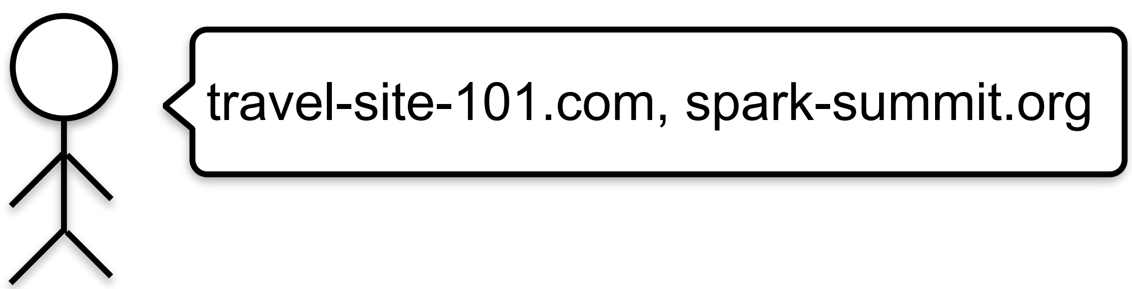
Using GraphX/Pregel on Browsing History to Discover Purchase Intent

Zhang, Lisa
Rubicon Project Buyer Cloud

SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

Problem

- Identify possible new customers for our advertisers using **intent data**, one of which is **browsing history**



Challenges

Sites are **numerous** and **ever-changing**

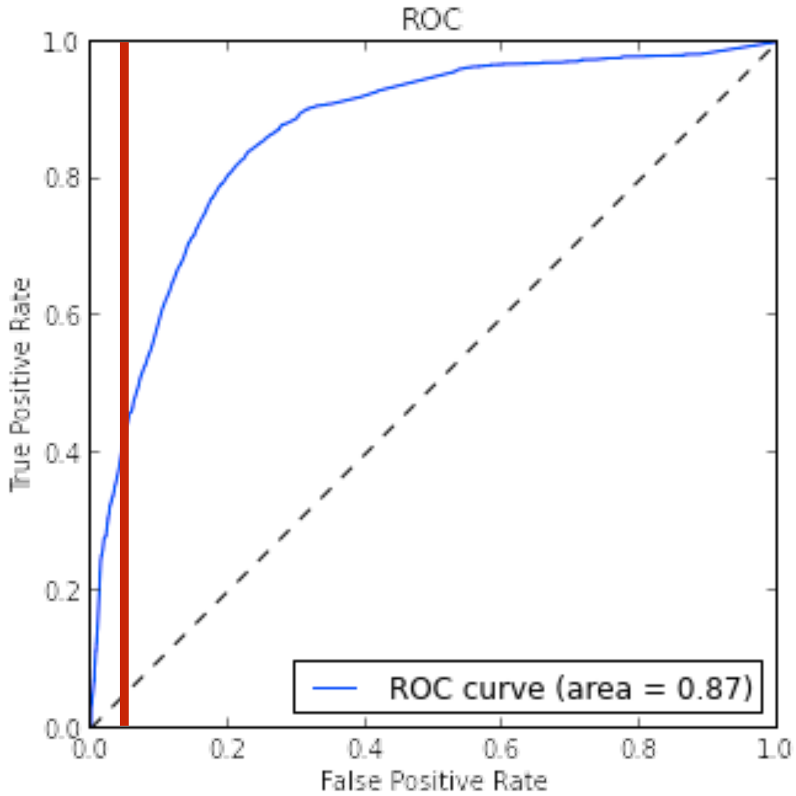
Need to build **one model per advertiser**

