SPARK+AI
SUMMIT 2018

Building a Scalable Record Linkage System

with Apache Spark, Python 3, and Machine Learning

Nicholas Chammas and Eddie Pantridge
MassMutual

#Py6SAIS

The Business Problem

- What: Comprehensive view of the customer
- Why: Marketing and underwriting
- Problem:
  - Customer information scattered across many systems
  - No global key to link them all together
  - Variations in name and address

The Technical Challenge

- 330M+ records that need to be linked
- Records come from various systems with differing schemas
- No global key; SSN generally not available
- No differential extracts of source systems
- Need to link everything, all at once, every night

Record Linkage Prior Art

- Dedupe
  - Mature, sophisticated
  - Local processing
  - Previous team found did not scale enough

Record Linkage Prior Art

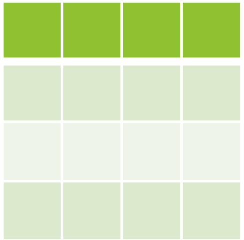
- Splinkr 2
  - In-house system built on Spark’s RDD API in late 2015 / early 2016
  - 8+ hours to run; problems with stability, code maintainability, and link quality
  - Successful as an experiment, but needed replacement for production

Splinkr 3

- All-new project started in late 2016, targeting Spark's DataFrame API
- Goal: Build a production-ready record linkage system, leveraging the lessons from Splinkr 2

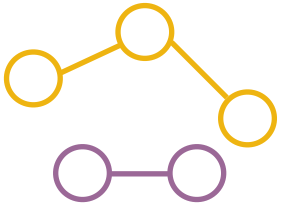
Record Linkage in the Abstract

Varied and
messy sources



Unify and
standardize

Generate pairs Score



Resolve
transitive links

Standardizing the Incoming Data

Full Name	…	ZIP
Nick Chammas		02138

First Name	Last Name	…	ZIP
Nicholas	Chammas		02138-1516

First Name	Last Name	…	ZIP
Nick	Chammas		02138
Nicholas	Chammas		02138

Generating Pairs Efficiently

- 330M records means (330M choose 2) ≈ 10^{16} possible pairs
- Need heuristic to quickly cut that 10^{16} down without losing many matches
- We accomplish this by extracting a “blocking key”
- We generate pairs only for records in the same block

Generating Pairs Efficiently

- Example:
  - Record: John Roberts, 20 Main St, Plainville MA 01111
  - Blocking key: JR01111
- Will be paired with: Jonathan Ray … 01111
- Won't be paired with: Frank Sinatra … 07030

Generating Pairs Efficiently

- Blocking as a crude model for predicting matches
  - We want a recall of 1.0
  - Don’t care about precision
  - Drastically shrink search space
- Blocking cuts down generated pairs from 10^{16} to 10^{8}

Generating Pairs Efficiently

```
blocked_people = (
    people
    .withColumn(
        '_blocking_key',
        blocking_key_udf('first_name', 'last_name', 'zip')
    )
)
```

Generating Pairs Efficiently

```
people_pairs = (
    blocked_people.alias('p1')
    .join(
        blocked_people.alias('p2'),
        on='_blocking_key')
    .where(
        concat(col('p1.source_name'), col('p1.source_pk')) <
        concat(col('p2.source_name'), col('p2.source_pk'))
    )
)
```

