Using Bayesian Generative Models with Apache Spark to Solve Entity Resolutions Problems (DeDup, Merging, Uniqueness) at Scale

Charles Adetiloye & Timo Mechler

MavenCode

About MavenCode

MavenCode is an Artificial Intelligence Solutions company located in Dallas, Texas - we do training, product development, and consulting services in the following areas:

- Provisioning Scalable Data Processing Pipelines and Cloud Infrastructure Deployment
- Development & Deployment of Machine Learning and Artificial Intelligence Platforms
- Streaming and Big Data Analytics Edge-IoT and Sensors

About Us!

Charles is an ML Platforms Engineer at MavenCode. He has well over 15 years of experience building large-scale, distributed applications. He has extensive experience working and consulting with companies like Google, Twitter, Lightbend and a lot of fortune 500 companies and startups

twitter.com/cadetiloye

Timo is an Engagement Manager and Solutions Architect at MavenCode. He has close to a decade of data modeling experience working both as an analyst and strategist in the energy commodities sector. At MavenCode he now works closely with clients and the rest of the consulting team to solve interesting big data modeling and infrastructure challenges.

twitter.com/mavencode

Our Goal

Identifying Profiles that belong to the same User/Entity but occurring in multiple places across different data sources

First name	Middle name	Last name	DOB	Address
James	Doe	Smith	11-01-2000	123 main street, Southlake Texas 76029

First name	Middle name	Last name	DOB	Address
James	D.	Simth	11-01-2000	4th street, Houston Texas 76029

First name	Middle name	Last name	DOB	Address
James	Doe	Smith	01-11-2000	123 Collins, Southlake Texas 76029