Positive training cases are **sparse**

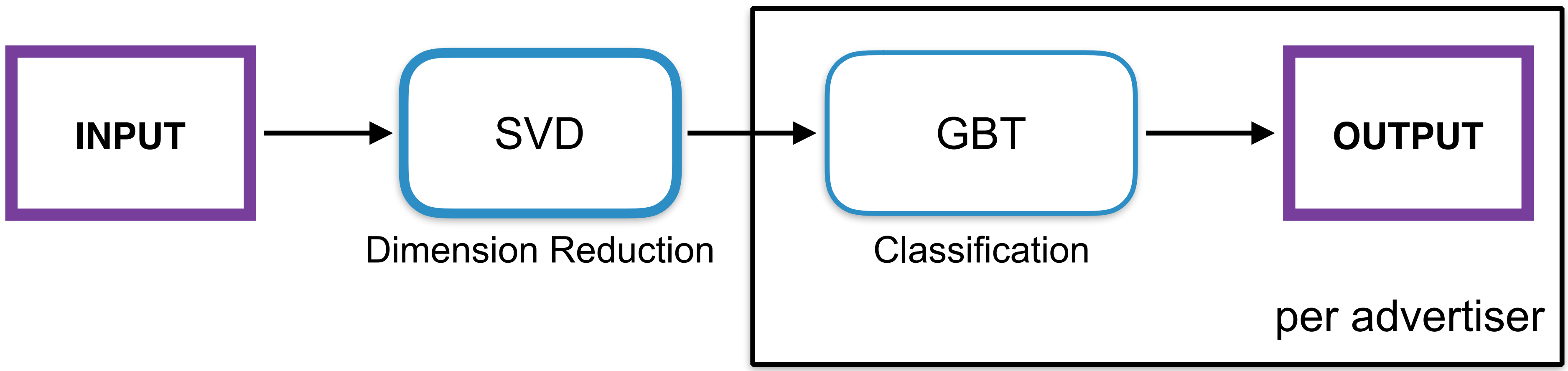
Models run **frequently**: every few hours

Offline Evaluation Metrics

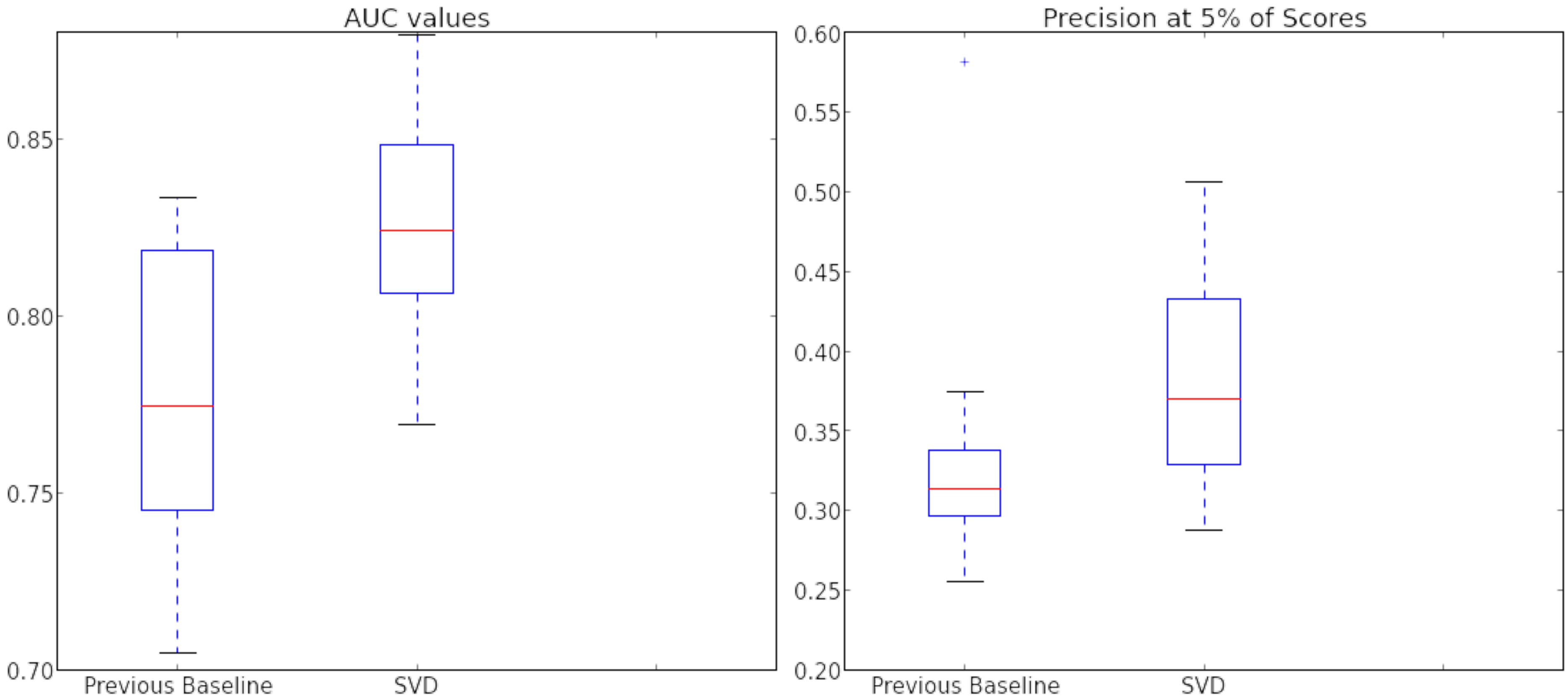
- **AUC**: area under ROC curve
- **Precision at top 5% of score**: model used to identify top users only
- **Baseline**: Previous solution prior to Spark



