

Big Data Analytics: Distributed Machine Learning Algorithms



Arjun Anand, Enmao Diao, Meera Kamath, Sairam Balani

Background

- Big Data becoming an important aspect of most analytical tasks.
- Today, there are various tools and frameworks available to support distributed tasks.

Size	Classification	Tools
Lines <i>Sample Data</i>	Analysis and Visualization	Whiteboard, bash,...
KBs - low MBs <i>Prototype Data</i>	Analysis and Visualization	Matlab, Octave, R, Processing, bash,...
MBs - low GBs <i>Online Data</i>	Storage	MySQL (DBs),...
	Analysis	NumPy, SciPy, Weka, BLAS/ LAPACK,...
	Visualization	Flare, AmCharts, Raphael, Protovis,...
GBs - TBs - PBs <i>Big Data</i>	Storage	HDFS, HBase, Cassandra,...
	Analysis	Hive, Mahout, Hama, Giraph,...

Mahout?

LIBLINEAR?

H₂O?

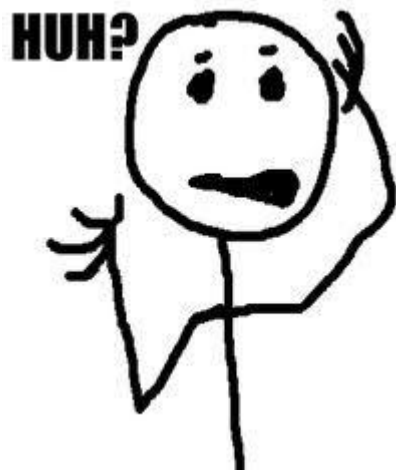
Vowpal Wabbit?

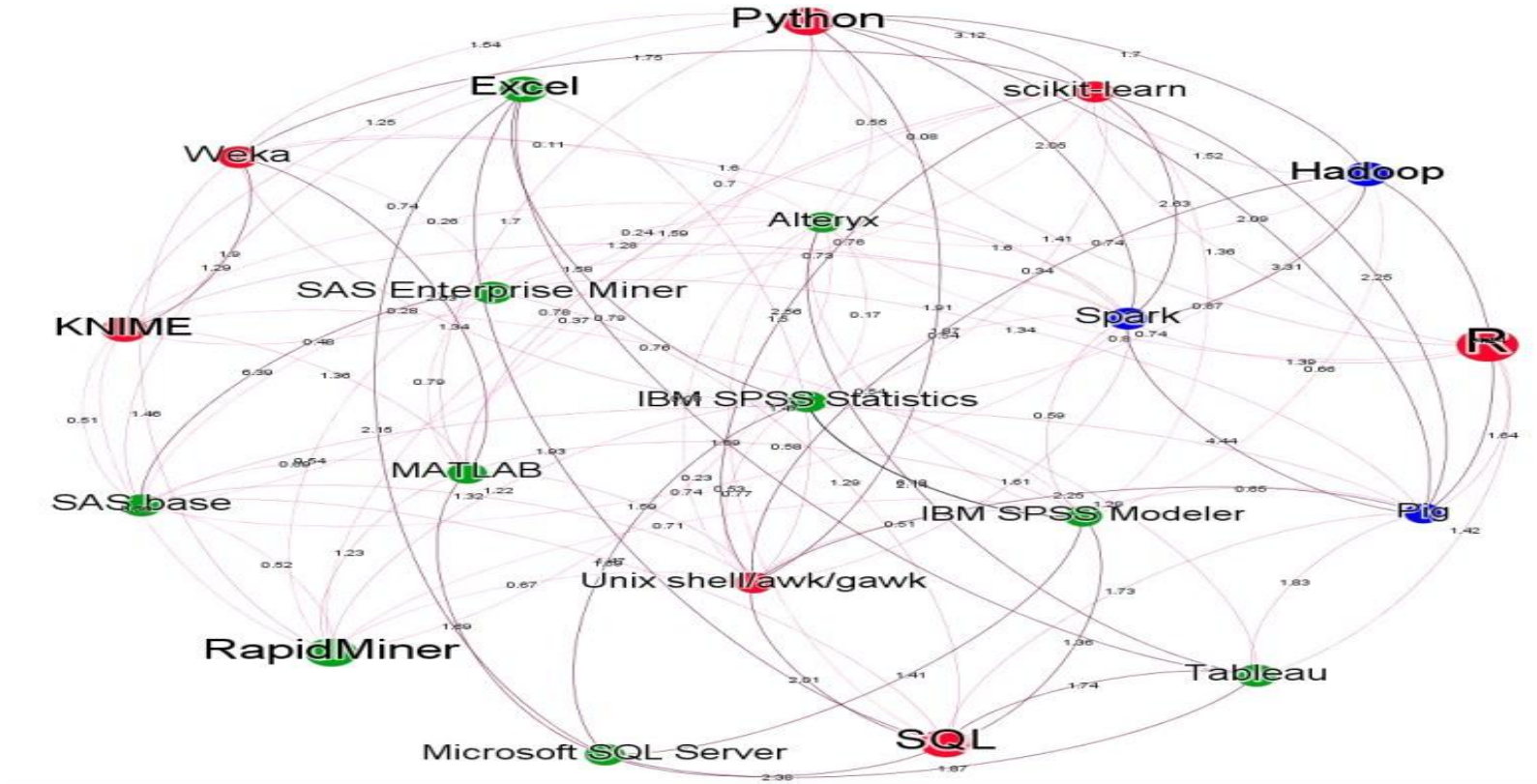
MATLAB?

R?

scikit-learn?

Weka?





Network graph of top 20 most popular tools

Machine Learning Algorithms

We focus on two types of machine learning problems

Classification - Supervised learning

- Using labelled training data to create a classifier that can predict output for unseen inputs.
- Eg: Loan Approval, Spam Detection, etc.

Clustering - Unsupervised learning

- Grouping unstructured data without any training data
- Self learning.
- Small intra-cluster distance and large inter-cluster distance
- Eg: Image Segmentation, Anomaly Detection, etc.

Apache Mahout

What is it?

- Scalable Machine Learning library that supports large data sets.
- Built on Hadoop , written in Java

Algorithm types:

- Classification
- Clustering
- Recommendation



What's in Mahout

Recommendation	Classification	Clustering
User-Based Collaborative Filtering - single machine	Logistic Regression - trained via SGD - single machine	Canopy Clustering - single machine / MapReduce (deprecated, will be removed once Streaming k-Means is stable enough)
Item-Based Collaborative Filtering - single machine / MapReduce	Naive Bayes/ Complementary Naive Bayes - MapReduce	k-Means Clustering - single machine / MapReduce
Matrix Factorization with Alternating Least Squares - single machine / MapReduce	Random Forest - MapReduce	Fuzzy k-Means - single machine / MapReduce
Matrix Factorization with Alternating Least Squares on Implicit Feedback- single machine / MapReduce	Hidden Markov Models - single machine	Streaming k-Means - single machine / MapReduce
Weighted Matrix Factorization, SVD++, Parallel SGD - single machine	Multilayer Perceptron - single machine	Spectral Clustering - MapReduce

Spark ML-Lib



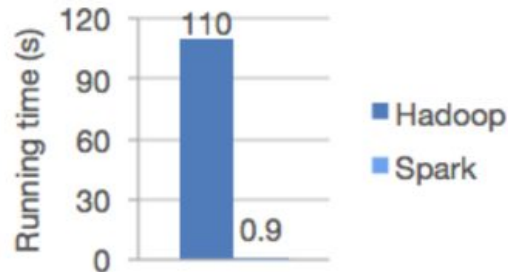
MLlib is a Spark subproject providing machine learning primitives.