Why data duplication occurs

1. Incomplete profile attributes or missing attributes

2. Data coming from disparate sources

3. Spatio-temporal data updates with entries arriving at different times

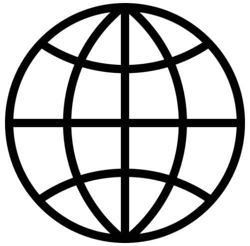
4. Unavoidable human errors

Data deduplication challenges faced today

Datasets are increasing in size leading to Quadratic Complexity

Integration of legacy and modern systems

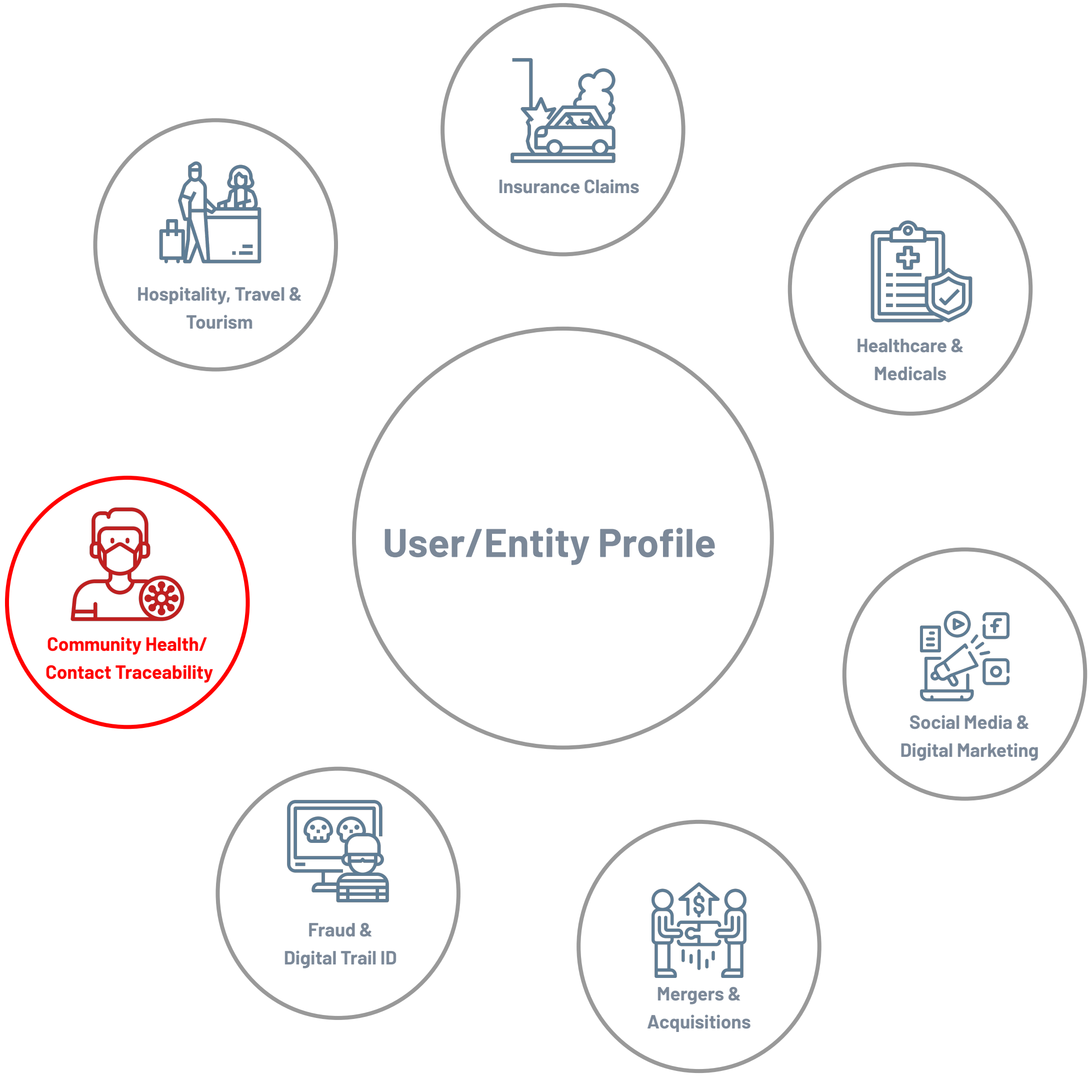
Building scalable systems



Inexact or “fuzzy” matching

John Smith <-?-> Jon Smith

Where do we see this problem?



Current solutions for deduplicating data

a) Naive or Brute Force approach:

Element Wise Comparison

Table A

First Name	Last Name	DOB	Address
James	Smith	11-01-1900	123 Main Street, Southlake, Texas 76092
Sue	Doe	01-02-1905	456 South Street, Southlake, Texas 76092
Michael	Johnson	07-07-1907	111 Data Circle, Dallas, Texas, 75001

Table B

First Name	Last Name	DOB	Address
James	Smith	11-01-1900	123 Main Street, Southlake, Texas 76092
Sandy	Doe	02-15-1920	419 South Street, Southlake, Texas 76092
Michael	Johnson	07-07-1907	111 Data Circle, Dallas, Texas, 75001

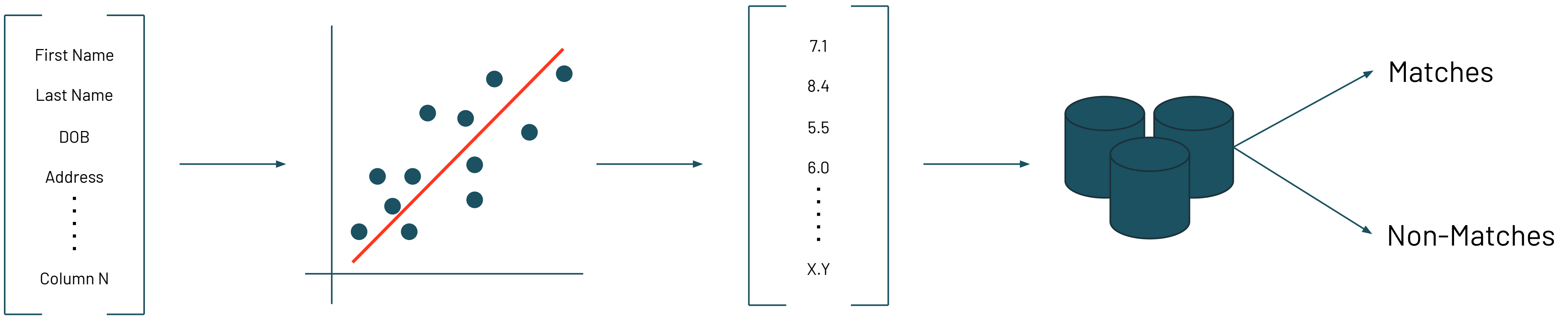
Pro(s):
- Easy to implement

Con(s):
- Difficulty scaling to large datasets
- Challenges with inexact/"fuzzy" matches

Current solutions for deduplicating data

b) Supervised learning (with labeled data):

