CREATING A RECOMMENDER SYSTEM

An Elasticsearch & Apache Spark approach
ÁLVARO SANTOS ANDRÉS

Big Data & Analytics Solution Architect in Ericsson with more than 12 years of experience focused on Big Data projects including Personalization services, Recommenders, Data Lakes and many others. Born with Java, now a great lover of Scala, Functional Programming Languages and Apache Spark.
ABOUT ERICSSON
# Ericsson at a Glance

<table>
<thead>
<tr>
<th>NETWORKS</th>
<th>IT</th>
<th>MEDIA</th>
<th>INDUSTRIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create one network for a million different needs</td>
<td>Achieve business agility with transformative IT</td>
<td>Delight the TV consumer every day</td>
<td>Connect industries for business acceleration</td>
</tr>
<tr>
<td>42,000</td>
<td>Patents: 1 BILLION</td>
<td>Subscribers managed by us: 1 BILLION</td>
<td>Net Sales: 222,6 B. SEK</td>
</tr>
<tr>
<td>23,700</td>
<td>R&amp;D Employees: 2.5 BILLION</td>
<td>Subscribers supported by us: 2.5 BILLION</td>
<td>Countries with customers: 180</td>
</tr>
<tr>
<td>32,8 B. SEK</td>
<td>In R&amp;D: 66,000</td>
<td>Services professionals: 66,000</td>
<td>Employees: 111,000</td>
</tr>
</tbody>
</table>

Full year 2016 figures
### Proven Solutions

<table>
<thead>
<tr>
<th>Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;10</td>
<td>Experience Manager &amp; SQM Customers</td>
</tr>
<tr>
<td>&gt;15</td>
<td>Expert Analytics Customers</td>
</tr>
<tr>
<td>&gt;90</td>
<td>Mobile Positioning Customers</td>
</tr>
<tr>
<td>&gt;400</td>
<td>Patents in Location &amp; Analytics</td>
</tr>
</tbody>
</table>

### Leading CEM & Analytics player

- OSS leader for fifth year running
- Best Mobile CEM solution

### References

- [OSS leader for fifth year running](#)
- [Best Mobile CEM solution](#)
- [OSS leader for fifth year running](#)
- [Best Mobile CEM solution](#)

*Professional Services: proven capabilities in managing CEM transformations*
# Customer Reference Update

## CEM Transformation

- 3M subs with world's highest data usage per customer (~13Gbit/month)
- Full CEM transformation in Customer Care & Operations
- 100's of users, 2-3sec response time
- **Great business case:**
  - -90% escalations to 2nd line
  - -20% average handling time
  - +65% customer satisfaction
- Market leading customer satisfaction thanks to CEM

## NPS Improvement

- CEM contract for **22 markets with standardized apps across mobile and fixed** (446M subs)
- Several ongoing deployments and one Tier1 live during 2016
- True multi-vendor solution integrated with Vodafone Big Data program
- Enabling CEM transformation across opcos and org, with **Service Level Index (SLI)**
- Showing strong correlation to technical-NPS

---

“This tool is revolutionary - a game-changer”
EXPERT ANALYTICS
BUSINESS MODELS AND DEPLOYMENT SCENARIOS

As-a-Service

EEA
in the cloud

EEA
As part of
Managed Services

EEA
As part of App
Experience Optimization

Stand alone EEA applications

Planning
Operations
Care
Marketing
Monetization
IoT
3PP ecosystem

Cross-Domain Analytics

Expert Analytics (EEA)

Platform - enabling IoT & 3PP

Network attached

EEA
Embedded

Domain Specific

End-user devices
Mobile
Fixed
TV & Media

DSS attached

OSS/BSS

EEA
Embedded

Big Data Environment

3PP
Data Discovery

3PP
Big Data Lake

Other
(Social, IoT, Security...)

Online
Offline

Embedded

Probes
INDEX

› What is a Recommender System?
› Generic Architecture
› Batch and Streaming ETLs with Spark
› Searching items with Elasticsearch
› Applying Machine Learning using Spark
› Putting all together
WHAT IS A RECOMMENDER SYSTEM?
DEFINITION

› [Wikipedia] A recommender system or a recommendation system (sometimes replacing "system" with a synonym such as platform or engine) is a subclass of information filtering system that seeks to predict the "rating" or "preference" that a user would give to an item.

