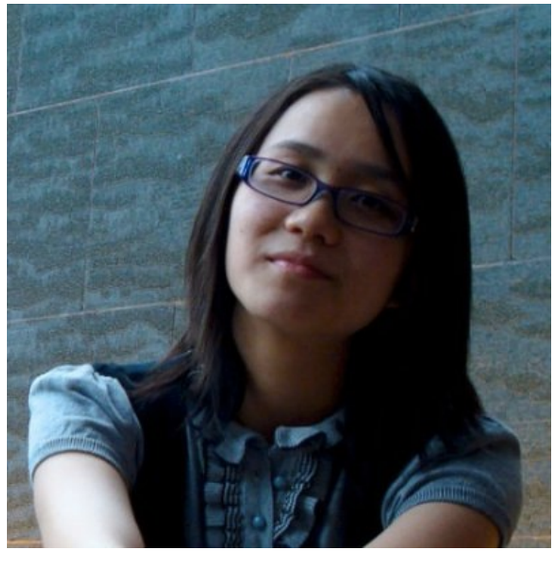
The ACM Conference Series on
Recommender Systems

RecSys 2015 Tutorial

Scalable Recommender Systems

Where Machine Learning
Meets Search!



Diana Hu
Senior Data Scientist

@sdianahu
diana.hu@verizon.com

Joaquin Delgado, PhD.
Director of Engineering

@joaquind
joaquin.a.delgado@verizon.com

Presenters

Disclaimer

The content of this presentation are of the authors' personal statements and does not officially represent their employer's view in anyway. Included content is especially not intended to convey the views of OnCue or Verizon.

Index

1. Introduction

What to expect from this tutorial?

- The focus is on practical examples of how to implement scalable recommender systems using search and learning-to-rank (machine learning) techniques
- What it is not
  - Deep dive into any specific areas (Search, RecSys, Learning to rank, or Machine learning)
  - Algorithmic survey
  - Comparative Analysis

Finding commonalities

What is a recommendation?

Beyond rating prediction

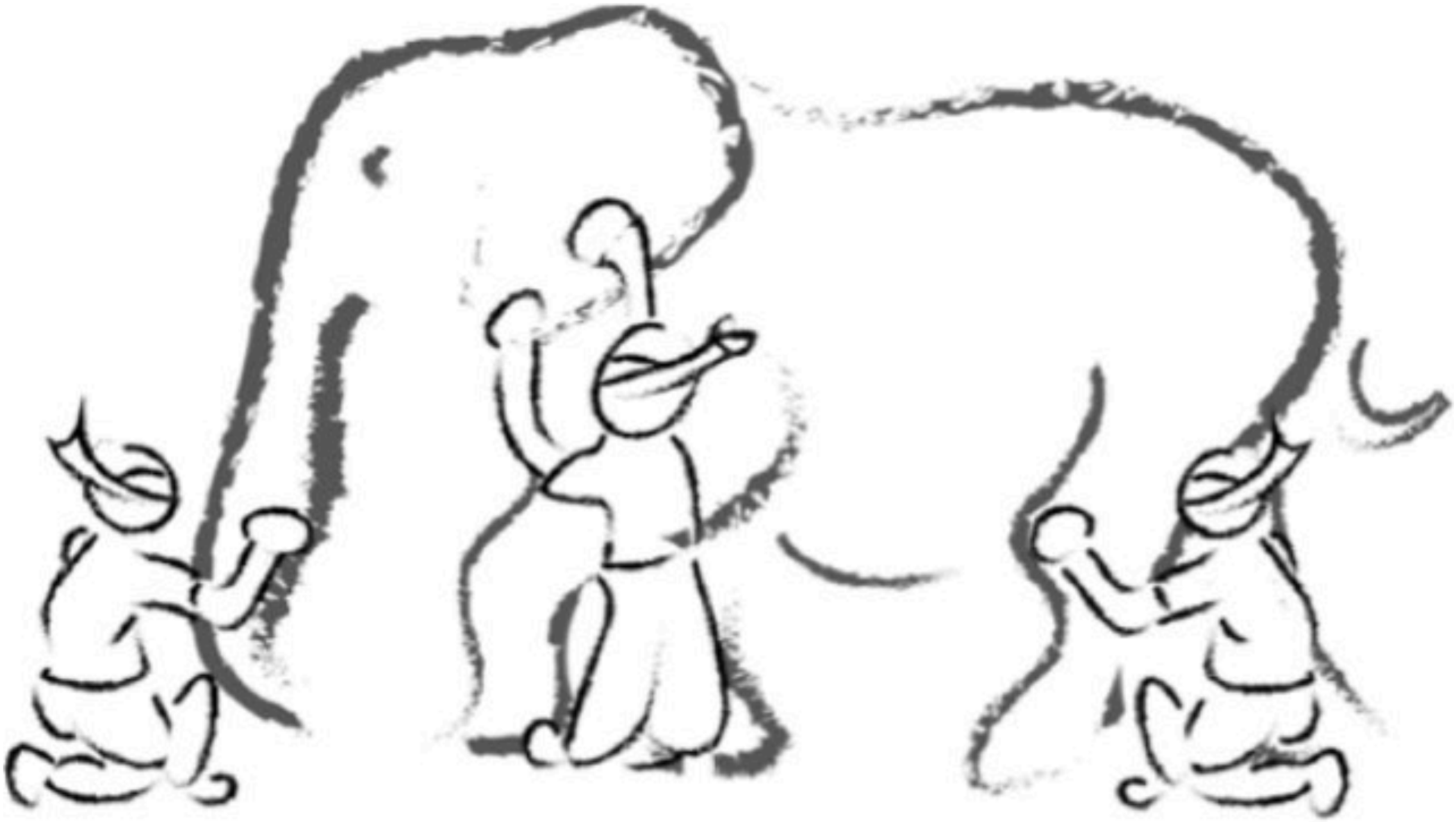


Image - "Everything Connects" (McGraw Hill, Feb 2014)

Paradigms of recommender systems

- Reduce information load by estimating relevance
- Ranking Approaches:
  - **Collaborative filtering**: “Tell me what is popular amongst my peers”
  - **Content Based**: “Show me more of what I liked”
  - **Knowledge Based**: “Tell me what fits my needs”
  - **Hybrid**

Model Type	Pros	Cons
Collaborative	• No metadata engineering effort • Serendipity of results • Learns market segments	• Requires rating feedback • Cold start for new users and new items
Content-based	• No community required • Comparison between items possible	• Content descriptions necessary • Cold start for new users • No serendipity
Knowledge-based	• Deterministic • Assured quality • No cold-start • Interactive user sessions	• Knowledge engineering effort to bootstrap • Static • Does not react to short-term trends

Scaling recommender systems is hard!

- *Millions of users*
- *Millions of items*
- *Cold start* for ever increasing size of catalog and new users added
- *Imbalanced Datasets* – power law distribution is quite common
- Many algorithms have not been fully tested at “Internet Scale”

2. Recommender System Problem as a Search Problem

Content-based methods inspired by IR

- **Rec Task**: Given a *user profile* find the best matching *items* by their *attributes*
- **Similarity calculation**: based on keyword overlap between user/items
  - Neighborhood method (i.e. nearest neighbor)
  - Query-based retrieval (i.e Rocchio's method)
  - Probabilistic methods (classical text classification)
  - Explicit decision models
- **Feature representation**: based on content analysis
  - Vector space model
  - TF-IDF
  - Topic Modeling

