Building a Scalable Recommender System with Apache Spark, Apache Kafka and Elasticsearch
About

- @MLnick
- Principal Engineer, IBM
- Apache Spark PMC
- Focused on machine learning
- Author of *Machine Learning with Spark*
Agenda

- Recommender systems & the machine learning workflow
- Data modelling for recommender systems
- Why Spark, Kafka & Elasticsearch?
- Kafka & Spark Streaming
- Spark ML for collaborative filtering
- Deploying & scoring recommender models with Elasticsearch
- Monitoring, feedback & re-training
- Scaling model serving
- Demo
Recommender Systems & the ML Workflow
Recommender Systems

Overview
The Machine Learning Workflow

Perception

Data

???

Machine Learning

???

$$$

The Machine Learning Workflow

Reality

Data
- Historical
- Streaming

Ingest
- Feature transformation & engineering

Data Processing
- Model selection & evaluation

Model Training
- Pipelines, not just models
- Versioning

Deploy
- Predict given new data
- Monitoring & live evaluation

Live System

Spark DataFrames

Various

???

Spark ML

Feedback Loop

Stream (Kafka)

Missing piece!
The Machine Learning Workflow

Data
- User & Item Metadata
- Events

Ingest
- Aggregation
- Handle implicit data

Data Processing
- ALS
- Ranking-style evaluation

Model Training
- Model size & complexity

Deploy
- User & item recommendations
- Monitoring, filters

Live System

Spark ML

Elasticsearch

Spark DataFrames

Stream (Kafka)

Feedback => another Event Type
Data Modeling for Recommender Systems
User and Item Metadata

```
{
    "user_id": "1",
    "name": "Joe Bloggs",
    "created_date": "1476884080",
    "updated_date": "1476946916",
    "last_active_date": "1476946962",
    "age": 32,
    "country": "Spain",
    "city": "Seville",
    ...
}
```

Data model

```
{
    "item_id": "10",
    "name": "LOL Cats",
    "description": "catscatscats",
    "category": ["Cat Videos", "Humour", "Animals"],
    "tags": ["cat", "lol", "funny", "cats", "felines"],
    "created_date": "1476884080",
    "updated_date": "1476884080",
    "last_played_date": "1476946962",
    "likes": 100000,
    "author_id": "321",
    "author_name": "ilikecats",
    "channel_id": "CatVideoCentral",
    ...
}
```
User and Item Metadata

```json
{
    "user_id": "1",
    "name": "Joe Bloggs",
    "created_date": 1476884080,
    "updated_date": 1476946916,
    "last_active_date": 1476946962,
    "age": 32,
    "country": "Spain",
    "city": "Seville",
    ...
}
```

```
{
    "item_id": "10",
    "name": "LOL Cats",
    "description": "catscatscats",
    "category": ["Cat Videos", "Humour", "Animals"],
    "tags": ["cat", "lol", "funny", "cats", "felines"],
    "created_date": 1476884080,
    "updated_date": 1476884080,
    "last_played_date": 1476946962,
    "likes": 100000,
    "author_id": "321",
    "author_name": "ilikecats",
    "channel_id": "CatVideoCentral",
    ...
}
```
Anatomy of a User Event

User Interactions

Implicit preference data
- Page view
- eCommerce - cart, purchase
- Media – preview, watch, listen

Explicit preference data
- Rating
- Review

Intent data
- Search query

Social network interactions
- Like
- Share
- Follow
Anatomy of a User Event

Data model

```json
{
    "user_id": "1",
    "item_id": "10",
    "event_type": "page_view",
    "timestamp": 1476884080,
    "referrer": "http://spark.tc",
    "ip": "123.12.12.12",
    "device_type": "Smartphone",
    "user_agent_os": "Android",
    "user_agent_type": "Mobile Browser",
    "user_agent_family": "Chrome Mobile",
    "geo": "50.8503, 4.3517"
    ...
}
```
Anatomy of a User Event

How to handle implicit feedback?

```json
{
    "user_id": "1",
    "item_id": "10",
    "event_type": "page_view",
    "weight": 1.0,
    "timestamp": 1476884080,
    "referrer": "http://spark.tc",
    "ip": "123.12.12.12",
    "device_type": "Smartphone",
    "user_agent_os": "Android",
    "user_agent_type": "Mobile Browser",
    "user_agent_family": "Chrome Mobile",
    "geo": "50.8503, 4.3517"
    ...
}
```
Why Kafka, Spark & Elasticsearch?
Why Kafka?

