Big Data at Spotify

Anders Arpteg, Ph D

*Analytics Machine Learning, **Spotify***

- Quickly about me
- Quickly about Spotify
- What is all the data used for?
- Quickly about Spark
- Hadoop MR vs Spark
- Need for (distributed) speed
- Logistic regression in Scikit vs Spark
- SGD optimizer in Spark
- General thoughts so far
- Demo?

Quickly about me

- 1995 University of Kalmar
- 1997 The Buyer's Guide
- 2000 Ph D student, Kalmar + Linköping
- 2005 Assistant Professor, Kalmar
- 2007 Venture capital, research project
- 2007 TestFreaks, Pricerunner
  - 15,000+ sites worldwide
- 2011 Campanja, AI-team
  - Optimized Netflix worldwide
- 2013 Spotify, Graph data lead
- 2014 Spotify, Analytics ML manager

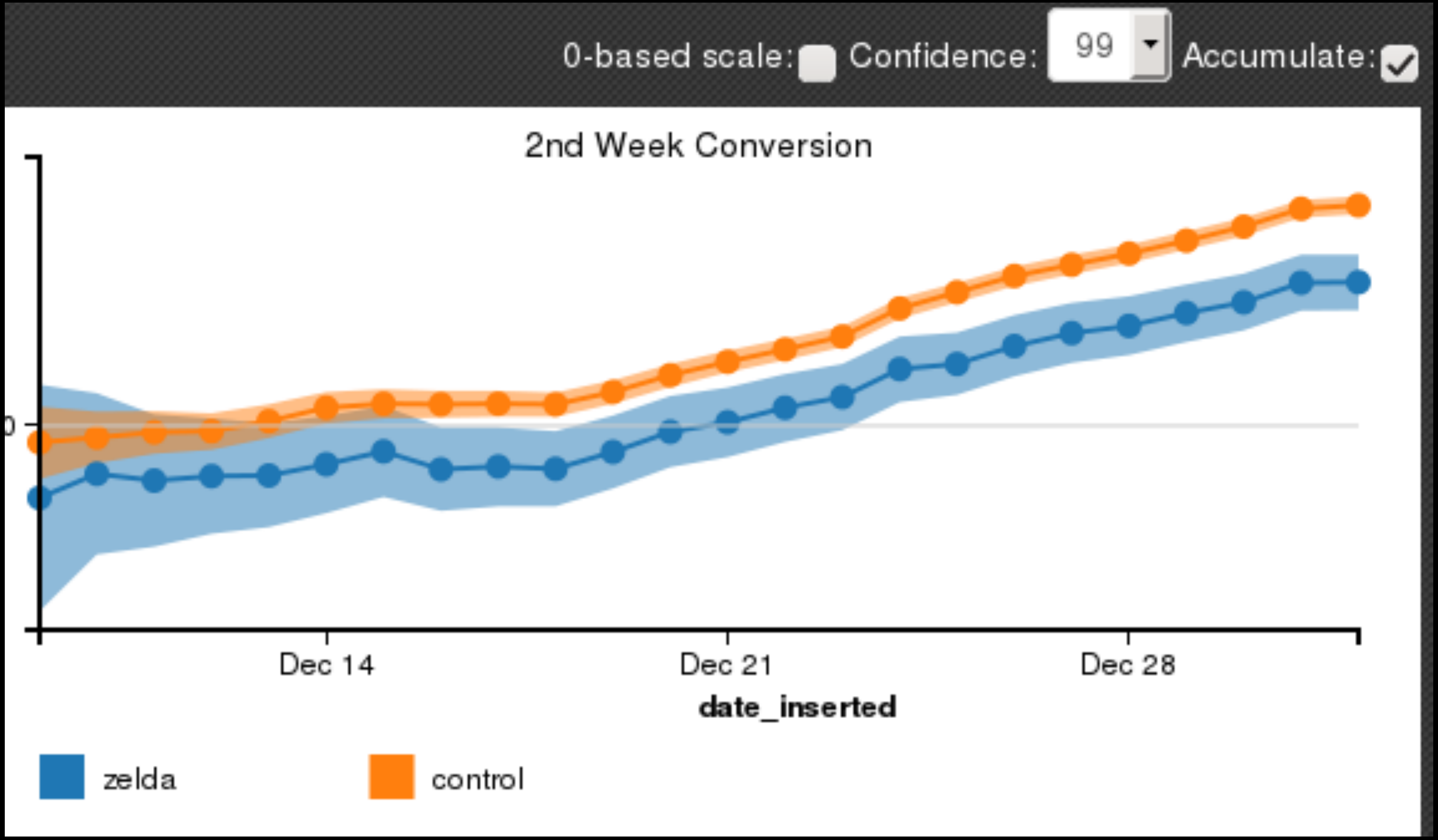
Quickly about Spotify

- 75+ million monthly active users
  - Launched in 58 different countries
  - 20+ million paying subscribers
- 30+ million licensed songs
  - 20,000 new songs every day
  - 1,5+ billion playlists created
- 14 TB of user/service-related log data per day
  - Expands to 170 TB per day
- 1200+ node Hadoop cluster
  - 50 PB of storage capacity, 48 TB of memory capacity

What is all the data used for?

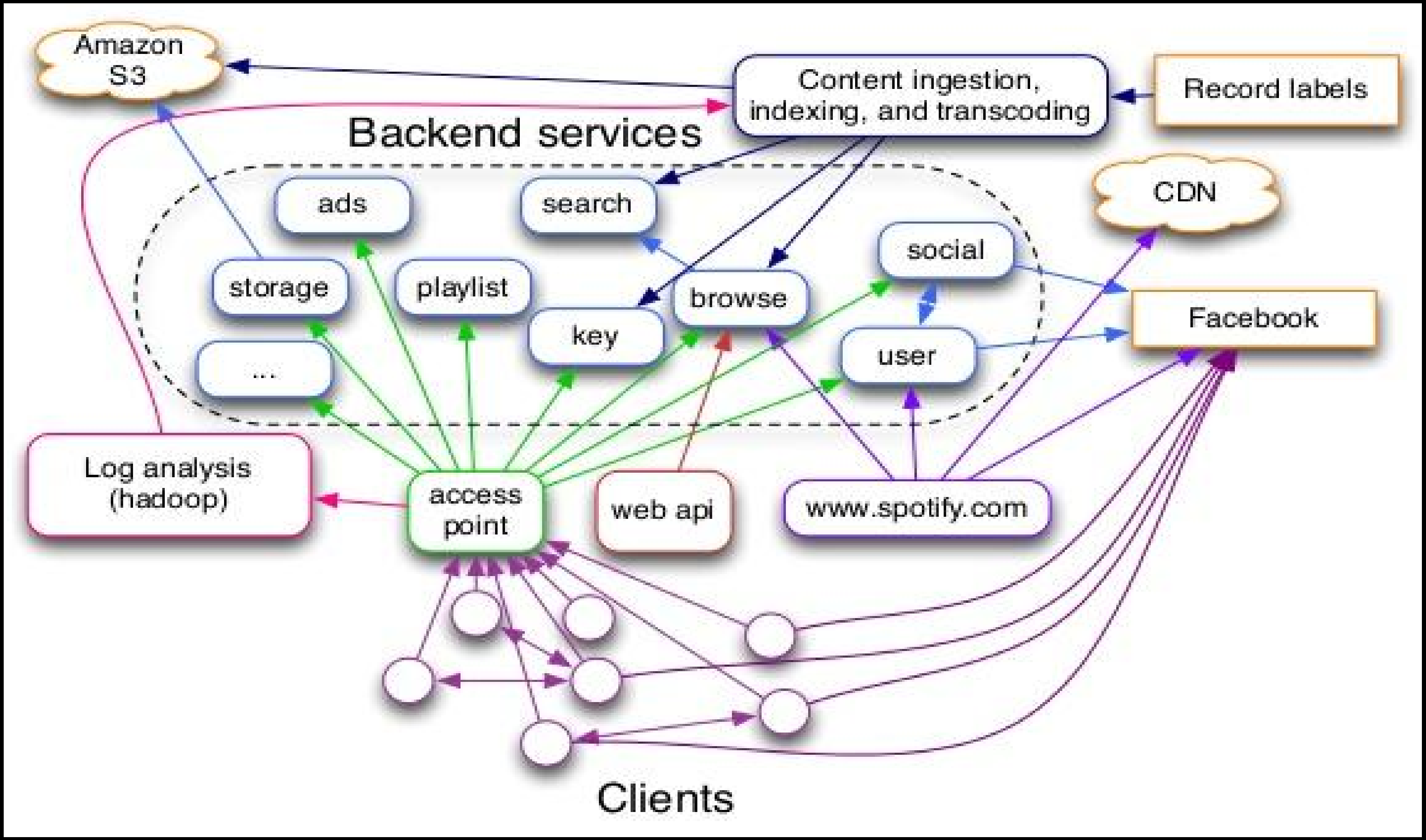
- Reporting to labels and right holders
- Product Features
  - Browse, search, radio, related artists, ...
  - A/B Testing
- Catalog quality
  - Artist disambiguation, track deduplication
- Business Analytics
  - KPI, DAU, MAU, SUBS, conversion, retention, ...
  - NPS analysis, understand the users
  - User funnel, awareness, activation, conversion, retention
- Marketing, growth, consumer insights
- Operational Analysis