Search queries as content-based recommendations

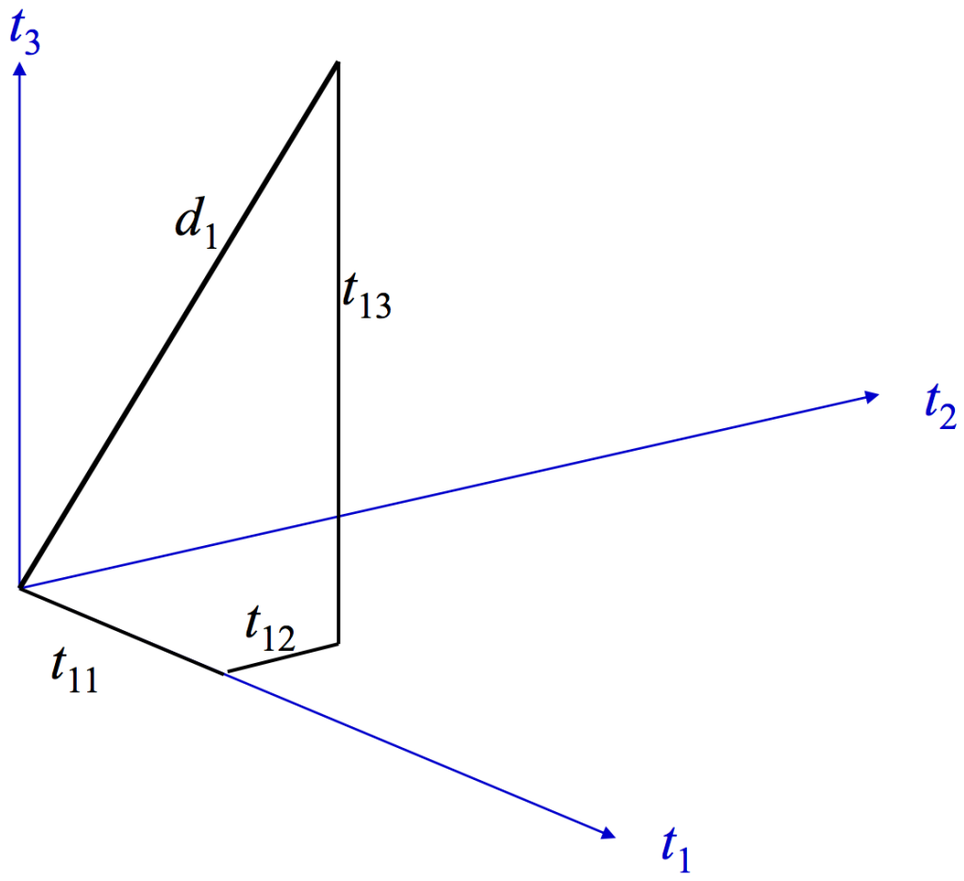
- Exact matching (Boolean)
  - Relevant or not relevant (no ranking)
- Ranking by similarity to query (Vector Space Model)
  - Text similarity: Bag of words, TF-IDF, Incidence Matrix

$$\text{score}(q,d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}.$$

- Ranking by importance (e.g. PageRank)

Content-based similarity measures

- Simple match $|Q \cap D|$
- Dice's Coefficient $2\dfrac{|Q \cap D|}{|Q| + |D|}$
- Jaccard's Coefficient $\dfrac{|Q \cap D|}{|Q \cup D|}$
- Cosine Coefficient $\dfrac{|Q \cap D|}{|Q|^{1/2} \times |D|^{1/2}}$
- Overlap Coefficient $\dfrac{|Q \cap D|}{\min(|Q|, |D|)}$



3D Term Vector Space

Knowledge-based methods inspired by IR

- **Rec Task**: Given *explicit recommendation rules* find the best matches between *user's requirements* and *item's* characteristics (i.e., which item should be recommended in which context?)
- **Similarity calculation**: based on constraint satisfaction problem and distance similarity requirements<->attributes
  - Conjunctive queries
  - Similarity metrics for item retrieval
- **Feature representation**: based on query representation
  - User defined preferences
  - Utility-based preferences
  - Conjoint analysis

Search queries as knowledge-based recommendations

- Constraint satisfaction problem (CSP) is a tuple (V,D,C)
  - V – set of variables
  - D – set of finite domains for V
  - C – set of constraints of possible V permutations
- Recommendation as CSP:
  (V,D,C) => (Vi *U* Vu, D, Cr *U* Ci *U* Cf *U* REQ)
  - Vu – user properties (possible user's requirements)
  - Vi – item properties
  - Cr – compatibility constraints (possible Vc permutations)
  - Ci – Item constraints (conjunction fully defines an item)
  - Cf – filter conditions (define Vu<->Vi relationships)
  - REQ – user's requirements

3. Introduction to Search and Information Retrieval

Search

Search is about finding specific things that are either known or assumed to exist, **Discovery** is about is about helping the user encounter what he/she didn't even know exists

Both Search and Discovery can be achieved through a query based data/information system.

Predicate Logic and Declarative Languages Rock!

Examples of query based systems

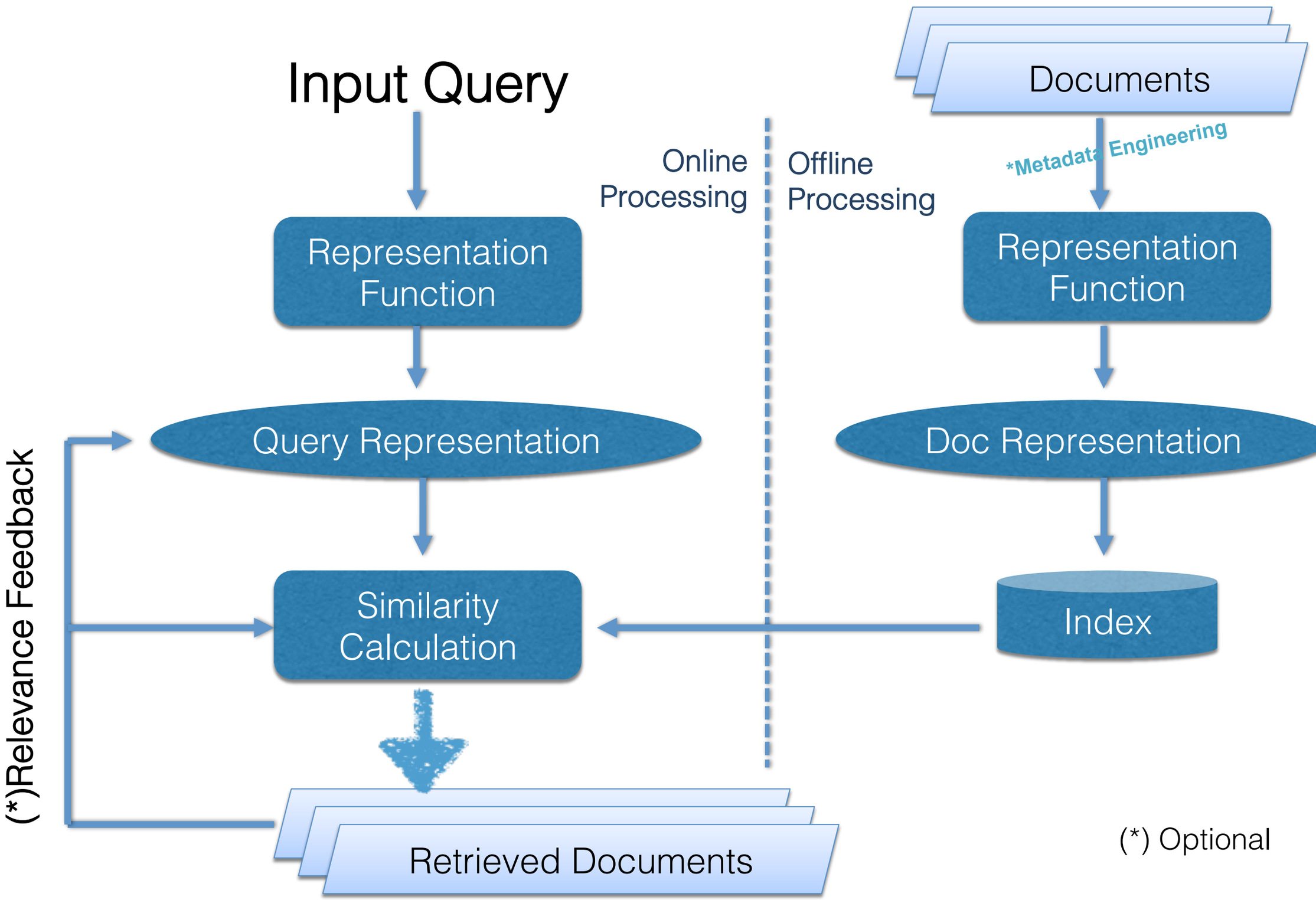
- Focused on Search
  - Search engines
  - Database systems
- Focus on Discovery
  - Recommender systems
  - Advertising systems

IR: The science behind search!

