# [SRC] 06.1. Analyze Free Text from the Tweets

import org.apache.spark.ml.feature.Tokenizer

val tokenizer = new Tokenizer()

.setInputCol("text")

.setOutputCol("words")

val words = tokenizer.transform(tweets)

words.select("words").take(5).foreach(println)

words.select("id", "words")

.write

.mode("overwrite")

.parquet("/dataset/twitter/features/f=words")

import org.apache.spark.ml.feature.StopWordsRemover

val remover = new StopWordsRemover()

.setInputCol("words")

.setOutputCol("noStopWords")

val noStopWords = remover.transform(words)

noStopWords.select("words", "noStopWords").take(5).foreach(println)

noStopWords.select("id", "noStopWords")

.write

.mode("overwrite")

.parquet("/dataset/twitter/features/f=noStopWords")

import org.apache.spark.ml.feature.{HashingTF, IDF}

val hashingtf = new HashingTF()

.setInputCol("words")

.setOutputCol("rawFeatures")

.setNumFeatures(20)

val tf = hashingtf.transform(words)

val idf = new IDF()

.setInputCol("rawFeatures")

.setOutputCol("features")

val idfModel = idf.fit(tf)

val tfidf = idfModel.transform(tf)

tfidf.select("features").take(5).foreach(println)

tfidf.select("id", "features")

.write

.mode("overwrite")

.parquet("/dataset/twitter/features/f=tf-idf")

import org.apache.spark.ml.feature.{CountVectorizer, CountVectorizerModel}

val cvModel = new CountVectorizer()

.setInputCol("words")

.setOutputCol("rawFeatures")

.setVocabSize(200)

.setMinDF(2)

.fit(words)

val countVector = cvModel.transform(words)

val idf = new IDF()

.setInputCol("rawFeatures")

.setOutputCol("features")

val idfModel = idf.fit(countVector)

val tfcv = idfModel.transform(countVector)

tfcv.select("features").take(5).foreach(println)

tfcv.select("id", "features")

.write

.mode("overwrite")

.parquet("/dataset/twitter/features/f=tf-cv")

import org.apache.spark.ml.feature.Word2Vec

val word2Vec = new Word2Vec()

.setInputCol("words")

.setOutputCol("vec")

.setVectorSize(100)

.setMinCount(0)

val model = word2Vec.fit(words)

val vec = model.transform(words)

vec.select("vec").take(3).foreach(println)

val synonyms = model.findSynonyms("twitter", 3)

synonyms.show

vec.select("id", "vec")

.write

.mode("overwrite")

.parquet("/dataset/twitter/features/f=word2vec")

# [SRC] 06.2. Deal with Stemming and Syntax Idioms as Hashtags...

import java.util.Properties

import edu.stanford.nlp.ling.CoreAnnotations.\_

import edu.stanford.nlp.pipeline.\_

import edu.stanford.nlp.sentiment.SentimentCoreAnnotations.SentimentClass

import org.scalatest.FunSuite

import scala.io.Source

import scala.collection.JavaConverters.\_

import scala.collection.JavaConversions.\_

import scala.collection.mutable.ArrayBuffer

def stem(text: String, stopWords: Set[String], pipeline: StanfordCoreNLP): Seq[String] = {

val doc = new Annotation(text)

pipeline.annotate(doc)

val stems = new ArrayBuffer[String]()

val sentences = doc.get(classOf[SentencesAnnotation])

for (sentence <- sentences.asScala; token <- sentence.get(classOf[TokensAnnotation]).asScala) {

val stem = token.get(classOf[LemmaAnnotation])

if (!stopWords.contains(stem)) {

stems += stem.toLowerCase

}

}

stems

}

val stopWords = sc.broadcast(Source.fromFile("/dataset/stop-words/stop-words.txt").getLines().toSet).value

val tweets = spark.read.parquet("/dataset/twitter/reference")

tweets.count

val stemmed = tweets.mapPartitions(it => {

val props = new Properties()

props.put("annotators", "tokenize, ssplit, pos, lemma")

val pipeline = new StanfordCoreNLP(props)

it.map { tweet => stem(tweet.getAs[String]("text"), stopWords, pipeline) }

})

stemmed.persist

stemmed.count

stemmed.take(5).foreach(println)

val stemmedTwitter = tweets.mapPartitions(it => {

val props = new Properties()

props.put("annotators", "tokenize, ssplit, pos, lemma")

props.put("pos.model", "/dataset/models/gate-EN-twitter.model")

val pipeline = new StanfordCoreNLP(props)

it.map { tweet => stem(tweet.getAs[String]("text"), stopWords, pipeline) }

})

stemmedTwitter.printSchema

stemmedTwitter.take(5).foreach(println)

tweets.count

val s = stemmed.flatMap(identity)

