STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Amazon Reviews for Sentiment Analysis using PySpark and AWS EMR

BIA-678 Big Data Technologies

——Instructor: Prof. David Belanger

Team Pictures

Shujaat Bakhsh
Fall 18 MSCS

Akshay Kirolikar
Fall 18 MSIS

Sumit Gupta
Fall 18 MSCS

Gaurav Venkatraman
Fall 18 BIA

PRESENTATION

AGENDA

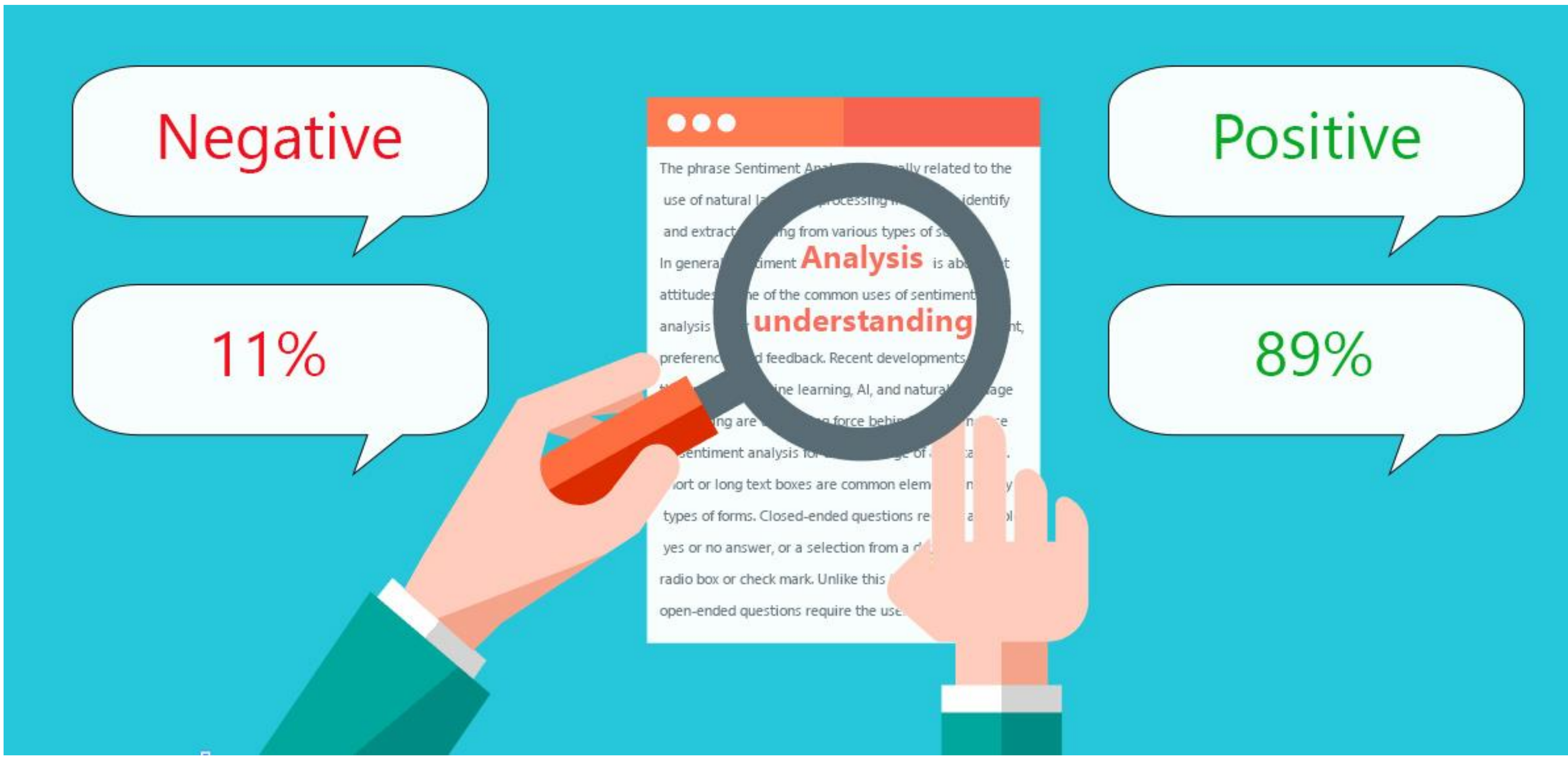
1. Introduction
2. About the Data
3. Model Evaluation
4. PySpark Comparison
5. Conclusion