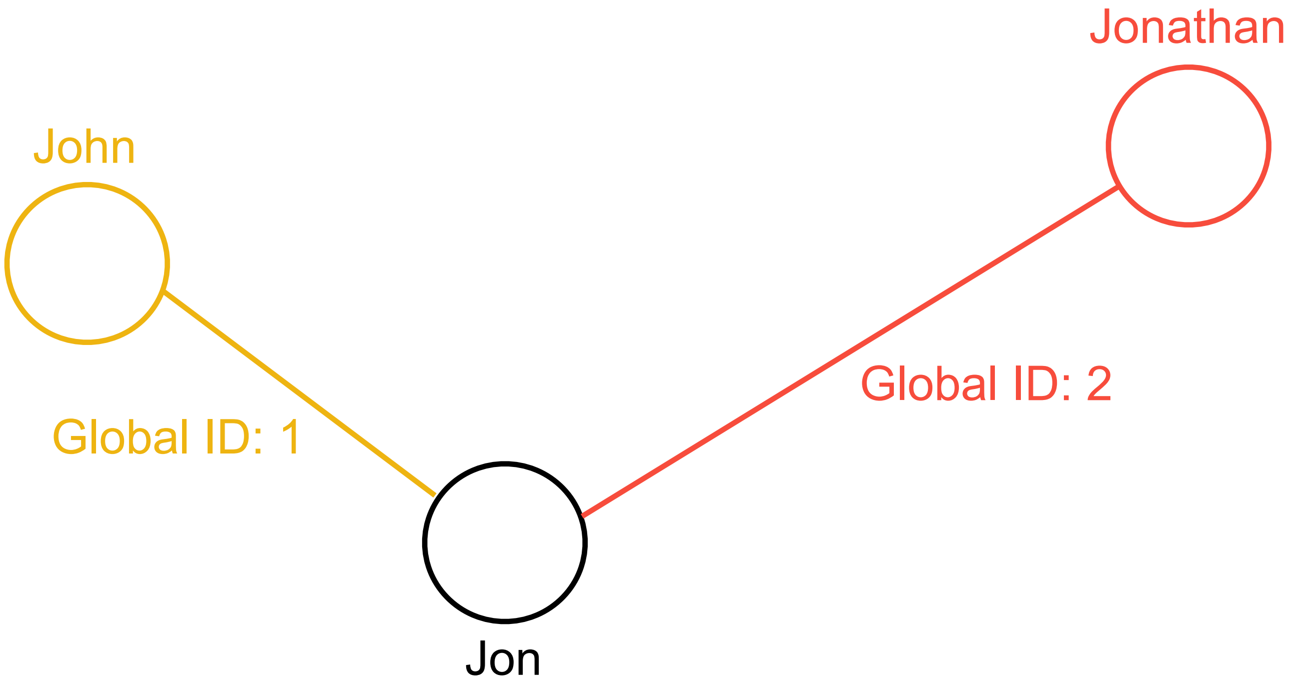
Identifying Matches

- Logistic regression model (Spark ML Guide)
- Trained on records where SSN was available
- Model features:
  - Phonetic equality on full name and city
  - String distance on full name, address, and city
  - Exact match on state and ZIP
- Jellyfish: Python library providing implementations of Metaphone, Jaro-Winkler, …