Logistic Regression - Deterministic



1. Determine Comparison Features / Attributes (Columns)
2. Run Regression Against Labeled Training Data (Candidate Pairs)
3. Regression Coefficients Represent Feature Weights
4. Run Trained Model Against Other Candidate Pairs and calculate total weight
5. Classify Pairs Into Matches & Non-Matches Based On Threshold

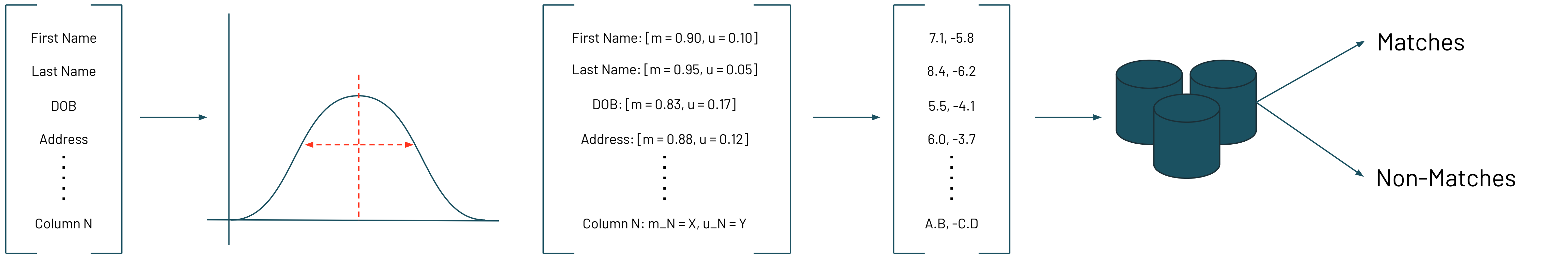
Pro(s):
- Easy to implement

Con(s):
- Requires high quality labeled training data
- May require tuning

Current solutions for deduplicating data

b) Supervised learning (with labeled data):

Naive Bayes Classifier - Probabilistic



1. Determine Comparison Features / Attributes (Columns)
2. Naive Bayes Classifier determines two types probabilities based on labeled training data:
   u = probability an attribute in a non-matching candidate pair agrees
   m = probability an attribute in a matching candidate pair agrees
3. Match and Non-Match weights are calculated from u and m probabilities
4. Run Classifier Against Other Candidate Pairs and calculate total weight
5. Classify Pairs Into Matches & Non-Matches Based On Threshold

Pro(s):

- Less tuning & intervention needed

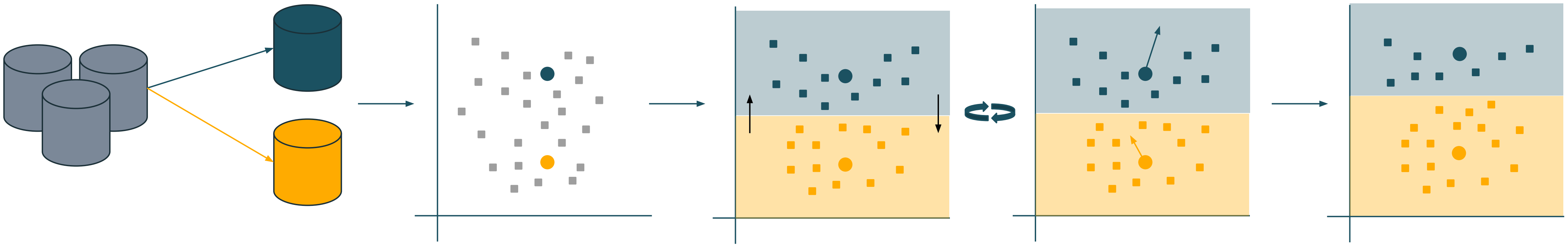
Con(s):

- Requires high quality labeled training data
- Does not scale well to large datasets

Current solutions for deduplicating data

c) Unsupervised Learning (no labeled data)

K-Means Clustering



1 Split Data (Candidate Pairs) into Matches and Non - Matches (K = 2)

2 Choose Center (Mean) Values or let algorithm assign them

3 Each Data Point is Assigned to Closest Center (Mean)

4 Centers (Means) are recalculated; Step 3 is then repeated

5 Once no additional data point reassignment is needed after step 4 has taken place, the algorithm has converged

Pro(s):

- Does not require labeled training data

Con(s):

- Heavily dependent on what centers are chosen, results can vary from run to run
- Convergence may not be optimum
- Better suited for initial assignment of partitions before further analysis