Introduction

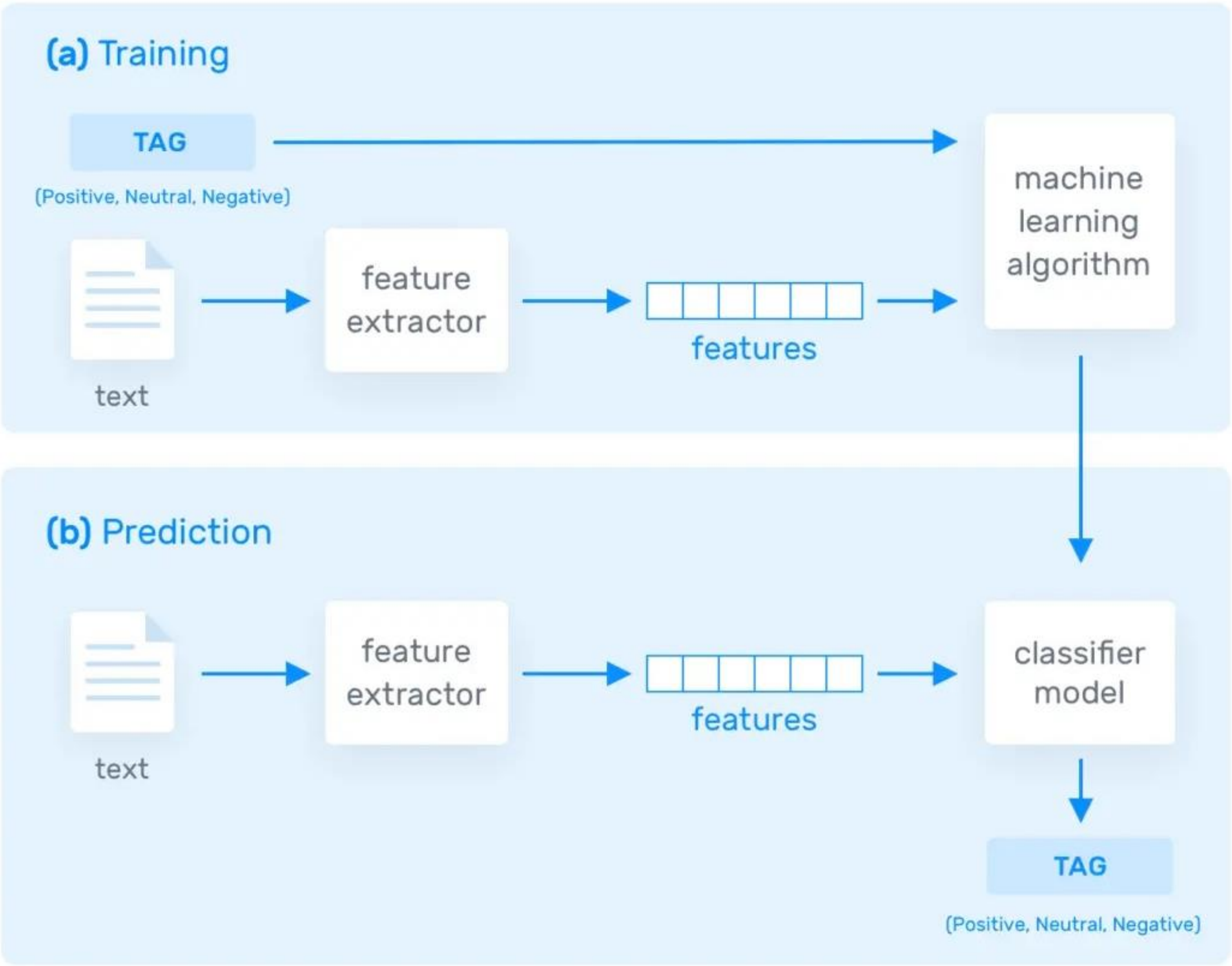
- Amazon is one of the largest online vendor in the World. People often gaze over the products and reviews of the product before buying the product on amazon itself
- Consequently large amount of data in the form of reviews is produced which helps prospective buyers to choose the right product
- Furthermore these reviews contain opinionated contents which can be useful for the company to identify the areas which need to be enhanced.