Information Retrieval (IR) is a query based on **data retrieval + relevance ranking** (scoring) usually applied to unstructured data (i.e. text documents and fields); often referred to as full-text or keyword search.

Have you heard of Bag-of-Words?
Vector Space Representation?
What about TF-IDF?

IR Architecture

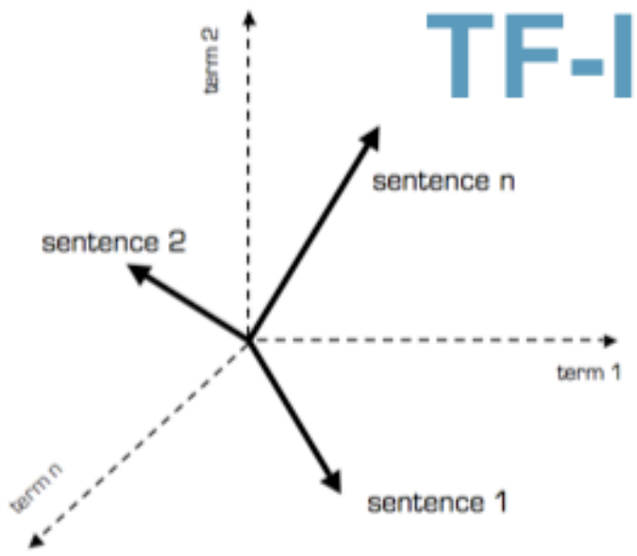


Retrieval Models

Model Type	Query Representation	Document Representation	Retrieval
Boolean	• Boolean expressions • Connected by AND, OR, NOT	• **Set** of keywords • Bag of words • Binary term weight	• Exact match • Binary relevance • No ranking
Vector Space Model	• Vector • Desired terms with optional weights	• Vectors • Bag of words with weight based on TF-IDF scheme	• Similarity score • Output documents are ranked • Relevance feedback support
Probabilistic	• Similarity with priors	• Document relevance	• Ranks documents in decreasing probability of relevance

Ranking in the Vector Space Model

TF-IDF



Language Model

P(optimization | search, engine) >>
P(walking | search, engine)

Probability Ranking Principle

P(R = 1 | d, q) or P(R = 0 | d, q)

Search Engines: the big hammer!

- Search engines are largely used to solve non-IR search problems, and here is why:
  - Widely available
  - Fast and scalable distributed systems
  - Integrates well with existing data stores (SQL and NoSQL)

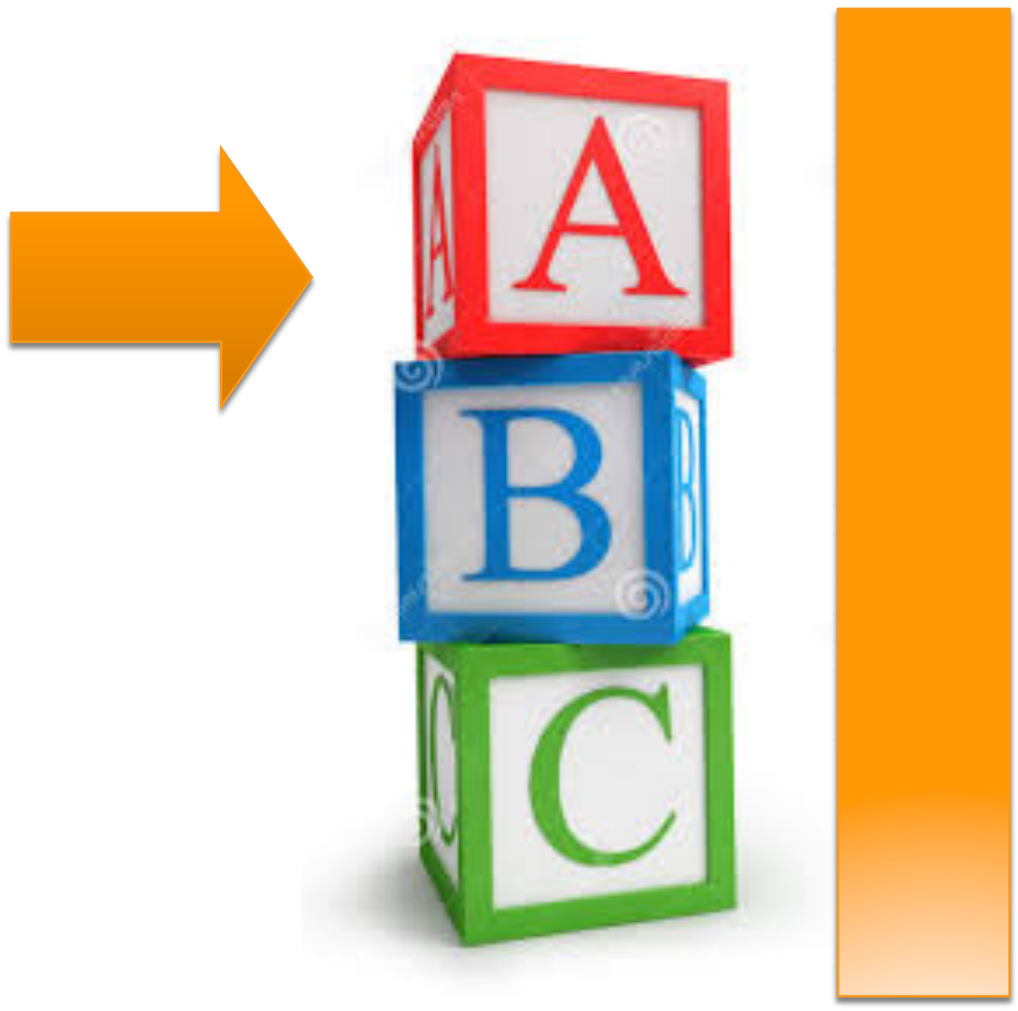
But are we using the right tool?

- Search Engines were originally designed for IR.
- More complex non-IR search/discovery tasks sometimes require a multi-phase, multi-system approach

Filter + Scoring: Two Phase Approach

Filter

Rank



Elasticsearch

- What is Elasticsearch?
  - Elasticsearch is an open-source search engine
  - Elasticsearch is written in Java
  - Built on top of Apache Lucene™
  - A distributed real-time document store where every field is indexed and searchable out-of-the box
  - A distributed search engine with real-time analytics
  - Has a plugin architecture that facilitates extending the core system
  - Written with NRT and cloud support in mind
  - Easy index, shard and replicas creation on live cluster
  - Has Optimistic Concurrency Control

Examples of scaling challenges

- More than 50 millions of documents a day
- Real time search
- Less than 200ms average query latency
- Throughput of at least 1000 QPS
- Multilingual indexing
- Multilingual querying