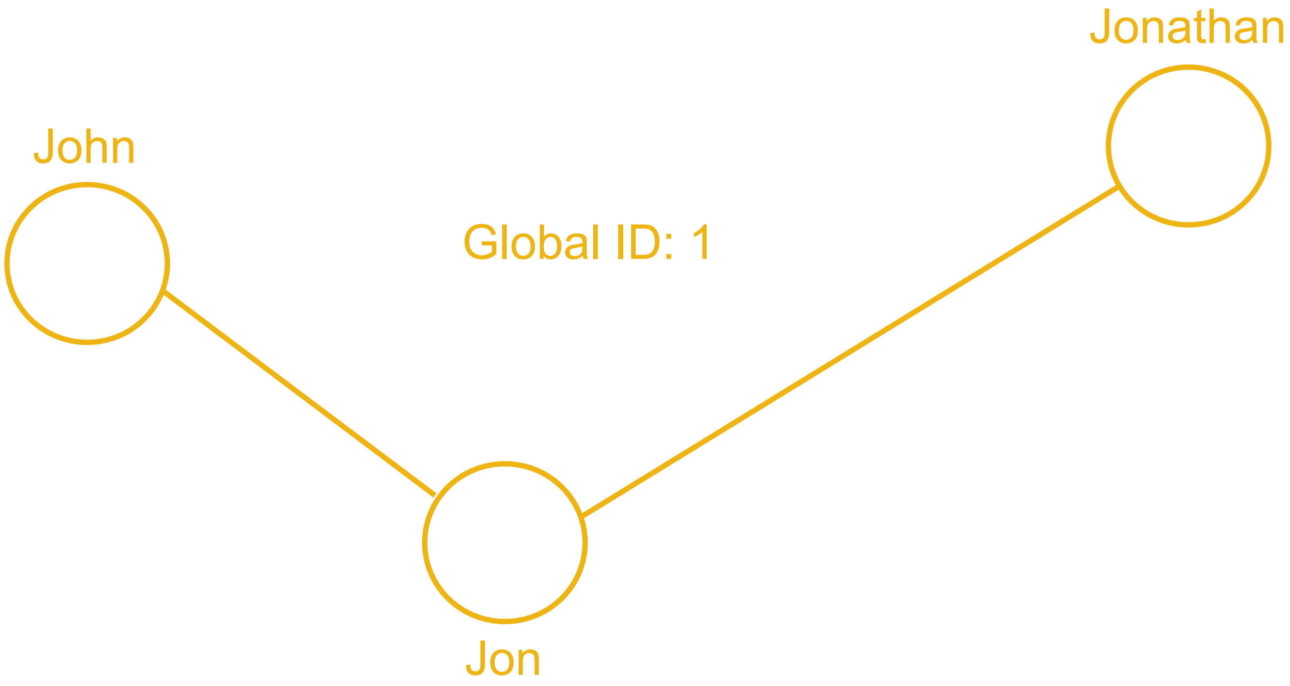
Identifying Matches

First Name	Last Name	Address	…	First Name	Last Name	Address	…	Match?
John	Jones	64 Plain St		Jonathan	Jones	64 Plain Street Apt. A		✅
John	Jones	64 Plain St		Jon	Jones	13 Washington Ave		❌
John	Jones	64 Plain St		Janet	Jackson	22 Regency Blvd		❌

Resolving Transitive Links



Resolving Transitive Links



Resolving Transitive Links

```
from graphframes import GraphFrame

graph = GraphFrame(vertices, edges)
connected_components = (graph.connectedComponents())

globally_linked_people = (
    connected_components
    .select(
        col('component').alias('global_id'),
        'person',
    ))
```

Splinkr 3 at Launch

- Cut runtime from 8+ hours (Splinkr 2) to 1.5 hours on same hardware
- Model performance better than previous in-house systems
- Code base easier to read thanks to declarative style of DataFrame API
- Less code weight by leveraging Spark and Python packages

Experiments with Neural Networks

- Prompted by idea shared during Riot Games talk at Spark Summit 2017
- Goals:
  - Handle edge cases better than logistic regression model
  - Simplify code while maintaining model performance

Experiments with Neural Networks

- Experimented with convolutional neural networks
- Model trained on raw text; no explicit features extracted
- Minor hacking required to integrate Keras model into Spark

Experiments with Neural Networks

- Results:
  - Marginal increase in accuracy; recall up, precision down
  - Neural network did not seem to learn anything not already captured by logistic regression
  - Code simplified at expense of interpretability, memory requirements, and run time

Biggest Hurdles in Building Splinkr 3

- Poor quality labels on our training data
  - SSN is not a great way to generate training pairs
  - Poor training data limited potential for linkage improvements
  - Manual cleanup helped, but turking or synthetic training data would have been better

Biggest Hurdles in Building Splinkr 3

- Bugs and API gaps working with Spark 2.0
  - Code generation bugs (SPARK-18492, SPARK-18866)
  - Optimizer problems related to UDFs (SPARK-18254, SPARK-18589)
  - GraphFrames Python 3 compatibility (graphframes#85)
  - GraphFrames correctness bug (graphframes#159)

Take-aways from Building Splinkr 3

- Spark makes building a scalable linkage pipeline approachable
- Declarative style of DataFrame API facilitates good design
- Access to Spark and Python ecosystems saves time (GraphFrames, Jellyfish)

Take-aways from Building Splinkr 3

- Good training data is critical and worth the upfront investment
- Good heuristics are often a better starting point than machine learning (Google's ML Engineering Rule #1)
- Building with cutting-edge tools (at the time, Spark 2.0) comes with risk

Building a Scalable Record Linkage System

with Apache Spark, Python 3, and Machine Learning

Nicholas Chammas and Eddie Pantridge
MassMutual

#Py6SAIS