Current solutions for deduplicating data

d) Blocking Keys to reduce search space

Blocking

First Name	Last Name	Sex	DOB	Address
James	Smith	M	11-01-1900	123 Main Street, Southlake, Texas 76092
Sue	Doe	F	01-02-1905	456 South Street, Southlake, Texas 76092
Michael	Johnson	M	07-07-1907	111 Data Circle, Dallas, Texas, 75001
John	Adams	M	11-05-1920	10 Any Drive, Dallas, Texas, 75002
Cindy	Kennedy	F	12-07-1934	456 Main Street, Southlake Texas 76092
Ashley	Michaels	F	01-01-1911	777 Data Court, Dallas, Texas 75001

1 Choose Blocking Key

First Name	Last Name	Sex	DOB	Address
James	Smith	M	11-01-1900	123 Main Street, Southlake, Texas 76092
Michael	Johnson	M	07-07-1907	111 Data Circle, Dallas, Texas, 75001
John	Adams	M	11-05-1920	10 Any Drive, Dallas, Texas, 75002

First Name	Last Name	Sex	DOB	Address
Sue	Doe	F	01-02-1905	456 South Street, Southlake, Texas 76092
Cindy	Kennedy	F	12-07-1934	456 Main Street, Southlake Texas 76092
Ashley	Michaels	F	01-01-1911	777 Data Court, Dallas, Texas 75001

2 Split Dataset apart by Blocking Key

Note:

- Care must be taken when choosing blocking key(s); these should be features/attributes with high confidence of being correct (undistorted)

Current solutions for deduplicating data

d) Use string comparison methods for fuzzy matching

Common comparison methods

Levenshtein Distance (LD)

Number of single characters edits required to change string A into string B

```
LD(John, Jon)   = 1
```

Add "h" to Jon for strings to be equivalent

```
LD(John, Jack)  = 3
```

Add "o", "h", and "n" to Jack for strings to be equivalent

Jaro Distance (JD)

Edit Distance between two strings A and B; normalized between 0 and 1

$$JD = \begin{cases} 0 \text{ if no common characters, i.e. } m = 0 \\ \frac{1}{3} * [\ m / |s1| + m / |s2| + (m - t) / m\ ] \end{cases}$$

where,
m = number of matching characters
|s1| = length of string 1
|s2| = length of string 2
t = half of the number of transpositions

```
JD(Jason,Jasno)= ⅓ *[5/5 + 5/5 + (5-1)/5] =.9333
```

Longest Common Subsequence (LCS)

Longest subsequence of characters common to both string (need not be consecutive)

```
LCS(Mike, Michael) = MIE = 3
LCS(Tim, Timo)     = TIM = 3
LCS(Frank, Bob)    =       0
```

Phonetic Comparison

The similarity between two strings when sounded out

```
Ashley -> Ashlee
Leigh  -> Lee
```

Our Approach

Perform distributed probabilistic record linkage on Spark using a Bayesian Generative Model[1]

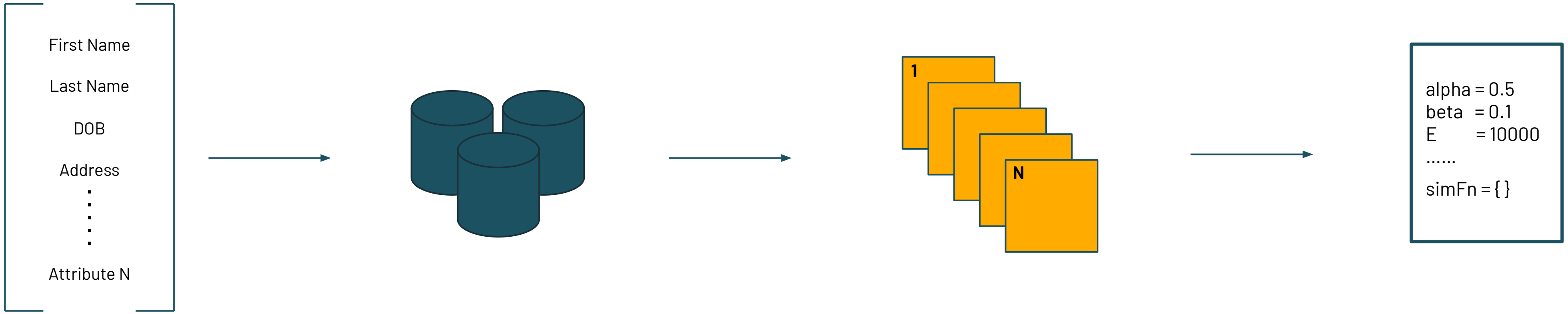
Key Features

- Completely Unsupervised - does not require large amounts of labeled data
- Supports both categorical and string data (can use string comparison functions)
- Preserves uncertainty between stages
- Scales across multiple compute nodes allowing for distributed computation and larger datasets

