2 - Testing Strategy and Feature Engineering

Sparkommender - Testing Stategy and Feature Engineering

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

import org.apache.spark.mllib.feature.PCA
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vector

import org.joda.time.Days
import org.joda.time.format.DateTimeFormat

val sqlContext = new SQLContext(sc)

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

Beating the benchmarks

Before engineering any features lets define first the goal and the method to score the models

The objective is to beat both:

1) Random Guess Benchmark: 0.02260

2) Most Frequent Benchmark: 0.05949

Note, that the benchmark was set for the main test data.

For the train-test data that's defined later the Most Frequent Benchmark score is `0.0695994`.

The effectivenes of the model is defined by MAP@5 which gives a score for up to 5 best predictions.

If the correct answer is given as a first option, the model for this prediction gets a score of 1. The score goes down the further the correct prediction is made. If it's not if the top 5, the score is 0.

```
case class Map5Score(position: Int, score: Double)
List(Map5Score(1,1.0),Map5Score(2,1.0/2),Map5Score(3,1.0/3),Map5Score(4,1.0/4),Map5Scor
```

Show:
10

Search:

position	score
1	1
2	0.5
3	0.3333333333333333
4	0.25
5	0.2
6	0

Showing 6 of 6 records

Pages: Previous 1 Next

The bussines rationale is that it's not essential to predict only one exact hotel cluster, but it should be present in the top 5 which will be recommended to the users

User Defined Function to calculate MAP@5

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

Splitting the data for training and testing

The whole train set includes bookings/clicks made in 2013 and 2014. The final test set includes only bookings made in 2015.

It makes sense to split the train set also based on dates for training the model and evaluating it's quality.

```
val trainFile = "/Users/radek.ostrowski/git/expedia-kaggle/data/train.csv"
val trainDf = sqlContext.read.format(csv).options(csvOptions).load(trainFile)
```

```
def splitTrainOnYear(df: DataFrame): (DataFrame, DataFrame) = {
  val train = df.where(to_date(df("date_time")).lt("2014-01-01"))
  val test = df.where(to_date(df("date_time")).geq("2014-01-01")).where("is_booking = 1
    .withColumn("id", monotonicallyIncreasingId.cast("string"))
  (train, test)
}
```

```
val (train2013, test2014) = splitTrainOnYear(trainDf)
```

Let's save them for later use

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

```
train2013.write.save(train2013File)
test2014.write.save(test2014File)
```

Feature Engineering

Most of the existing features will be used for building models plus several new features designed to improve the quality of the recommendations

Date-based features

Date features are not well understood by ML algorithms, so let's create several useful features out of them

Number of nights per stay - the length of stay may infuence the hotel cluster chosen

```
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
        }
      }
})
```

Month of check-in - bookings of particular clusters will depend on the time of the year/season

```
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
        }
      }
})
```

Number of days to check-in - how long in advance was the hotel cluster clicked/booked should also infuence the selection

```
val getDaysToCi = udf((dateTime: String, ci: String) => {
      val date = dateTime.split(" ")(0)
      val format = DateTimeFormat.forPattern("yyyy-MM-dd")
      if(ci == null || ci.isEmpty || date == null || date.isEmpty) 0
      else {
        try {
          val daysCount = Days.daysBetween(format.parseLocalDateTime(dateTime), format.
          if(daysCount < 0) 0 else daysCount
        } catch {
          case e: Exception => 0
        }
      }
})
```

Family related feature

A family related feature that looks like it could work is whether or not the hotel cluster in a given destination allows children, as this affects the choice for parents traveling with their dependents

```
val acceptChildrenDf = train2013.where("srch_children_cnt IS NOT NULL AND srch_children
  .select("hotel_cluster", "srch_destination_id").distinct
```

```
acceptChildrenDf.count
```

```
144208
```

Let's assume that if there are no clicks or bookings made for children at all in certain hotel clusters per destination, those hotels don't accept children

```
val noChildrenDf = train2013.select("hotel_cluster", "srch_destination_id").distinct.ex
```

```
noChildrenDf.count
```

```
105134
```

Destinations Feature

Let's also use the destinations file which describes the similarity of different destinations and add it as a feature

```
val destinationsFile = "/Users/radek.ostrowski/git/expedia-kaggle/data/destinations.csv
val destinationsDf = sqlContext.read.format(csv).options(csvOptions).load(destinationsF
```

As it contains 149 numeric columns lets apply dimensionality reduction with PCA

```
//TODO

val inputColumns = (1 to 149).map(x => "d" + x).toArray

val assembler = new VectorAssembler()
      .setInputCols(inputColumns)
      .setOutputCol("destinations")

destinationsDf.map(r => (r.getInt("srch_destination_id "), )) .transform(destinationsDf

val destVectors = destDf.select("destinations").map(r => r.getAs[Vector](0))

val pcaVectors = new PCA(1).fit(destVectors).transform(destVectors).map(v => v(0))
```

```
pcaVectors.count
```

```
62106
```

```
destinationsDf.count
```

```
62106
```

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* |.