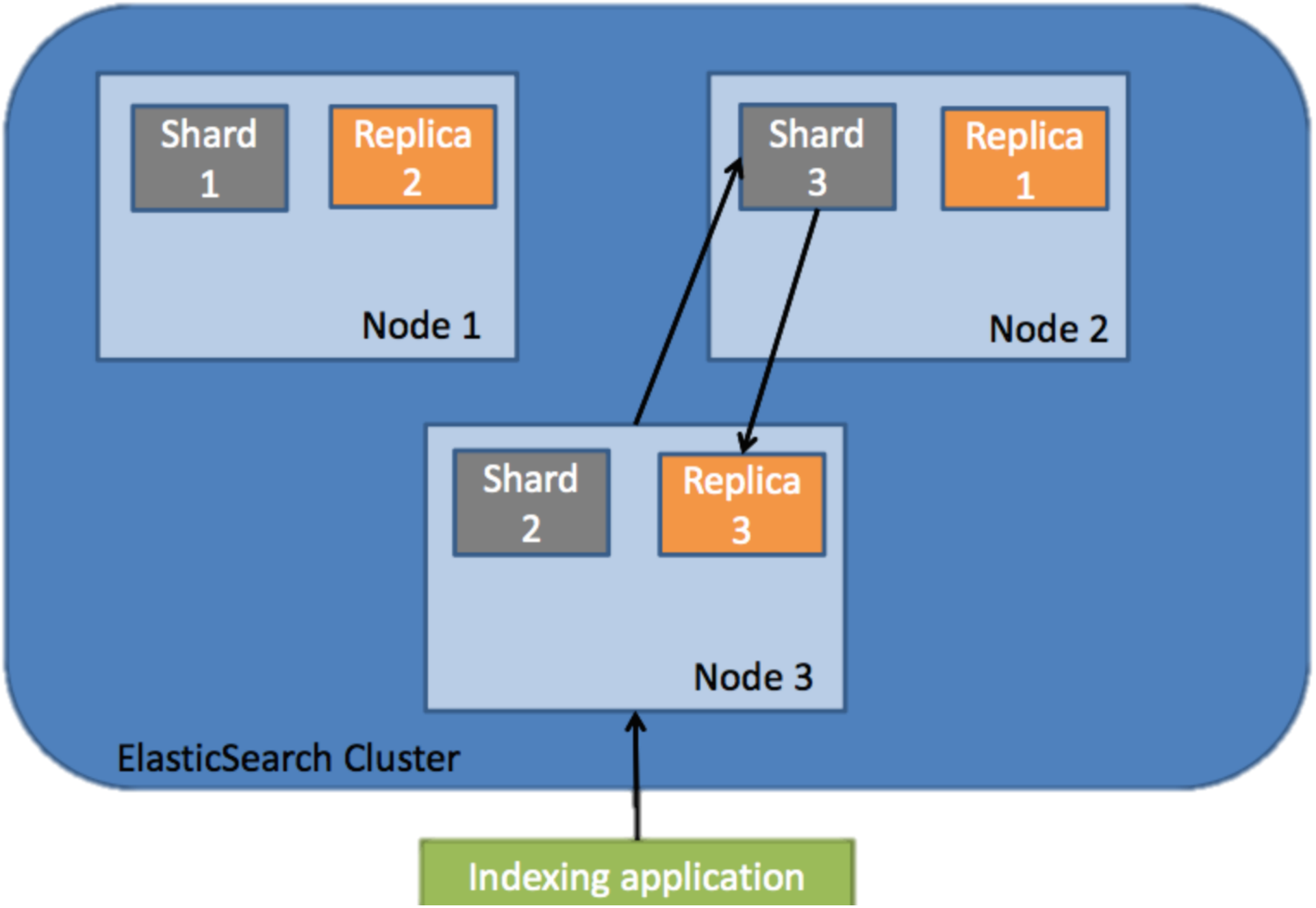
Who uses ES?

- Wikipedia
  - Uses Elasticsearch to provide full-text search with highlighted search snippets, and *search-as-you-type* and *did-you-mean* suggestions.
- The Guardian
  - Uses Elasticsearch to combine visitor logs with social -network data to provide real-time feedback to its editors about the public's response to new articles.
- Stack Overflow
  - Combines full-text search with geo-location queries and uses more-like-this to find related questions and answers.
- GitHub
  - Uses Elasticsearch to query 130 billion lines of code.

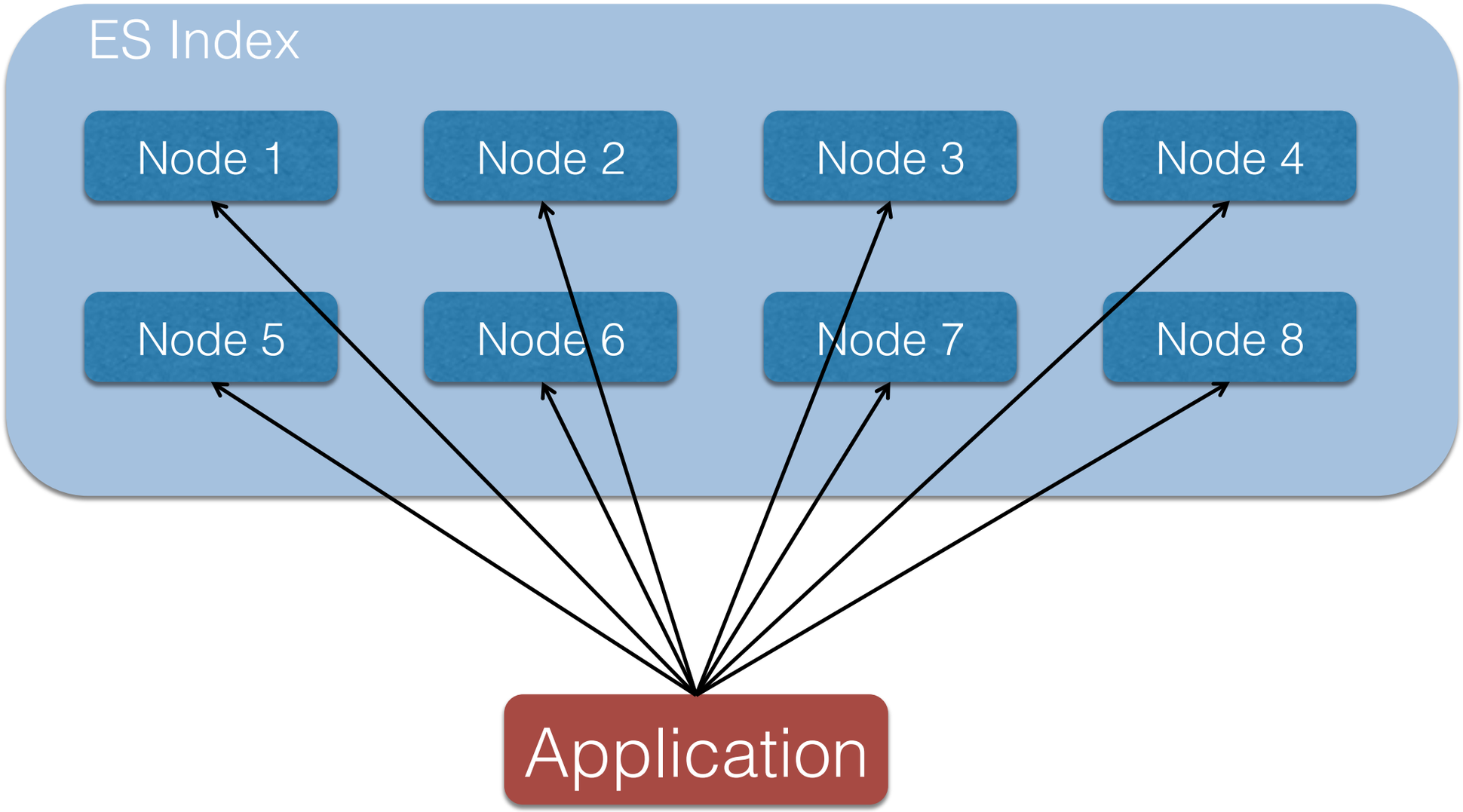
How ES scales?