- initial contribution from AMPLab, UC Berkeley
- shipped with Spark since version 0.8

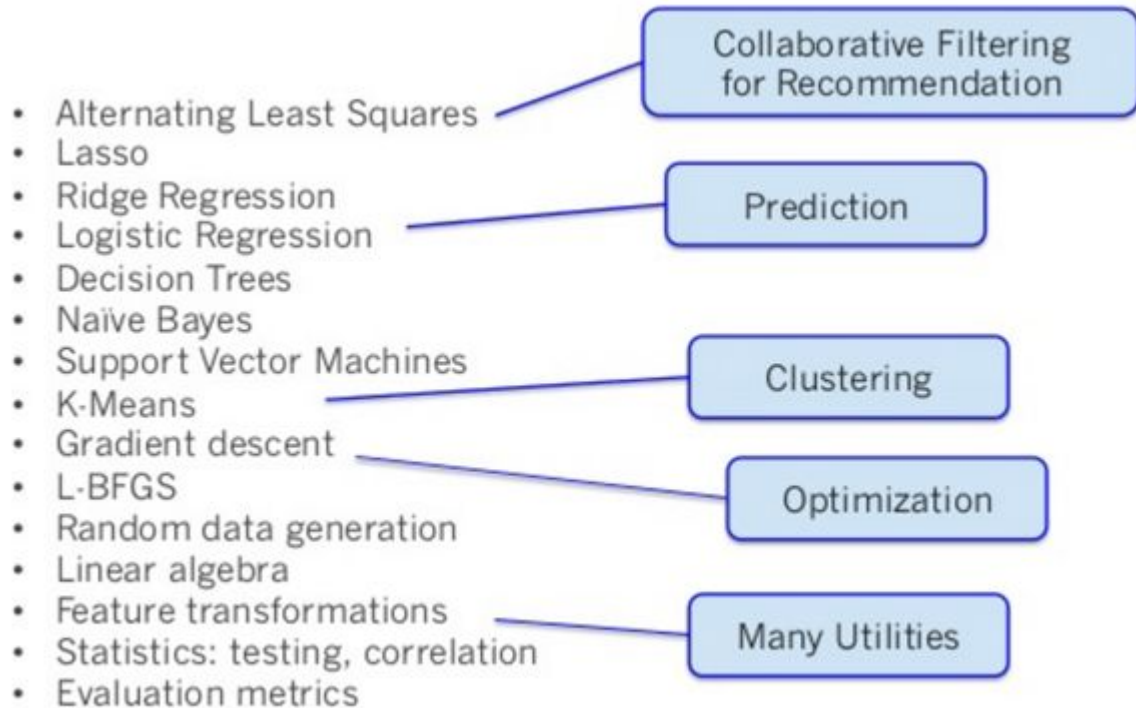
It is built on Apache Spark, a fast and general engine for large-scale data processing.

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Write applications quickly in Java, Scala, or Python.



What's in MLlib



Dataset I - Million Songs Dataset

- **Source:** <http://labrosa.ee.columbia.edu/millionsong/blog/11-2-28-deriving-genre-dataset>
- Original Size : 280 GB with a million songs
- Our representative set : 60k songs, ~16.8GB
- Primary Features used - Loudness, energy, timbre, tempo, song key

Our labels - 10 Genres

- folk
- metal
- pop
- classical
- hip-hop
- dance and electronica
- classic pop and rock
- punk
- soul and reggae
- jazz and blues

Why is Genre Classification important?

- Categorical labels created by humans
- Currently, musical genre annotation is done manually - Process can be AUTOMATED!
- First feature used to make effective recommendations
- Expresses first degree of song similarity

Clustering

- Why? - To understand what truly represents a “genre”
- How? - Evaluate the dataset based on combination of representative features and understand which best groups the data
- Ok, but really how? - K-Means! Your friendly neighbourhood clustering algorithm!

K-Means Setup

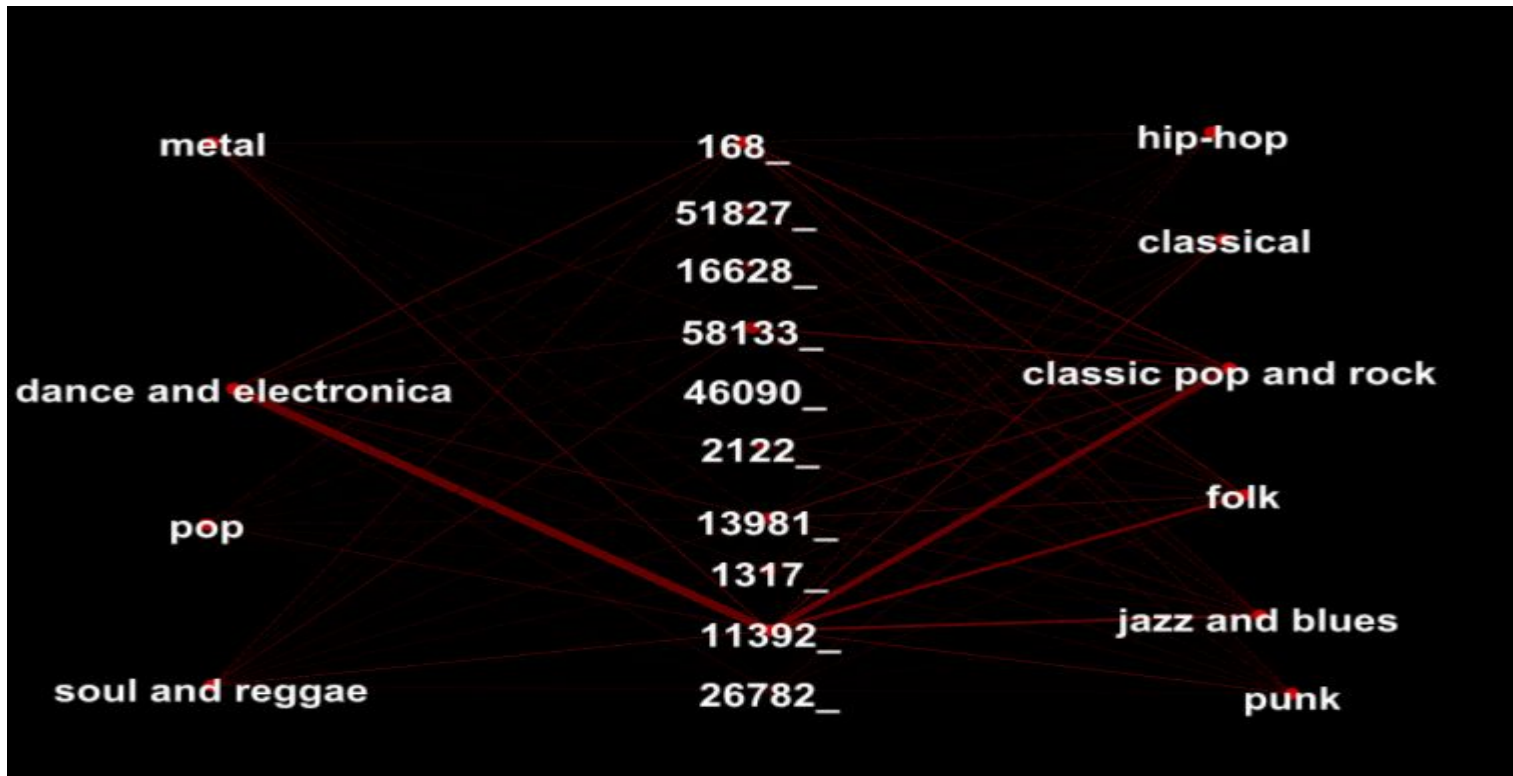
- First step was to select features that are more indicative and cluster data using these values
- Based on results from first run, select different combination of features or features not considered previously
- Setup used
 - Apache Mahout on Amazon EC2
 - 60k songs from Million Song Dataset
 - Distance metric used is Cosine Similarity
 - Number of clusters = Number of observed genres = 10

What did we find?