What is sentiment analysis ?

Sentiment analysis is uses to identify positive, negative and neutral opinions in text. Also known as opinion mining, it helps businesses track customer sentiments over time, and gain valuable insights about their brand to make data-driven decisions.



How does Sentiment Analysis Work?



About the Data

- Source: The data was taken from Kaggle.
- Data consist of 3.6 million rows of training data and 400K test data.
- the classes are __label__1 and __label__2, and there is only one class per row.
  __label__1 corresponds to 1- and 2-star reviews
  __label__2 corresponds to 4- and 5-star reviews.
  (3-star reviews i.e. reviews with neutral sentiment were not included in the original)
- Goal : This is a large dataset, and the version used here only has the text as a feature. We will process the data and predict the sentiment of the review given by the user as positive or negative.
- We will implement two models Logistic Regression and Naive Bayes in PySpark and compare the computations.

Data Processing

Raw Data

```
__label__2 Whispers of the Wicked Saints: This was a easy to read book
__label__1 The Worst!: A complete waste of time. Typographical errors,
__label__2 Great book: This was a great book,I just could not put it d
```

- __label__2 is a positive review
- __label__1 is a negative review

Data Cleaning

- ❖ The text file was parsed and the single column was converted into two.
- ❖ __label__2 that is the positive review is changed to 1.0
- ❖ __label__1 that is the negative review is changed to 0.0
- ❖ The data was converted to a Pyspark dataframe .

label	reviewText
1.0	Stuning even for ...
1.0	The best soundtra...
1.0	Amazing!: This so...
1.0	Excellent Soundtr...
1.0	Remember, Pull Yo...
1.0	an absolute maste...
0.0	Buyer beware: Thi...
1.0	Glorious story: I...
1.0	A FIVE STAR BOOK:...
1.0	Whispers of the W...
0.0	The Worst!: A com...
1.0	Great book: This ...
1.0	Great Read: I tho...
0.0	Oh please: I gues...
0.0	Awful beyond beli...
0.0	Don't try to fool...
1.0	A romantic zen ba...
1.0	Fashionable Compr...
1.0	Jobst UltraSheer ...
0.0	sizes recomended ...

only showing top 20 rows

Text Data to TFIDF Vectorizer

- Text data requires special preparation before you can start using it for predictive modeling
- The text must be parsed to remove words, called tokenization.
- Then the words need to be encoded as integers or floating point values for use as input to a machine learning algorithm, called feature extraction (or vectorization).
- the Bag-of-Words Model, or BoW which throws away all of the order information in the words and focuses on the occurrence of words in a document.
- TF-IDF - Term Frequency – Inverse Document Frequency
  - Term Frequency: This summarizes how often a given word appears within a document.
  - Inverse Document Frequency: This downscales words that appear a lot across documents.

Bag-of-words

Two simple text documents:

(1) Sam likes to watch cartoons. Lucy likes cartoons too.
(2) Sam also likes to watch soccer games.