(1) "d-blink: Distributed End-to-End Bayesian Entity Resolution"; https://arxiv.org/pdf/1909.06039.pdf

Approach description

Data preparation & feature engineering



1 Choose appropriate attributes/features (columns) for comparison and chosen String Similarity Functions

Attributes/features with low distortion are preferred to simplify computation

2 Split dataset entities into n partitions, n proportional to number of nodes in the Spark cluster ensure balanced workload distribution.

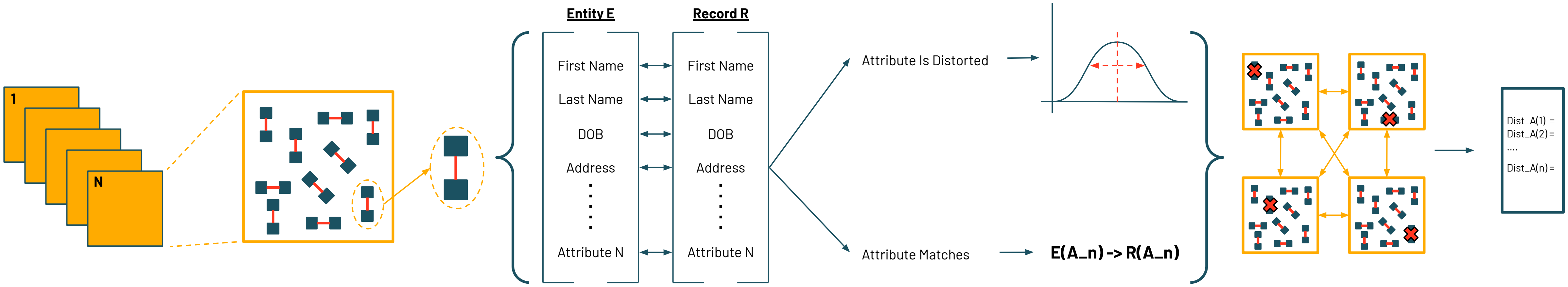
There is trade off between balanced partitions and making sure similar entities start in the same partition (to minimize movement)

3 Prepare configuration file for algorithm run, which includes:

- Hyperparameters (for distortion distribution)
- String Similarity Function definitions and cutoff thresholds
- Expected number of unique entities

Approach Description

High Level Overview[1]



1. A partition is chosen and then within that partition a given entity E linked to a record R is chosen
2. For each of the attributes A, a distortion indicator is first determined (0 or 1). If 0 (no distortion) the attribute A of E is copied to R (the attributes values A from E and R equal). If not 0, the value A for R is drawn from the A's distortion distribution. Then it is determined whether the given attribute has already been observed (0 or 1).
3. Entity - Record links continue to be updated in all partitions (entity attribute values and assigned records)

   Movement to new partitions occurs as needed.
4. Update distortion probability (distributions and summary statistics)

Note:

- Steps 1) - 4) is an iterative process proportional to length of linkage chain desired

[1]**CREDITS : d-blink: Distributed End-to-End Bayesian Entity Resolution (Marchant et al, 2019)**
Entity Resolution with Empirically Motivated Priors (Steorts, 2015)

Probabilistic Entity Model Representation

Entities (User Profiles): Datasets with Entities E, and having Attributes A, we assume that each Entity profile is independent and identically distributed