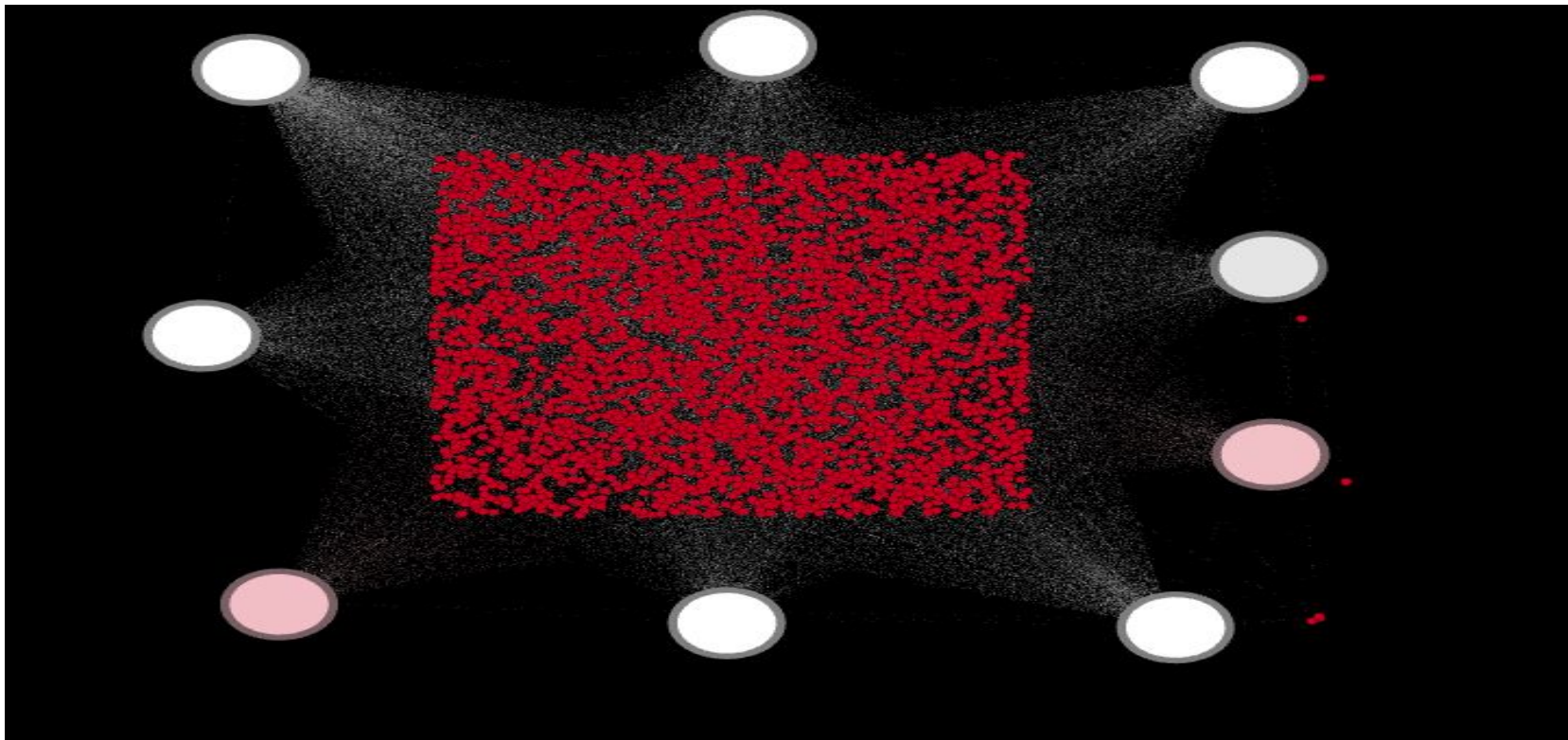


Features like energy, loudness, tempo,
song key, timbre etc. yield upsetting
results