List constructed as follows for each document:

"Sam", "likes", "to", "watch", "cartoons", "lucy", "likes", "cartoons", "too"

"Sam ", "also", "likes", "to", "watch", "soccer", "games"

Representing each word respect to there term frequency:

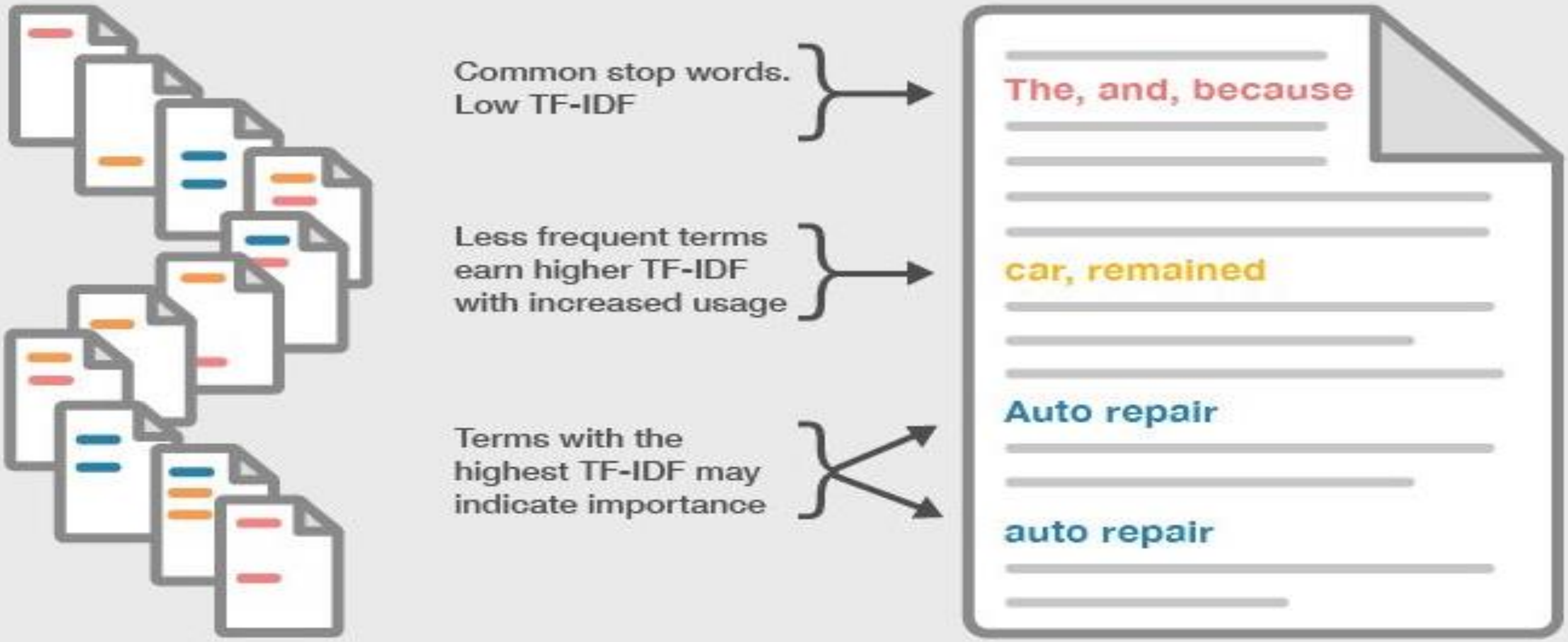
BoW1 = {"Sam":1,"likes":2,"to":1,"watch":1,"cartoons":2,"lucy":1,"too":1..
BoW2 = {"Sam":1,"also":1,"likes":1,"to":1,"watch":1,"soccer":1,"games":1};

Final representation of bag of words

BoWF = {"Sam":2,"likes":3,"to":2,"watch":2,"cartoons":2,"lucy":1,"too":1,"also":1,"soccer":1,"games":1};