› [sciencedirect.com] Recommender systems are information filtering systems that deal with the problem of information overload by filtering vital information fragment out of large amount of dynamically generated information according to user’s preferences, interest, or observed behavior about item.
APPROACHES

› Non personalized
› Content-based filtering
› Collaborative filtering
› Hybrid recommender systems
A non-personalized recommender makes the same recommendations for everyone despite of the nature of the customer.

The simplest kind of recommendations would be lists of items sorted by rating or purchases.
Examples:
› Average of ratings
› Purchases / Clicks sums
› Trending products
› Advanced rankings (Reddit algorithm)
Content-based filtering provides alternatives items that are contextualized to the item viewed.

The system matches the attributes of the item viewed with attributes of other items to generate recommendations in order to find those attributes which are relevant for a certain product.
In order to find those attributes more relevant for a certain item, we use **NLP** techniques such as **TF/IDF**.

\[ \text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D) \]

\[ \text{tf}(t, d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}} \]

\[ \text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|} \]

- **TF**
- **DF**

High TF/IDF
Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from other users (collaborating).
COLLABORATIVE FILTERING II

It assumes that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

Types:
- **User to Item**: better conversion but harder to calculate
- **Item to Item**: less time dependent but less effective.
Alternating Least Squares or ALS is the more common CF algorithm and it is a Matrix Factorization technique.

It decompose the user-item ratings matrix in two:

› User matrix with its latent factors (characteristics).
› The transpose of the items matrix with its latent factor.
HYBRID RECOMMENDER SYSTEMS

› Hybrid recommenders are different combinations of the NP, CB and CF.
› These hybrids overcome the limitations of CB and CF improving their performance.
› Usually most commercial recommender systems are hybrid.
GENERIC ARCHITECTURE
3 STEPS ARCHITECTURE

Ingest  Pre-calculate  Retrieve
Every Recommender system needs data of the items and users in order to work.

› **Item Catalog**: information about the items. Daily/hourly updated.

› **User Ratings**: implicit or explicit ratings are generated in a continuous stream of data.
Recommender System drawbacks:
› Latency is a key factor in any personalization service.
› Calculate recommendations just-in-time are very expensive and slow.

The best approach is to pre-calculate Recommendations:
› Content Based: modern indexers store the data in a way that executing retrieval queries is fast.
› Collaborative filtering: train the ALS model and pre-calculate recs for the existing users and items.
STEP 3 – RETRIEVE

When a recommendation requests is received, just:

› **Content Based**: create a query and it will be executed by the indexer.
› **Collaborative filtering**: read the DB for the pre-calculated recommendations.
› **Hybrid approaches**: just ask to multiples sources and merge the results.
BATCH AND STREAMING ETLS WITH SPARK
DATA INGESTION - BATCH

1. Catalog is stored in files or in external DB tables.
2. Spark SQL parse and convert the catalog.
3. Using Dataframes, items are written in multiples databases.
SPARK SQL FOR BATCH

Spark SQL permit us:

› Work with *any kind of data* (structured, semi-structured or raw).
› Read from *multiple sources* (files, queues, NoSQL/SQL DBs…).
› *Parse, remove and transform* the data using SQL commands.
› Write to *different outputs* (MongoDB and Elasticsearch).
SPARK CODE EXAMPLE

1. **Read Raw Data**
2. **Parse and convert data to common structures**
3. **Write into MongoDB**
4. **Write into Elasticsearch**
DATA INGESTION - STREAMING

1. Rating Tracking are stored in a Kafka topic.
2. Spark Streaming reads and process the data from Kafka.
3. Spark SQL writes the rating in MongoDB.
Spark Streaming is the right choice:

- *Micro-batches* satisfy the latency requirements for a recommender system.
- *Scalable* and *fault-tolerant* streaming engine.
- *New Structured Streaming* will permit us to work with *Spark SQL* in a streaming way.
SPARK CODE EXAMPLE