- Sharding and Replicas
  - Several indices (at least one index for each day of data)
  - Indices divided into multiple shards
  - Multiple replicas of a single shard
  - Real-time, synchronous replication
  - Near-real-time index refresh (1 to 30 seconds)

Indexing the data



Querying ES


ES Index
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Node 7
Node 8
Application

Using Search Engines for RS

- Its not just about rating prediction and ranking
  - Business filtering logic
    - Age restrictions
    - Catalog navigation context (e.g. e-commerce)
    - Promotional materials
  - Low latency and scale
    - SLAs on response times including query, responses and presentation
    - Actual time for computing recommendations is just a small fraction of total allocated time

Stacking things up

Offline

Online

Query Generation and Contextual Pre-filtering

Model Building

Retrieval

Index Building

Contextual Post Filtering

Data Analytics

Ranking

Visualization / UI

Experimentation

Data/Events Collections

Ranking in Elasticsearch

```
score(q,d)  =  ❶
           queryNorm(q)  ❷
         · coord(q,d)    ❸
         · Σ (           ❹
             tf(t in d)   ❺
           · idf(t)²      ❻
           · t.getBoost() ❼
           · norm(t,d)    ❽
           ) (t in q)    ❹
```

❶ `score(q,d)` is the relevance score of document `d` for query `q`.

❷ `queryNorm(q)` is the *query normalization* factor (new).

❸ `coord(q,d)` is the *coordination* factor (new).

❹ The sum of the weights for each term `t` in the query `q` for document `d`.

❺ `tf(t in d)` is the term frequency for term `t` in document `d`.

❻ `idf(t)` is the inverse document frequency for term `t`.

❼ `t.getBoost()` is the *boost* that has been applied to the query (new).

❽ `norm(t,d)` is the field-length norm, combined with the index-time field-level boost, if any. (new).

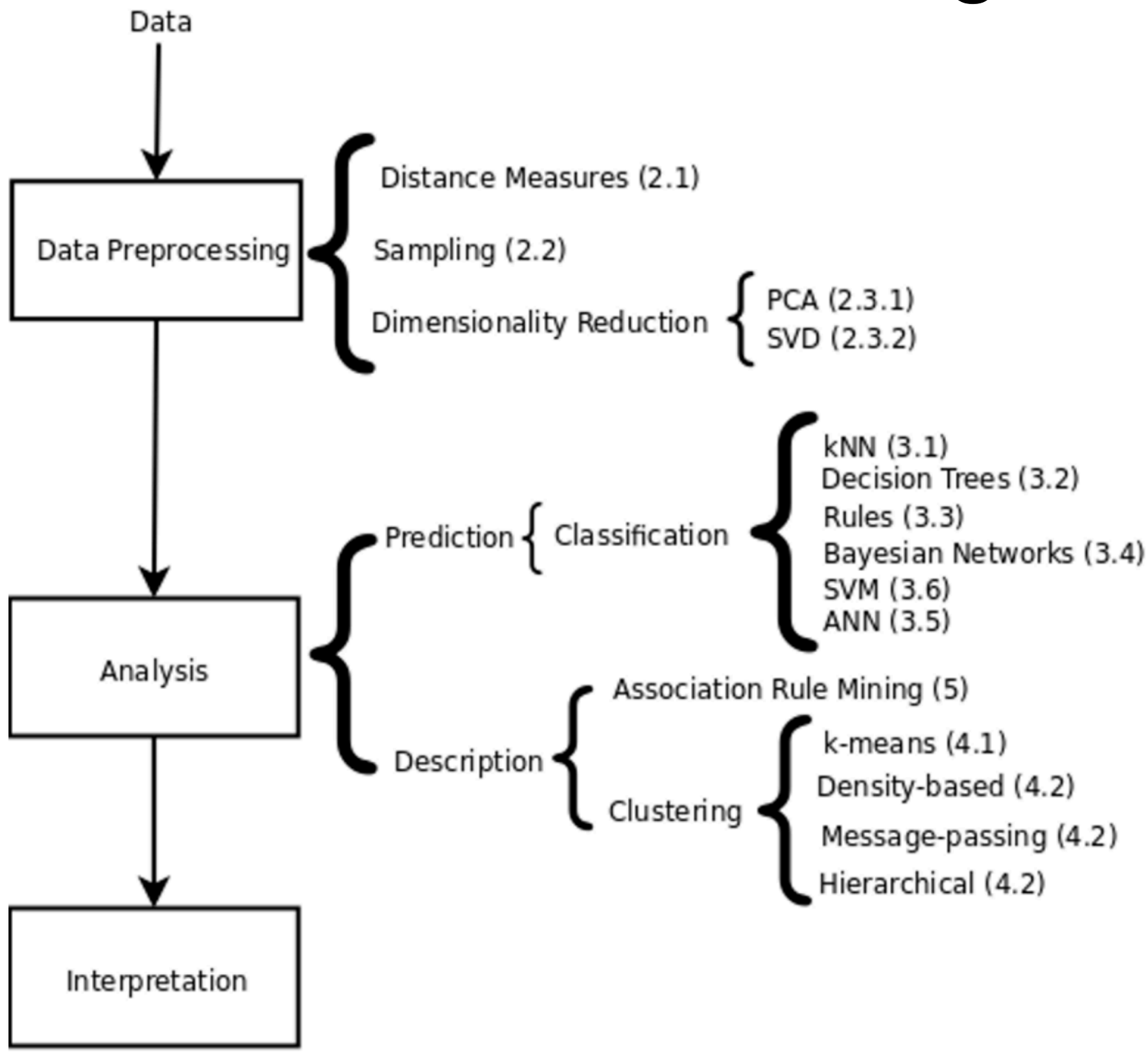
4. Overview of Machine Learning Techniques for Recommender Systems

Machine Learning

Machine Learning in particular *supervised learning* refer to techniques used to learn how to classify or score previously unseen objects based on a training dataset

Inference and Generalization are the Key!

Recommendations as data mining



Amatriain, Xavier, et al. "Data mining methods for recommender systems." Recommender Systems Handbook. Springer US, 2011. 39-71.