Scalability
- De facto standard for a centralized enterprise message / event queue

Integration
- Integrates with just about every storage & processing system
- Good Spark Streaming integration – 1\textsuperscript{st} class citizen
- Including for Structured Streaming (but still very new & rough!)
Why Spark?

DataFrames
- Events & metadata are “lightly structured” data
- Suited to DataFrames
- Pluggable external data source support

Spark ML
- Spark ML pipelines – including scalable ALS model for collaborative filtering
- Implicit feedback & NMF in ALS
- Cross-validation
- Custom transformers & algorithms
Why Elasticsearch?

Storage
- Native JSON
- Scalable
- Good support for time-series / event data
- Kibana for data visualisation
- Integration with Spark DataFrames

Scoring
- Full-text search
- Filtering
- Aggregations (grouping)
- Search ~== recommendation (more later)
Kafka for Recommender Systems
Event Data Pipeline

```json
{
  "user_id": "1",
  "item_id": "10",
  "event_type": "page_view",
  "timestamp": 1476884080,
  "referrer": "http://spark.tc",
  "ip": "123.12.12.12",
  "device_type": "Smartphone",
  "user_agent_os": "Android",
  "user_agent_type": "Mobile Browser",
  "user_agent_family": "Chrome Mobile",
  "geo": "50.8503, 4.3517"
  ...
}
```
Write to Event Store

eventStream.foreachRDD { rdd =>
  rdd.map { case (key, event) =>
    val doc = Map(
      "userId"  -> event.userId,
      "itemId"  -> event.itemId,
      "eventType" -> event.eventType,
      "timestamp" -> event.timestamp,
      "weight"  -> event.weight,
      ...
    )
    val meta = Map(Metadata.ID -> event.eventId)
    (meta, doc)
  }.saveToEsWithMeta(Map("es.resource" -> "events-2016.11.14/events"))
}
Kibana Dashboards
Item Metadata Analytics

{  
    "item_id": "10",
    "name": "LOL Cats",
    "description": "catscatscats",
    "category": ["Cat Videos", "Humour", "Animals"],
    "tags": ["cat", "lol", "funny", "cats", "felines"],
    "created_date": 1476884080,
    "updated_date": 1476884080,
    "last_played_date": 1476946962,
    "likes": 100000,
    "author_id": "321",
    "author_name": "ilikecats",
    "channel_id": "CatVideoCentral",
    ...
}
User Metadata Analytics

```
{
    "user_id": "1",
    "name": "Joe Bloggs",
    "created_date": 1476884080,
    "updated_date": 1476946916,
    "last_active_date": 1476946962,
    "age": 32,
    "items": [{"id":"10","event_type":"purchase"},...]
    "country": "Spain",
    "city": "Seville",
...
```
Structured Streaming

```scala
val rawStream = spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "events")
  .load()
val eventStream = rawStream
  .selectExpr("CAST(value AS STRING)")
  .select(readEventUdf(...))
  .writeStream
  .foreach(new ESForeachWriter)
  .start()
```

**Status**

- Still early days
- Initial Kafka support in Spark 2.0.2
- No ES support yet – not clear if it will be a full-blown datasource or ForeachWriter
- For now, you can create a custom ForeachWriter for your needs
Spark ML for Collaborative Filtering
Collaborative Filtering

Matrix Factorization

\[
\begin{bmatrix}
3 \\
1 \\
5 \\
1 \\
2
\end{bmatrix}
\begin{bmatrix}
-1.1 & 3.2 & 4.3 \\
0.2 & 1.4 & 3.1 \\
2.5 & 0.3 & 2.3 \\
4.3 & -2.4 & 0.5 \\
3.6 & 0.3 & 1.2
\end{bmatrix}
\begin{bmatrix}
0.2 & 1.7 & 2.3 & 0.1 \\
1.9 & 0.4 & 0.8 & -0.3 \\
1.5 & -1.2 & 0.3 & 1.2
\end{bmatrix}
\]
Collaborative Filtering

Prediction

\[
\begin{bmatrix}
3 \\
1 \\
5 \\
1 \\
1 \\
2
\end{bmatrix}
\begin{bmatrix}
-1.1 & 3.2 & 4.3 \\
0.2 & 1.4 & 3.1 \\
2.5 & 0.3 & 2.3 \\
4.3 & -2.4 & 0.5 \\
3.6 & 0.3 & 1.2
\end{bmatrix}
\begin{bmatrix}
0.2 & 1.7 & 2.3 & 0.1 \\
1.9 & 0.4 & 0.8 & -0.3 \\
1.5 & -1.2 & 0.3 & 1.2
\end{bmatrix}
\]
Collaborative Filtering

