//Triplets using OpenIE

**object** SparkIE {  
  
 **def** main(ags: Array[String]) {  
 *// Configuration* **val** sparkConfnnss = **new** SparkConf().setAppName(**"SparkOpenie"**).setMaster(**"local[\*]"**)  
  
 **val** sc12 = **new** SparkContext(sparkConfnss)  
  
 **val** input1345 = sc12.textFile(**"E:\\UMKC\\Sum\_May\\KDM\\week1\\bbcsport\\cricket\\001.txt"**).map(line234 => {  
 *//Getting OpenIE Form of the word using lda.CoreNLP* **val** triples=NLP.*callTriplets*(line234)  
 triplets  
 }) *println*(input1345.collect().mkString(**"\n"**))  
  
  
  
 }  
  
}

**public class** NLP {  
 **public static** String callTriplets(String sent) {  
  
 Document documennts11 = **new** Document(sent);  
 String lema=**""**;  
 **for** (Sentence s : documennst.sent()) { Collection<Quadruple<String, String, String, Double>> col=s.openie();  
  
 lema+= col.toString();}  
  
 **return** lema;  
 }  
  
}

//Semantic Meaning using ConceptNet

**public class** concept {  
  
 **public final static void** main(String[] args) {  
  
 HttpClient Client111 = **new** DefaultHttpClient();  
 String line1111 = **""**;  
 **try** {  
 HttpGet Request1 = **new** HttpGet(**"http://conceptnet5.media.mit.edu/data/5.4/search?rel=/r/PartOf&end=/c/en/animal&limit=10"**);  
 HttpResponse Response111 = httpClient.execute(Request1);  
  
   
  
 HttpEntity entity1111 = Response.getEntity();  
  
 **byte**[] buffer111 = **new byte**[1024];  
 **if** (entity111 != **null**) {  
 InputStream Stream123 = entity.getContent();  
 **int** bytesRead112 = 0;  
 BufferedInputStream bis121 = **new** BufferedInputStream(Stream123);  
 **while** ((bytesRead112 = bis121.read(buffer111)) != -1) {  
 String chunkccc = **new** String(buffer, 0, bytesRead112);  
 System.***out***.println(chunkccc);  
 line1111 += chunkccc;  
 }  
  
   
 }  
 JSONParser parser1122 = **new** JSONParser();  
 Object obj12 = parser1122.parse(line111);  
 JSONObject b11 = (JSONObject) obj12;  
 JSONArray ja12 = (JSONArray) b11.get(**"edges"**);  
 **for** (**int** k = 0; k < ja12.size(); k++) {  
 JSONObject object2222 = (JSONObject) ja12.get(k);  
 System.***out***.println(ob2222.get(**"surfaceText"**));  
 }  
  
 }  
}

//Synonyms using WordNet

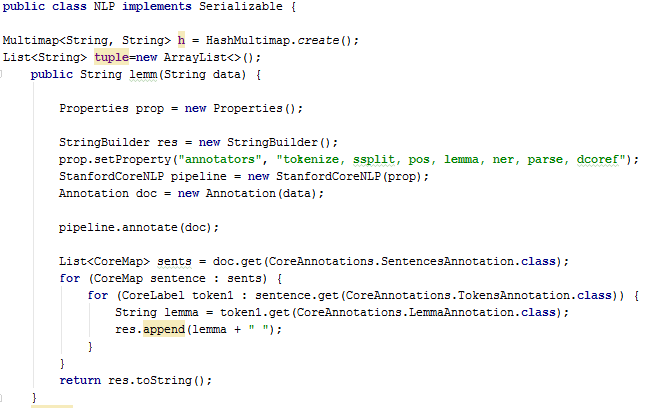
**val** dd55=data555.map(f11=>{  
 **val** wordnet1223 = **new** RiWordNet(**"E:\\UMKC\\Sum\_May\\KDM\\WordNet-3.0"**)  
 **val** farr111=f11.split(**" "**)  
 *getSynoymns143*(wordnet1223,**"cricket"**)  
 })  
 dd55.take(1).foreach(f33=>*println*(f33.mkString(**"\n"**)))  
}  
**def** getSynoymns143(wordnet12:RiWordNet,word12:String): Array11[String] ={  
 *println*(word112)  
 **val** pos111=wordnet12.getPos(word12)  
 *println*(pos111.mkString(**" "**))  
 **val** syn111=wordnet12.getAllSynonyms(wordnet12, pos111(0), 10)  
 syn111  
}