Linear Dimensionality Reduction



Evaluation


AUC values
0.85
0.80
0.75
0.70
Previous Baseline
SVD
Precision at 5% of Scores
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
Previous Baseline
SVD

SVD: Top Sites

Home Improvement Advertiser
deal-site-101.com
chat-site-001.com
ecommerce-site-001.com
chat-site-002.com
invitation-site-001.com
classified-site-001.com

Telecom Advertiser
developer-forum-001.com
chat-site-001.com
invitation-site-001.com
deal-site-101.com
college-site-001.com
chat-site-002.com

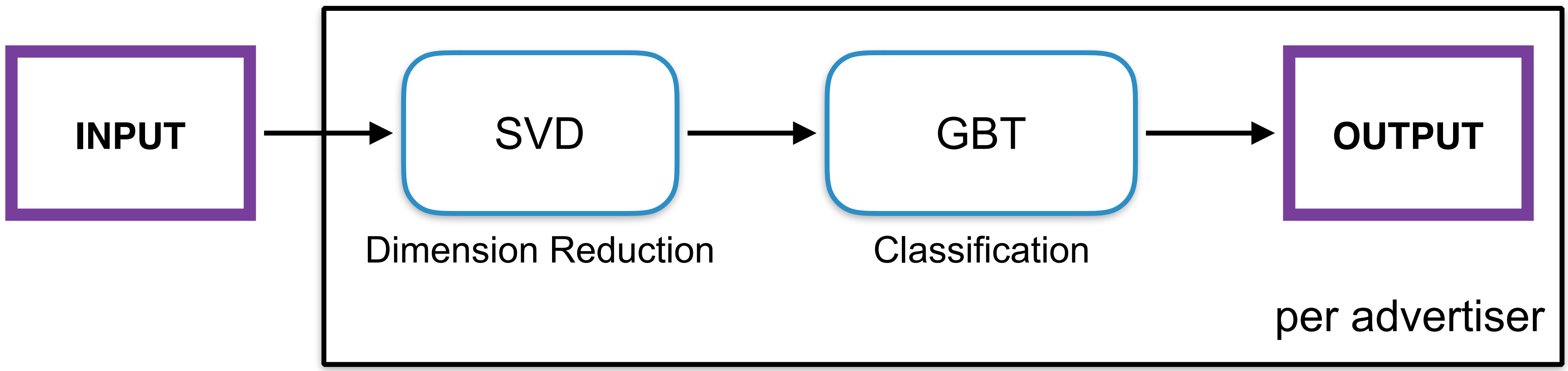
The Issue with SVDs

- Dominated by the **same** signal across **all** advertisers

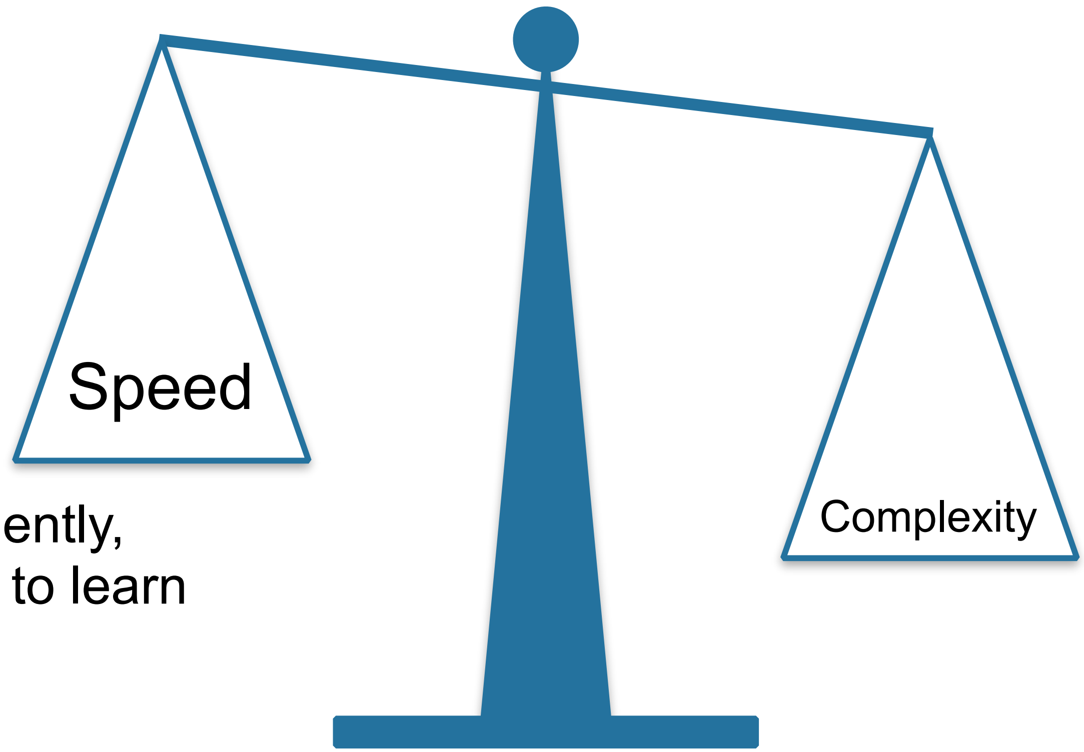
- Identify online buyers, but **not those specific to each advertiser**