1. Create Kafka Stream

2. Operate every Batch

3. Parse Raw Data

4. Write into MongoDB

```scala
val ds = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder] (ssc, params, Set(topic))

ds.foreachRDD({ rdd =>
    import spark.implicits._

    val dataDF = sqlContext.read.json(rdd.map(_.2))
    .as[Rating]

    dataDF.write
    .option("uri", mongoConf.uri)
    .option("collection", RATINGS_COLLECTION_NAME)
    .mode("append")
    .format("com.mongodb.spark.sql")
    .save()
})
```
SEARCHING ITEMS WITH ELASTICSEARCH
Elasticsearch is a search engine based on Lucene with very rich functionality:

› Provides a distributed, multitenant-capable full-text search engine.
› It has an HTTP web interface and schema-free JSON documents.
› Its engine permits to search items by text or similarity among items based on selected attributes.
ES QUERY EXAMPLES

1. **Search by text**

```json
{
  "more_like_this": {
    "fields": [
      "product.name"
    ],
    "like_text": "smartphone cover",
    "min_term_freq": 1,
    "max_query_terms": 12
  }
}
```

2. **Search similar items**

```json
{
  "more_like_this": {
    "fields": [
      "title",
      "description"
    ],
    "like": [
      {
        "_index": "imdb",
        "_type": "movies",
        "_id": "1"
      }
    ],
    "min_term_freq": 1,
    "max_query_terms": 12
  }
}
```
APPLYING MACHINE LEARNING USING SPARK
Apache Spark provides out of the box a very powerful machine learning library (MLLib):

- **Great variety of Machine Learning algorithms**: classification, regression, clustering and collaborative filtering.
- **Featurization**: feature extraction, transformation, dimensionality reduction and selection
- **Pipelines**: tools for constructing, evaluating and tuning ML Pipelines
- **Persistence**: saving and load algorithms, models, and Pipelines
- **Utilities**: linear algebra, statistics, data handling, etc.
Train the ALS model

1. Read rating from MongoDB

```scala
val ratings = spark.read
  .option("uri", mongoConf.uri)
  .option("collection", DatasetIngestion.REVIEWS_COLLECTION_NAME)
  .format("com.mongodb.spark.sql")
  .load()
  .select("$userId".as("user"), "$productId".as("item"), "$overall".cast(FloatType).as("rating"))
  .where("$userId".isNotNull && "$productId".isNotNull && "$overall".isNotNull)
  .cache

val als = new ALS()
  .setMaxIter(5)
  .setRegParam(0.01)
  .setUserCol("user")
  .setItemCol("item")
  .setRatingCol("rating")
val model = als.fit(ratings)
```

2. Train ALS model
Pre-calculate User-to-Items recommendations:

1. **Create the user-item matrix**
   ```scala
   val userProductsJoin = users.crossJoin(products)
   val userRating = userProductsJoin.map { row => Rating(row.getAs[Int](0), row.getAs[Int](1), 0) }
   ```

2. **Obtain all possible predictions**
   ```scala
   val recommendations = model.transform(userRating)
   .filter(col(model.getPredictionCol) > 0 && !col(model.getPredictionCol).isNaW )
   .groupByKey(p => (p.getAs[Int](model.getUserCol))
   .mapGroups { case (userId, predictions) =>
     val recommendations = predictions.toSeq.sorted(RatingOrder)
     .take(MAX_RECOMMENDATIONS)
     .map(p => Rec(p.getAs[Int](model.getItemCol), p.getAs[Float](model.getPredictionCol).toDouble))
     UserRecommendation(userId, recommendations)
   }
   ```

3. **Get the X biggest predictions for every user**

4. **Write to MongoDB**
   ```scala
   recommendations.write
   .option("uri", mongoConf.uri)
   .option("collection", USER_RECS_COLLECTION_NAME)
   .mode("overwrite")
   .format("com.mongodb.spark.sql")
   .save
   ```
Pre-calculate Item-to-Items recommendations:

