4 - Building Models - Part Two

Building Models - Part Two

```
import org.apache.spark.sql.{DataFrame, Row, SQLContext}
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._

val sqlContext = new SQLContext(sc)

val csv = "com.databricks.spark.csv"
val csvOptions = Map("delimiter" -> ",", "header" -> "true", "inferSchema" -> "false")
```

```
val train2013File = "/Users/radek.ostrowski/git/expedia-kaggle/data/train-2013"
val test2014File = "/Users/radek.ostrowski/git/expedia-kaggle/data/test-2014"
val destinationsFile = "/Users/radek.ostrowski/git/expedia-kaggle/data/destinations.csv
```

```
val trainDf = sqlContext.read.load(train2013File)
val testDf = sqlContext.read.load(test2014File)
```

```
val map5 = udf((correct: String, answers: String) => {
      if(answers == null || answers.isEmpty) 0.0
      else {
        val l = answers.split(" ").toList
        val i = l.indexOf(correct)
        if(i == -1) 0.0
        else 1.0/(i+1)
      }
})
```

Hand-made Models

How would we deal with new users? We couldn't use CF.

What if we find top 5 `hotel_clusters` per `srch_destination_id`?

Top 5 `hotel_clusters` per `srch_destination_id`

```
def getTopClustersPerDestination(dfTrain: DataFrame, top: Int=5): DataFrame = {
    val ordered = dfTrain.select("srch_destination_id", "hotel_cluster", "is_booking")
      .groupBy("srch_destination_id", "hotel_cluster").agg(sum("is_booking").as("s"), c
    val merge = udf((sum: Long, count: Long) => {
      //book counts as 1, click counts as 0.15
      sum * 0.85 + count * 0.15
    })
    val agg = ordered.withColumn("counts", merge(ordered("s"), ordered("c")))
      .select("srch_destination_id", "hotel_cluster", "counts")
    val sorted = agg.rdd.map(r => (r.getString(0), (r.getDouble(2), r.getString(1))))
      .groupByKey().mapValues(x =>
        x.toList.sortWith(_._1 > _._1).take(top).map(z => z._2).mkString(" "))
    sqlContext.createDataFrame(sorted.map(x => Row.fromTuple(x)), StructType(
      Seq(StructField("srch_destination_id", StringType, false),
          StructField("hotel_cluster_top_per_dest", StringType, false))))
}
```

```
val topPerDestinationDf = getTopClustersPerDestination(trainDf)
```

```
display(topPerDestinationDf)
```

```
val toScoreDf = testDf.select("id","srch_destination_id","hotel_cluster")
    .join(topPerDestinationDf, Seq("srch_destination_id"))
toScoreDf.withColumn("score", map5(toScoreDf("hotel_cluster"), toScoreDf("hotel_cluster
      .agg(avg("score")).show()
```

```
+------------------+
|        avg(score)|
+------------------+
|0.30815156937413646|
+------------------+
```

Wow, the score is so much better then the best model so far.

And not all the five guesses were made, so it could improved be filling up with top clusters in general

Users' previous `hotel_clusters` per `srch_destination_id`

Let's check if suggesting to the users hotels from their past would score well

```
def getPreviousYearsHotelClusters(dfTrain: DataFrame): DataFrame = {
    val previousYearsRdd = dfTrain.select("user_id", "srch_destination_id", "hotel_clus
      .groupBy("user_id", "srch_destination_id", "hotel_cluster")
      .agg(sum("is_booking").as("s"), count("is_booking").as("c"))

    val merge = udf((sum: Long, count: Long) => {
      //booking = 1.5, click = 0.6
      sum * 0.9 + count * 0.6
    })

    val agg = previousYearsRdd.withColumn("counts", merge(previousYearsRdd("s"), previo
      .select("user_id", "srch_destination_id", "hotel_cluster", "counts")

    val sorted = agg.rdd.map(r => ((r.getString(0), r.getString(1)), (r.getString(2), r
      .mapValues(k => k.toList.sortBy(_._2)(Ordering.Double.reverse).take(5).map(x => x
      .map(x => (x._1._1, x._1._2, x._2))

    sqlContext.createDataFrame(sorted.map(x => Row.fromTuple(x)), StructType(
      Seq(StructField("user_id", StringType, false), StructField("srch_destination_id",
        StructField("hotel_cluster_past", StringType, false))))
}
```

```
val pastHotelClustersDf = getPreviousYearsHotelClusters(trainDf)
```

```
display(pastHotelClustersDf)
```

```
val toScoreDf = testDf.select("user_id","srch_destination_id","hotel_cluster")
    .join(pastHotelClustersDf, Seq("user_id","srch_destination_id"))
toScoreDf.withColumn("score", map5(toScoreDf("hotel_cluster"), toScoreDf("hotel_cluster
      .agg(avg("score")).show()
```

```
+------------------+
|        avg(score)|
+------------------+
|0.35289974815776975|
+------------------+
```