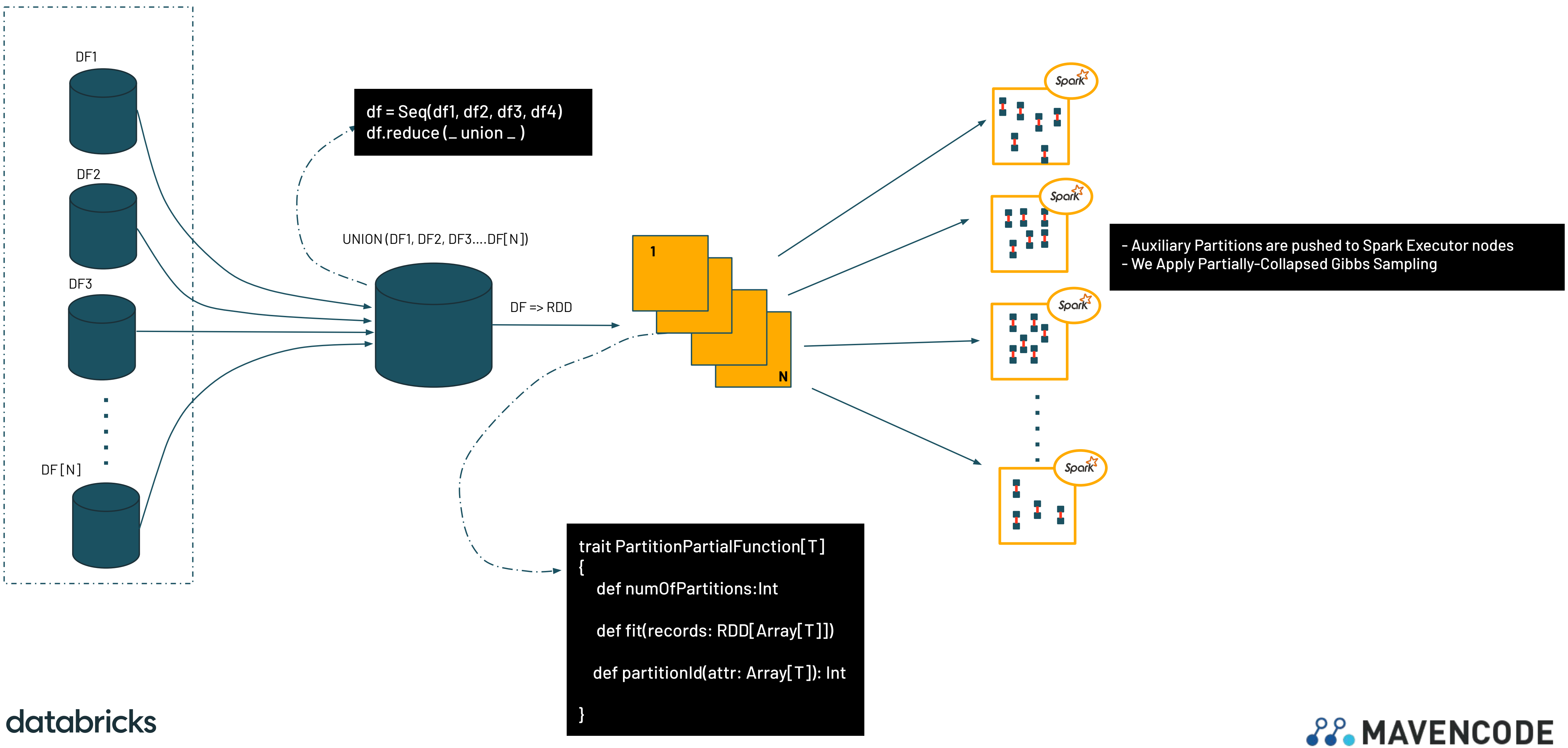
Partitions: We assume that our datasets can be almost evenly partitioned by a categorical attribute A which is part of the Entity Profile, this will allow us to do distributed parallel processing on the Spark cluster

Attribute Distortion: We assume the likelihood of a non-conformal distortion in the Entity or Profile Attributes

Profile Records: Representation of randomly selected Entity Attribute subject to distortion

d-blink: Distributed End-to-End Bayesian Entity Resolution (Marchant et al, 2019)