A/B Testing

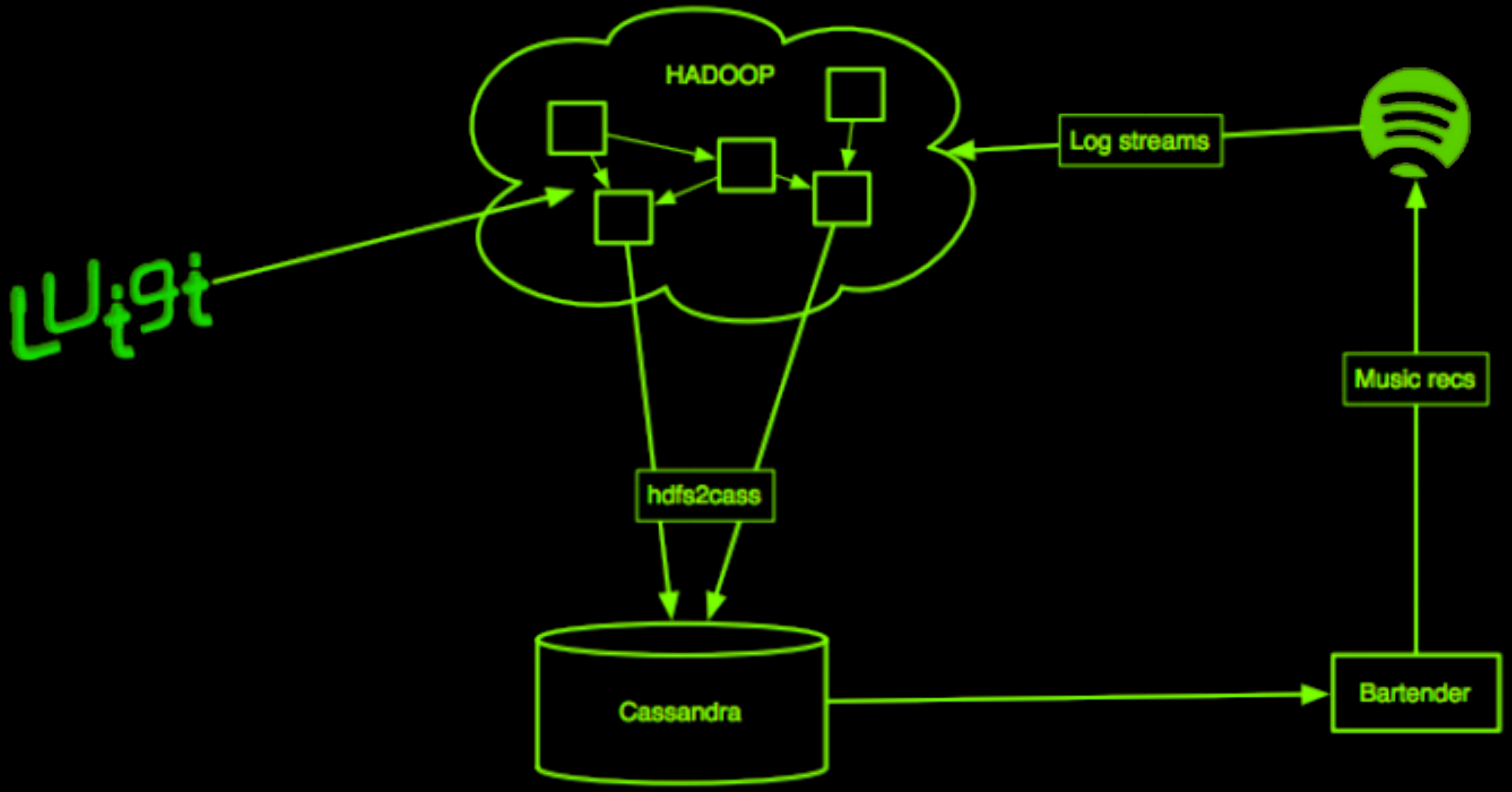


0-based scale:
Confidence:
99
Accumulate:
2nd Week Conversion
Dec 14
Dec 21
Dec 28
date_inserted
zelda
control

Spotify data architecture



The discovery data pipeline



Collaborative filtering

$$P = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \approx \begin{pmatrix} & X & \end{pmatrix} \begin{pmatrix} Y^T \end{pmatrix}$$

Y is all item vectors, X is all user vectors

- Approximate 60M users x 4M songs with 40 latent factors, ALS
- In short, minimize the cost function:

$$\sum_{u,i} c_{ui} \left(p_{ui} - x_u^T y_i\right)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