Even better then the previous model! But for a smaller number of records, though.

```
toScoreDf.count
```

243274

And we have many records with only one guess, so the score could be improved by adding up to five predictions

Mixed Models

Let's combine the two best models so far

```
val mergeModels = udf((past: String, topDest: String) => {

      def toList(s: String) = {
        if (s != null && !s.equals("null")) s.split(" ").toList else List()
      }

      val pastList = toList(past)
      val topDestList = toList(topDest)

      //merge the models, remove duplicate hotel clusters, and only take top 5
      (pastList ++ topDestList).distinct.take(5).mkString(" ")
})
```

```
val mixedModelDf = testDf.select("id","user_id","srch_children_cnt","srch_destination_i
  .join(topPerDestinationDf, Seq("srch_destination_id"))
  .join(pastHotelClustersDf, Seq("user_id","srch_destination_id"))
```

```
val toScoreDf = mixedModelDf.withColumn("hotel_cluster_prediction",
          mergeModels(mixedModelDf("hotel_cluster_past"), mixedModelDf("hotel_cluster_t
```

```
toScoreDf.withColumn("score", map5(toScoreDf("hotel_cluster"), toScoreDf("hotel_cluster
      .agg(avg("score")).show()
```

```
+------------------+
|        avg(score)|
+------------------+
|0.41067068408461144|
+------------------+
```

Wow, the mixed model is even better.

Improving the best model

Let's try topping up, up to 5 predictions, using top 5 `hotel_clusters` in general

```
def getTopClusters(dfTrain: DataFrame, top: Int=5) = {
    val ordered = dfTrain.select("hotel_cluster", "is_booking")
      .groupBy("hotel_cluster").agg(sum("is_booking").as("s"), count("is_booking").as("
    val merge = udf((sum: Long, count: Long) => {
      //book counts as 1, click counts as 0.15
      sum * 0.85 + count * 0.15
    })
    val agg = ordered.withColumn("counts", merge(ordered("s"), ordered("c")))

    agg.select("hotel_cluster", "counts").orderBy(agg("counts").desc)
      .take(top).map(r => r.getString(0)).toList.mkString(" ")
}
```

```
val topClusters = getTopClusters(trainDf)
```

So, the top `hotel_clusters` in general are 91, 48, 59, 41, 64. Let's top up the mixed model.

```
val mergeModels2 = udf((past: String, topDest: String, top: String) => {

      def toList(s: String) = {
        if (s != null && !s.equals("null")) s.split(" ").toList else List()
      }

      val topList = toList(top)
      val pastList = toList(past)
      val topDestList = toList(topDest)

      //merge the models, remove duplicate hotel clusters, top up with most popular in
      (pastList ++ topDestList ++ topList).distinct.take(5).mkString(" ")
})
```

```
val mixedModelWithTop = mixedModelDf.withColumn("topClusters", lit(topClusters))

val toScoreDf = mixedModelWithTop.withColumn("hotel_cluster_prediction",
          mergeModels2(mixedModelWithTop("hotel_cluster_past"),
                       mixedModelWithTop("hotel_cluster_top_per_dest"),
                       mixedModelWithTop("topClusters")))

toScoreDf.withColumn("score", map5(toScoreDf("hotel_cluster"), toScoreDf("hotel_cluster
      .agg(avg("score")).show()
```

```
+------------------+
|        avg(score)|
+------------------+
|0.41070185606901316|
+------------------+
```

It improved the score only slightly.