Implementation Pipeline

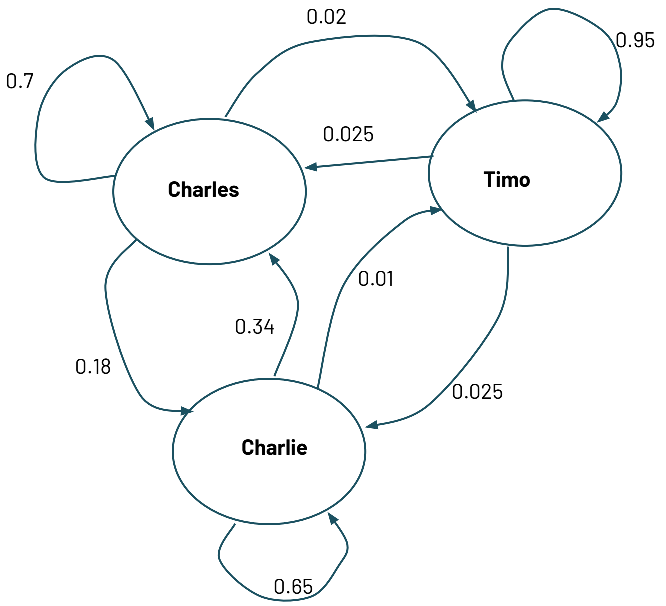


databricks

MAVENCODE

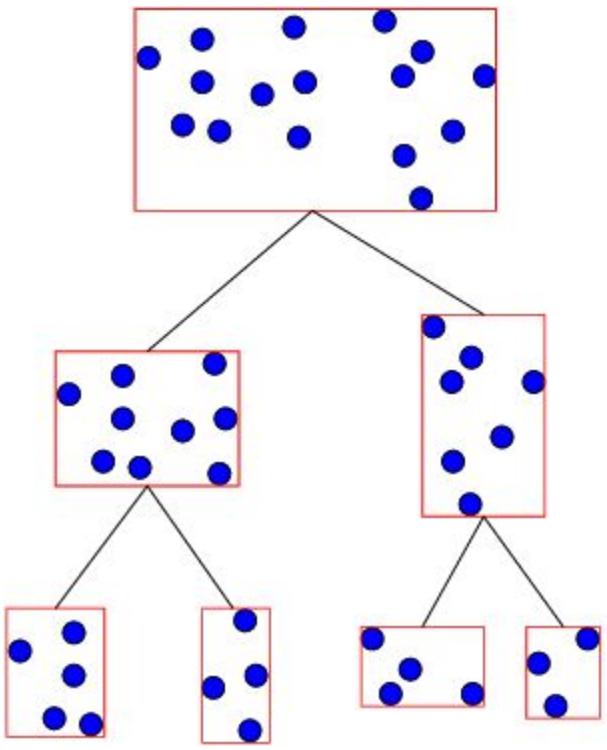
Bayesian Generative Model

A way of modeling how the set of **Observed Data** could have been produced from a set of known **Prior Information.** $X_1 = PX_0$



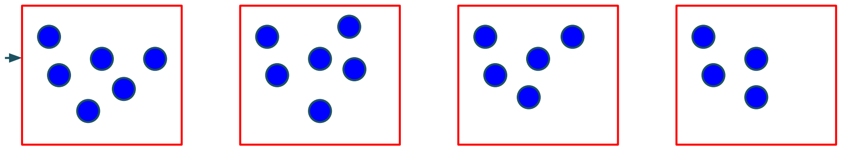
$$\begin{bmatrix} x1_1 \\ x1_2 \\ x1_3 \end{bmatrix} = \begin{array}{c} \\ \text{Charles} \\ \text{Timo} \\ \text{Charlie} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Charles} & \text{Timo} & \text{Charlie} \end{array} \\ \begin{bmatrix} 0.7 & 0.02 & 0.18 \\ 0.025 & 0.95 & 0.025 \\ 0.34 & 0.01 & 0.65 \end{bmatrix} \end{array} \begin{bmatrix} x0_1 \\ x0_2 \\ x0_3 \end{bmatrix}$$

