3 - Building Models - Part One

Building Models - Part One

```
import org.apache.spark.sql.{DataFrame, Row, SQLContext}
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
import org.apache.spark.mllib.classification.{NaiveBayes, NaiveBayesModel}
import org.apache.spark.mllib.recommendation.{ALS, MatrixFactorizationModel, Rating}
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.{Vector, Vectors}
import org.joda.time.format.DateTimeFormat
import org.joda.time.{Days, LocalDateTime}

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)
```

Selecting features we are going to use

And adding some new as described previously

```
val getNightsStayed = udf((ci: String, co: String) => {
      val format = DateTimeFormat.forPattern("yyyy-MM-dd")
      if(ci == null || ci.isEmpty || co == null || co.isEmpty) 0
      else {
        try{
          val daysCount = Days.daysBetween(format.parseLocalDateTime(ci), format.parseL
          if(daysCount < 0) 0 else daysCount
        } catch {
          case e: Exception => 0
        }
      }
})
val getMonth = udf((ci: String) => {
      val format = DateTimeFormat.forPattern("yyyy-MM-dd")
      if(ci == null || ci.isEmpty) 0
      else {
        try {
          format.parseLocalDateTime(ci).getMonthOfYear
        } catch {
          case e: Exception => 0
        }
      }
})
val getDaysToCi = udf((dateTime: Long, ci: String) => {
      val format = DateTimeFormat.forPattern("yyyy-MM-dd")
      if(ci == null || ci.isEmpty) 0
      else {
        try {
          val firstDate = new LocalDateTime(dateTime*1000)
          val daysCount = Days.daysBetween(firstDate, format.parseLocalDateTime(ci)).ge
          if(daysCount < 0) 0 else daysCount
        } catch {
          case e: Exception => 0
        }
      }
})

val getHasChildren = udf((srch_children_cnt: String) => {
      if(srch_children_cnt == null || srch_children_cnt.isEmpty || srch_children_cnt.to
      else 1
})

def asDouble(a: Any): Double = {
  if(a == null) 0.0
  else {
    val s = a.toString
    if(s.equals("null") || s.isEmpty) 0.0 else s.toDouble
  }
}
```

```
val selectedTrainFeaturesDf = trainDf
  .withColumn("children", getHasChildren(trainDf("srch_children_cnt")))
  .withColumn("number_of_nights", getNightsStayed(trainDf("srch_ci"),trainDf("srch_co")
  .withColumn("month", getMonth(trainDf("srch_ci")))
  .withColumn("days_to_ci", getDaysToCi(unix_timestamp(trainDf("date_time")),trainDf("s
  .select("hotel_cluster",
          "user_location_country","user_location_region","user_location_city",
          "is_mobile","is_package","channel",
          "srch_destination_id", "srch_destination_type_id",
          "hotel_continent", "hotel_country", "hotel_market",
          "number_of_nights", "month", "days_to_ci", "children")
```

```
display(selectedTrainFeaturesDf)
```

```
val selectedTestFeaturesDf = testDf
  .withColumn("children", getHasChildren(testDf("srch_children_cnt")))
  .withColumn("number_of_nights", getNightsStayed(testDf("srch_ci"),testDf("srch_co")))
  .withColumn("month", getMonth(testDf("srch_ci")))
  .withColumn("days_to_ci", getDaysToCi(unix_timestamp(testDf("date_time")),testDf("src
  .select("id",
          "user_location_country","user_location_region","user_location_city",
          "is_mobile","is_package","channel",
          "srch_destination_id", "srch_destination_type_id",
          "hotel_continent", "hotel_country", "hotel_market",
          "number_of_nights", "month", "days_to_ci", "children")
```

```
display(selectedTestFeaturesDf)
```

Naive Bayes

Build the model

Treating clicks equal to bookings as various experiments showed they have similar weights. Also ignoring `user_id` for NB as it has high cardinality.

```
def dataFrameToTrainFeatures(df: DataFrame) = {
//first is hotel_cluster, our target for prediction
  df.map(r => LabeledPoint(asDouble(r.getString(0)),
        Vectors.dense(asDouble(r.getString(1)),asDouble(r.getString(2)),asDouble(r.getS
                      asDouble(r.getString(4)),asDouble(r.getString(5)),asDouble(r.getS
                      asDouble(r.getString(7)),asDouble(r.getString(8)),asDouble(r.getS
                      asDouble(r.getString(10)),asDouble(r.getString(11)),
                      asDouble(r.getInt(12)),asDouble(r.getInt(13)), asDouble(r.getInt(
        )))
}

def buildBayes(df: DataFrame) = {
      val featuresNB = dataFrameToTrainFeatures(df)
      val model = NaiveBayes.train(featuresNB, lambda = 1.0, modelType = "multinomial")
      //model.save(sc, "NaiveBayesModel-simple")
      model
}
```

```
val modelNB = buildBayes(selectedTrainFeaturesDf)
```