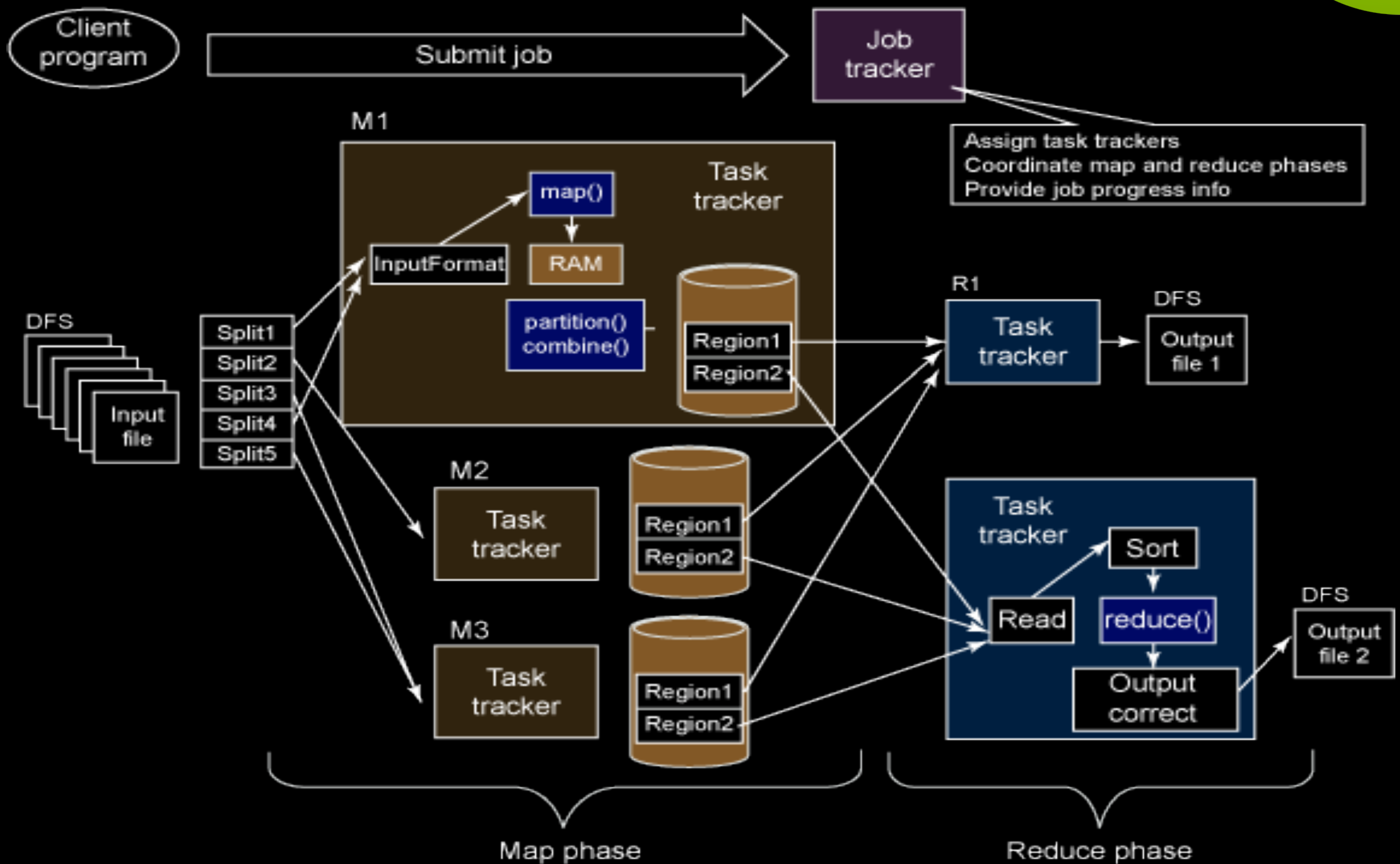
Next-generation Data Analytics

- Analytics 1.0 - Traditional statistical analysis
  - Statistical significance with ~1000 users
  - Centralized relational databases
- Analytics 2.0 - Big Data
  - Moving algorithms to data
  - Make it possible to handle big data
  - Volume, Variety, and Velocity
- Analytics 3.0 - Machine Learning & Real-time
  - Simplify distributed data processing
  - Decrease latency between incoming data and decision
  - Intelligent distributed machine learning algorithms

Next-generation Data Analytics (2)

- Hadoop 2+, YARN application
  - Killing classical Map/Reduce
  - Iterative algorithms in Spark, Tez, and Flink
- Streaming data (not just music)
  - Kafka, Storm, och Spark Streaming
  - Lambda architecture
- Improved storage formats
  - Columnar data storage
  - Parquet, ORC
- Simplified machine learning toolkits
  - Scikit-learn, Spark MLlib, IPython notebooks, R
  - Ubiquitous machine learning, ML for everyone
- Better tools for datawarehousing and dashboarding

