to remove brackets and special characters as these does have no effect in sentiment analysis model. Then we tokenise the reviews and convert each token to lower case letter and remove any white spaces. then we filter the tockens using the stopwords. Then we stem the tokens using nltk PorterStemmer. Stemming is the process of producing morphological variants of a root/base word.

In [4]:
```
def cleaner(text):
    soup = BeautifulSoup(text, "html.parser")
    text = soup.get_text()
    text = re.sub(r'\[[^]]*\]', '', text)
    text = re.sub(r'[^a-zA-z\s]','',text)
    tokens = tokenizer.tokenize(text)
    tokens = [token.lower().strip() for token in tokens]
    filtered_tokens = [token for token in tokens if token not in stopword_list]
    text = ' '.join(filtered_tokens)
    ps=nltk.porter.PorterStemmer()
    text = ' '.join([ps.stem(word) for word in text.split()])
    return text
```

Store Processed reviews

In [5]:
```
imdb_data['review']=imdb_data['review'].apply(cleaner)

imdb_data.to_csv('data/cleanedReviews.csv',index=False)
```

Model Preparation

Import required libraries

In [1]:
```
import findspark
findspark.init()
import pyspark as ps
import warnings
from pyspark.sql import SQLContext
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import CountVectorizer
from pyspark.ml.evaluation import BinaryClassificationEvaluator
import pyspark.sql.functions as f
import matplotlib.pyplot as plt
```

Initialize Apache Spark in local cluster

In [2]:
```
try:
    sc = ps.SparkContext("local[*]",'test')
    sqlContext = SQLContext(sc)
    print("Just created a SparkContext")
except ValueError:
    warnings.warn("SparkContext already exists in this scope")
```

```
Just created a SparkContext
```

Read PreProcessed Data

In [3]:
```
df = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('data/cleanedReviews.csv
df = df.dropna()
df.show()
df.count()
```

```
+--------------------+---------+
|              review|sentiment|
+--------------------+---------+
|one review mentio...| positive|
|wonder littl prod...| positive|
|thought wonder wa...| positive|
|basic there famil...| negative|
|petter mattei lov...| positive|
|probabl alltim fa...| positive|
|sure would like s...| positive|
|show amaz fresh i...| negative|
|encourag posit co...| negative|
|like origin gut w...| positive|
|phil alien one qu...| negative|
|saw movi came rec...| negative|
|im big fan boll w...| negative|
|cast play shakesp...| negative|
|fantast movi thre...| positive|
|kind drawn erot s...| negative|
|film simpli remad...| positive|
|movi made one top...| negative|
|rememb filmit fir...| positive|
|aw film must real...| negative|
+--------------------+---------+
only showing top 20 rows
```

Out[3]:
```
50000
```

Split data into Train set and Validation set

In [4]:
```
(train_set, val_set) = df.randomSplit([0.9, 0.1], seed = 300)
print("Train set count:",train_set.count())
print("Validation set count:",val_set.count())
```

```
Train set count: 44948
Validation set count: 5052
```

Functions for training data using tf_idf or cv_idf

In [5]:
```
def tf_idf(textCol,labelCol):
    tokenizer = Tokenizer(inputCol=textCol, outputCol="words")
    hashtf = HashingTF(numFeatures=1000, inputCol="words", outputCol='tf')
    idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5)
    label_stringIdx = StringIndexer(inputCol = labelCol, outputCol = "label")
    lr = LogisticRegression(maxIter=100)
    pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_stringIdx, lr])
    return pipeline

def cv_idf(textCol,labelCol):
    tokenizer = Tokenizer(inputCol=textCol, outputCol="words")
    cv = CountVectorizer(vocabSize=1000, inputCol="words", outputCol='cv')
    idf = IDF(inputCol='cv', outputCol="features", minDocFreq=5)
    label_stringIdx = StringIndexer(inputCol = labelCol, outputCol = "label")
    lr = LogisticRegression(maxIter=100)
    pipeline = Pipeline(stages=[tokenizer, cv, idf, label_stringIdx, lr])
    return pipeline
```

We use Term Frequency-Inverse Document Frequency or Counter Vector-Inverse Document Frequency for selecting the features, they provide the weights for the token generated which is used for analyzing the sentiment of reviews.