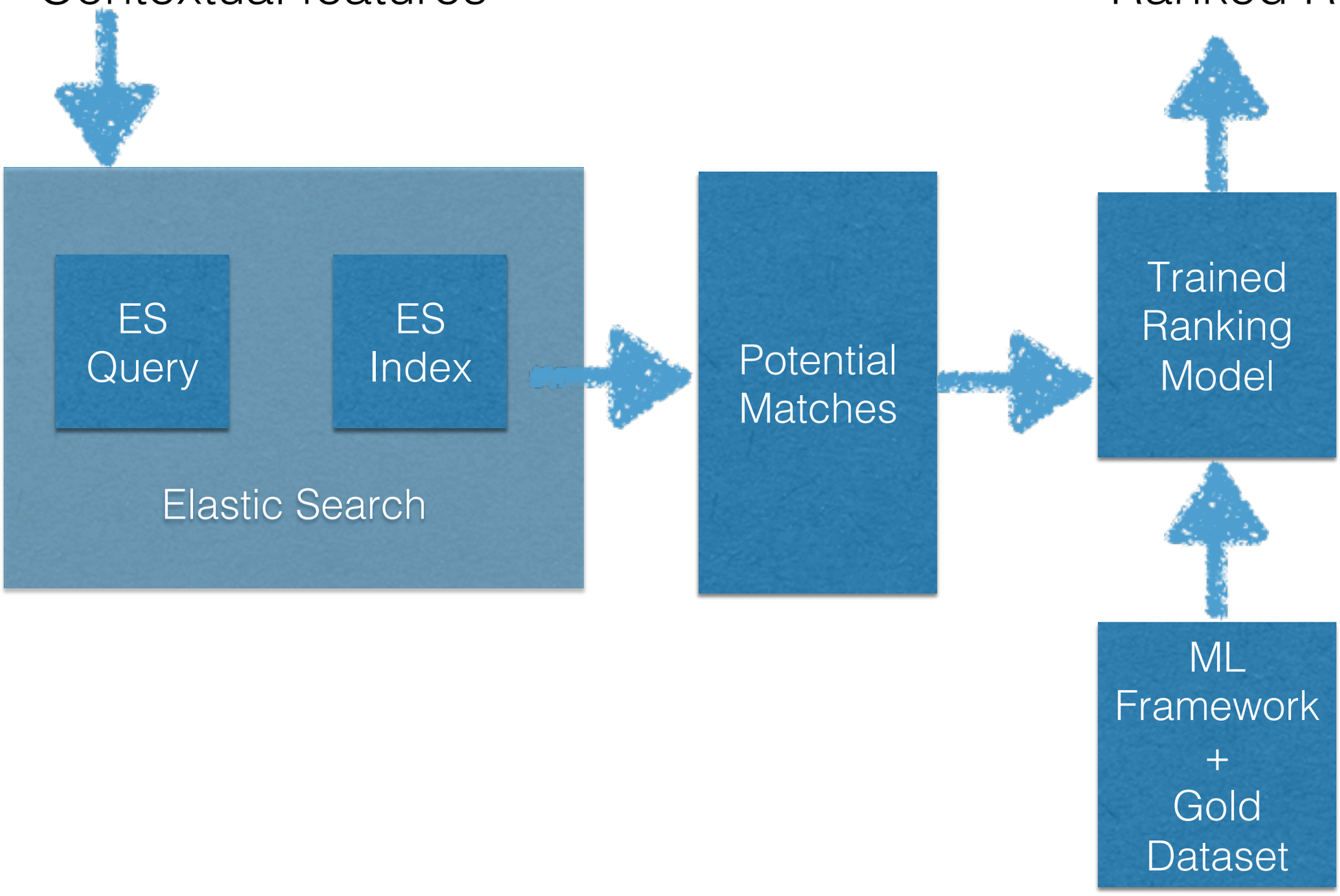
Learning to rank

- Formulate the problem as standard supervised learning
- Training data can be cardinal or binary
- Various approaches:
  - Pointwise: Typically approximated by regression
  - Pairwise: Approximated via binary classifier
  - Listwise: Directly optimize whole list (difficult!)
- A trick with ES is to include raw scores returned by ES into the feature vector

Learning to rank with ES

Input:
Contextual features

Output:
Ranked Results



Web scale ML challenges

- Massive amount of examples
- Billions of features
- Big models don’t fit in a single machine’s memory
- Variety of algorithms that need to be scaled up

A Note of Caution….

"Invariably, ***simple models*** *and* ***a lot of data*** *trump more elaborate models* **based on less data**."

Alon Halevy, Peter Norvig, and Fernando Pereira, Google

http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//pubs/archive/35179.pdf

Scalability in Machine Learning

- Distributed systems – *Fault tolerance, Throughput vs. latency*
- Parallelization Strategies – *Hashing, trees*
- Processing – *Map reduce variants, MPI, graph parallel*
- Databases – *Key/Value Stores, NoSQL*

What is Spark?

Fast, expressive cluster computing system

BlinkDB
approx queries

Spark SQL
structured data

Spark Streaming
real-time

MLlib
machine learning

GraphX
graph Analytics

Spark Core

What is Spark?

- Work on distributed collections like local ones
- RDD:
  - Immutable
  - Parallel transforms
  - Resilient and configurable persistence
- Operations
  - Transforms: Lazy operations (map, filter, join,…)
  - Actions: Return/write results (collect, save, count,…)

ML Software Framework: Spark MLlib

- Subproject with ML primitives
- Building blocks (as a framework vs. library)
  - Large scale statistics
  - Classification
  - Regression
  - Clustering
  - Matrix factorization
  - Optimization
  - Frequent pattern mining
  - Dimensionality reduction

What is ML-Scoring?

- Creates an Elastic Search (ES) document index of instances
- Trains a supervised learning ML model from a dataset of instances + labels
- Generate an Elasticsearch plugin that uses the trained ML model to score documents at query time

- A
-

An Open Source POC!

Remember the elephant?

Offline

Online

Query Generation and Contextual Pre-filtering

Model Building

Retrieval

Index Building

Contextual Post Filtering

Data Analytics

Ranking

Visualization / UI

Experimentation

Data/Events Collections

Simplifying the Stack!

Offline

Online

Query Generation and Contextual Pre-filtering

Model Building

Retrieval

Contextual Post Filtering

Ranking

Index Building

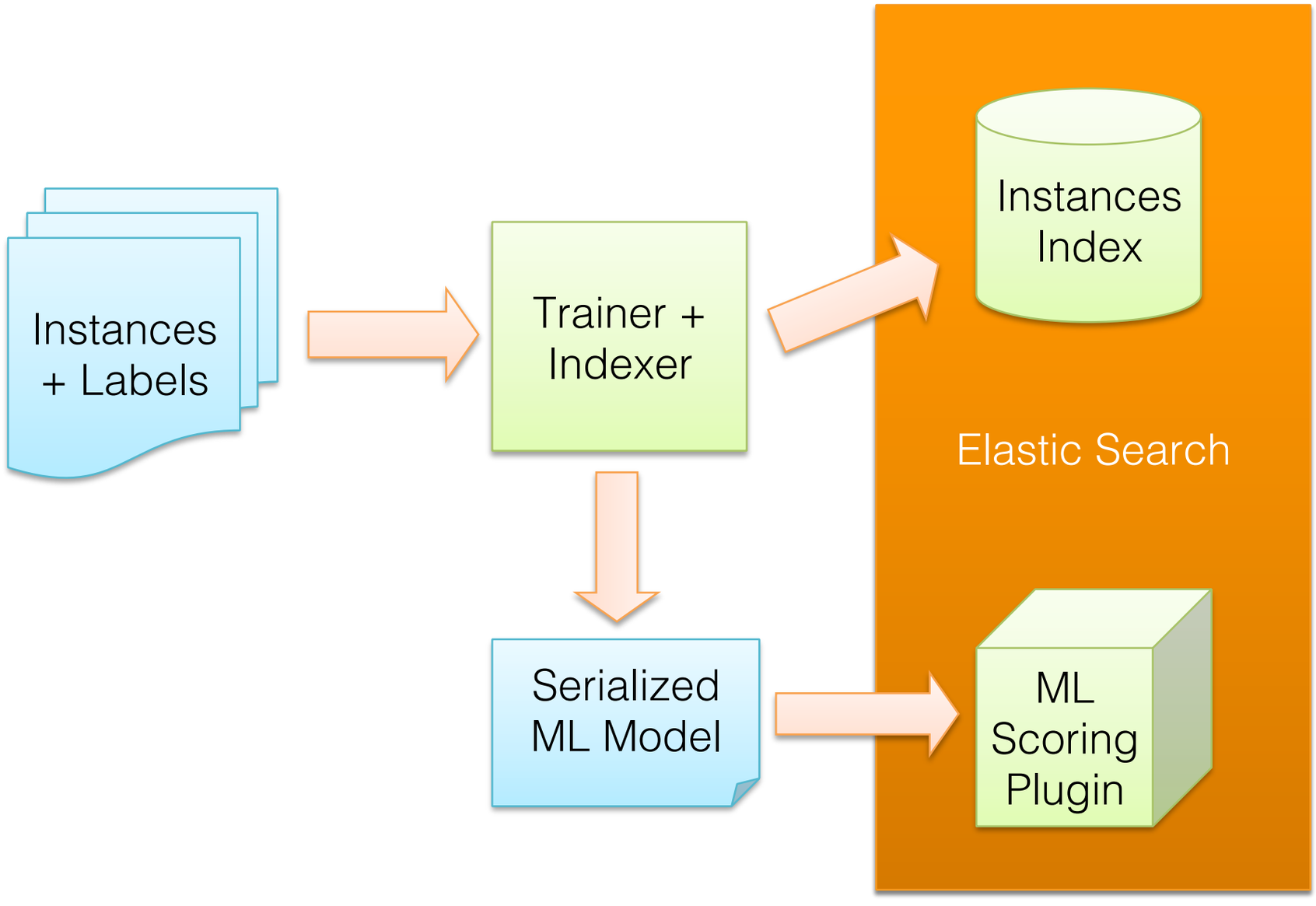
Data Analytics

Visualization / UI

Experimentation

Data/Events Collections

ML-Scoring Architecture



5. Re-writing the ranking function

Using ML-Scoring

- Creating an ES Index
- Boolean queries
- More-Like-This queries
- Built-in scoring functions
- Scoring script
- Scoring plugin
- ML-Score evaluator using Spark
- ML-Score query