Loading Data in Spark ML

```python
{,
    "user_id": "1",
    "item_id": "10",
    "event_type": "page_view",
    "weight": 1.0,
    "timestamp": 1476884080,
    "referrer": "http://spark.tc",
    "ip": "123.12.12.12",
    "device_type": "Smartphone",
    "user_agent_os": "Android",
    "user_agent_type": "Mobile Browser",
    "user_agent_family": "Chrome Mobile",
    "geo": "50.8503, 4.3517"
}
```

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

df = spark
    .read
    .format("es")
    .load("demo/events")

+----------+--------+-------------+----------+
<table>
<thead>
<tr>
<th>user_id</th>
<th>item_id</th>
<th>event_type</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>page_view</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>page_view</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>page_view</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>purchase</td>
<td>5.0</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>add_to_cart</td>
<td>2.0</td>
</tr>
</tbody>
</table>
+----------+--------+-------------+----------+
```
Alternating Least Squares

Implicit Preference Data

+----------------+----------------+----------------+----------------+
<table>
<thead>
<tr>
<th>user_id</th>
<th>item_id</th>
<th>event_type</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>page_view</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>page_view</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>page_view</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>purchase</td>
<td>5.0</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>add_to_cart</td>
<td>2.0</td>
</tr>
</tbody>
</table>
+----------------+----------------+----------------+----------------+

ratings = df
    .select("user_id", "item_id", "weight")
    .groupBy("user_id", "item_id")
    .sum().toDF("user_id", "item_id", "rating")

from pyspark.ml.recommendation import ALS
als = ALS(userCol="user_id",
          itemCol="item_id",
          ratingCol="rating",
          implicitPrefs=True)
model = als.fit(ratings)
Deploying & Scoring Recommendation Models
Prelude: Search

Full-text Search & Similarity

Analysis

Term vectors

Scoring

Ranking

"cat videos"

[0 1 ... 1 0 ...]

Similarity

[0 0 ... 0 1 ...]

[0 1 ... 1 1 ...]

[1 1 ... 0 0 ...]

[1 0 ... 0 1 ...]

Sort results
Can we use the same machinery?

**Recommendation**

**Analysis**
- Cannot use the same machinery for analysis.

**Term vectors**
- Cannot use term vectors for recommendations.

**Scoring**
- Can use similarity measures such as dot product and cosine similarity.

**Ranking**
- Can sort results based on similarity scores.

User (or item) vector: 
- User vector: [1.2, ..., -0.2, 0.3]

Similarity matrix:
- Similarity scores for different items.

Dot product & cosine similarity:
- The same as we need for recommendations!
Elasticsearch Term Vectors

Delimited Payload Filter

Raw vector → Custom analyzer → Term vector with payloads

\[
\begin{bmatrix}
1.2 & \cdots & -0.2 & 0.3
\end{bmatrix} \rightarrow 0|1.2 \cdots 3|-0.2 4|0.3
\]

```
'terms': {
'0': {
'term_freq': 1,
'tokens': [
{
'payload': 'P5mZmg==',
'position': 0
}
]
}
},
```
Custom scoring function

- Native script (Java), compiled for speed
- Scoring function computes dot product by:
  - For each document vector index ("term"), retrieve payload
  - \( \text{score} += \text{payload} \times \text{query}(i) \)
- Normalizes with query vector norm and document vector norm for cosine similarity

```json
{
  "function_score": {
    "query": {
      ...
    },
    "script_score": {
      "script": "payload_vector_score",
      "lang": "native",
      "params": {
        "field": "@model.factor",
        "vector": [1.2, ... , -0.2, 0.3],
        "cosine": True
      }
    },
    "boost_mode": "replace"
  }
}
```
Can we use the same machinery?

Recommendation

Analysis

Term vectors

Scoring

Ranking

User (or item) vector

Delimited payload filter

Custom scoring function

Sort results

\[
\begin{bmatrix}
1.2 & \cdots & -0.2 & 0.3
\end{bmatrix}
\]

\[
\begin{bmatrix}
-1.1 & 1.3 & \cdots & 0.4
\end{bmatrix}
\]

\[
\begin{bmatrix}
1.2 & -0.2 & \cdots & 0.3
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.5 & 0.7 & \cdots & -1.3
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.9 & 1.4 & \cdots & -0.8
\end{bmatrix}
\]
Elasticsearch Scoring

We get search engine functionality for free!