Training model using tf_idf and predictions of validation datasets

In [6]:
```
%%time
pipelineFit = tf_idf("review","sentiment").fit(train_set)

predictions = pipelineFit.transform(val_set)

evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
tf_accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(val_set.count())
tf_roc_auc = evaluator.evaluate(predictions)

print("Accuracy:",tf_accuracy)
print("Area under curve:",tf_roc_auc)
```

```
Accuracy: 0.8032462391132225
Area under curve: 0.8837053830312113
Wall time: 34.2 s
```

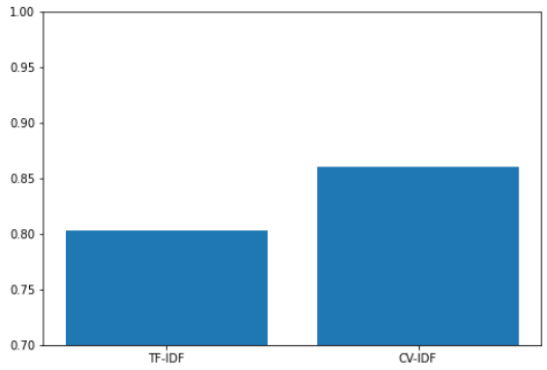
Training model using cv_idf and predictions of validation datasets

In [7]:
```
%%time
pipelineFit = cv_idf("review","sentiment").fit(train_set)

predictions = pipelineFit.transform(val_set)

evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
cv_accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(val_set.count())
cv_roc_auc = evaluator.evaluate(predictions)

print("Accuracy:",cv_accuracy)
print("Area under curve:",cv_roc_auc)
```

```
Accuracy: 0.8606492478226445
Area under curve: 0.9318968813965228
Wall time: 24.2 s
```

Accuracy plot

In [8]:
```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
XAxis = ['TF-IDF','CV-IDF']
Height = [tf_accuracy,cv_accuracy]
ax.bar(XAxis,Height)
plt.ylim(0.7, 1)
plt.show()
```



We train our model using both tf-idf and cv-idf. Then we analyze the accuracy of both the methods. The above graph depicts that tf-idf has accuracy of 88.37% and cv-idf has accuracy of 93.18%. It's obvious that cv-idf has more accuracy and we use cv-idf for further analysis.

Result Analysis

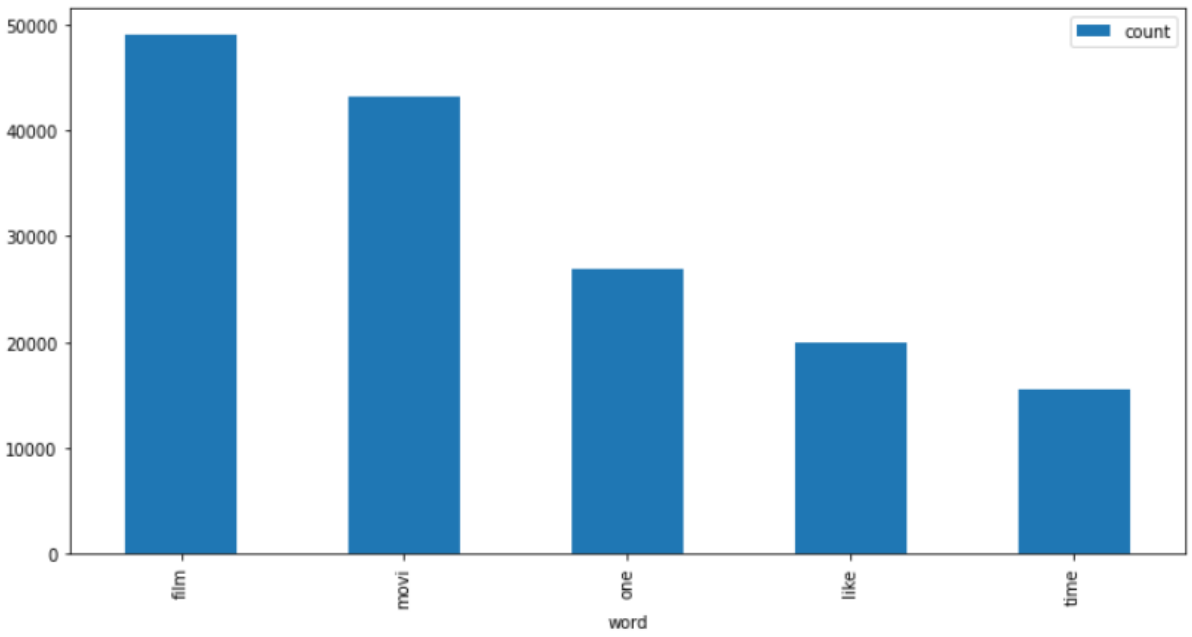
Here we provide the frequencies of popular words used in both positive and negative review. We have also visualized the frequency count of top 5 words in both positive review and negative review using the bar-graph. The positive review has words like see, good, like etc. while negative review has words like don't, bad etc.