- Not appropriate for our use case

SVD per Advertiser?



Non-linear Approaches?



Too Complex:
Cannot run frequently,
we become slow to learn
about new sites

Too Simple:
Possibly same
problem as SVD

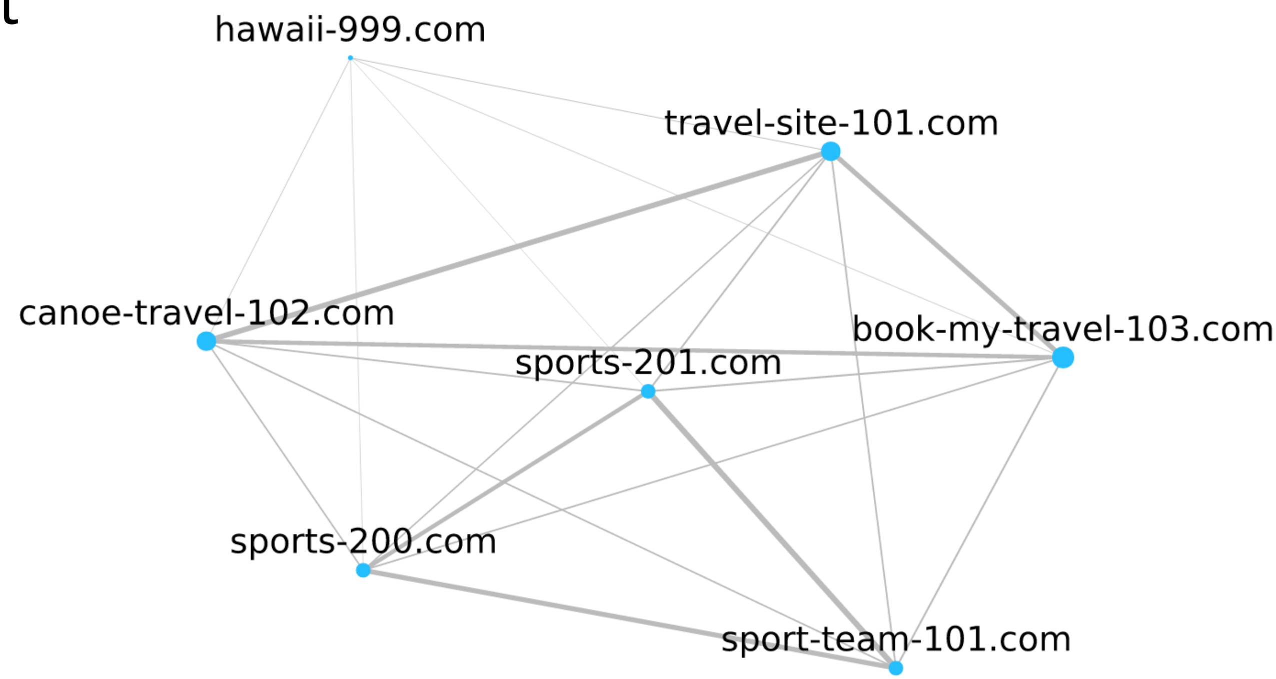
Can We Simplify?

Intuition:
Given a known positive training case, target other users that have **similar site history** as the current user.

One natural way is to treat sites as a **graph**.