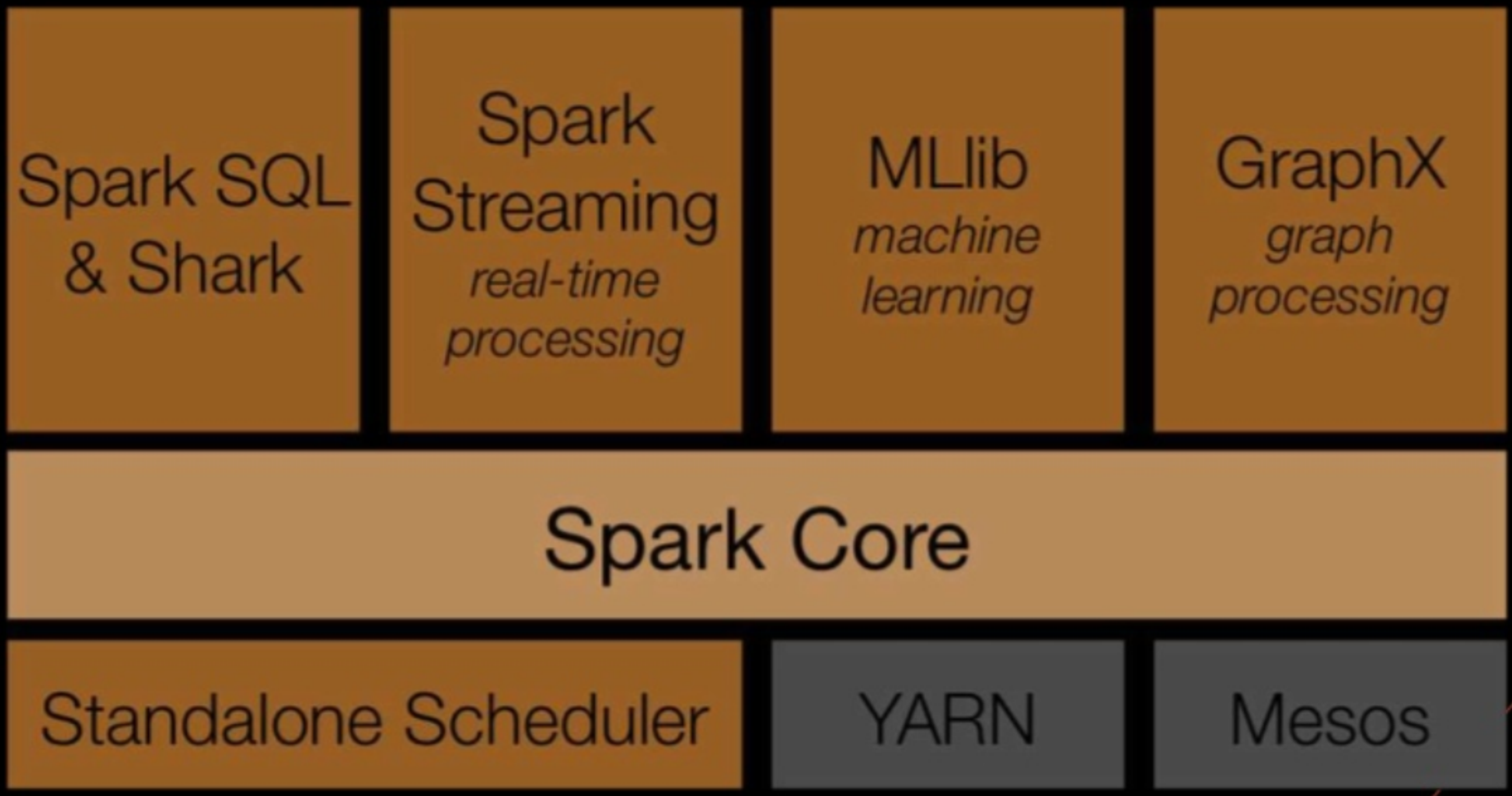
Quickly about classical map/reduce



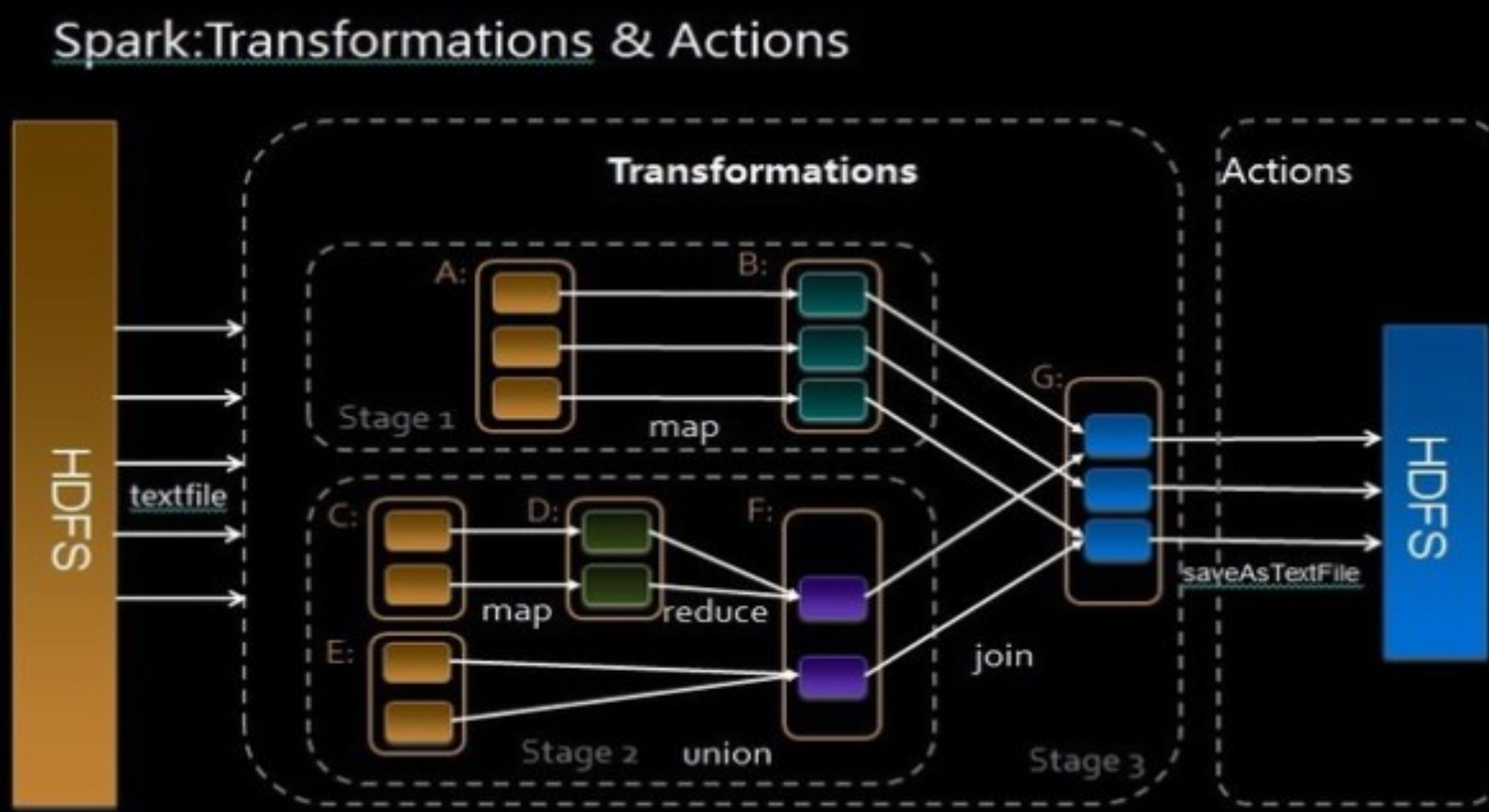
Quickly about classical map/reduce (2)

```python
class AbnormalExitJoinUserAggregated(hadoop.JobTask):
    """
    .. owner:: Elias Freider
       :email: freider@spotify.com
    """
    retention_days = spotify.luigi.retention.RETAIN_FOREVER
    date = luigi.DateParameter()
    secondary_sort = True

    def output(self):
        return hdfs.HdfsTarget(
            "/pipeline/insights/tech/abnormal_exit/AbnormalExitJoinUserAggregated/%s" % self.date
            )

    def requires(self):
        yield AbnormalExitJoinUser(date=self.date)

    def mapper(self, line):
        rec = line.strip('\n').split('\t')
        yield (rec[2], rec[3], rec[4], rec[5], rec[7]), rec[1], rec[1]

    def reducer(self, key, values):
        events = 0
        unique_users = 0
        last_user = None
        for v in values:
            events += 1
            if v != last_user:
                unique_users += 1
                last_user = v

        yield key, (events, unique_users)
```

Quickly about Spark



Quickly about Spark (2)



Spark Example with the RDD API

```
new ds.ad_reporting_metrics_anonym("2015-05-02".toDateTime).load(sc)
  .map(ad => (ad.getUserId.toStringOrEmpty.length, 1))
  .reduceByKey(_+_)
  .map(r => List(r._1.toString, r._2.toString).mkString("\t"))
  .saveAsTextFile("ad-counts")
```

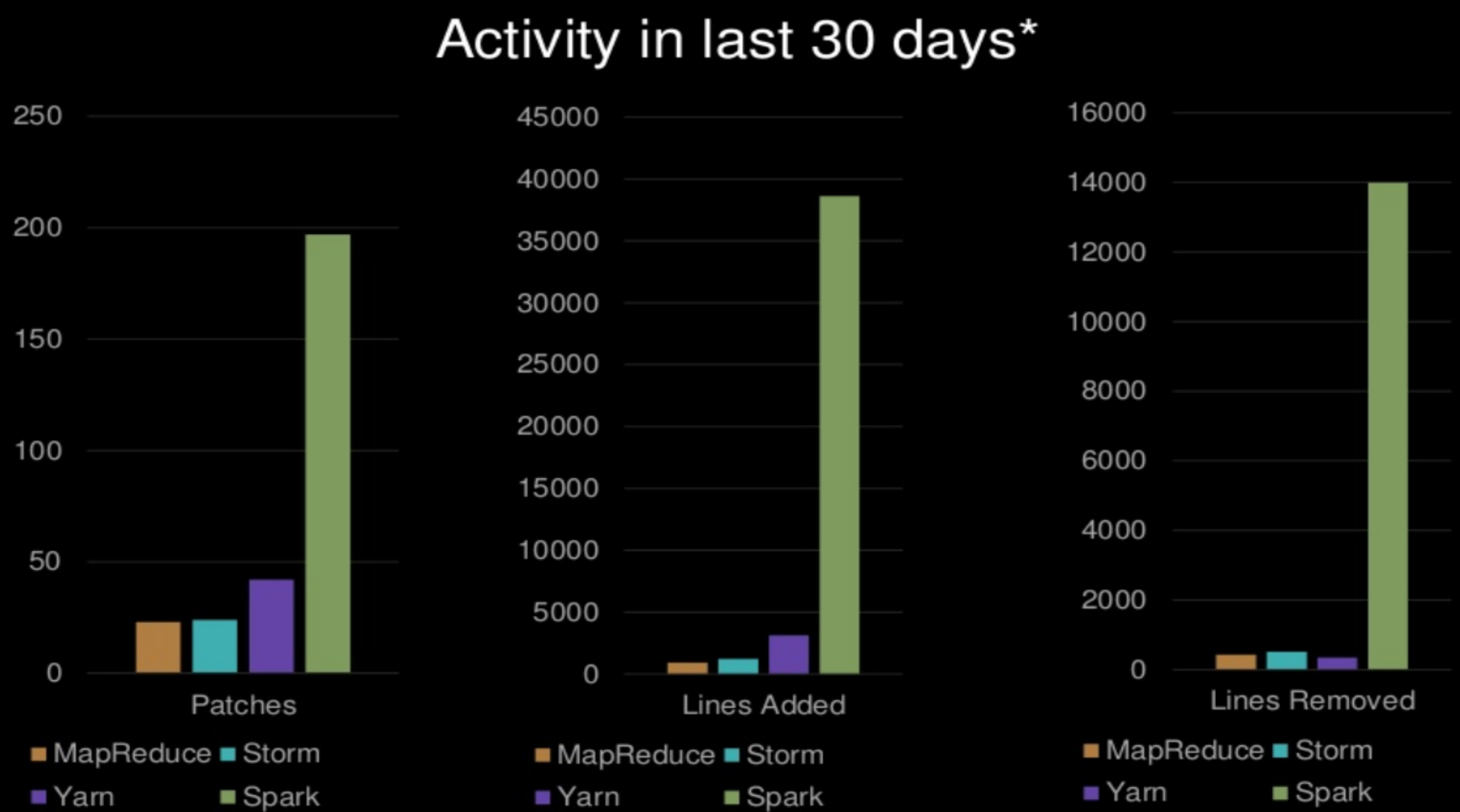
- Array of data distributed of workers
- Same API as normal arrays
  - Transforming: map, filter, reduceByKey, groupByKey, ...
  - Joining: joinByKey, leftOuterJoin, cogroup, zip, ..
  - Actions: count, saveAsAvro, saveAsText, ...
- Failure recovery, reruns failed tasks