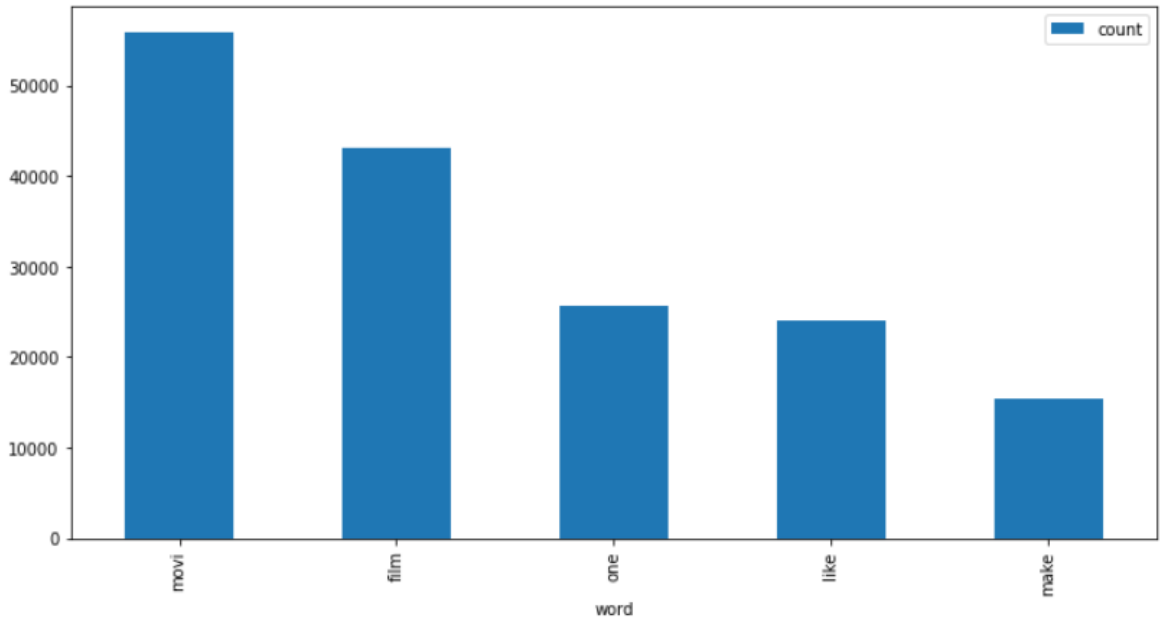
Positive and Negative word frequencies

In [9]:
```
def word_count(sentiment):
    d=df.filter(f.col('sentiment')==sentiment)
    plot = d.withColumn('word', f.explode(f.split(f.col('review'), ' ')))\
        .groupBy('word')\
        .count()\
        .sort('count', ascending=False)
    plot.show()
    print(plot.count())
    pandas_df = plot.toPandas().head(5)
    pandas_df.set_index('word', inplace=True)
    pandas_df.plot(kind='bar',figsize=(12,6));

word_count("positive")
word_count("negative")
```

```
+-------+-----+
|   word|count|
+-------+-----+
|   movi|55893|
|   film|43095|
|    one|25719|
|   like|23944|
|   make|15402|
|    get|15142|
|   even|15098|
|  watch|14582|
|   time|14360|
|   good|14345|
|    bad|14157|
|charact|13944|
|  would|13631|
|    see|12889|
| realli|12225|
|   look|11315|
|  scene|11013|
|  stori|10582|
|    act|10141|
|   dont|10008|
+-------+-----+
only showing top 20 rows

106286
```

```
+-------+-----+
|   word|count|
+-------+-----+
|   film|49021|
|   movi|43129|
|    one|26958|
|   like|19885|
|   time|15480|
|    see|14687|
|   good|14573|
|  stori|13654|
|charact|13630|
|   make|13167|
|  great|12798|
|    get|12577|
|  watch|12507|
|   love|12064|
|   well|11309|
|   show|10813|
| realli|10676|
|   also|10550|
|  would|10368|
|   play| 9840|
+-------+-----+
only showing top 20 rows

108517
```





The average word count per review is 119 words which is shown below.

Average word count per review

```
In [10]: df.withColumn('count', f.size(f.split(f.col('review'), ' '))).agg({"count": "avg"}).show()
```

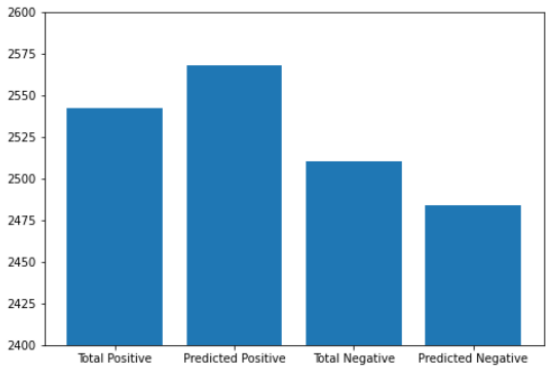
```
+----------+
|avg(count)|
+----------+
| 118.62438|
+----------+
```