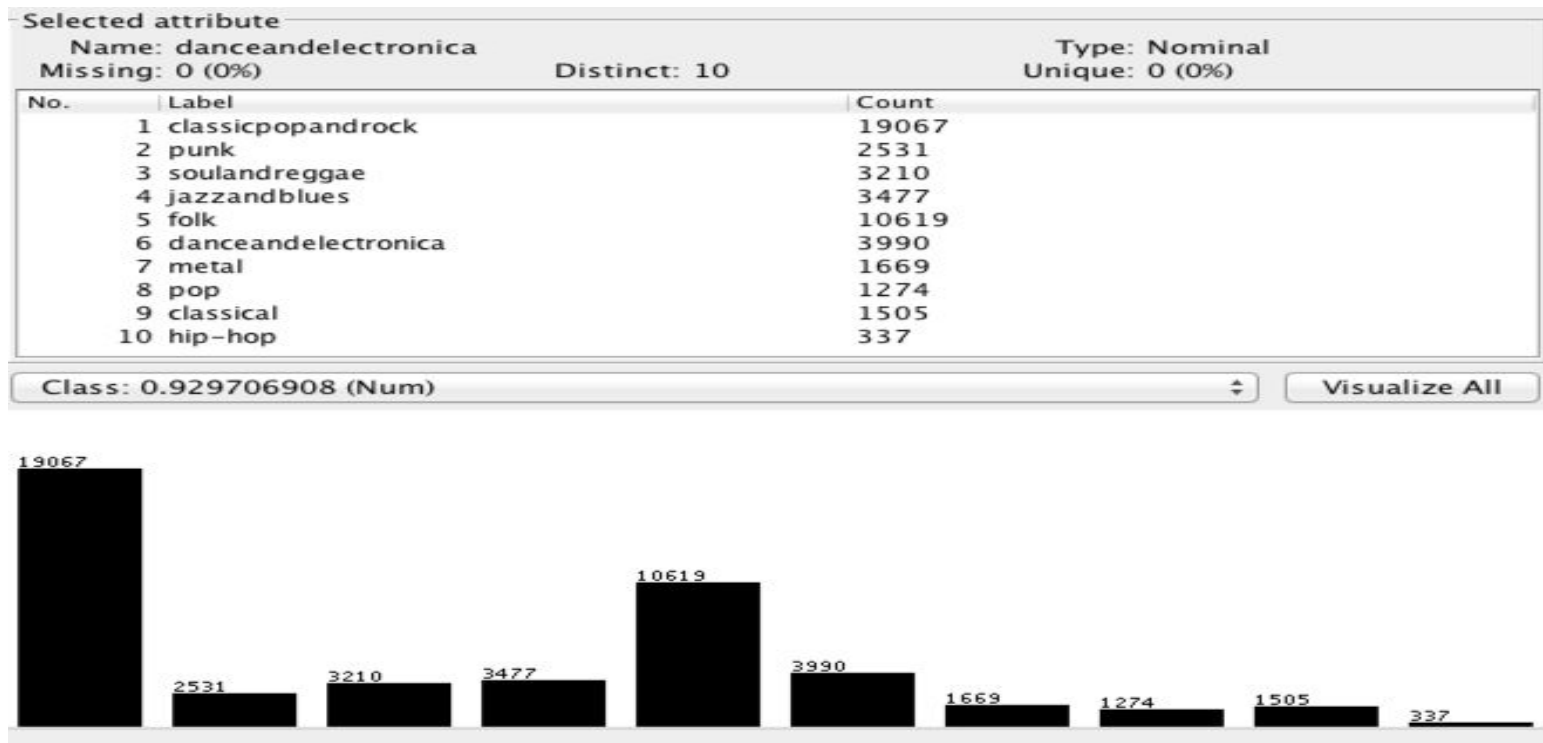
Cluster Visualization : Song features

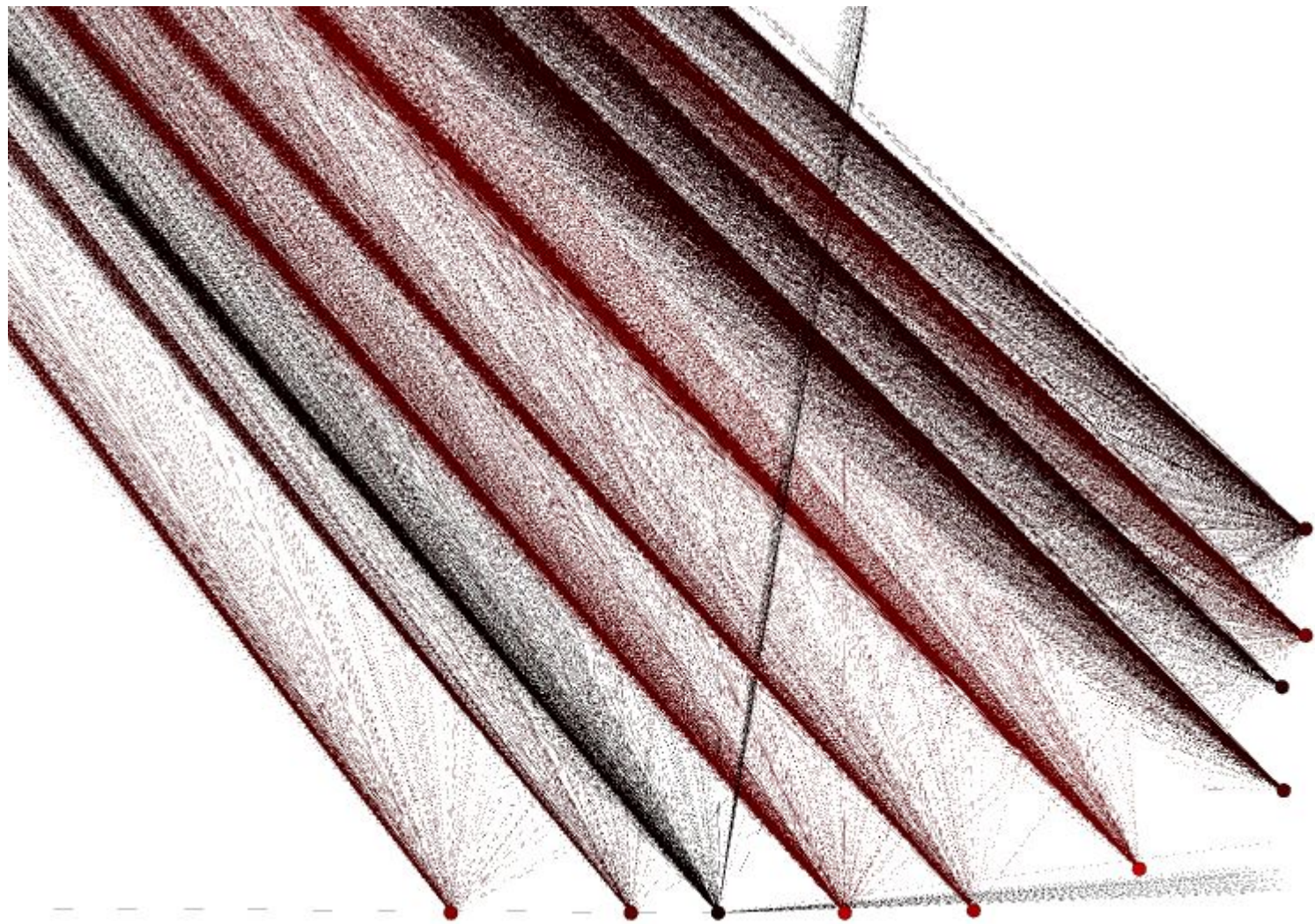


Cluster Visualization : Artists



Distribution of Labels





Comparing the two techniques

- Evaluation criterion
 - $Q = -\log (1/N) \sum (i - c_i)^2$
 - c_i = cluster center of point i , N = total no.of data points
- For song features, $Q = 7.462$
- For just using the artist, $Q = 7.499$

Now let's look at how Mahout scored as opposed to Spark MLlib

Mahout vs. MLlib - Million Songs

- We used Spark's MLlib library for scala.
- We do not consider the artist feature in the clustering.
- After basic preprocessing it is simple to execute K-means by specifying the number of trials, maximum iterations, initialization mode, etc.
- 10 clusters were created in ~ 20 seconds with the sum of squared distances of points to their nearest center of 1.025 (Within set sum of squared error - WSSSE).

Naive-Bayes

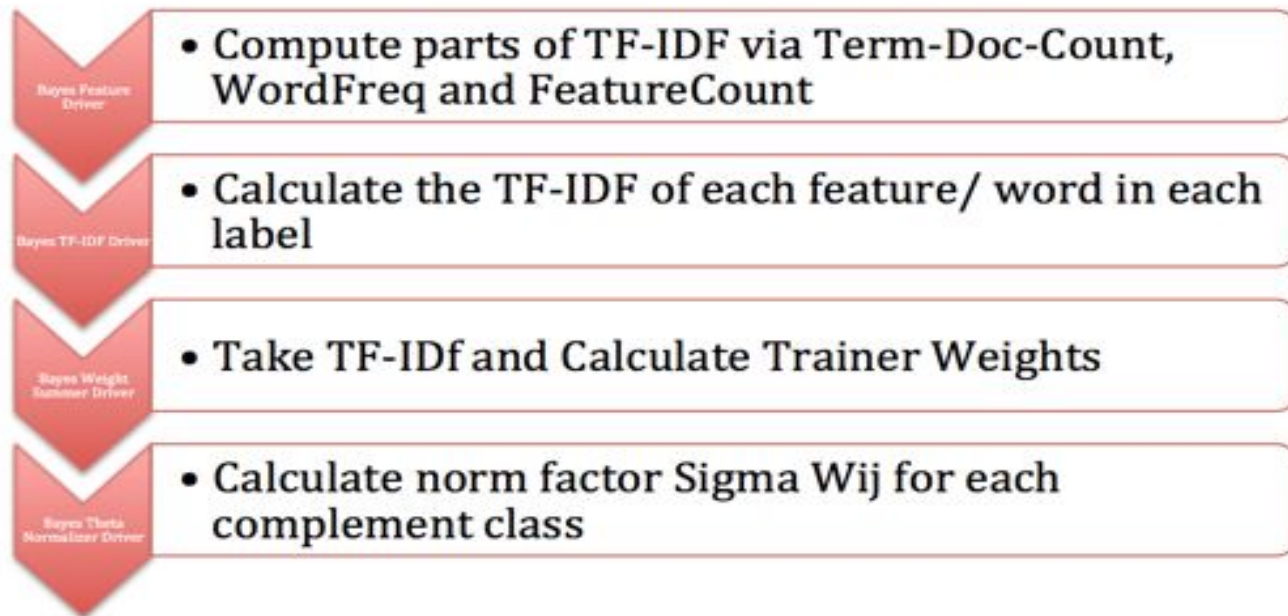
- Simple Probabilistic Classifier based on:
 - Applying Baye's theorem (From Bayesian statistics).
 - Assumes that presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature.
- Both Spark MLlib and Mahout have the Naive Bayes classifier.
- Runs in parallel and single pass.