Spark Example with the DataFrame API

```
val ads = new ds.ad_reporting_metrics_anonym_df(dt).load(sc)
val stats = ads
  .filter(s"line_item_id IN (${items.keys.mkString(",")})")
  .groupBy("line_item_id", "user_id", "product", "impressions", "clicks",
    "platform", "ad_type_name", "country", "city", "flight_name")
  .count()
store_performance_stats(stats, items_bc, p, dt)
//calc_group_ctr(stats, items_bc, p, dt)
//calc_group_promiscuity(stats, items_bc, p, dt)
calc_num_conversions(base_path, sc, p, dt)
```

- Higher level of abstraction than RDD
- Make use of schema-free data sources
  - Dynamic schema-awareness
- Additional optimizations performed automatically
- Same performance in Python as in Scala
- Similar API as Pandas and R

Quickly about Spark (5)



*as of June 1, 2014

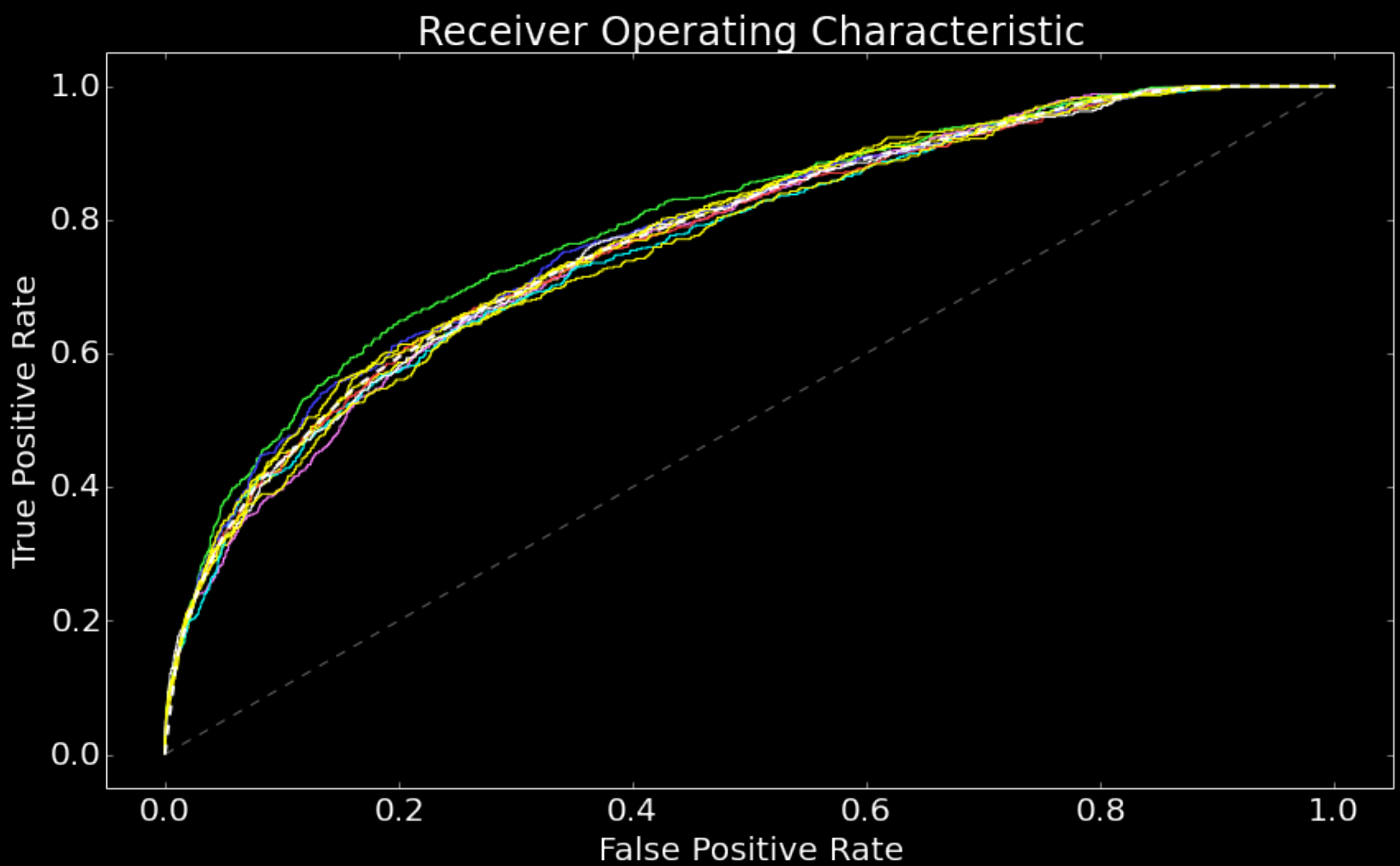
Problem Definition + Hypothesis

- Improve user targeting for house ads
  - *Identify users that are likely to convert given that they've seen house ads*
  - *Target less people with house ads, and retain as many conversions as possible*

$$P(C|A)$$

- *Hypothesis*
  - *By making use of information about users behaviour, demographics, and ad data, it will be possible to estimate likelihood of conversion with a logistic regression model.*
  - *Alternative algorithms*
    - *Navie Bayes, Decision Trees, Boosted Trees*
    - *Random Forest, SVM, ...*

Evaluation of the model



Need for (distributed) speed

- Steps to build the model
  - Extract data for training
  - Transform data into features
  - Train the model using the features
  - Evaluate the performance of the model
  - Tune the parameters
  - Extract data for prediction
  - Transform prediction data into features
  - Predict probability of conversion for all the users

- Main tools used
  - IPython notebook
  - Scikit learn library
  - Spark + MLlib

Running data extraction in Spark

Spark 1.3.0 | Jobs | Stages | Storage | Environment | Executors | com.spotify.analytics.house_ad_m...