Sites as Graphs

- Easy to interpret
- Easy to visualize
- Graph algos well studied



Spark GraphX

- Spark's API for parallel graph computations
- Comes with some common graph algorithms
- API for developing new graph algorithms: e.g. via **pregel**

Pregel API

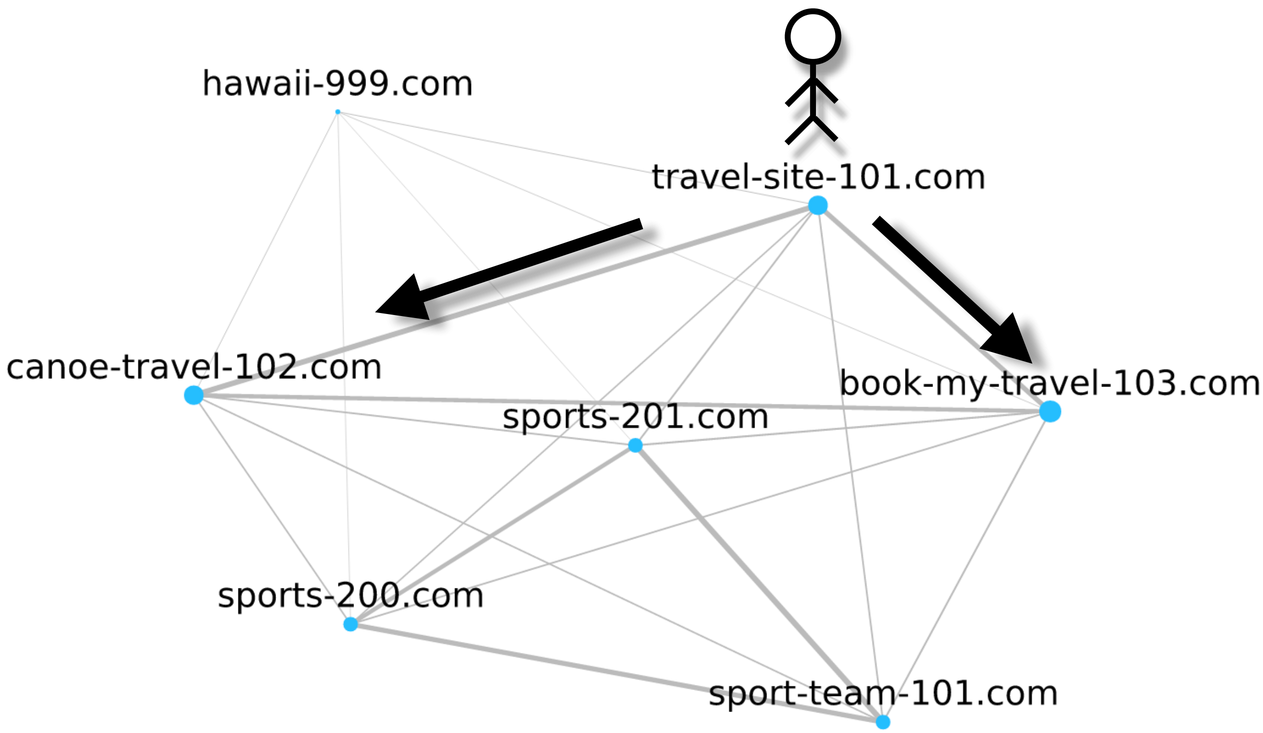
- Pass **messages** from vertices to other, typically adjacent, vertices: “Think like a vertex”

- Define an algorithm by stating:

repeat
: how to **send messages**
: how to **merge** multiple messages
: how to **update a vertex** with message