Creating an Index in ES

```
POST /my_movie_catalog/movies/_bulk
{ "index": { "_id": 1 }}
{ ”genre" : “Documentary”, ”productID" : "XHDK-A-1293-#fJ3" , “title” : “Olympic Sports”, “content” : “Olympic greateness…“, price” : 20}
{ "index": { "_id": 2 }}
{ ”genre" : “Sports”, ”productID" : "KDKE-B-9947-#kL5", “title” : “NY Yankees: Winning the World Series”, , “content” : “There is no better team than the NY…“ “price” :20}
{ "index": { "_id": 3 }}
{ ”genre" : “Action”, “productID" : "JODL-X-1937-#pV7",”title” : “Rambo III”, , “content” : “Sylvester Stallone is evermore…“ “price” : 18}
{ "index": { "_id": 4 }}
{ ”genre" : “Children”, ”productID" : "QQPX-R-3956-#aD8", “title” : “Fairy Tale”, , “content” : “Once upon a time…“, “price” : 30}
```

Boolean queries

- SQL representation

SELECT movie

FROM movies

WHERE (price = 20 OR productID = "XHDK-A-1293-#fJ3")

AND (price != 30)

- ES DSL

```
GET /my_movie_catalog/movies/_search
    {
       "query" : {
          "filtered" : {
             "filter" : {
                "bool" : {
                  "should" : [
                     { "term" : {"price" : 20}},
                     { "term" : {"productID" : "XHDK-A-1293-#fJ3"}}
                  ],
                  "must_not" : {
                     "term" : {"price" : 30}
…
```

Content based similarity queries (MLT)

- The More Like This Query (MLT Query) finds documents that are "like" a given set of documents. In order to do so, MLT selects a set of representative terms of these input documents, forms a query using these terms, executes the query and returns the results.

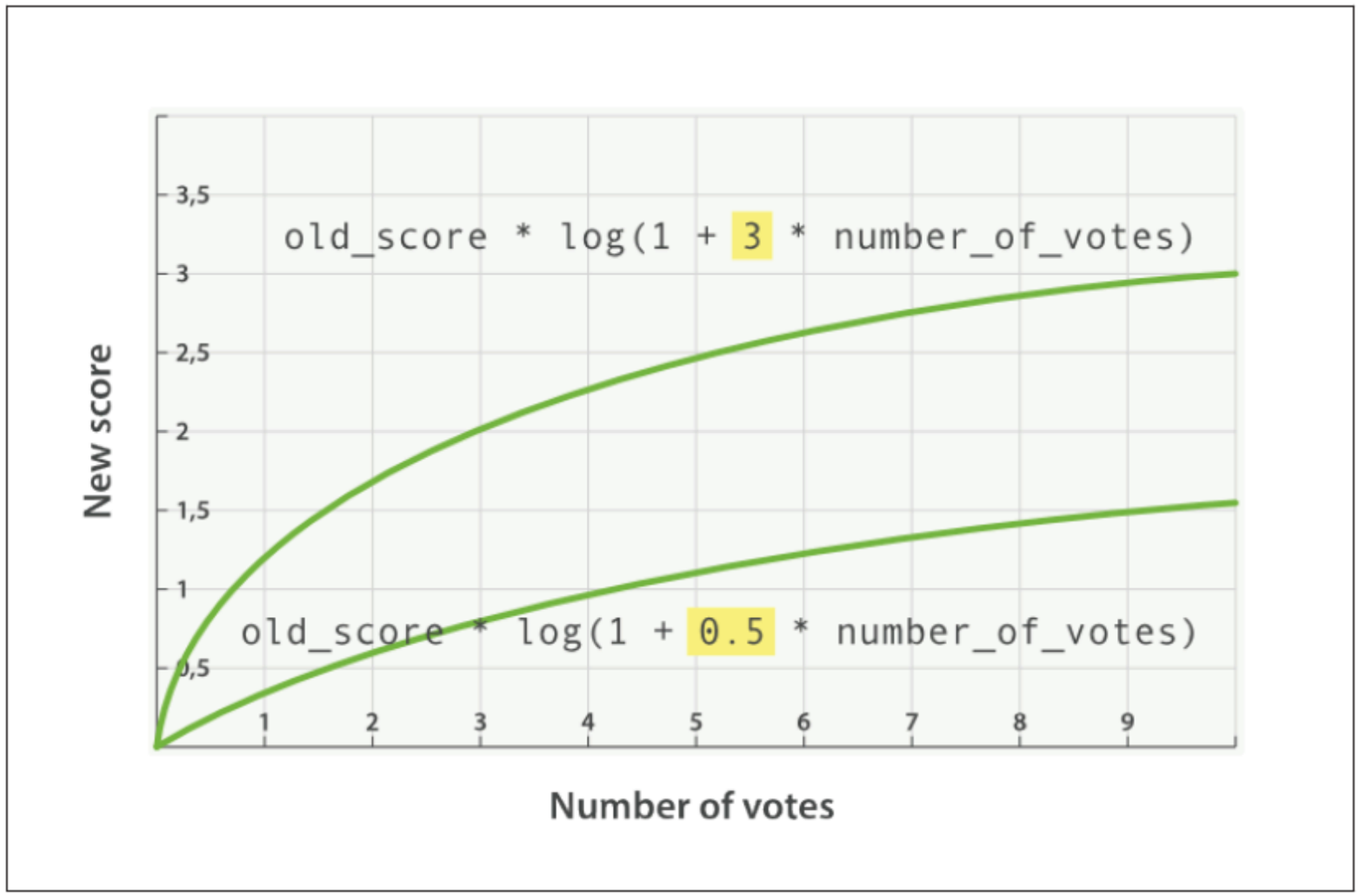
```
{
    "more_like_this" : {
        "fields" : ["title", "description"],
        "like_text" : "Once upon a time",
        "min_term_freq" : 1,
        "max_query_terms" : 12
    }
}
```

Similar to a given document

```
{
    "more_like_this" : {
        "fields" : ["title",
"description"],
        "docs" : [
        {
            "_index" : "imdb",
            "_type" : "movies",
            "_id" : "1"
        },
        {
            "_index" : "imdb",
            "_type" : "movies",
            "_id" : "2"
        }],
        "min_term_freq" : 1,
        "max_query_terms" : 12
    }
}
```

Built-in functions

- Suppose we want to boost movies by popularity (base-line of many RS)