Details for Job 3

Status: RUNNING
Active Stages: 24
Pending Stages: 63
Completed Stages: 35

Active Stages (24)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
121	keyBy at DataExtraction.scala:281 (kill) +details	2015/03/23 12:43:18	2.0 min	49/141	341.9 MB			804.5 MB
115	keyBy at DataExtraction.scala:281 (kill) +details	2015/03/23 12:43:18	2.0 min	70/138	491.9 MB			1157.4 MB
113	keyBy at DataExtraction.scala:281 (kill) +details	2015/03/23 12:43:18	2.0 min	79/138	553.8 MB			1302.6 MB
111	keyBy at DataExtraction.scala:281 (kill) +details	2015/03/23 12:43:18	2.0 min	69/135	495.9 MB			1167.8 MB
109	keyBy at DataExtraction.scala:281 (kill) +details	2015/03/23 12:43:17	2.0 min	89/124	640.1 MB			1507.2 MB
107	keyBy at DataExtraction.scala:281 (kill)	2015/03/23	2.0 min	94/125	623.8			1464.0

More often like this

8	filter at DataExtraction.scala:279 +details	2015/03/23 12:46:52	41 s	2649/2649	33.8 GB	650.1 MB
119	keyBy at DataExtraction.scala:133 +details	2015/03/23 12:46:14	35 s	2649/2649	5.3 GB	5.3 GB

Failed Stages (30)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write	Failure Reason
122	filter at DataExtraction.scala:279 +details	2015/03/23 12:46:53	12 min	187/2649 (5	687.2 MB			23.4 MB	org.apache.spark.shuffle.MetadataFetchFailedExcep Missing an output location for shuffle 1 +d
116	filter at DataExtraction.scala:279 +details	2015/03/23 12:46:53	12 min	159/2649 (2	584.4 MB			38.9 MB	org.apache.spark.shuffle.MetadataFetchFailedExcep Missing an output location for shuffle 1 +d
114	filter at DataExtraction.scala:279 +details	2015/03/23 12:46:53	12 min	173/2649 (1	635.7 MB			62.3 MB	org.apache.spark.shuffle.MetadataFetchFailedExcep Missing an output location for shuffle 1 +d
112	filter at DataExtraction.scala:279 +details	2015/03/23 12:46:53	12 min	151/2649 (3	555.0 MB			71.2 MB	org.apache.spark.shuffle.MetadataFetchFailedExcep Missing an output location for shuffle 1 +d
110	filter at DataExtraction.scala:279 +details	2015/03/23 12:46:53	12 min	105/2649 (1	386.0 MB			61.0 MB	org.apache.spark.shuffle.MetadataFetchFailedExcep Missing an output location for shuffle 1 +d
108	filter at DataExtraction.scala:279	2015/03/23 12:46:53	12 min	273/2649 (16	1007.1 MB			188.1 MB	org.apache.spark.shuffle.MetadataFetchFailedExcep Missing an output location for shuffle 1 +d

Quickly about Logistic Regression

$$Y = user\ converted\ and\ seen\ house\ ads$$
$$X = \{behaviour, demographics, ads\}$$
$$P(Y|X) = likelihood\ that\ user\ will\ convert$$
$$\log \frac{P}{1-P} = w^T x = w_0 + w_1 x_1 + \ldots + w_d x_d$$
$$P(Y|X, w) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}$$
$$MLE = \operatorname*{argmax}_w P(Y|X, w)$$

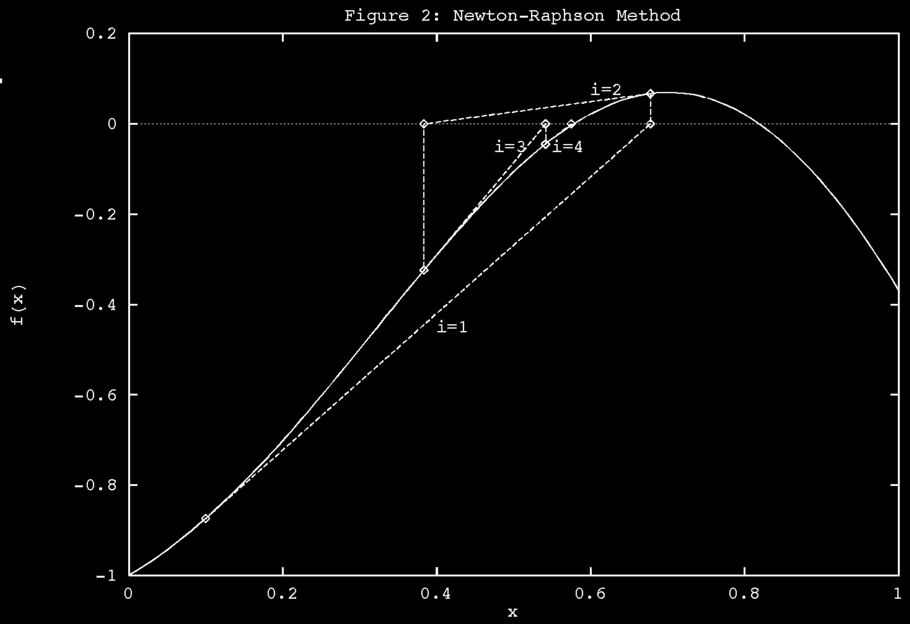
Logistic Regression in Scikit-learn

- L2 regularized optimization problem in liblinear

$$\min_{w} \quad \frac{1}{2}w^T w + C \sum_{i=1}^{l} \log(1 + e^{-y_i w^T x_i}).$$

- Newton Raphson solver

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$



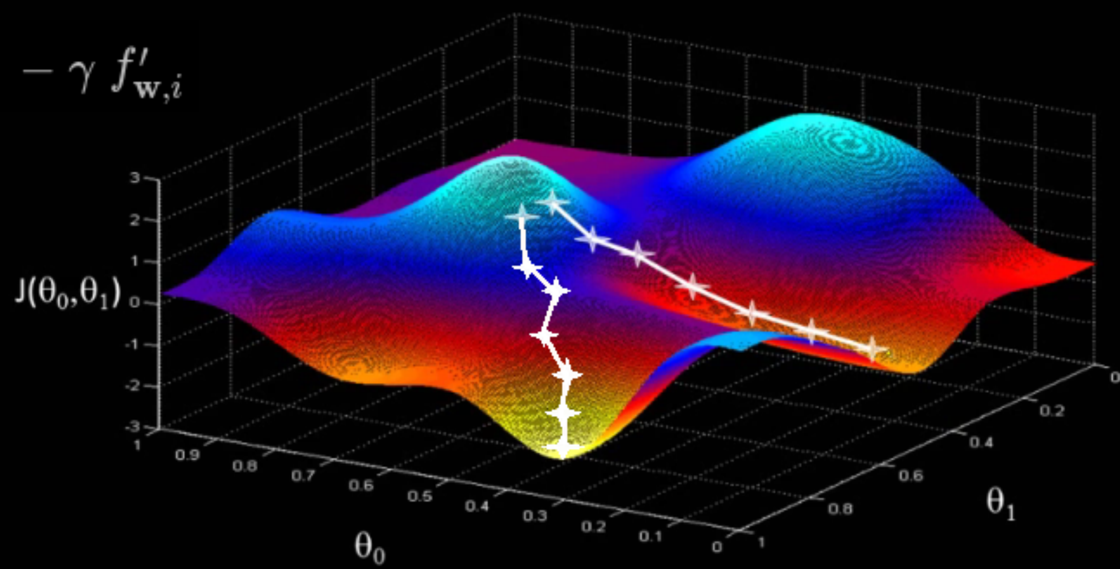
Logistic Regression in Spark

- Stochastic gradient descent
    - Params: step size, use intercept, regularization, batch size

$$f(\mathbf{w}) := \lambda\, R(\mathbf{w}) + \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{w}; \mathbf{x}_i, y_i)\ .$$

$$f'_{\mathbf{w},i} := L'_{\mathbf{w},i} + \lambda\, R'_{\mathbf{w}}\ ,$$

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \gamma\, f'_{\mathbf{w},i}$$