Now let us compare the predicted value against the actual results. The below bar-graph provides the visualization of Total Positive reviews and Total Negative reviews that are actually present and the Total Positive reviews and Total Negative reviews that our model has predicted. We can see that some of the negative reviews has been predicted as positive. The technology has to be developed with the advancement in deep learning to achieve full accuracy. We also have plotted the accuracy of positive and negative reviews and we can see that positive review accuracy is slightly above 87 and negative review accuracy is slightly below 85.

Plot Total positive and negative with predicted positive and negative

In [11]:
```
posCount = val_set.filter(df.sentiment=="positive").count()
negCount = val_set.filter(df.sentiment=="negative").count()
posPred = predictions.filter(1.0 == predictions.prediction).count()
negPred = predictions.filter(0.0 == predictions.prediction).count()
print(posCount,negCount,posPred,negPred)

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
XAxis = ['Total Positive','Predicted Positive','Total Negative','Predicted Negative']
Height = [posCount,posPred,negCount,negPred]
ax.bar(XAxis,Height)
plt.ylim(2400, 2600)
plt.show()
```

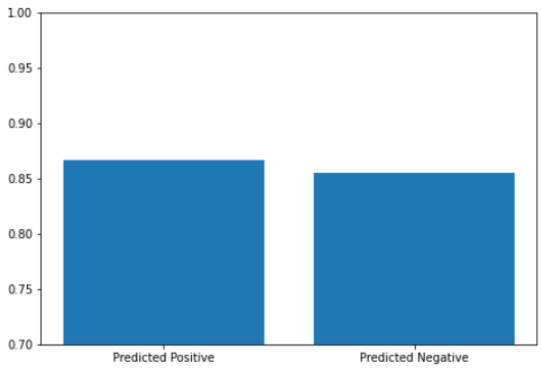
```
2542 2510 2568 2484
```



Positive and negative reviews Accuracy

In [12]:
```
pos = predictions.filter(predictions.label==1.0)
neg = predictions.filter(predictions.label==0.0)
posCount = pos.count()
negCount = neg.count()
posPred = pos.filter(1.0 == predictions.prediction).count()
negPred = neg.filter(0.0 == predictions.prediction).count()
print(posCount,negCount,posPred,negPred)

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
XAxis = ['Predicted Positive','Predicted Negative']
Height = [posPred/posCount,negPred/negCount]
ax.bar(XAxis,Height)
plt.ylim(0.7, 1)
plt.show()
```

```
2542 2510 2203 2145
```