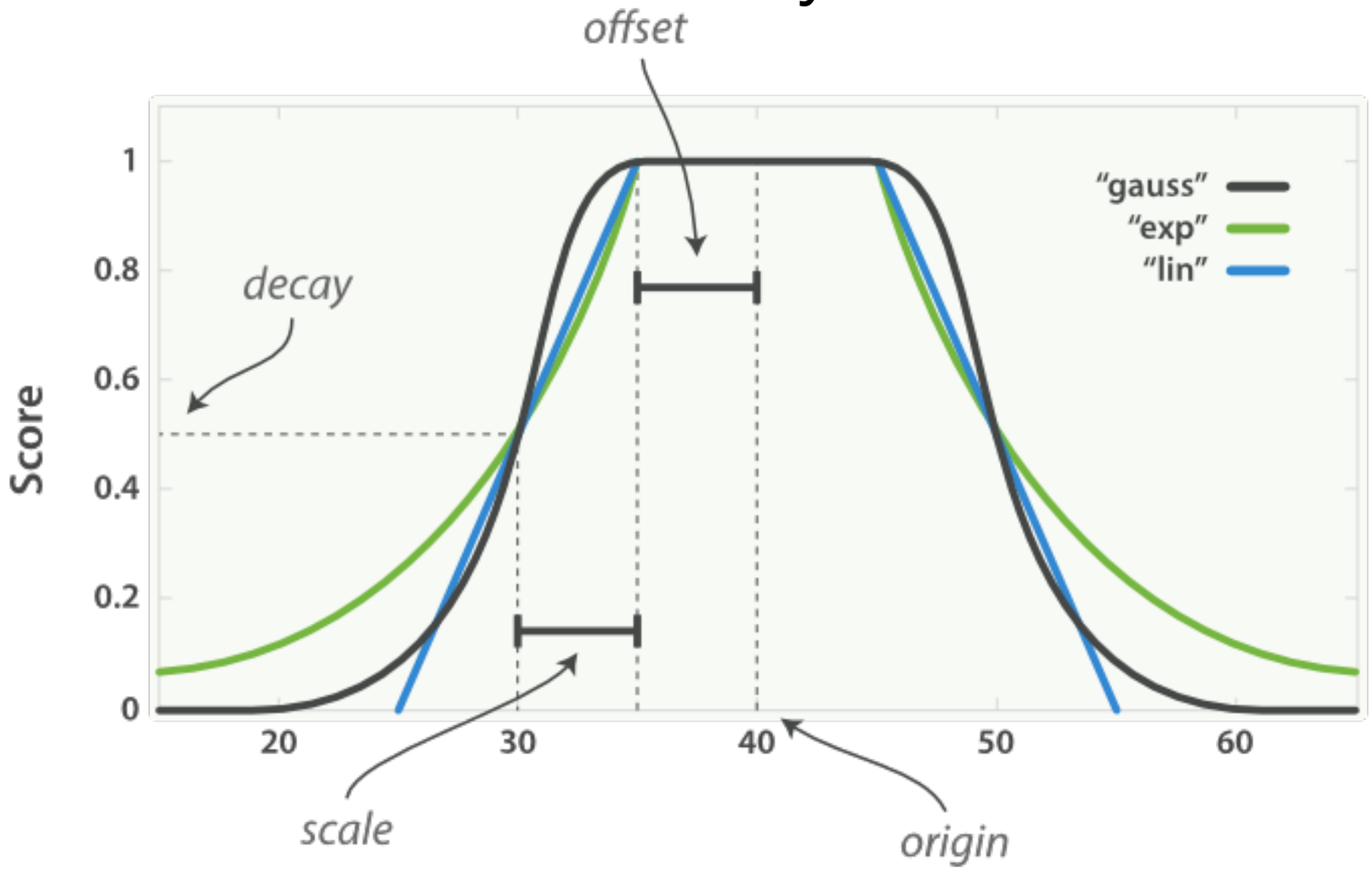


Popularity-based boosting

```
GET /my_movie_catalog/movies/post/_search
{
"query": {
   "function_score": {
      "query": {
         "multi_match": {
         "query": "popularity",
         "fields": [ "title", "content" ]
         }
      },
      "field_value_factor": {
         "field": "votes",
         "modifier": "log1p"
         }
      }
   }
}
```

Geo-Location

- Suppose we want to build a location-aware recommender system



Decay functions

- Supported decay functions
  - Linear
  - Gauss
  - Exp
- Also supported
  - random_score

```
GET /_search
{
"query": {
"function_score": {
    "functions": [
    {
        "gauss": {
        "location": {
            "origin": { "lat": 51.5, "lon": 0.12 },
            "offset": "2km",
            "scale": "3km"
        }
    }
},
{
    "gauss": {
    "price": {
        "origin": "50",
        "offset": "50",
        "scale": "20"
    }
},
"weight": 2
…
```

ES scoring script

- Trickier pricing and margin based scoring

```
if (price < threshold) {
   profit = price * margin
} else {
   profit = price * (1 - discount) * margin
}
return profit / target
```

ES Scoring Script

```
GET /_search
{
"function_score": {
 "functions": [
   { ...location clause... },
   { ...price clause... },
   {
    "script_score": {
       "params": { "threshold": 80, "discount": 0.1, "target":
10 },
       "script": "price = doc['price'].value; margin =
doc['margin'].value; if (price < threshold) { return price *
margin / target };return price * (1 - discount) * margin /
target; "}
    …
```

Limitations of ranking using ES practical scoring function

- Stateless computation
- Meant primarily for text search
- Hard to represent context and history
- Limited complexity (simple math functions only)
- Nevertheless, original score should not be discarded as it may become handy!

Scoring plugin in ES

```
public class PredictorPlugin extends AbstractPlugin {

   @Override
   public String name() {
    return getClass().getName();
   }

   @Override
   public String description() {
    return "Simple plugin to predict values.";
   }

   public void onModule(ScriptModule module) {
    module.registerScript(
       PredictorScoreScript.SCRIPT_NAME,
       PredictorScoreScript.Factory.class);
   }
}
```

ML-Scoring evaluator using Spark

```
class SparkPredictorEngine[M](val readPath: String, val spHelp:
SparkModelHelpers[M]) extends PredictorEngine {

  private var _model: ModelData[M] = ModelData[M]()

  override def getPrediction(values: Collection[IndexValue]) = {
   if (_model.clf.nonEmpty) {
     val v = ReadUtil.cIndVal2Vector( values, _model.mapper)
     _model.clf.get.predict(v)
   } else {
     throw new PredictionException("Empty model");
   }
  }

  def readModel() = _model = spHelp.readSparkModel(readPath)

  def getModel: ModelData[M] = _model

  …
```

ML-Scoring query

```
{
  "query": {
    "function_score": {
      "query": {
        "match_all": {}
      },
      "functions": [
        {
          "script_score": {
            "script": "search-predictor",
            "lang": "native",
            "params": {}
          }
        }
      ],
    "boost_mode": "replace"
    }
  }
}
```

https://github.com/sdhu/elasticsearch-prediction

Potential issues

- Performance
  - It may be a problem if the search space is very large and/or the computation to intensive
- Operations
  - Code running on a key infrastructure
  - Versioning and binary compatibility

Summary

- Importance of the whole picture – RS seen from the lenses of the whole elephant
- RS research is a new field in comparison to IR
- Scalability is hard! Why not learn from all of RS's cousins:
  - Search
  - Distributed systems
  - Databases
  - Machine learning
  - Content analysis
  - …
- Bridging the gap between research and engineering is an ongoing effort

References

- Baeza-Yates, R., & Ribeiro-Neto, B. 2011. *Modern information retrieval*. New York: ACM press.
- Chirita, P. A., Firan, C. S., & Nejdl, W. 2007. Personalized query expansion for the web. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 7-14). ACM.
- Croft, W. B., Metzler, D., & Strohman, T. 2010. *Search engines: Information retrieval in practice*. Reading: Addison-Wesley.
- Dunning, T. 1993. Accurate methods for the statistics of surprise and coincidence. *Computational linguistics*, *19*(1), 61-74.
- Elastic, Elasticsearch: RESTful, Distributed Search & Analytics. 2015. https://www.elastic.co/products/elasticsearch.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. 2009. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, *11*(1), 10-18.
- Ihaka, R., & Gentleman, R. 1996. R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, *5*(3), 299-314.

References

- Kantor, P. B., Rokach, L., Ricci, F., & Shapira, B. 2011. *Recommender systems handbook*. Springer.
- Manning, C. D., Raghavan, P., & Schütze, H. 2008. *Introduction to information retrieval*. Cambridge: Cambridge university press.
- Qiu, F., & Cho, J. 2006. Automatic identification of user interest for personalized search. In *Proceedings of the 15th international conference on World Wide Web* (pp. 727-736). ACM.
- Sun, J. T., Zeng, H. J., Liu, H., Lu, Y., & Chen, Z. 2005. Cubesvd: a novel approach to personalized web search. In *Proceedings of the 14th international conference on World Wide Web* (pp. 382-390). ACM.
- Xing, B., & Lin, Z. 2006. The impact of search engine optimization on online advertising market. In *Proceedings of the 8th international conference on Electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet* (pp. 519-529). ACM.
- Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. 2010. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (Vol. 10, p. 10)

Additional Credits

- Doug Kang
  - Data Scientist, Verizon OnCue
- Federico Ponte
  - System Engineer from Mahisoft
- Yessika Labrador
  - Data Engineer from Mahisoft