```
{
    "function_score": {
        "query": {
            ...
        },
        "script_score": {
            "script": "payload_vector_score",
            "lang": "native",
            "params": {
                "field": "@model.factor",
                "vector": [1.2,...,-0.2,0.3],
                "cosine": true
            }
        },
        "boost_mode": "replace"
    }
}
```

```
{
    "item_id": "10",
    "name": "LOL Cats",
    "description": "catscatscats",
    "category": ["Cat Videos", "Humour", "Animals"],
    "tags": ["cat", "lol", "funny", "cats", "felines"],
    "created_date": 1476884080,
    "updated_date": 1476884080,
    "last_played_date": 1476946962,
    "likes": 100000,
    "author_id": "321",
    "author_name": "ilikecats",
    "channel_id": "CatVideoCentral",
    ...
}
```
Alternating Least Squares

Deploying to Elasticsearch

<table>
<thead>
<tr>
<th>id</th>
<th>features</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>[-0.31136435, 0.4...]</td>
</tr>
<tr>
<td>20</td>
<td>[0.35291243, 0.13...]</td>
</tr>
<tr>
<td>30</td>
<td>[-0.19601235, 0.6...]</td>
</tr>
<tr>
<td>40</td>
<td>[-0.23222291, 0.8...]</td>
</tr>
<tr>
<td>50</td>
<td>[-0.14678353, 0.4...]</td>
</tr>
</tbody>
</table>

movie_vectors.write.format("es")

.option("es.mapping.id", "id")

.option("es.write.operation", "update")

.save("demo/movies", mode="append")
Monitoring & Feedback
System Events

Logging Recommendations Served

```
{
    "event_id": "rec123",
    "user_id": "1",
    "item_ids": ["10", "20", "34", "13", "43"],
    "event_type": "recommendation",
    "model_id": "model456",
    "timestamp": 1476884080,
    ...
}
```
System Events

Logging Recommendation Actions

```json
{
    "event_id": "rec123",
    "user_id": "1",
    "item_ids": ["10", "20", "34", "13", "43"],
    "actions": [
        {
            "item_id": "20",
            "action": "click",
            "timestamp": ...
        },
        ...
    ],
    "event_type": "recommendation",
    "model_id": "model456",
    "timestamp": 1476884080,
    ...
}
```
Tracking Performance

- Kafka
- Spark Streaming

Performance monitoring & alerts

Event store

Dashboards

Impression capping / fatigue
Scaling Model
Scoring
Scoring Performance

Scoring time per query, by factor dimension & number of items

- k=20
- k=50
- k=100

*3x nodes, 30x shards*
Scoring Performance

Increasing number of shards

Scoring time per query, by number of shards & number of items

*3x nodes, k=50
Locality Sensitive Hashing

- LSH hashes each input vector into $L$ “hash tables”. Each table contains a “hash signature” created by applying $k$ hash functions.
- Standard for cosine similarity is Sign Random Projections
- At indexing time, create a “bucket” by combining hash table id and hash signature
- Store buckets as part of item model metadata
- At scoring time, filter candidate set using term filter on buckets of query item
- Tune LSH parameters to trade off speed / accuracy
- LSH coming soon to Spark ML – SPARK-5992
Scoring Performance

Locality Sensitive Hashing

Scoring time per query - brute force vs LSH

*3x nodes, 30x shards, k=50, 1,000,000 items
Comparison to “score then search”

Scoring time per query – LSH vs score-then-search

*3x nodes, 30x shards, k=50, 1,000,000 items
Future Work
Future Work

- Apache Solr version of scoring plugin (any takers?)
- Investigate ways to improve Elasticsearch scoring performance
  - Performance for LSH-filtered scoring should be better!
  - Can we dig deep into ES scoring internals to combine efficiency of matrix-vector math with ES search & filter capabilities?
- Investigate more complex models
  - Factorization machines & other contextual recommender models
  - Scoring performance
- Spark Structured Streaming with Kafka, Elasticsearch & Kibana
  - Continuous recommender application including data, model training, analytics & monitoring
References

- Elasticsearch
- Elasticsearch Spark Integration
- Spark ML ALS for Collaborative Filtering
- Collaborative Filtering for Implicit Feedback Datasets
- Factorization Machines
- Elasticsearch Term Vectors & Payloads
- Delimited Payload Filter
- Vector Scoring Plugin
- Kafka & Spark Streaming
- Kibana
Thanks!

https://github.com/MLnick/elasticsearch-vector-scoring