Predicting new reviews

In [13]:
```
def predict_sentence(sentence):
    columns=["review","sentiment"]
    data=[(sentence,"positive")]
    sentenceDF = sqlContext.createDataFrame(data).toDF(*columns)
    prediction = pipelineFit.transform(sentenceDF)
    if prediction.head().prediction==1.0:
        print("Positive review")
    else:
        print("Negative review")

positive_sentence = '''It is a very good movie'''
predict_sentence(positive_sentence)
negative_sentence = '''This movie will be flop. Its boring'''
predict_sentence(negative_sentence)
```

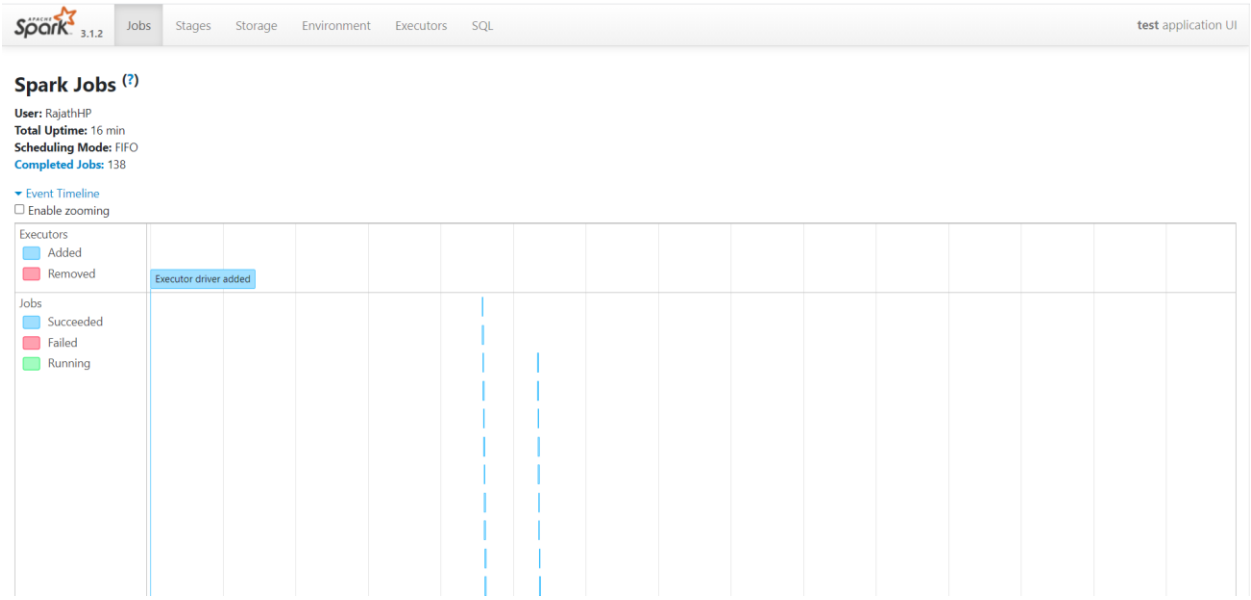
```
Positive review
Negative review
```

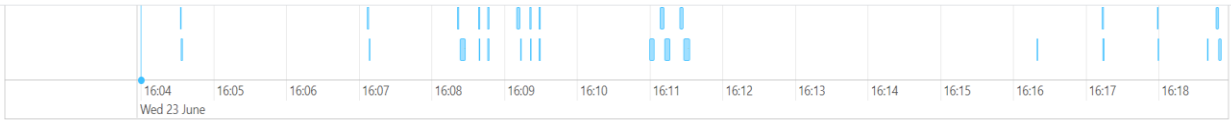
In [14]:
```
sc.stop()
```

Here we check our model against some custom reviews and get the desired output.

