Real-time Recommendations using Spark

Jan Neumann & Sridhar Alla
Comcast Labs, Washington, DC
Who we are

- Jan Neumann, Lead of Big Data Research Team
- Sridhar Alla, Director/Big Data Architect, Enterprise BI
- Comcast Labs DC
  - Content Discovery for Comcast
  - Innovation
    - Machine Learning/Data Science Expertise for all of Comcast
    - Prototype new product ideas
What do we do?

**METADATA LIKE**

**IMDb**

- Powering millions of devices

**SEARCH LIKE**

**Google**

- Taking into account your TV channels, subscriptions and tastes

**RECOMMENDATIONS LIKE**

**Netflix**

- Including live programming
Goal: Real-time TV Recommendations

= “Trending For You”
How can we do this?

- Challenges
  - We have millions of users, thousands of programs
  - Programs on live TV are constantly changing (Cold-Start)
  - Approach inspired by “Google news personalization: scalable online collaborative filtering”, Das et al., 2007
Real-time Recommendations Algorithm

- Cluster user by taste profiles and geographic proximity
- Calculate Top K trending programs for each cluster
- Look up cluster for user and return trending programs
Real-time Recommendations in Spark

- Thanks to Spark we could implement this quickly

User History from HDFS → Batch: User Clustering with MLlib

Live Tune Activity via Kafka → Real-time: TopK Trending Programs per Cluster w/ Spark Streaming

Real-time Program recommendations per user
Batch: Cluster Users based on their Tastes

- Compute user taste vector from viewing history
- Cluster users to find groups with similar tastes
Implicit Matrix Factorization

- **ALS.trainImplicit**(view_count, k, numIter, alpha, lambda)

- For more info see Music Recommendations with Spark, Chris Johnson (Spotify), Spark Summit 2014

\[
\begin{bmatrix}
2.4 & 0 & 14.3 \\
0 & 7.1 & 0 \\
9.1 & 6.9 & 8.7 \\
23 & 0 & 0 \\
0 & 0 & 2 \\
\end{bmatrix}
\begin{bmatrix}
1.3 & 1 & 4.3 \\
1 & 2.1 & 1 \\
3.1 & 2 & 3 \\
7 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix} \times
\begin{bmatrix}
1 & 0 & 1 \\
0 & 1 & 0 \\
1 & 1 & 1 \\
1 & 0 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
- \begin{bmatrix}
U_1 \\
U_2 \\
U_3 \\
\end{bmatrix}
\begin{bmatrix}
M_1 & M_2 & M_3 \\
\end{bmatrix}
\]
Cluster Normalized User Vectors

- Spark Kmeans only provides clustering in Euclidean space
- $\|P_1 - P_2\|^2 = \|U_1 M - U_2 M\|^2 = (U_1 - U_2) M M^T (U_1 - U_2)^T \neq \|U_1 - U_2\|^2$
- `M.computeSVD` $\rightarrow M^T = X D Y^T$ where $X^T X = I, Y^T Y = I, D = \text{diag}$
- $(U_1 - U_2) M M^T (U_1 - U_2)^T = (U_1 - U_2) D D D (U_1 - U_2)^T = \|\tilde{U}_1 - \tilde{U}_2\|^2$
- With $\tilde{U} = U Y D$
- `KMeans.train(\tilde{U},numClusters,numIter)`
Real-time: Compute TopK TV programs per cluster

- Sessionize live tune activity for each viewer
- Aggregate popular programs across users for each cluster
- Keep Top K (using Twitter Algebird TopK Monoid)
Batch Implementation

# convert user viewing history to ratings (hash user_id to int)
val user_history = sc.textFile("user_history.dat")
val ratings = user_history.flat_map(parse_ratings)

# build matrix factorization model
val mf_model = ALS.train_implicit(ratings, rank, n, lambda, alpha)
val productRows = mf_model.productFeatures.map(s=>Vectors.dense(s._2))
val productRowMatrix = new RowMatrix(productRows)
# transform the movie feature matrix
val productSVD = productRowMatrix.computeSVD(svdRank)

val userFeatures = userRowMatrix.multiply(productSVD.V)
  .multiply(Matrices.diag(productSVD.s))

# use latent taste space to cluster users
val cluster_model = KMeans(userFeatures.rows,numClusters,numIter)

# compute map from user_ids to cluster ids
val user2cluster = mf_model.userFeatures.map(
    (bHashToUser.value(_._1),
     bClusterData.value.predict(Vectors.dense(_._2)))
)
Real-Time Implementation

// format
event_time|device_id|program_id|station_id|dma_title|tune_type

// get data from Kafka
val tuneEventsPerUser = KafkaUtils.createStream(ssc, zkQuorum, groupId, topics, storageLevel).flatMap(parseTuneEventByUser)

// what is being watched by each user
val userState =
tuneEventsPerUser.updateStateByKey(updateUserHistory).cache()

// aggregate tunes per program per cluster
val tvTunes = userState.map { case (userId, tuneInfo) =>
  ((tuneInfo.programId, user2cluster(userId)), 1) }
  .reduceByKey(_+_)
import com.twitter.algebird.TopKMonoid

case class ProgramCount (val programId: Long, val count: Int) extends Ordered[ProgramCount] {
    def compare(that: ProgramCount): Int = {...}
}

val topKMonoid = new TopKMonoid[ProgramCount](topk)

val tvTopK = tvTunes.map { case ((programId, clusterId), cnt) =>
    (clusterId, topKMonoid.build(ProgramCount(programId, cnt)))
}.reduceByKey(topKMonoid.plus)

// export top tunes to Hbase for lookup by web service ...

tvTopK.foreachRDD(rdd =>
    rdd.foreachPartition(p => (saveToHBase(p))))
Trending For You

- The Big Bang Theory
  - Rank: Score: 0.9999475
- Full House
  - Rank: Score: 0.97854805
- Duck Dynasty
  - Rank: Score: 0.90365034
- Bubble Guppies
  - Rank: Score: 0.882251
- MLB Baseball
  - Rank: Score: 0.81806295
- Victorious
  - Rank: Score: 0.7538549
- South Beach Tow
  - Rank: Score: 0.7431552
- Martin
  - Rank: Score: 0.7324555
- Family Feud
  - Rank: Score: 0.59335865
- Bad Girls Club
  - Rank: Score: 0.5184619
Results

- Leverage existing Hadoop infrastructure and data
- Compute 10 user clusters for 100k users in less than 10 minutes on a 25 Node Cluster (100 cores, 128GB RAM)
- Consume STB events on a real time basis directly from Kafka
- Calculate Top K trending programs for each cluster in 10 second micro batches storing the results using SparkOnHBase.
- Service requests for Personalized Trending Shows = Happy Customer😊
Final Words

- Thanks to Spark we implemented first version in a week
- Example accelerated adoption of Spark in dev & research
- Many further improvements possible
  - Do time-dependent clustering of user tastes
  - Gather feedback from real users
- We are hiring! Contact us at jobs.comcast.com