SGD implementation in Spark

```scala
for (i <- 1 to numIterations) {
  val bcWeights = data.context.broadcast(weights)
  // Sample a subset (fraction miniBatchFraction) of the total data
  // compute and sum up the subgradients on this subset (this is one map-reduce)
  val (gradientSum, lossSum, miniBatchSize) = data.sample(false, miniBatchFraction, 42 + i)
    .treeAggregate((BDV.zeros[Double](n), 0.0, 0L))(
      seqOp = (c, v) => {
        // c: (grad, loss, count), v: (label, features)
        val l = gradient.compute(v._2, v._1, bcWeights.value, Vectors.fromBreeze(c._1))
        (c._1, c._2 + l, c._3 + 1)
      },
      combOp = (c1, c2) => {
        // c: (grad, loss, count)
        (c1._1 += c2._1, c1._2 + c2._2, c1._3 + c2._3)
      })

  if (miniBatchSize > 0) {
    /**
     * NOTE(Xinghao): lossSum is computed using the weights from the previous iteration
     * and regVal is the regularization value computed in the previous iteration as well.
     */
    stochasticLossHistory.append(lossSum / miniBatchSize + regVal)
    val update = updater.compute(
      weights, Vectors.fromBreeze(gradientSum / miniBatchSize.toDouble), stepSize, i, regParam)
    weights = update._1
    regVal = update._2
  } else {
    logWarning(s"Iteration ($i/$numIterations). The size of sampled batch is zero")
  }
}
```

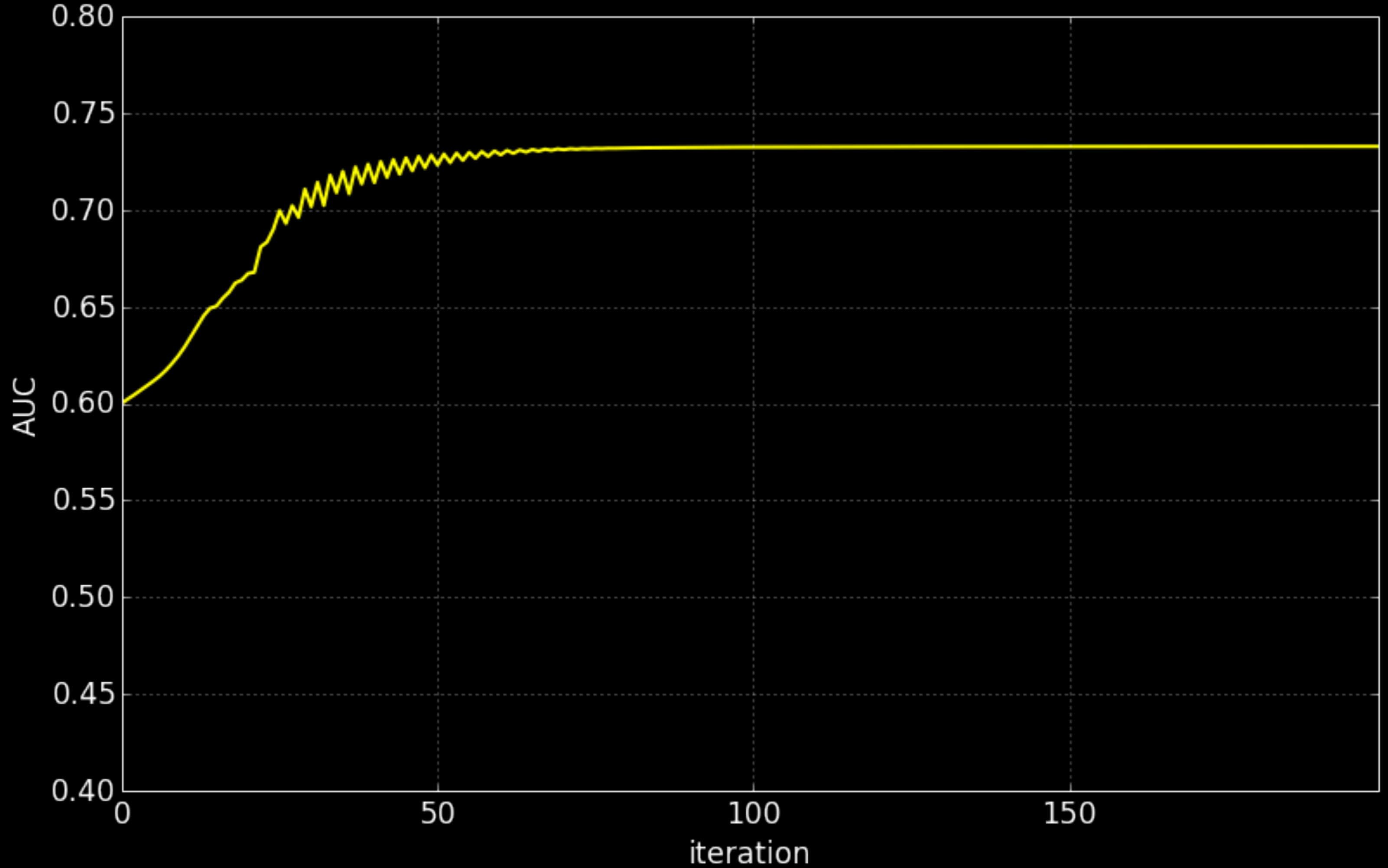
Calculation of the gradient

```scala
/**
 * For Binary Logistic Regression.
 *
 * Although the loss and gradient calculation for multinomial one is more generalized
 * and multinomial one can also be used in binary case, we still implement a speciali
 * binary version for performance reason.
 */
val margin = -1.0 * dot(data, weights)  data: "[1.0,0.04415584415584415,0.56228956229
val multiplier = (1.0 / (1.0 + math.exp(margin))) - label
axpy(multiplier, data, cumGradient)
if (label > 0) {
  // The following is equivalent to log(1 + exp(margin)) but more numerically stable.
  MLUtils.log1pExp(margin)
} else {
  MLUtils.log1pExp(margin) - margin
}
```