Reorder the predictions

What if the same `hotel_cluster` is present in user's past and in the most popular? Let's promote it and find out.

```
def promoteCommonClusters(a: List[String], b: List[String]): List[String] = {
    val bufLeft = scala.collection.mutable.ListBuffer.empty[String]
    val bufRight = scala.collection.mutable.ListBuffer.empty[String]
    for (x <- a) {
      if (b.contains(x)) bufLeft.append(x) else bufRight.append(x)
    }
    (bufLeft ++ bufRight ++ b).toList.distinct.take(5)
}
```

```
val mergeModels3 = udf((past: String, topDest: String) => {

      def toList(s: String) = {
        if (s != null && !s.equals("null")) s.split(" ").toList else List()
      }

      val topList = List("91", "48", "59", "41", "64")
      val pastList = toList(past)
      val topDestList = toList(topDest)

     (promoteCommonClusters(pastList,topDestList) ++ topList).distinct.take(5).mkString
})
```

```
val toScoreDf = mixedModelDf.withColumn("hotel_cluster_prediction",
          mergeModels3(mixedModelDf("hotel_cluster_past"), mixedModelDf("hotel_cluster_

toScoreDf.withColumn("score", map5(toScoreDf("hotel_cluster"), toScoreDf("hotel_cluster
      .agg(avg("score")).show()
```

```
+------------------+
|        avg(score)|
+------------------+
|0.40911304400250964|
+------------------+
```

It's slightly worse, so let's forget it.

Adjust the model for parents

Assumption: if some hotels don't allow children, they can be removed from the prediction if the user is booking in with children

```
val acceptChildrenDf = trainDf.where("srch_children_cnt IS NOT NULL AND srch_children_c
  .select("hotel_cluster", "srch_destination_id").distinct

val noChildrenDf = trainDf.select("hotel_cluster","srch_destination_id").distinct
  .except(acceptChildrenDf)
```

```
val grouped = noChildrenDf.rdd.map(x => (x.getString(1), x.getString(0))).groupByKey().

val noChildrenClustersDf = sqlContext.createDataFrame(grouped.map(x => Row.fromTuple(x)
      Seq(StructField("srch_destination_id", StringType, false), StructField("no_childr
```

```
display(noChildrenClustersDf)
```

```
val mergeModels4 = udf((past: String, topDest: String, hasChildren: String, childrenNot

      def toList(s: String) = {
        if (s != null && !s.equals("null")) s.split(" ").toList else List()
      }

      val isParent = if(hasChildren != null && !hasChildren.equals("null")) hasChildren
      val noChildrenList = toList(childrenNotAllowed)

      val topList = List("91", "48", "59", "41", "64")
      val pastList = toList(past)
      val topDestList = toList(topDest)

      val predictions = (pastList ++ topDestList ++ topList)
      val parentPredictions = predictions.filter(x => !noChildrenList.contains(x))

      (if(isParent) parentPredictions else predictions).distinct.take(5).mkString(" ")
})
```

```
val mixedModelWithNoChildClustersDf = mixedModelDf
      .join(noChildrenClustersDf, Seq("srch_destination_id"), "left_outer")

val toScoreDf = mixedModelWithNoChildClustersDf.withColumn("hotel_cluster_prediction",
          mergeModels4(mixedModelWithNoChildClustersDf("hotel_cluster_past"),
                       mixedModelWithNoChildClustersDf("hotel_cluster_top_per_dest"),
                       mixedModelWithNoChildClustersDf("srch_children_cnt"),
                       mixedModelWithNoChildClustersDf("no_children_clusters")
                      ))

toScoreDf.withColumn("score", map5(toScoreDf("hotel_cluster"), toScoreDf("hotel_cluster
      .agg(avg("score")).show()
```

```
+------------------+
|        avg(score)|
+------------------+
|0.41070000630290543|
+------------------+
```

This slighlty decreased the score, so let's forget it too. It seems that the definition of hotels not accepting children is not correct, and it removed valid hotels from the prediction.

Summary

The best model found is a mixed model combining users' history of clicks and bookings, most common hotel clusters per destination and the top hotel clusters in general.

Achieved score is `0.410701856` which is equivalent on average to recommending the correct hotel half time at second and half time at third place. Second place would be `0.5` and third would be `0.333`.

Success! It significantly beats the `Most Frequent Benchmark: 0.0695994`

```
```

Build: | **buildTime**-*Sat Mar 12 18:40:47 UTC 2016* | **formattedShaVersion**-*0.6.3-61efca56c2a45613ec609f350dc4911b166b3f28-SNAPSHOT* | **sbtVersion**-*0.13.8* | **scalaVersion**-*2.11.7* | **sparkNotebookVersion**-*0.6.3* | **hadoopVersion**-*2.7.2* | **jets3tVersion**-*0.7.1* | **jlineDef**-*(jline,2.12)* | **sparkVersion**-*1.6.1* | **withHive**-*true* | **withParquet**-*true* |.