Frequency of term in a large set of documents

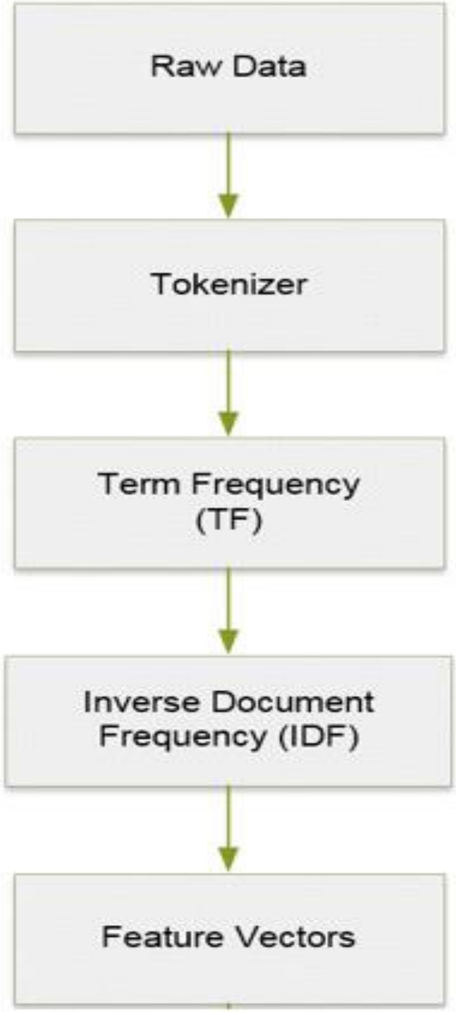
Frequency of term on a single page



TF-IDF

Term frequency–inverse document frequency (TF-IDF) measures the importance of a keyword phrase by comparing it to the frequency of the term in a large set of documents. Many advanced textual analysis techniques use a version of TF-IDF as a base.

Data Processing Pipeline



- Create a Datframe from raw text
- Tokenize the sentences
- Create the Frequency matrix of the words in each sentence.
- Calculate TermFrequency and generate a matrix
- Creating a table for documents per words
- Calculate IDF and generate a matrix
- Calculate TF-IDF and generate a matrix

Classification methods

❖ Logistic Regression

Confusion Matrix

	True Positive	True Negative
Predicted Positive	176900	23100
Predicted Negative	21370	178630

Sensitivity	0.8922	**Accuracy**	0.8888
Specificity	0.8855	**F1 Score**	0.8883
Precision	0.8845	**False Negative Rate**	0.1078
False Positive Rate	0.1145		

❖ Naïve Bayes

Confusion Matrix

	True Positive	True Negative
Predicted Positive	167433	32567
Predicted Negative	33311	166689

Sensitivity	0.8341	**False Positive Rate**	0.1634
Specificity	0.8366	**False Negative Rate**	0.1659
Precision	0.8372	**F1 Score**	0.8356
Accuracy	0.8353		

PySpark , Python and AWS EMR

This project used IPython notebooks attached to AWS EMR clusters.

The IPython interface has been preconfigured so that it can be used easily with PySpark API with minimum overhead.

Environment Setup

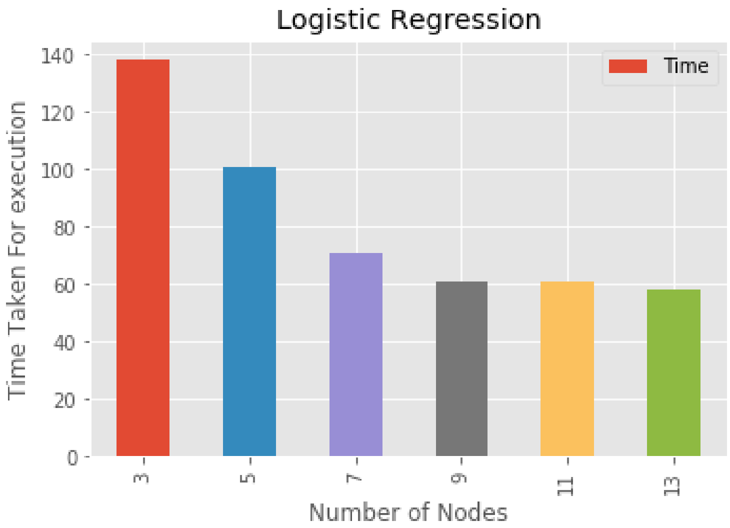
- The AWS EMR cluster uses specified number of ec2 instances to create a spark cluster.
- The user can then create an IPython notebook and attach it to a running cluster.
- This IPython notebook can be used to perform different operations as supported by PySPark.
- The input dataset is stored on a s3 bucket and is used to create a spark Resilient Distributed Dataset (RDD).
- A vast array of operations is provided by the Spark API which can be performed on this RDD in a parallelized fashion.
- The user can also use user-defined functions to manipulate the RDD. It can also be converted to a dataframe using the SQLContext API which converts the whole RDD into a row and column based data store.