Propagation Based Approach

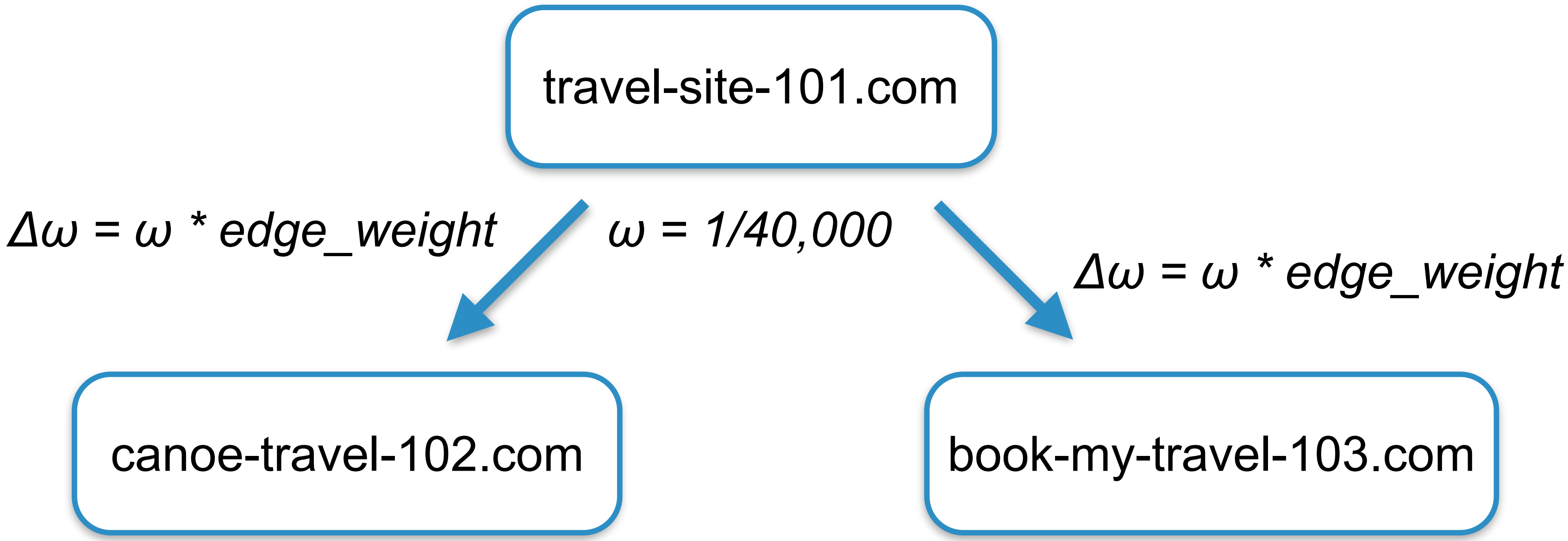
- Pass positive (converter) information across edges
- Give credit to "similar" sites



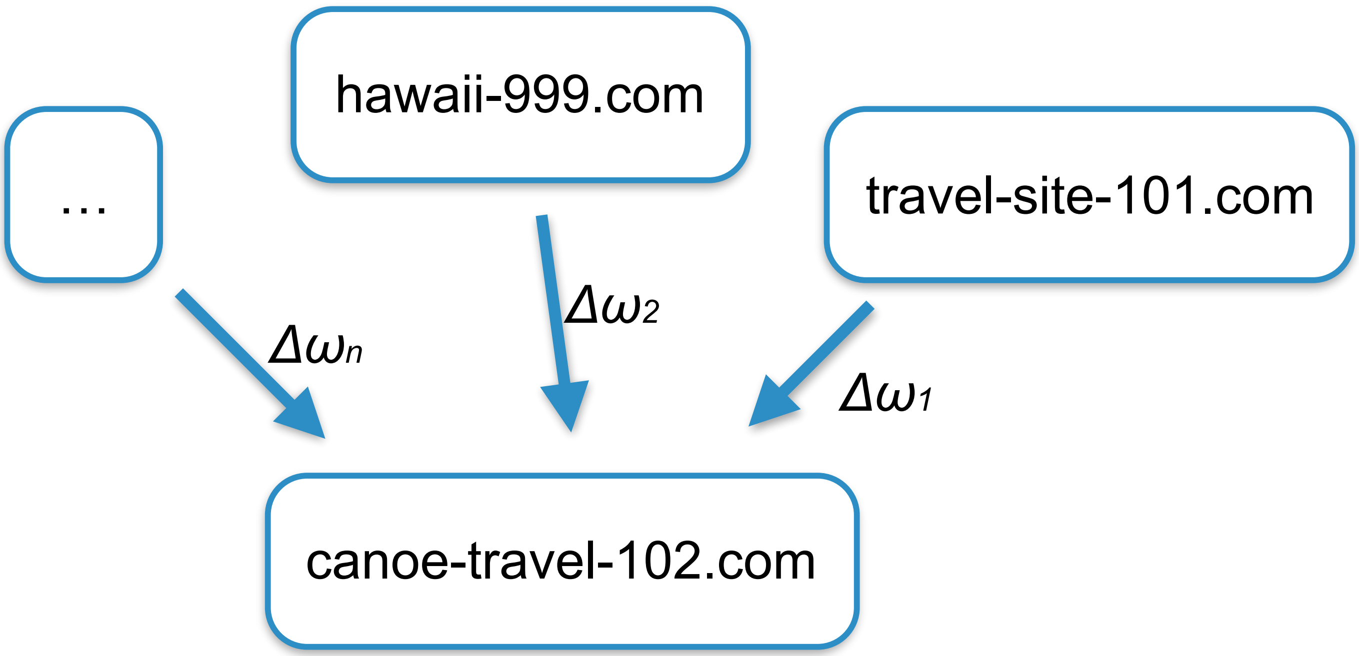
Example Scenario

travel-site-101.com	1 converter / 40,000 visitors
canoe-travel-102.com	0 converter / 48,000 visitors
book-my-travel-103.com	0 converter / 41,000 visitors