Apache Spark Web UI

Here we review the system performance and various resources used, time taken to complete the jobs, computational resources used and the primary and secondary resources used, stages of all the jobs, environments, executors and SQL, and other system resources analysis provided by Spark Web UI.





▾ **Completed Jobs (138)**

Page: 1 2 >

2 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
137	head at <ipython-input-13-52c86a37caa5>:6 head at <ipython-input-13-52c86a37caa5>:6	2021/06/23 16:18:49	2 s	1/1	3/3
136	head at <ipython-input-13-52c86a37caa5>:6 head at <ipython-input-13-52c86a37caa5>:6	2021/06/23 16:18:46	3 s	1/1	4/4
135	head at <ipython-input-13-52c86a37caa5>:6 head at <ipython-input-13-52c86a37caa5>:6	2021/06/23 16:18:45	0.8 s	1/1	1/1
134	head at <ipython-input-13-52c86a37caa5>:6 head at <ipython-input-13-52c86a37caa5>:6	2021/06/23 16:18:43	2 s	1/1	3/3
133	head at <ipython-input-13-52c86a37caa5>:6 head at <ipython-input-13-52c86a37caa5>:6	2021/06/23 16:18:40	3 s	1/1	4/4
132	head at <ipython-input-13-52c86a37caa5>:6 head at <ipython-input-13-52c86a37caa5>:6	2021/06/23 16:18:39	0.9 s	1/1	1/1
131	count at <unknown>:0 count at <unknown>:0	2021/06/23 16:17:58	0.3 s	2/2	9/9
130	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2021/06/23 16:17:58	0.3 s	2/2	9/9
129	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2021/06/23 16:17:58	0.2 s	2/2	9/9

Spark 3.1.2 Jobs Stages Storage Environment Executors SQL **test** application UI

Stages for All Jobs

Completed Stages: 264
Skipped Stages: 13

▾ **Completed Stages (264)**

Page: 1 2 3 >

3 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
276	head at <ipython-input-13-52c86a37caa5>:6 +details	2021/06/23 16:18:49	2 s	3/3				
275	head at <ipython-input-13-52c86a37caa5>:6 +details	2021/06/23 16:18:46	3 s	4/4				
274	head at <ipython-input-13-52c86a37caa5>:6 +details	2021/06/23 16:18:45	0.7 s	1/1				
273	head at <ipython-input-13-52c86a37caa5>:6 +details	2021/06/23 16:18:43	2 s	3/3				
272	head at <ipython-input-13-52c86a37caa5>:6 +details	2021/06/23 16:18:40	3 s	4/4				
271	head at <ipython-input-13-52c86a37caa5>:6 +details	2021/06/23 16:18:39	0.9 s	1/1				
270	count at <unknown>:0 +details	2021/06/23 16:17:59	6 ms	1/1			472.0 B	
269	count at <unknown>:0 +details	2021/06/23 16:17:58	0.2 s	8/8	36.4 MiB			472.0 B
268	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:58	5 ms	1/1			472.0 B	
267	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:58	0.3 s	8/8	36.4 MiB			472.0 B
266	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:58	5 ms	1/1			472.0 B	
265	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:58	0.1 s	8/8	36.4 MiB			472.0 B
264	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:57	6 ms	1/1			472.0 B	
263	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:57	0.2 s	8/8	36.4 MiB			472.0 B

172	treeAggregate at RDDLossFunction.scala:61	+details	2021/06/23 16:09:21	43 ms	8/8	31.0 MiB			69.3 KiB
171	treeAggregate at RDDLossFunction.scala:61	+details	2021/06/23 16:09:21	18 ms	2/2			69.3 KiB	
170	treeAggregate at RDDLossFunction.scala:61	+details	2021/06/23 16:09:21	48 ms	8/8	31.0 MiB			69.3 KiB