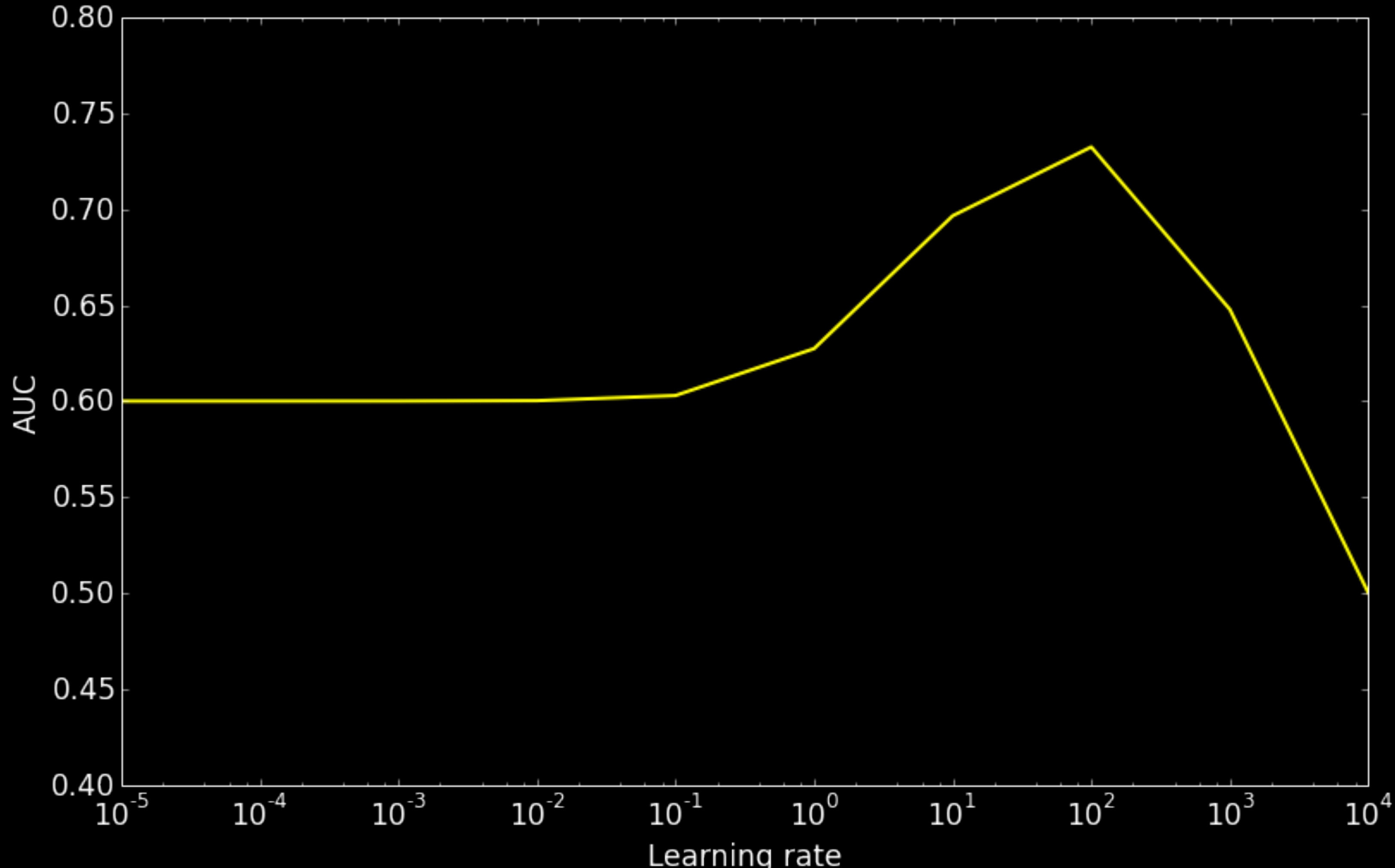
Updating of the weights

```scala
/**
 * :: DeveloperApi ::
 * Updater for L2 regularized problems.
 *          R(w) = 1/2 ||w||^2
 * Uses a step-size decreasing with the square root of the number of iterations.
 */
@DeveloperApi
class SquaredL2Updater extends Updater {
  override def compute(
      weightsOld: Vector,
      gradient: Vector,
      stepSize: Double,
      iter: Int,
      regParam: Double): (Vector, Double) = {
    // add up both updates from the gradient of the loss (= step) as well as
    // the gradient of the regularizer (= regParam * weightsOld)
    // w' = w - thisIterStepSize * (gradient + regParam * w)
    // w' = (1 - thisIterStepSize * regParam) * w - thisIterStepSize * gradient
    val thisIterStepSize = stepSize / math.sqrt(iter)
    val brzWeights: BV[Double] = weightsOld.toBreeze.toDenseVector
    brzWeights :*= (1.0 - thisIterStepSize * regParam)
    brzAxpy(-thisIterStepSize, gradient.toBreeze, brzWeights)
    val norm = brzNorm(brzWeights, 2.0)

    (Vectors.fromBreeze(brzWeights), 0.5 * regParam * norm * norm)
  }
}
```

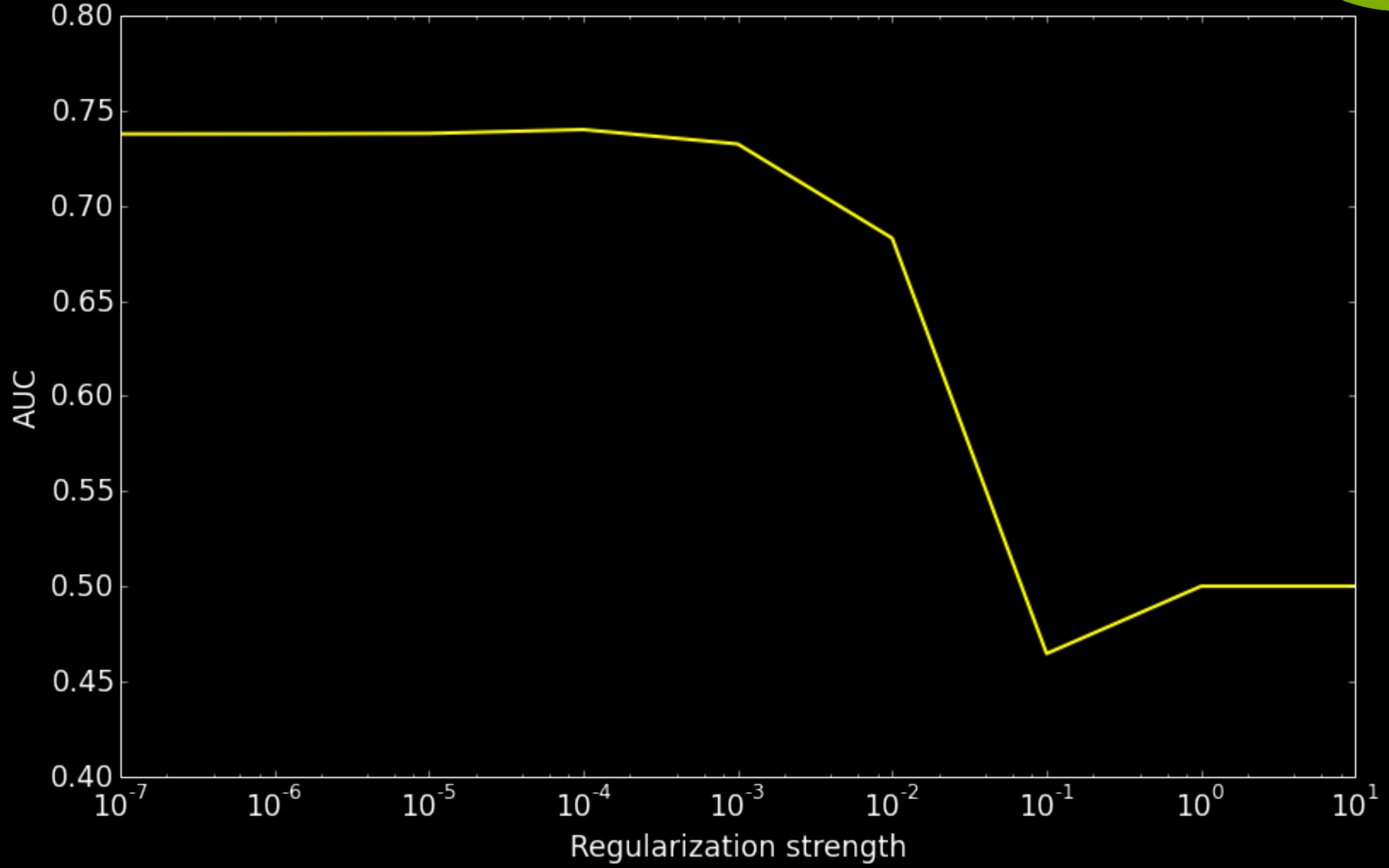
SGD Convergence



Learning rate (step size) tuning



Regularizaton tuning



Thoughts about Spark

- Advantages with Spark
    - General purpose engine (batch, streaming, sql, graph)
    - Faster Yarn engine, DAG optimization and less IO
    - High level machine learning library
    - RDD, failure recovery, data locality
    - Generic caching and accumulators
    - Nice development environment, local debugging, ...
    - Huge community and activity

- Disadvantages and things to consider
    - Still rather immature, unexpected error messages
    - Beware number of executors
    - Avoid references to outer classes
    - Be careful about partition tunining

Thanks!

Deep learning for identifying similar songs