Sending Messages



Receiving Messages



$$\omega_{new} = \omega_{old} + \lambda \cdot \Sigma\, \Delta\omega_i$$

Weights After One Iteration

travel-site-101.com	2.5 x 10^(-5)
canoe-travel-102.com	1.2 x 10^(-5)
book-my-travel-103.com	0.8 x 10^(-5)

Simplified Code

```
type MT = Double; type ED = Double; type VD = Double
val lambda = …; val maxIterations = …
val initialMsg = 0.0

def updateVertex(id: VertexId, w: VD, delta_w: MT): VD =
  w + lambda * delta_w
def sendMessage(edge: EdgeTriplet[VD, ED]): Iterator[(VertexId, MT)] = {
  Iterator((edge.srcId, edge.attr * edge.dstAttr),
           (edge.dstId, edge.attr * edge.srcAttr))
}
def mergeMsgs(w1: MT, w2: MT): MT = x + y

val graph: Graph[VD, ED] = …
graph.pregel(initialMessage, maxIterations, EdgeDirection.out)(
  updateVertex, sendMessage, mergeMessage)
```

Model Output & Application

- **Model output** is a mapping of sites to final scores

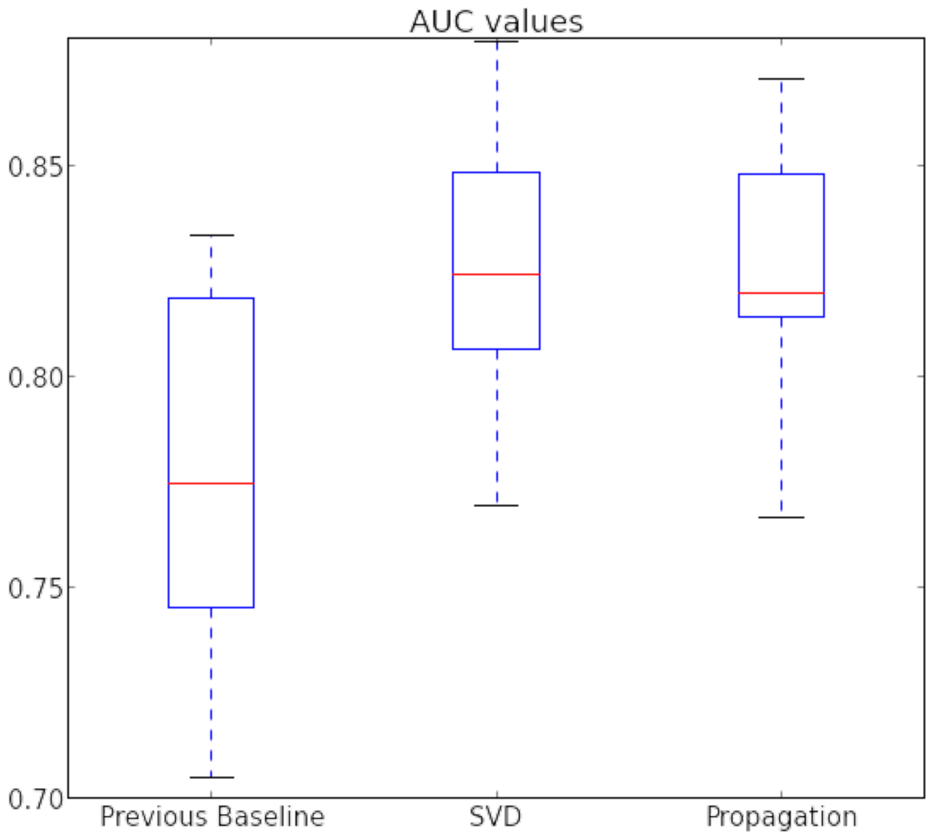
- To **apply** the model, aggregate scores of sites visited by user