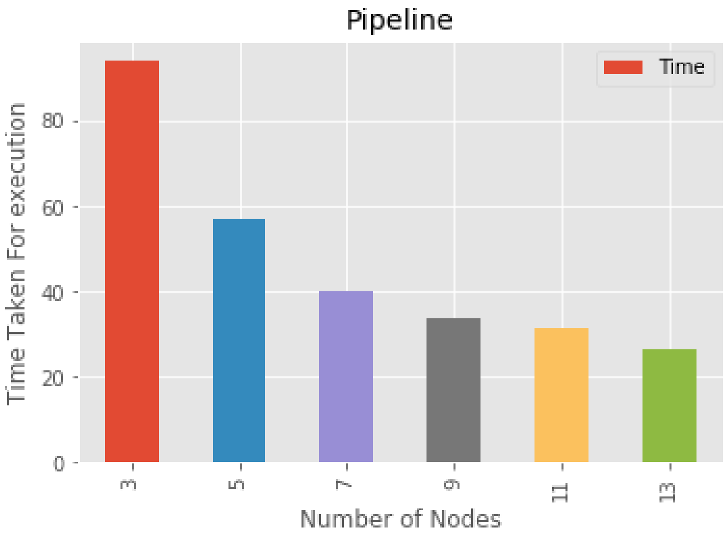
Pyspark Computations with different configurations

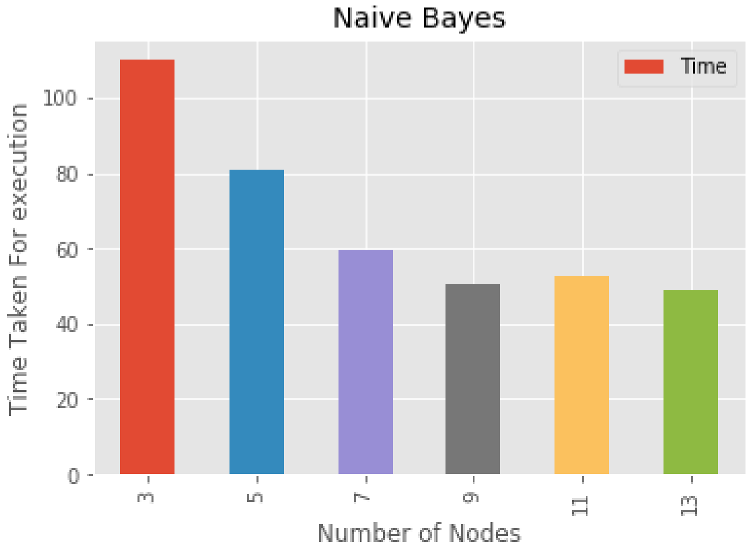
This project used AWS EMR cluster with different configurations.

- 3 nodes (1 master 2 Slaves)
- 5 nodes (1 master 4 Slaves)
- 7 nodes (1 master 6 Slaves)
- 9 nodes (1 master 8 slaves)
- 11 nodes (1 master 10 slaves)
- 13 nodes (1 master 12 slaves)

Compute Time Variation across Cluster Configurations







Conclusion

The main takeaways from this project have been:

- Spark is a very powerful framework and is best suited for large scale datasets. Smaller datasets tend to perform better on local machines because of the network and cluster configuration overhead.
- There is a direct relation between the number of nodes in a cluster and compute power.
- PySpark is a very thorough Python wrapper for using the Spark framework but is still evolving and gaining maturity.
- PySpark MLLib is an excellent library with all the required tools for training models on a spark cluster. It provides a lot of abstraction and hence, it is easy to implement with working knowledge of existing machine learning frameworks.

Thank you!