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

$$p(C_k|\mathbf{x}) = \frac{p(C_k) p(\mathbf{x}|C_k)}{p(\mathbf{x})}$$

BayesDriver Training Workflow

Naive Bayes Training Map Reduce Workflow in Mahout



BayesDriver Training Workflow

Our Implementation

1. Text based

- a. separate and sample raw data into training and test set.
- b. generate sequence files first
- c. key values pairs as **Text.class**
 - i. key: /genre/id (/classic pop and rock/3600)
 - ii. value: any values like artist_name, loudness, and tempo separated by whitespace
 - iii. eg. BlueOysterCult -8.697 155.007
- d. generate vector files with ***mahout seq2sparse***
- e. train and test classifier with ***mahout trannb*** and ***mahout testnb***
- f. Widely applicable but low accuracy

BayesDriver Training Workflow

Our Implementation

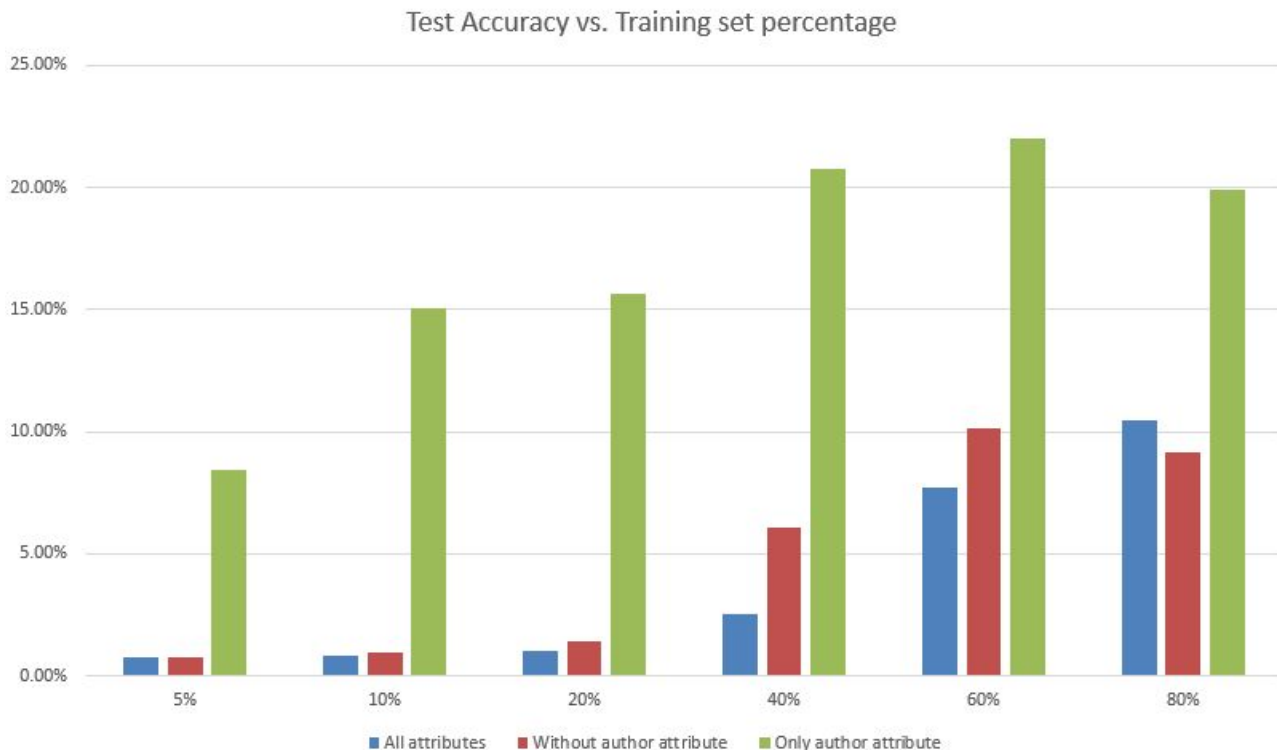
2. Numeric based

- a. directly generate vector files
- b. key as **Text.class** and value as **VectorWritable.class**
 - i. key: /genre/id (/classic pop and rock/3600)
 - ii. value: put numeric values in each row inside **DenseVector** and put vectors inside **VectorWritable**.
 - iii. eg. 6,0,45,235,158
- c. train and test classifier with ***mahout trannb*** and ***mahout testnb***
- d. Accurate but dataset dependent


Naive Bayes Performance - Mahout



Naive Bayes Performance - Mahout

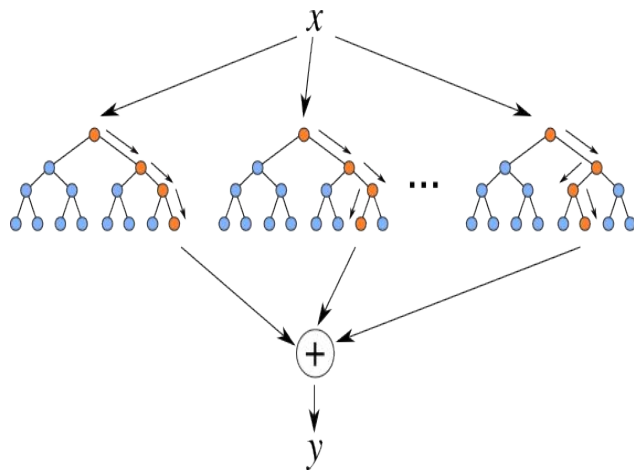


Naive Bayes Performance - MLlib

- Naive Bayes cannot read negative values.
We had to transform the entire dataset again.(!!) 
- Implementation much more simple to code.
- Did not consider the author attribute in any of the trials.
- Results showed an accuracy of 20% with a random 80-20 train-test split.

Random Forest

- Ensemble learning method.
- Created from a multitude of decision trees (each trained uniquely) and aggregating the results of each of the trees to output a single value.
- Both Mahout and MLlib support Map Reduce versions of Random Forest.



Random Forest - Mahout

- Dataset transformations - delimiters and descriptor file
- Requires descriptors similar to ARFF format
- Partial implementation
- Runs only in single instance mode
- Documentation incomplete
- Map jobs time out or never complete

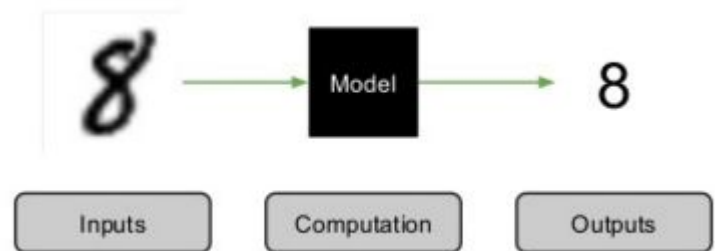


Random Forest - MLlib

- Data needs to be in LIB SVM format. (Lot of preprocessing work again!!)
- Implementation is straightforward to use in Scala
- Trained with 100 Trees
- Accuracy of 55%.

Dataset II - MNIST

- Dataset for handwritten digits images.
- A popular dataset used in many benchmarking studies.
- Each image is scaled to 28 x 28 size, computing center of pixels and translating the point to the center of the image.
- 60,000 training samples
- 10,000 test samples
- 10 classes
- 784 features



K-Means Clustering

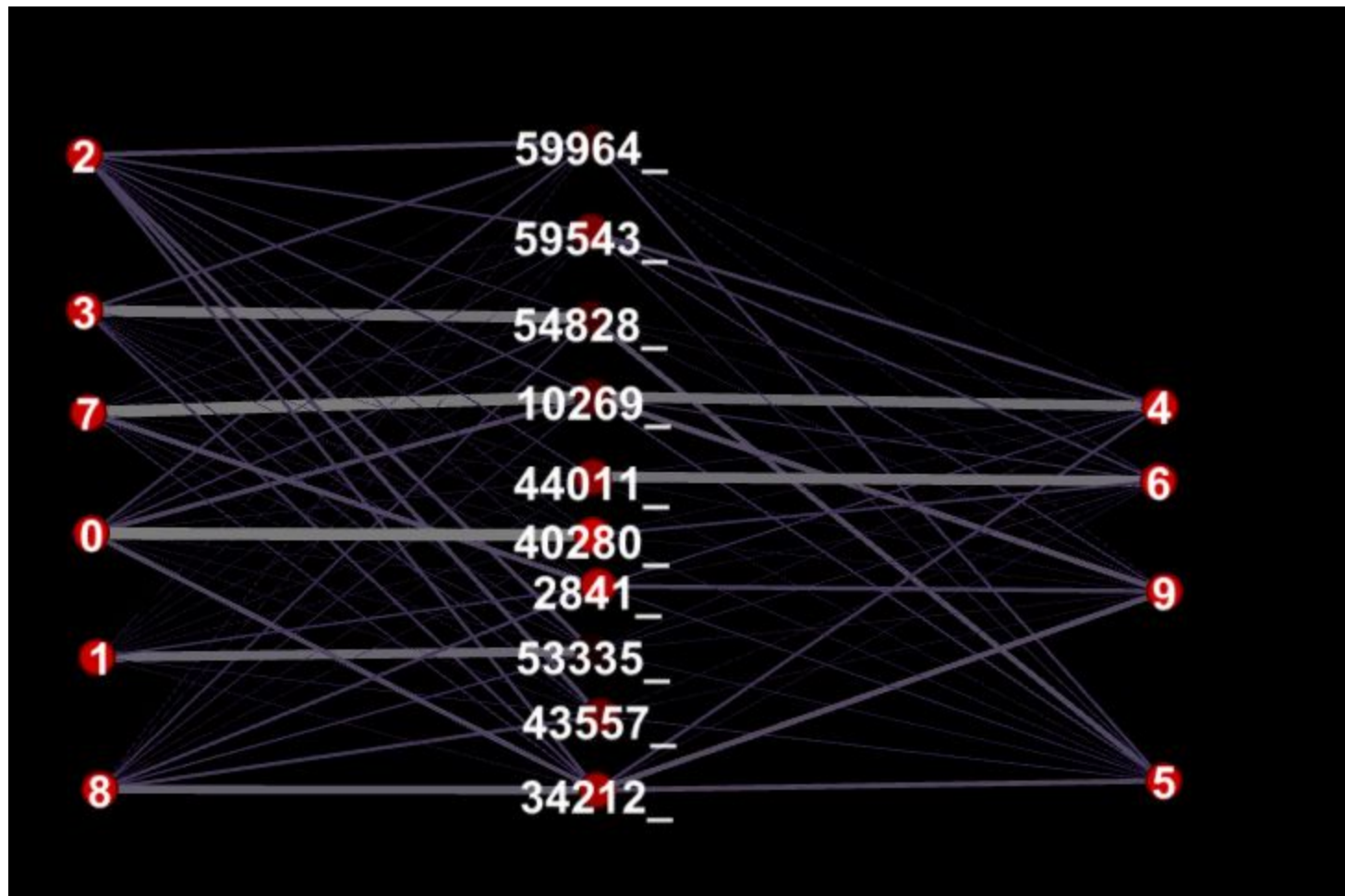
- Purpose here is to understand the distinctness of features
- Setup used
 - Apache Mahout on Amazon EC2
 - 70k image samples from MNIST Dataset
 - 784 pixels(features) per image
 - Distance metric used is Cosine Similarity
 - Number of clusters = Number of observed digits = 10

Results?

Second time's a charm!

Success!





How do you define success?

- Quality factor(Q) = 2.376
- Despite being lower than 7.3 and 7.5(from Million Song dataset)
- Considerably more number of features
- Significantly more data points

Mahout vs. MLlib - MNIST

- We used the same setting for each trial (M1 large instance on AWS)
- Could not yet compare cluster quality results

Metric	Mahout	Spark
Run Time on 1 Slave	113s	136s
Run Time on 5 Slaves	102s	90s
Run Time on 10 Slaves	100s	80s
Quality	2.376 Quality Factor	1.5345 WSSSE

Naive Bayes Performance

Metric	Mahout	MLlib
Test Accuracy	83.65%	83.65%
Run Time	11.46s	10.1s

Random Forest Performance

- Mahout's random forest fails again.
- MLlib is easy to use once the data is in Lib SVM format.
- Important parameters to tune - maxDepth and numTrees

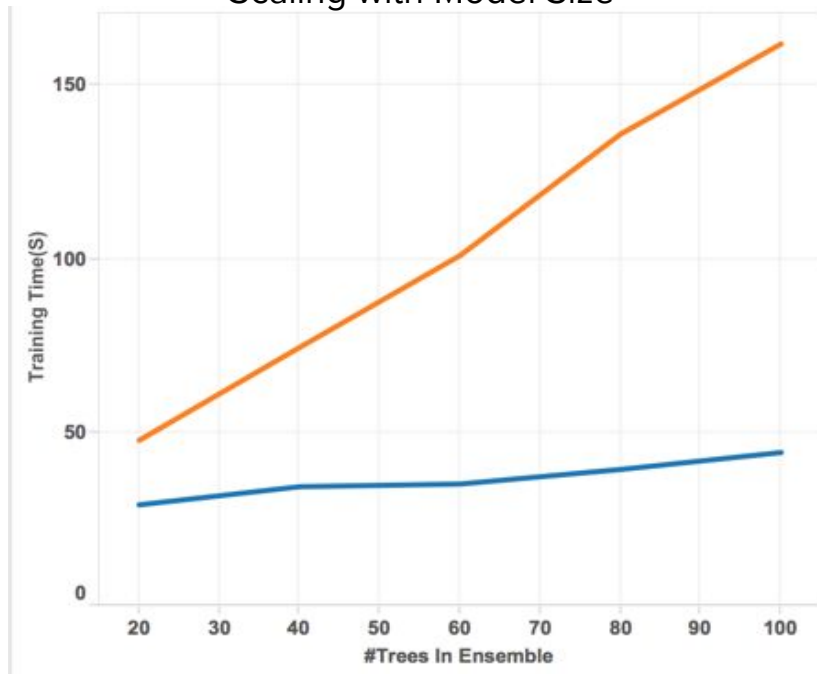
Metric	Mahout	MLlib
Test Accuracy(Default parameters)	87%	84%
Test Accuracy (10 depth, 8 bins)	73%	94.7%
Run Time on 1 Slave	8m	167s
Run Time on 5 Slaves	2m	110s

Results and Findings

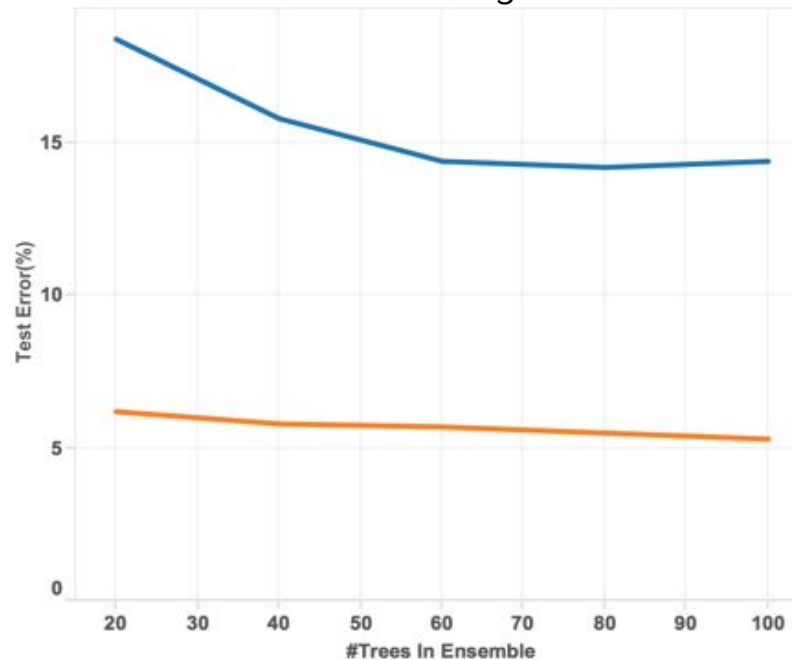
- Accuracy performances of the implementations.
- Runtime and scalability analysis.
- Differences in implementations and pre requisites.

Scaling model size: Training time and Test error of Random Forest

Scaling with Model Size



Lower error with larger ensembles



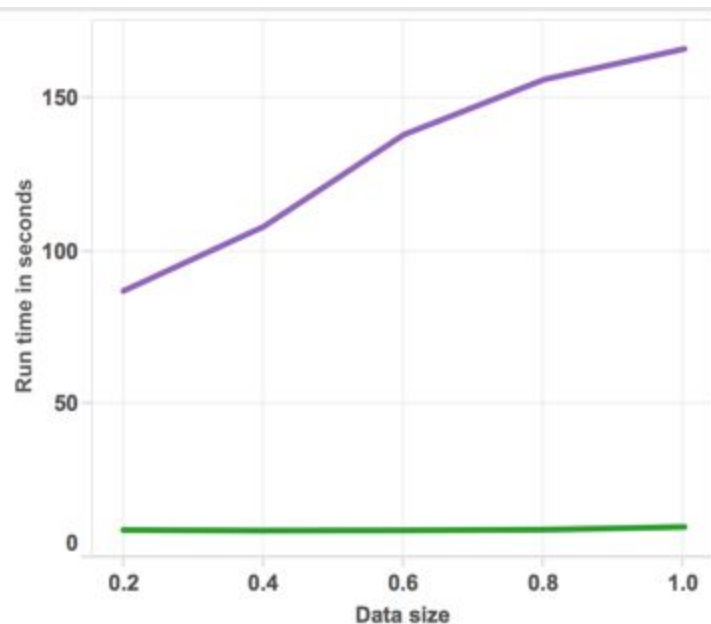
Depth

■ 5

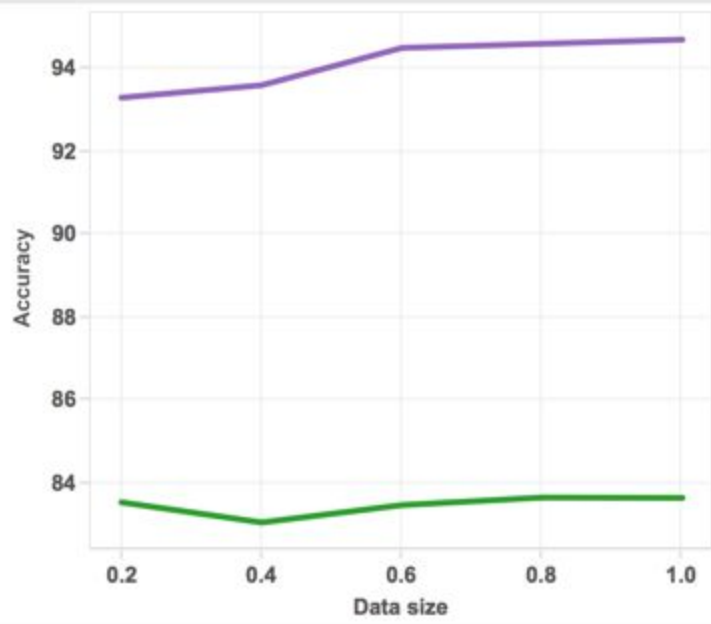
■ 10

Scaling training set size: Training time and Test error

Scaling with training set size



Higher accuracy with more training data



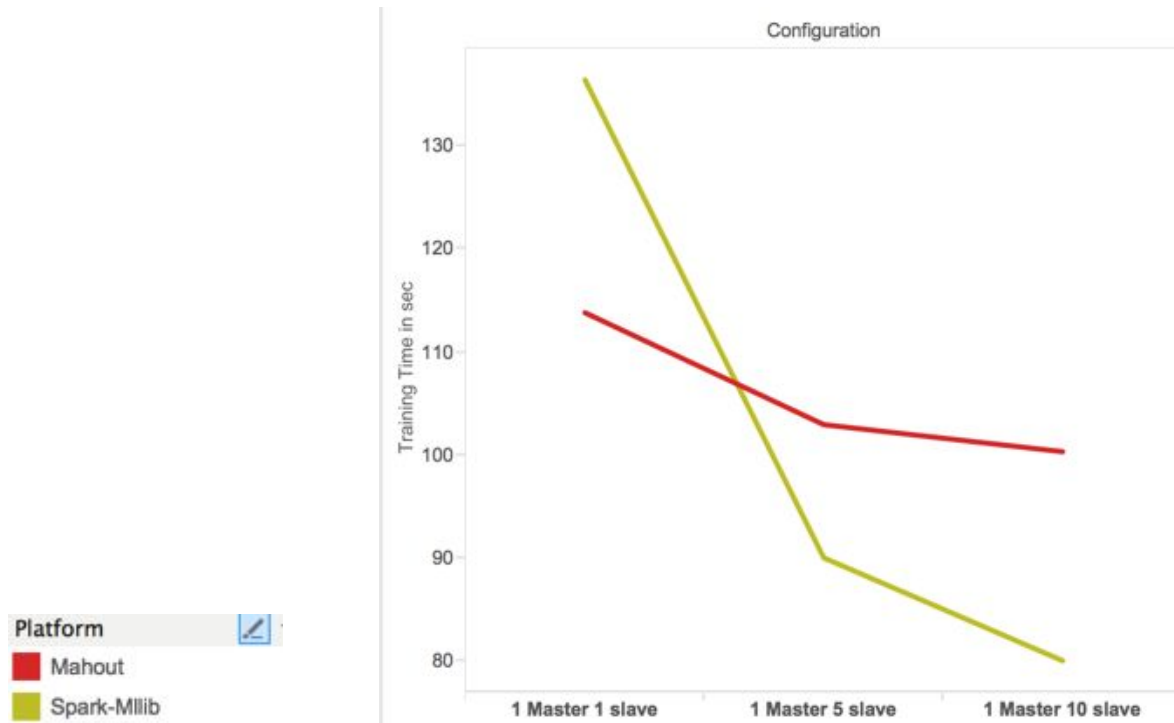
Algorithm

■ Naïve Bayes

■ Random Forest

MLLib vs. Mahout: Overview

Runtime statistics on MNIST- K - Means & Naive Bayes(TBD) on AWS - EMR clusters



Hadoop Configuration

Memory tuning issues may appear as one or more of the following symptoms:

- The job gets stuck in mapper or reducer phase.
- The job hangs for long periods of time or finishes much later than expected.
- The job completely fails and or terminates.

These symptoms are typical when the master instance type or slave nodes run out of memory.

Hadoop Configuration

Three options:

a. Limit the number of simultaneous map/reduce tasks

In Amazon EMR, a JVM runs for each slot and thus a high number of map slots require a large amount of memory

```
--args -m,mapred.tasktracker.map.tasks.maximum=maximum number of simultaneous map tasks
```

```
--args -m,mapred.tasktracker.reduce.tasks.maximum=maximum number of simultaneous reduce tasks
```

Hadoop Configuration

Three options:

b. Increase the JVM heap size and task timeout

Tasks are run within containers launched by YARN. We can set the memory size of the container being used to run the map or reduce task. If the task grows beyond this limit, YARN will kill the container.

```
mapreduce.{mapreduce}.memory.mb
```

To execute the actual map or reduce task, YARN will run a JVM within the container. The hadoop property `mapreduce.{mapreduce}.java.opts` is intended to set max heap size of the JVM.

Consequently, you should ensure that the heap you specify in `{mapreduce}.java.opts` is set to be less than than the memory specified by `{mapreduce}.memory.mb`.

Hadoop Configuration

Three options:

c. Increase NameNode heap size

- It is rare that NameNodes will get out of memory errors
- If it does, you need to change configuration while bootstrapping the cluster using json configuration.

Hadoop Configuration

Results:

mapreduce.map.memory.mb	mapreduce.map.java.opts	Time	Memory
2048	256	13m 18s 182	Physical memory (bytes) snapshot=14696734720 Virtual memory (bytes) snapshot=50950901760 Total committed heap usage (bytes)=9413066752
2048	1536	12m 47s 1	Physical memory (bytes) snapshot=20322983936 Virtual memory (bytes) snapshot=133063290880 Total committed heap usage (bytes)=20627587072
1024	700	8m 18s 185	Physical memory (bytes) snapshot=18557845504 Virtual memory (bytes) snapshot=79443869696 Total committed heap usage (bytes)=18265145344
1024	400	8m 41s 201	Physical memory (bytes) snapshot=17288671232 Virtual memory (bytes) snapshot=60284788736 Total committed heap usage (bytes)=12152995840
512	300	8m 55s 590	Physical memory (bytes) snapshot=14736785408 Virtual memory (bytes) snapshot=53693014016 Total committed heap usage (bytes)=10431758336
32	21	10m 49s 603	Physical memory (bytes) snapshot=9687326720 Virtual memory (bytes) snapshot=36020396032 Total committed heap usage (bytes)=1209008128

Hadoop Configuration

Results:

Effect of changing `mapred.min.split.size` - If the average mapper running time is shorter than one minute, you can increase the `mapred.min.split.size`, so that less mappers are allocated in slot and thus reduces the mapper initializing overhead. The number of partitions is controlled by the `-Dmapred.max.split.size` argument that indicates to Hadoop the max. size of each partition.