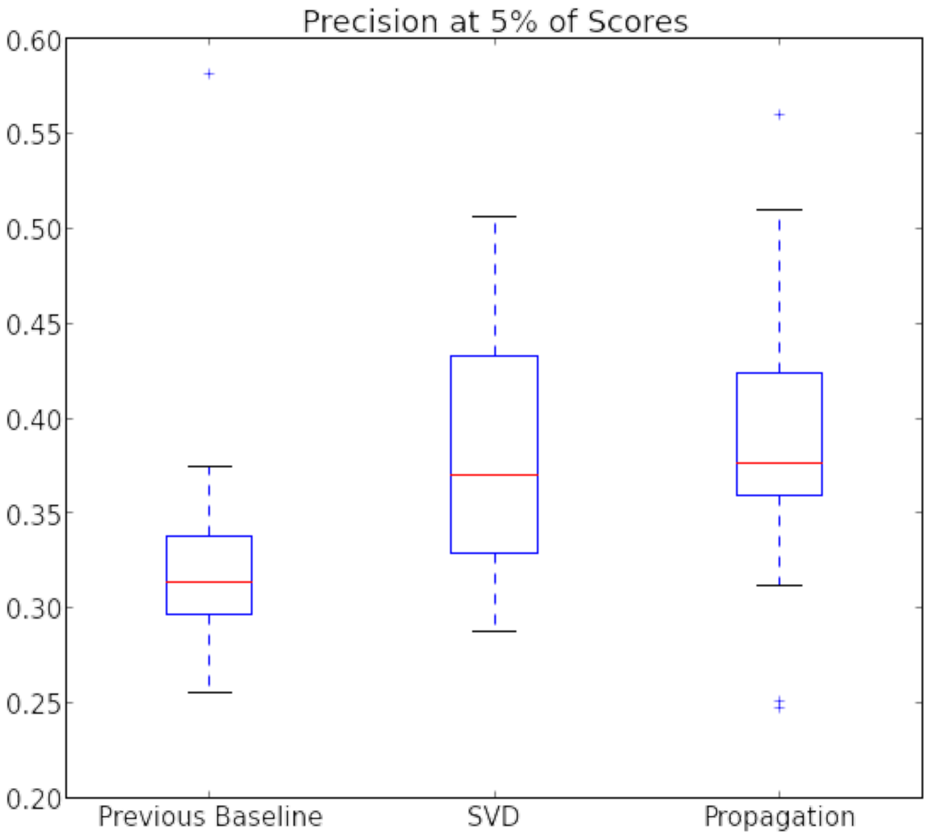
SITE	SCORE
travel-site-101.com	*0.5*
canoe-travel-102.com	*0.4*
sport-team-101.com	*0.1*
…	*…*

Other Factors

- **Edge Weights**: Cosine Similarity, Jaccard Index, Conditional Probability
- **Edge/Vertex Removal**: Remove sites and edges on the long-tail
- **Hyper parameter Tuning**: lambda, numIterations and others through testing (there is no convergence)

Evaluation


AUC values
0.85
0.80
0.75
0.70
Previous Baseline
SVD
Propagation


Precision at 5% of Scores
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
Previous Baseline
SVD
Propagation

Propagation: Top Sites

Home Improvement Advrt.	Telecom Advertiser
label-maker-101.com	canada-movies-101.ca
laptop-bags-101.com	canadian-news-101.ca
renovations-101.com	canadian-jobs-101.ca
fitness-equipment-101.com	canadian-teacher-rating-101.ca
renovations-102.com	watch-tv-online.com
buy-realestate-101.com	phone-system-review-101.com

Renovations

Canadian

Telecom

Challenges (from earlier)

Sites are **numerous** and **ever-changing**

Need to build **one model per advertiser**

Positive training cases are **sparse**

Models run **frequently**: every few hours

Resolutions

Graph built **just in time** for training

Need to build **one model per advertiser**

Positive training cases are **sparse**

Models run **frequently**: every few hours

Resolutions

Graph built **just in time** for training

Graph built **once**; propagation runs **per advertiser**

Positive training cases are **sparse**

Models run **frequently**: every few hours

Resolutions

Graph built **just in time** for training

Graph built **once**; propagation runs **per advertiser**

Propagation resolves sparsity: **intuitive** and **interpretable**

Models run **frequently**: every few hours

Resolutions

Graph built **just in time** for training

Graph built **once**; propagation runs **per advertiser**

Propagation resolves sparsity: **intuitive** and **interpretable**

Evaluating users **fast**; does **not** require GraphX

General Spark Learnings

- **Many small jobs > one large job**: We split big jobs into multiple smaller, concurrent, jobs and increased throughput (more jobs could run concurrently).

- **Serialization**: Don't save SparkContext as a member variable, define Python classes in a separate file, check if your object serializes/deserializes well!

- Use **rdd.reduceByKey()** and others over **rdd.groupByKey()**.

- Be careful with **rdd.coalesce()** vs **rdd.repartition()**, **rdd.partitionBy()** can be your friend in the right circumstances.

THANK YOU.

lzhang@rubiconproject.com

SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY