20180111.lda-analy…

READY

```
println("Spark version " + sc.version)
println("Scala version " + util.Properties.versionString)

// Global variables
val dir_data = "data/dev-test-exploratory-analysis"
val fcomm_clean = dir_data + "/224564804326967_facebook_comments_clean/part-*"
// End of Global variables

// Read text file
val commentsDF = sc.textFile(fcomm_clean)
                   .filter(_.nonEmpty)
                  .filter(l => !l.contains("null"))
                  .filter(l => !l.contains("0"))

val commentsArrDF = commentsDF
                        .map(line => line.split(" "))
                        .filter(s => s.length > 1)
                        .toDF("words")
```

```
Spark version 2.1.0
Scala version version 2.11.8
dir_data: String = data/dev-test-exploratory-analysis
fcomm_clean: String = data/dev-test-exploratory-analysis/224564804326967_facebook_comments_cl
ean/part-*
commentsDF: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[190] at filter at <console>:8
2
commentsArrDF: org.apache.spark.sql.DataFrame = [words: array<string>]
```

READY

```
import org.apache.spark.ml.feature._
import org.apache.spark.ml.clustering.LDA
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.clustering.DistributedLDAModel

// val commentsArrDF = commentsDF.map(line => line.split(" ")).toDF("words")



val vectorizer = new CountVectorizer()
  .setInputCol("words")
  .setOutputCol("features")
  .setVocabSize(3 * 2048)

val numTopics = 10

val lda = new LDA()
  .setK(numTopics)
  .setMaxIter(50)
  .setOptimizer("em")

val pipeline = new Pipeline().setStages(Array(vectorizer, lda))

// Save pipleines
pipeline.write.overwrite().save(dir_data + "/lda/pipeline")



val pipelineModel = pipeline.fit(commentsArrDF)
```

```
println("Stages " + pipelineModel.stages)

val vectorizerModel = pipelineModel.stages(0).asInstanceOf[CountVectorizerModel]
// Since we trained with the default optimizer (EM), we get back a DistributedLDAModel
val ldaModel = pipelineModel.stages(1).asInstanceOf[DistributedLDAModel]

// Save Vectorizer Model
vectorizerModel.write.overwrite().save(dir_data + "/vectorizer/model")

// Let's save our LDA Model for later reuse.
ldaModel.write.overwrite().save(dir_data + "/lda/model")
```

```
import org.apache.spark.ml.feature._
import org.apache.spark.ml.clustering.LDA
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.clustering.DistributedLDAModel
vectorizer: org.apache.spark.ml.feature.CountVectorizer = cntVec_9cb728283195
numTopics: Int = 10
lda: org.apache.spark.ml.clustering.LDA = lda_1ceba4b44d21
pipeline: org.apache.spark.ml.Pipeline = pipeline_100407a7ebef
pipelineModel: org.apache.spark.ml.PipelineModel = pipeline_100407a7ebef
Stages [Lorg.apache.spark.ml.Transformer;@34aee95d
vectorizerModel: org.apache.spark.ml.feature.CountVectorizerModel = cntVec_9cb728283195
ldaModel: org.apache.spark.ml.clustering.DistributedLDAModel = lda_1ceba4b44d21
```

READY

```
// Data log likelihood gives us a statistic for evaluation.
// This statistics is always negative, and closer to 0 is better.
ldaModel.trainingLogLikelihood
```

```
res104: Double = -64499.561394447934
```

READY

```
ldaModel.describeTopics(maxTermsPerTopic = 5).show()
```

```
+-----+-----------------+--------------------+
|topic|       termIndices|         termWeights|
+-----+-----------------+--------------------+
|    0|  [0, 22, 7, 4, 2]||[0.08361479546064...|
|    1| [0, 2, 24, 10, 4]||[0.09066305546892...|
|    2|[3, 23, 25, 35, 0]||[0.04940099267992...|
|    3| [0, 2, 4, 17, 27]||[0.07379670508899...|
|    4|  [0, 4, 2, 33, 6]||[0.06444162366566...|
|    5| [0, 6, 37, 4, 24]||[0.06869716963353...|
|    6| [0, 2, 4, 26, 15]||[0.07036937965935...|
|    7| [1, 8, 20, 0, 18]||[0.11087448945391...|
|    8|[28, 0, 11, 5, 13]||[0.04025302995210...|
|    9|  [0, 2, 5, 4, 14]||[0.06031717166363...|
+-----+-----------------+--------------------+
```

READY

```
// Get vocab
val vocabList = vectorizerModel.vocabulary
val termsIdx2Str = udf { (termIndices: Seq[Int]) => termIndices.map(idx => vocabList(idx)) }
```

```
vocabList: Array[String] = Array(byt, cena, on, ta, vajce, ja, lidl, kriz, vajicko, ani, kupi
t, este, vas, sty, letak, slovensky, ist, nebyt, vy, vsetok, buda, nemat, my, tam, boliet, cl
```

```
ovek, u, dakovat, d, taky, chciet, ci, rad, iny, lebo, predajny, lidli, mama, toto, ktory, ak
y, slovensko, mata, dat, moct, to, nic, kupovat, ziaden, preco, nizky, tato, cela, dostat, ve
la, nevediet, prosit, akcia, malit, sliepka, obchod, draha, nas, bola, kto, asi, nemecky, sup
er, vediet, 2, ona, no, stat, maslo, maj, 1, lacne, retazec, dobry, musiet, rakusky, trh, vel
mi, 3, v, potravina, photo, dol, pravda, vec, nechodit, vidiet, viest, rovnaky, ano, jeden, p
ot, vianoce, iba, robit, kde, zly, vobec, skoncit, cas, kazdy, doma, problem, rok, mysliet, k
onecna, hned, keby, sk, nakupovat, ponuka, dreveny, n...
termsIdx2Str: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<fun
ction1>,ArrayType(StringType,true),Some(List(ArrayType(IntegerType,false))))
```

```
// Show 5 terms from every topic.
termDF.show(51)
```

READY

```
|      6|        u|0.011202748810891753|
|      6|slovensky|0.009717220318741867|
|      7|     cena| 0.11087448945391011|
|      7|  vajicko| 0.04366654958345922|
|      7|     buda| 0.03015896091387578|
|      7|      byt|0.025505137890753933|

|      7|       vy| 0.02301149415116919|
|      8|        d| 0.04025302995210583|
|      8|      byt| 0.03696157046525404|
|      8|     este|0.026605247871373663|
|      8|       ja|0.025858232951204794|
|      8|      sty|0.023926218685289366|
|      9|      byt| 0.06031717166363496|
|      9|       on| 0.02089676272925423|
|      9|       ja|0.015296806115767957|
|      9|    vajce|0.010878176810651112|
|      9|    letak| 0.00983320718336385|
```

READY