Page: 1 2 3 >

3 Pages. Jump to 1 . Show 100 items in a page. Go

Skipped Stages (13)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
250	toPandas at <ipython-input-9-8216ee827b5f>:9	+details	Unknown	Unknown	0/8				
240	toPandas at <ipython-input-9-8216ee827b5f>:9	+details	Unknown	Unknown	0/8				
231	combineByKey at BinaryClassificationMetrics.scala:188	+details	Unknown	Unknown	0/8				
230	map at BinaryClassificationMetrics.scala:48	+details	Unknown	Unknown	0/8				
228	combineByKey at BinaryClassificationMetrics.scala:188	+details	Unknown	Unknown	0/8				
227	map at BinaryClassificationMetrics.scala:48	+details	Unknown	Unknown	0/8				
224	map at BinaryClassificationMetrics.scala:48	+details	Unknown	Unknown	0/8				
126	flatMap at CountVectorizer.scala:212	+details	Unknown	Unknown	0/8				
122	combineByKey at BinaryClassificationMetrics.scala:188	+details	Unknown	Unknown	0/8				
121	map at BinaryClassificationMetrics.scala:48	+details	Unknown	Unknown	0/8				
119	combineByKey at BinaryClassificationMetrics.scala:188	+details	Unknown	Unknown	0/8				
118	map at BinaryClassificationMetrics.scala:48	+details	Unknown	Unknown	0/8				
115	map at BinaryClassificationMetrics.scala:48	+details	Unknown	Unknown	0/8				

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

Spark 3.1.2 Jobs Stages Storage Environment Executors SQL test application UI

Environment

Runtime Information

Name	Value
Java Home	C:\Program Files\Java\jdk-13.0.1
Java Version	13.0.1 (Oracle Corporation)
Scala Version	version 2.12.10

Spark Properties

Resource Profiles

Hadoop Properties

Name	Value
adl.feature.ownerandgroup.enableupn	false
adl.http.timeout	-1
dfs.ha.fencing.ssh.connect-timeout	30000
file.blocksize	67108864
file.bytes-per-checksum	512
file.client-write-packet-size	65536
file.replication	1
file.stream-buffer-size	4096
fs.AbstractFileSystem.abfs.impl	org.apache.hadoop.fs.azurebfs.Abfs
fs.AbstractFileSystem.abfss.impl	org.apache.hadoop.fs.azurebfs.Abfss

Executors

Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(1)	0	147.7 KiB / 434.4 MiB	0.0 B	8	0	0	2959	2959	9.8 min (14 s)	4.5 GiB	45.8 MiB	37.3 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	147.7 KiB / 434.4 MiB	0.0 B	8	0	0	2959	2959	9.8 min (14 s)	4.5 GiB	45.8 MiB	37.3 MiB	0

Executors

Show 20 entries

Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump
driver	LAPTOP-HEU33EDE:55211	Active	0	147.7 KiB / 434.4 MiB	0.0 B	8	0	0	2959	2959	9.8 min (14 s)	4.5 GiB	45.8 MiB	37.3 MiB	Thread Dump

Showing 1 to 1 of 1 entries

Previous 1 Next

SQL

Completed Queries: 28

Completed Queries (28)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

ID	Description	Submitted	Duration	Job IDs
27	head at <ipython-input-13-52c86a37caa5>:6 +details	2021/06/23 16:18:45	5 s	[135][136][137]
26	head at <ipython-input-13-52c86a37caa5>:6 +details	2021/06/23 16:18:39	6 s	[132][133][134]
25	count at <unknown>:0 +details	2021/06/23 16:17:58	0.3 s	[131]
24	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:58	0.4 s	[130]
23	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:58	0.2 s	[129]
22	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:57	0.3 s	[128]
21	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:13	0.4 s	[127]
20	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:12	0.6 s	[126]
19	count at NativeMethodAccessorImpl.java:0 +details	2021/06/23 16:17:12	0.3 s	[125]

Conclusion and Future Work

Movie reviews are an important way to gauge the performance of a movie. Sentiment analysis refers to the use of natural language processing, text analysis and computational linguistics to extract and identify subjective information in source materials. The main objective is to predict the sentiment for several movie reviews obtained from the Internet Movie Database (IMDb). We have used pyspark an API which uses Apache Spark for performing analysis which is faster than traditional python scripts. We gather useful insights from the analysis which can be used for further enhancement of film industry, recommendation system etc.

The future scope of work is to develop other classifiers like SVM, CNN and to verify whether a hybrid technique can be used by applying the permutations and combinations of other classifiers in order to achieve better accuracy.