Partition partial function: KD-Tree partitioning

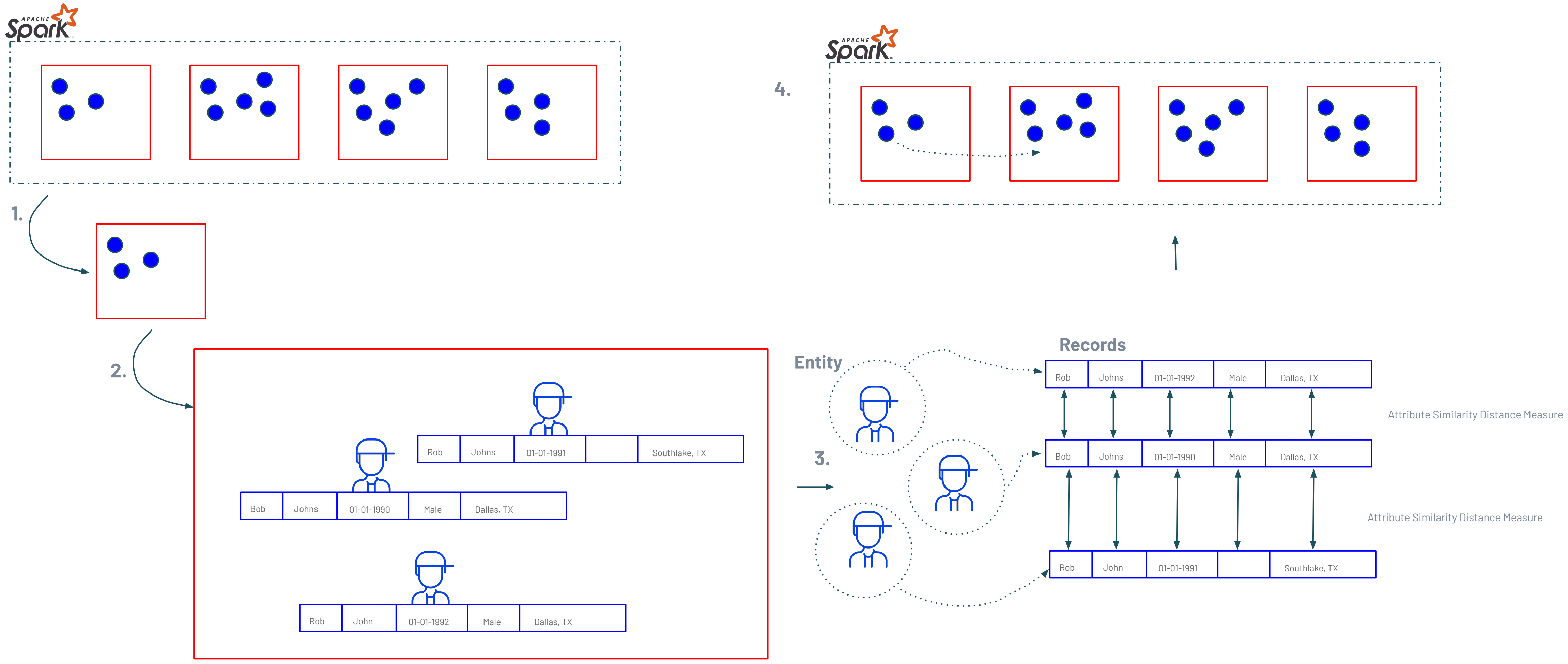


```
class KDTreeParition[T : Ordering](levels: Int, attr: Traversable[Int])
{
    def numOfPartitions:Int

    def fit(records: RDD[Array[T]])

   def partitionId(attr: Array[T]): Int

}
```

Spark Cluster



Entity resolution & attribute similarity measures



CREDITS : Rebecca Steorts (resteorts.github.io)

databricks

MAVENCODE

Attribute Similarity measures

- Levenshtein Metric
- Jaro-Winkler Metric
- Jaccard Metric
- Refined NYSIIS Metric
- Metaphone Metric
- Match-Rating Approach
- Geo-Spatial Distance

Demo

Algorithm running on GCP Dataproc cluster

Why Spark?

The Apache Spark framework was the right choice for us to to run this distributed record-linkage algorithm because:

- Spark is a battle tested distributed computing framework that is readily supported in different environments (cloud, on-premise, etc.)
- Variables updates using on a given partition of records and entities only depends on the variables on that same partition, allowing for distributed parallel computation across multiple worker nodes and sharing of variables when needed as Spark broadcast variable
- With well selected partitions we were able to evenly distribute our partitioned datasets on the Executor nodes, leveraging the full distributed processing power of Spark
- Easily Scalable, framework leveraging Spark RDD and Dataframe for efficient data distribution

Summary

Running a Bayesian Generative Model for data linkage and deduplication with Apache Spark has led us to conclude the following to date:

- By partitioning datasets and leveraging multiple nodes with Spark are we are able to achieve scalability to larger datasets and decrease run time
- Support for inexact of fuzzy matching via string comparison/distance functions
- Achieve acceptable match accuracy despite being an unsupervised modeling approach
- Easy cross-platform support by using Spark (major CSP's or on-premise)

Thank You!

If you are interested in record linkage, deduplication, and wrangling lots of data:

Drop us a mail	**hello@mavencode.com**
Visit Us Online	**https://www.mavencode.com**
Follow Us	**https://www.twitter.com/mavencode**

A big thank you to **Neil Marchant, Rebecca C. Steorts** and the open source community behind the project https://github.com/cleanzr/dblink

Q & A

If you are interested in record linkage, deduplication, and wrangling lots of data:

Drop us a mail	**hello@mavencode.com**
Visit Us Online	**https://www.mavencode.com**
Follow Us	**https://www.twitter.com/mavencode**

databricks