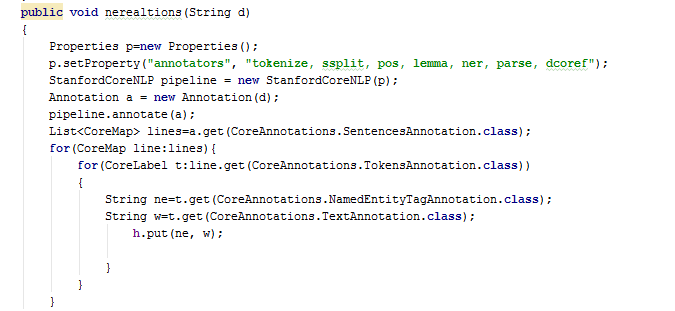
//LDA

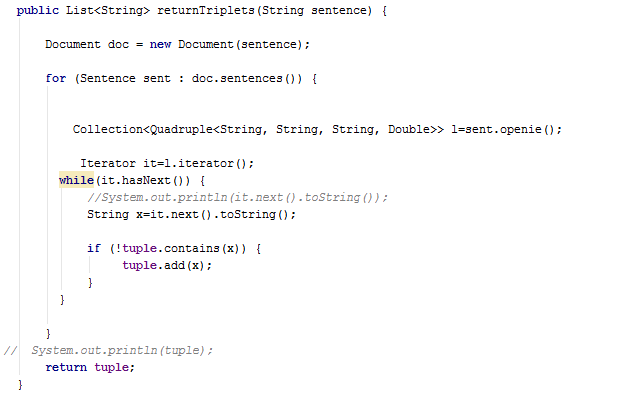
**private def** run(params: Params11332331) {  
 System.*setProperty*(**"hadoop.home.dir"**, **"E:\\UMKC\\Sum\_May\\KDM\\winutils"**)  
 **val** conf111= **new** SparkConf().setAppName(**s"LDAExample with $**params**"**).setMaster(**"local[\*]"**).set(**"spark.driver.memory"**, **"4g"**).set(**"spark.executor.memory"**, **"4g"**)  
 **val** schwvfwfvgr = **new** SparkContext(conf1111)  
  
 Logger.*getRootLogger*.setLevel(Level.*WARN*)  
  
 **val** topic\_output12ugug = **new** PrintStream(**"data/Results.txt"**)  
 *// Load documents, and prepare them for LDA.* **val** preprocessStartfhhvsfvhvkh = System.*nanoTime*()  
 **val** (corpusjvbkdv, vocabArraywbfub, actualNumTokensjnv) =  
 *preprocess*(scjkbc, params.input)  
 corpusjsbdd.cache()  
 **val** actualCorpusSizehsvkdk = corpusjsbds.count()  
 **val** actualVocabSizesvdhs = vocabArraybsjd.length  
 **val** preprocessElapsedsddd = (Systemjsjs.*nanoTime*() - preprocessStartjbfbfd) / 1e9  
  
  
  
 *// Run LDA.* **val** lda = **new** LDA()  
  
 **val** optimizerjbsjd = paramsbjcbjcbj.algorithm11.toLowerCase **match** {  
 **case "em"** => **new** EMLDAOptimizer  
 *// add (1.0 / actualCorpusSize) to MiniBatchFraction be more robust on tiny datasets.* **case "online"** => **new** OnlineLDAOptimizer().setMiniBatchFraction(0.05 + 1.0 / actualCorpusSizedcnd,)  
 **case** \_ => **throw new** IllegalArgumentException(  
 **s"Only em, online are supported but got $**{paramsdhdvd.algorithmbhvsd}**."**)  
 }  
  
 ldajsdbdsbdj.setOptimizer(optimizerbfdfhfd)  
 .setK(params.k)  
 .setMaxIterations(50)  
  
 **val** startTimejbsdsdjbdjbs = Systemhsvd.*nanoTime*()  
 **val** ldaModeldbsjd = ldajbsjd.run(corpusbsdbs)  
 **val** elapsedsdbs = (Systembsagd.*nanoTime*() - startTimejbsjdjb) / 1e9   
  