Make top 5 `hotel_cluster` predictions, not just one

```
def dataFrameToTestFeatures(df: DataFrame) = {
//first is id
  df.map(r => (r.getString(0),
        Vectors.dense(asDouble(r.getString(1)),asDouble(r.getString(2)),asDouble(r.getS
                      asDouble(r.getString(4)),asDouble(r.getString(5)),asDouble(r.getS
                      asDouble(r.getString(7)),asDouble(r.getString(8)),asDouble(r.getS
                      asDouble(r.getString(10)),asDouble(r.getString(11)),
                      asDouble(r.getInt(12)),asDouble(r.getInt(13)),asDouble(r.getInt(1
        )))
}

def predictBayes(df: DataFrame, loadedModel: NaiveBayesModel) = {

  def getTopPredictions(a: Vector, top: Int = 5): String = {
        a.toArray.zipWithIndex.sortBy(_._1)(Ordering.Double.reverse).take(top).map(_._2
  }

  val features = dataFrameToTestFeatures(df)
  //val loadedModel = NaiveBayesModel.load(sc, "NaiveBayesModel-simple")
  val bcModel = sc.broadcast(loadedModel)
  val rec = features.mapPartitions { iter =>
    val model = bcModel.value
    iter.map(x => (x._1, getTopPredictions(model.predictProbabilities(x._2)))) //take t
  }

  val modelPredictionsDf = sqlContext.createDataFrame(rec.map(x => Row.fromTuple(x)), S
        Seq(StructField("id", StringType, false),
          StructField("hotel_cluster_predicted", StringType, false)
        )))

  //modelPredictionsDf.write.format(csv).options(csvOptions).save("nb-simple")
  modelPredictionsDf
}
```

```
val predictionsNB = predictBayes(selectedTestFeaturesDf, modelNB)
```

```
display(predictionsNB)
```

Test NB model

```
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)
      }
})
```

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

```
+--------------------+
|          avg(score)|
+--------------------+
|0.019367280202505964|
+--------------------+
```

It seems that NB model is doing slightly worse than `Random Guess Benchmark: 0.02260` so let's continue with other approaches

Random Forest

A model based on Random Forest took a very long time to calculate and wasn't any good, so it's skipped here for clarity.

Collaborative Filtering

Let's build a simple collaborative filtering model with implicit feedback where product is the `hotel_cluster`

As rating we'll use a sum of bookings and clicks for particular `user_id` for particular `hotel_cluster`

```
def scoreProductsPerUser(dfTrain: DataFrame) = {
  val previousYearsDf = dfTrain.select("user_id", "hotel_cluster", "is_booking")
        .groupBy("user_id", "hotel_cluster").agg(sum("is_booking").as("s"), count("is_b

  val merge = udf((sum: Long, count: Long) => {
        // booking is counted as 2 and a click is counted as 1
        sum + count
  })

  previousYearsDf.withColumn("counts", merge(previousYearsDf("s"), previousYearsDf("c")
    .select("user_id", "hotel_cluster", "counts")
}

def buildCFModel(train: DataFrame) = {
    val ratings = scoreProductsPerUser(train).map(r => Rating(r.getString(0).toInt, r.g
    val rank = 10
    val numIterations = 10
    val lambda = 0.01
    val blocks = -1
    val alpha = 1.0
    val seed = 42L
    val model = ALS.trainImplicit(ratings, rank, numIterations, lambda, blocks, alpha,
    //model.save(sc, fileName)
    model
  }
```

```
val modelCf = buildCFModel(trainDf)
```

```
val recommendedProducts = modelCf.recommendProductsForUsers(5)
  .map(r => (r._1.toString, r._2.map(x => x.product).mkString(" ")))

val predictionsCF = sqlContext.createDataFrame(recommendedProducts.map(x => Row.fromTup
      Seq(StructField("user_id", StringType, false), StructField("hotel_cluster_cf", St
```

```
display(predictionsCF)
```

Show: 10

Search:

user_id	hotel_cluster_cf
1160000	44 63 92 28 48
100000	68 41 98 91 18
330000	1 79 45 54 24
560000	46 64 58 82 67
270000	58 46 82 78 67
1010000	6 28 4 95 21
10000	91 5 90 42 18
1050000	65 52 66 87 26
940000	91 42 48 18 34
790000	18 19 91 41 95

Showing 1 to 10 of 1000 records

Pages: Previous 1 2 3 … 100 Next

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

```
+------------------+
|        avg(score)|
+------------------+
|0.09821154068468636|
+------------------+
```

Yeah, this beats `Most Frequent Benchmark: 0.05949`.

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* |.