1. **Create item-item matrix**

```scala
val recommendations = model.itemFactors.crossJoin(model.itemFactors)
  .filter(r => r.getAs[Int](0) != r.getAs[Int](2))
  .map { r =>
    (idA, idB, cosineSimilarity(new DoubleMatrix(featuresA), new DoubleMatrix(featuresB)))
  }
  .filter(col("_3") > 0 && !col("_3").isNa)
  .groupBy(p => p._1)
  .mapGroups { case (productId, predictions) =>
    val recommendations = predictions.toSeq.sorted(RatingOrder)
    .take(MAX_RECOMMENDATIONS)
    .map(p => Rec(p._2, p._3.toDouble))

    ProductRecommendation(productId, recommendations)
  }.toDF

recommendations.write
  .option("uri", mongoConf.uri)
  .option("collection", PRODUCT_RECS_COLLECTION_NAME)
  .mode("overwrite")
  .format("com.mongodb.spark.sql")
  .save
```

2. **Get all similarity between two items using their factors**

3. **Get the X biggest similarities for every item**

4. **Write to MongoDB**
PUTTING ALL TOGETHER
After having populated data to Elasticsearch and written pre-calculated the CF recommendations to DB, we only have to read them:

› Launch a query to Elasticsearch:

```scala
val query = QueryBuilders.moreLikeThisQuery(Array("id"),
  Array("features"),
  Array(new MoreLikeThisQueryBuilder.Item(indexName,
    PRODUCTS_INDEX_NAME,
    productId.toString()))

parseESResponse(esClient.prepareSearch()
  .setQuery(query)
  .setSize(MAX_RECOMMENDATIONS)
  .execute().actionGet())
```

› Get the proper pre-calculated recommendation:

```scala
val listRecs = mongoClient(mongoConf.db)(PRODUCT_RECS_COLLECTION_NAME)
  .findOne(MongoDBObject("id" -> productId))
  .orElse(MongoDBObject())

parseProductRecs(listRecs, MAX_RECOMMENDATIONS)
```
HYBRID RECS

Hybrid recommendations are a piece of cake:
› Launch a query to Elasticsearch for the CB recs.
› Read the proper pre-calculated CF recs from DB.
› Merge all the recs using some weights to give more importance to some of them:

```scala
val cfRecs = findProductCFRecs(productYmd, MAX_RECOMMENDATIONS)
  .map(x => new HybridRecommendation(x.productId, x.rating, x.rating * cfRatingFactor))
val cbRecs = findContentBasedMoreLikeThisRecommendations(productId, MAX_RECOMMENDATIONS)
  .map(x => new HybridRecommendation(x.productId, x.rating, x.rating * cbRatingFactor))
val finalRecs = cfRecs ::: cbRecs
```
TO FINISH
TIPS

Update Recs Periodically
CB and CF recommendations are very different. CF recs will required to be updated more periodically.

Use Spark cleverly
ALS is an heavy resources consumption algorithm and it requires to process great volumes of data. Pay attention to your Spark code, maybe caching or re-partition data could make big improvements.

Coldstart problem
When an item or user are new in the system (Coldstart problem), our approach may not be very effective. Therefore, an hybrid way is the key to success.
TAKE-INS

OPEN SOURCE TOOLS
However, there are plenty of tools that ease our duty (Spark MLLib, Elasticsearch...)

CHALLENGING
Creating a Recommender System is not easy and require some Mathematical background

READ DOCS
Spark requires a great knowledge of distributed programming, reading the official documentation will help you a lot
MORE INFO

› Online courses:
  - Coursera - Master Recommender Systems
  - Coursera - Machine Learning: Recommender Systems & Dimensionality Reduction

› Recommended books:
  - Mining of Massive Datasets [http://www.mmds.org]
  - Recommender Systems Handbook

› Articles:
  - Creating a Recommender System Part I & II:
    [http://blog.stratio.com/creating-a-recommender-system-part-i/]
    [http://blog.stratio.com/creating-recommender-system-part-two/]
THANKS A LOT
QUESTIONS & ANSWER

› Code Example:
  https://github.com/alvsanand/spark_recommender

› Social:
  https://github.com/alvsanand
  https://www.linkedin.com/in/alvsanand

› Email:
  alvaro.santos.andres@ericsson.com
  alvsanand@gmail.com

ÁLVARO SANTOS ANDRÉS