 **if** (ldaModel.isInstanceOf[DistributedLDAModel]) {  
 **val** distLDAModeljsdbs = ldaModeljbdjcb.asInstanceOf[DistributedLDAModeljbjaa]  
 **val** avgLogLikelihoodabjab = distLDAModelknsnc.*logLikelihood* / actualCorpusSizeckbkfk.toDouble  
 *println*(**s"\t Training data average log likelihood: $**avgLogLikelihoodjbsbjvbjs**"**)  
 *println*()  
 topic\_output.println(**s"\t Training data average log likelihood: $**avgLogLikelihoodknkncnkd**"**)  
 topic\_output.println()  
 }  
**val** topicIndicesjcbjcc = ldaModelccnjnc.describeTopics(maxTermsPerTopicjjbdbj = actualVocabSizevjnnvv)  
 **val** topicsjjnjd = topicIndicesjbdbj.map { **case** (termskndnnk, termWeightskdnknn) =>  
 termsjjjd.zip(termWeightsdnknfvnk).map { **case** (termdunfk, weightdjdkkd) => (vocabArraynnnkfkn(termjjbbjed.toInt), weightbjdjbwbjd) }  
 }  
 *println*(**s"$**{paramssncjns.k} **topics:"**)  
 topic\_output.println(**s"$**{paramsnjdjb.k} **topics:"**)  
 topicsbsbjs.take(5).zipWithIndex.foreach { **case** (topichbsbhs, j) =>  
 *println*(**s"TOPIC $**j**"**)  
 topicjsjj.take(5).foreach { **case** (termjsjd, weightjbd) =>  
 *println*(**s"$**termsjjd**\t$**weightjjd**"**)  
 topic\_output.println(**s"TOPIC\_$**j**;$**termdhd**;$**weightdhdh**"**)  
 }  
 *println*()  
 topic\_output.println()  
 }

**private def** preprocesdjffds(scjfjd: SparkContext,pathsdnnf: Seq[String]): (RDD[(Long, Vector)], Arraydjfbj[String], Longnfbbjfj) = {  
**val** stopWordssjhdhds=scjsfs.textFile(**"data/stopwords.txt"**).collect()  
 **val** stopWordsBroadCastshsbdv=scdjbbjbjd.broadcast(stopWordsbjdbjfbjbje)  
  
 **val** dfjsjdjs = scjsfs.textFile(pathsjnfnbnsn.mkString(**","**)).map(fbfbf => {  
 *// val lemmatised=CoreNLP.returnLemma(f)* **val** splitString111 = fbfbf.split(**" "**)  
 splitString111.toSeq  
 })  
  
  
  *val stopWordRemovedDjjdsF=dfsdsd.map(f=>{  
 val filteredbjdbjbjF=fdbjjb.map(\_.replaceAll("[^a-zA-Z]"," "))  
   
 .filter(ffsdndsn=>{  
 if(stopWordsBroadCastjnsnd.value.contains(ffsbjdbjsbj.toLowerCase))  
 false  
 else  
 true  
 })  
 filteredFdsls  
 })  
  
 val dfseqsdsd=stopWordRemovedDFbsbjdbj.map(\_.toSeq)* **val** hashingTFjjdjdjb = **new** HashingTF(dfbjbjdjb.count().toInt) **val** tfahssabh = hashingTFdjndjdsjjd.transform(dfsnnsk).zipWithIndex().map(\_.swap)  
 tfbhasash.cache()**val** dffsdbds= dfbdjsjdj.flatMap(f=>f)  
 **val** vocabjsjjd=dffsnskn.distinct().collect()  
 (tfhsbdhs,vocabjsjjd, dffjdsjbjbd.count())}  
}

//Question and answer system







**public** String ma(String d11, List<String> li) {  
 *// System.out.println(li);* StringBuffer res11 = **new** StringBuffer();  
  
 **int** count1=0;  
 Iterator i1 = li.iterator();  
 **while** (i1.hasNext()) {  
 String get11 = i1.next().toString();  
 StringTokenizer st1 = **new** StringTokenizer(d,**" "**);  
 **while** (st1.hasMoreTokens()) {  
 String x2=st1.nextToken();  
 **if** (get.contains(x2)) {  
 count1++;  
 *// System.out.println(count);* **if**(count1>=2) {  
 **int** n1 = get11.length();  
 get11= get11.replaceAll(**","**, **" "**);  
  
 res11.append(get11.substring(1, n - 4) + **"\n"**);  
 **break**;  
 }  
  
 }  
 }  
 count1=0;  
  
 }  
 **return** res11.toString();  
}  
 }