s.count

s.filter(t => t.startsWith("#")).count

s.filter(t => t.startsWith("@")).count

s.filter(t => t.startsWith("http")).count

# [SRC] 06.3. Detect Tweet Sentiment

import java.util.Properties

import edu.stanford.nlp.ling.CoreAnnotations.\_

import edu.stanford.nlp.pipeline.\_

import edu.stanford.nlp.sentiment.SentimentCoreAnnotations.SentimentClass

import org.scalatest.FunSuite

import scala.io.Source

import scala.collection.JavaConverters.\_

import scala.collection.JavaConversions.\_

import scala.collection.mutable.ArrayBuffer

def sentiment(text: String, pipeline: StanfordCoreNLP) = {

val document = new Annotation(text)

pipeline.annotate(document)

var sentiments = 0

val sentences = document.get(classOf[SentencesAnnotation])

for (sentence <- sentences.asScala; token <- sentence.get(classOf[TokensAnnotation]).asScala) {

val sentiment = token.get(classOf[SentimentClass]).toLowerCase

sentiment match {

case "very positive" => sentiments += 2

case "positive" => sentiments += 1

case "neutral" =>

case "negative" => sentiments -= 1

case "very negative" => sentiments -= 2

case \_ =>

}

}

(sentiments, text)

}

val sentiments = tweets.mapPartitions(it => {

val props = new Properties()

props.put("annotators", "tokenize, ssplit, pos, parse, sentiment")

props.put("pos.model", "/dataset/models/gate-EN-twitter.model")

val pipeline = new StanfordCoreNLP(props)

it.map { tweet => sentiment(tweet.getAs[String]("text"), pipeline) }

})

sentiments.take(10).foreach(println)

%sh grep " estimate#" /dataset/sentiment/SentiWordNet-v0.3/sentiwordnet.tsv

import java.{util => ju}

import scala.io.Source

import scala.collection.JavaConversions.\_

var synsets = new ju.HashMap[String, Double]

def init(sentinetword: String) {

val lines = Source.fromFile(sentinetword)

val dictionary = new ju.HashMap[String, ju.HashMap[Integer, Double]]

for (line <- lines.getLines()) {

if (! line.trim.startsWith("#")) {

val data = line.split("\t")

/\*

n NOUN

v VERB

a ADJECTIVE

s ADJECTIVE SATELLITE

r ADVERB

\*/

val wordTypeMarker = data(0)

val synsetScore = data(2).toDouble - data(3).toDouble

val synTermsSplit = data(4).split(" ")

for (synTermSplit <- synTermsSplit) {

val synTermAndRank = synTermSplit.split("#")

val synTerm = synTermAndRank(0) + "#" + wordTypeMarker

val synTermRank = synTermAndRank(1).toInt

if (! dictionary.containsKey(synTerm)) dictionary.put(synTerm, new ju.HashMap[Integer, Double])

dictionary.get(synTerm).put(synTermRank, synsetScore)

}

}

}

for (entry <- dictionary.entrySet) {

val word = entry.getKey

val synSetScoreMap = entry.getValue

var score = 0.0

var sum = 0.0

for (setScore <- synSetScoreMap.entrySet) {

score += setScore.getValue / setScore.getKey

sum += 1.0 / setScore.getKey

}

score /= sum

synsets.put(word, score)

}

}

def sentiment(word: String, pos: String): Double = {

return synsets.get(word + "#" + pos)

}

init("/dataset/sentiment/SentiWordNet-v0.3/sentiwordnet.tsv")

println("good # a " + sentiment("good", "a"))

println("bad # a " + sentiment("bad", "a"))

println("yellow # a " + sentiment("blue", "a"))

println("yellow # n " + sentiment("blue", "n"))

# [SRC] 06.4. Identify Topics with LDA

val sample = stemmedTwitter

.rdd

.sample(fraction = 0.1, withReplacement = false)

.collect

val df = sc.parallelize(sample).toDF("words")

import org.apache.spark.ml.feature.CountVectorizer

import org.apache.spark.mllib.clustering.LDA

import org.apache.spark.ml.linalg.Vector

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.sql.Row

val cvModel = new CountVectorizer()

.setInputCol("words")

.setOutputCol("features")

.setVocabSize(300)

.setMinDF(2)

.fit(df)

val vocabulary = cvModel.vocabulary

val features = cvModel

.transform(df)

.select("features")

val corpus = features.rdd.map {

case Row(features: Vector) => Vectors.fromML(features)

}.zipWithIndex().map(\_.swap)

val lda = new LDA()

lda.setK(3)

.setMaxIterations(10)

.setTopicConcentration(10)

val ldaModel = lda.run(corpus)

val topicIndices = ldaModel.describeTopics(maxTermsPerTopic = 20)

val topics = topicIndices.map {

case (terms, termWeights) =>

terms.zip(termWeights).map {

case (term, weight) => (vocabulary(term.toInt), weight) }

}

topics.zipWithIndex.foreach { case (topic, i) =>

println(s"Topic $i")

topic.foreach { case (term, weight) =>

println(s"$term\t$weight")

}

println()

}