<code>mapred.min.split.size</code>	Time
18742310	1m 29s 877
1874231	8m 33s 661

Hadoop Scaling - Random Forest

Slaves	Parameter	Run Time
1 slave	<code>mapred.min.split.size = 18742310</code>	1m 29s 877
5 slaves	<code>mapred.min.split.size = 18742310</code>	1m 02s
1 slave	<code>mapred.min.split.size = 1874231</code>	8m 33s
5 slaves	<code>mapred.min.split.size = 1874231</code>	2m 2s

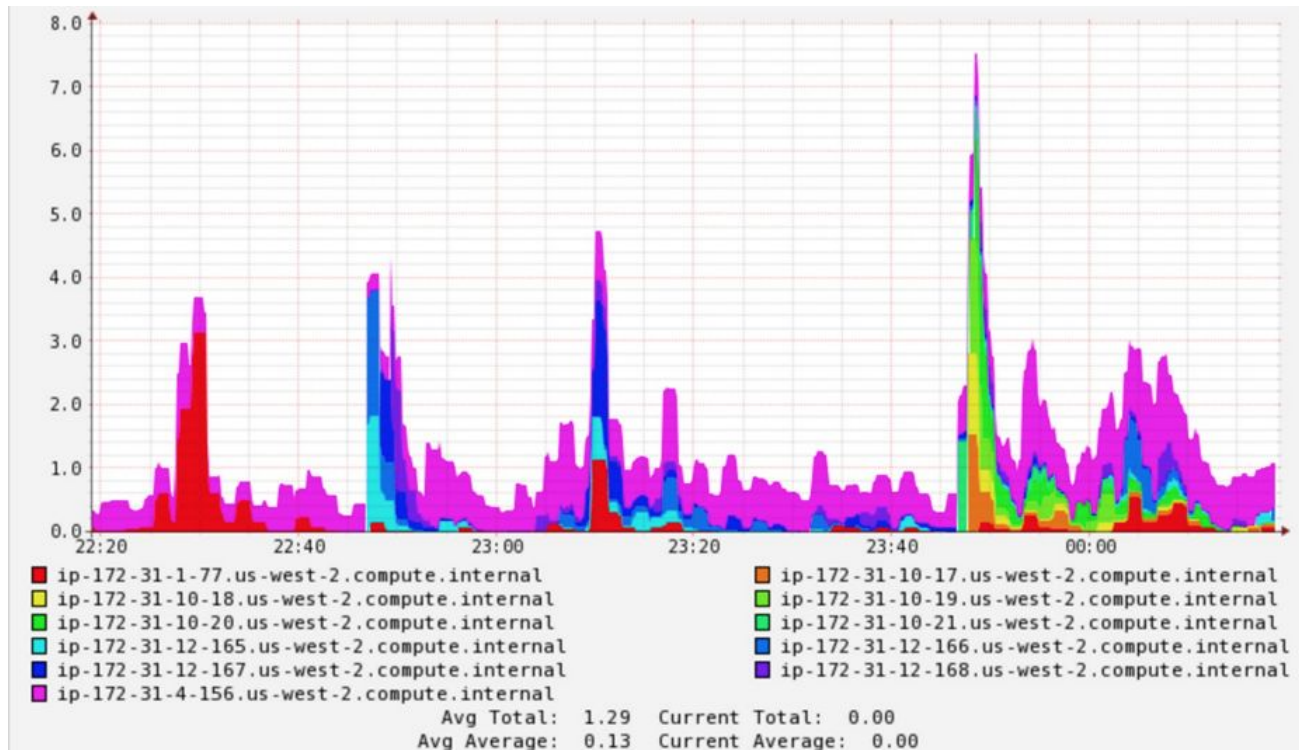
Hadoop Configuration

Random Forest

#Trees	Time	Accuracy
20	8m 24s 127	73.09
40	8m 27s 711	84
60	8m 29s 33	85.1
100	8m 33s 661	87.13

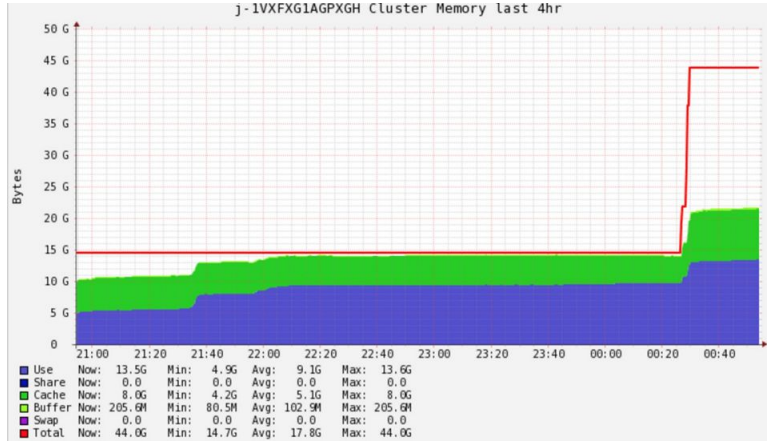
Cluster Usage Analytics using Ganglia

Aggregated CPU Load when running Mahout on EMR clusters

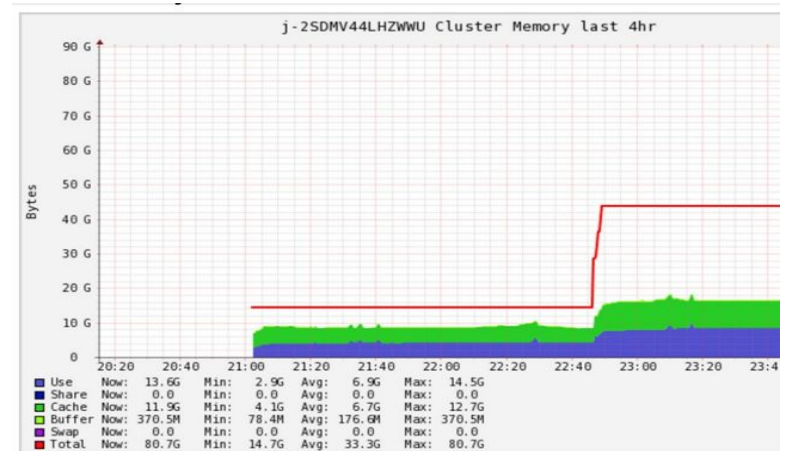


Cluster memory analysis

Spark - MLlib



Hadoop - Mahout



In Summary

- 2 Datasets
- 3 algorithms
- 2 distributed platforms
- 1 happy project group :)

Conclusions

- Both environments have similar algorithm implementations
- But vary a lot in parameters provided, data formats, results format, etc.
- Hadoop requires lot of tuning to get good performance.
- Spark was found to be better documented and easier to obtain initial results.

Some of the lessons

- Each implementation has its own prerequisites that need to be complied.
 - Eg: MLlib's random forest needs lib svm input format.
- There are many restrictions on the data points.
 - Eg: MLlib's Naive Bayes cannot read negative values.
- Documentation is incomplete sometimes.
 - Eg: Mahout's random forest does not specify the delimiters.
- Different measurement metrics
 - Eg: K-means on Mahout and Spark provide different cluster quality metrics by default.

Thank you!