indexer

```
import org.apache.spark.ml.feature.{StringIndexer, StringIndexerModel}
val data = sc.textFile("label_output_new.txt")
```

FINISHED

```
import org.apache.spark.ml.feature.{StringIndexer, StringIndexerModel}
data: org.apache.spark.rdd.RDD[String] = label_output_new.txt MapPartitionsRDD[32] at te
xtFile at <console>:32
```

```
data.collect()
```

FINISHED

```
res14: Array[String] = Array(neu, fru, neu, fru, neu, fru, fru, neu, fru, fru, fru, fru,
fru, fru, fru, fru, fru, fru, fru, fru, fru, ang, ang, fru, neu, ang, fru, sad, sad, sad, sad
, neu, sad, fru, sad, neu, sad, sad, sad, fru, sad, sad, sad, sad, sad, sad, neu, sad, s
ad, neu, sad, sad, neu, neu, sad, sad, fru, neu, hap, neu, hap, hap, exc, hap, hap, hap,
neu, hap, hap, neu, hap, neu, hap, neu, hap, hap, hap, hap, exc, hap, hap, neu, neu, neu
, neu, neu, hap, neu, neu, neu, neu, hap, exc, hap, hap, neu, hap, neu, neu, neu, neu, n
eu, fru, neu, neu, neu, neu, neu, neu, neu, fru, neu, neu, neu, neu, neu, neu, neu, neu,
neu, neu, neu, fru, neu, fru, neu, fru, neu, fru, neu, neu, neu, fru, fru, neu, neu, fru
, neu, fru, neu, fru, neu, fru, neu, fru, neu, fru, ang, fru, ang, neu, ang, neu, ang, f
ru...
```

```
val array = data.collect()
```

FINISHED

```
array: Array[String] = Array(neu, fru, neu, fru, neu, fru, fru, neu, fru, fru, fru, fru,
fru, fru, fru, fru, fru, fru, fru, fru, fru, ang, ang, fru, neu, ang, fru, sad, sad, sad, sad
, neu, sad, fru, sad, neu, sad, sad, sad, fru, sad, sad, sad, sad, sad, sad, neu, sad, s
ad, neu, sad, sad, neu, neu, sad, sad, fru, neu, hap, neu, hap, hap, exc, hap, hap, hap,
neu, hap, hap, neu, hap, neu, hap, neu, hap, hap, hap, hap, exc, hap, hap, neu, neu, neu
, neu, neu, hap, neu, neu, neu, neu, hap, exc, hap, hap, neu, hap, neu, neu, neu, neu, n
eu, fru, neu, neu, neu, neu, neu, neu, neu, fru, neu, neu, neu, neu, neu, neu, neu, neu,
neu, neu, neu, fru, neu, fru, neu, fru, neu, fru, neu, neu, neu, fru, fru, neu, neu, fru
, neu, fru, neu, fru, neu, fru, neu, fru, neu, fru, ang, fru, ang, neu, ang, neu, ang, f
ru...
```

```
val df: Seq[String] = array
```

FINISHED

```
df: Seq[String] = WrappedArray(neu, fru, neu, fru, neu, fru, fru, neu, fru, fru, fru, fr
u, fru, fru, fru, fru, fru, fru, fru, fru, ang, ang, fru, neu, ang, fru, sad, sad, sad,
sad, neu, sad, fru, sad, neu, sad, sad, sad, fru, sad, sad, sad, sad, sad, sad, neu, sad
, sad, neu, sad, sad, neu, neu, sad, sad, fru, neu, hap, neu, hap, hap, exc, hap, hap, h
ap, neu, hap, hap, neu, hap, neu, hap, neu, hap, hap, hap, hap, exc, hap, hap, neu, neu,
neu, neu, neu, hap, neu, neu, neu, neu, hap, exc, hap, hap, neu, hap, neu, neu, neu, neu
, neu, fru, neu, neu, neu, neu, neu, neu, neu, fru, neu, neu, neu, neu, neu, neu, neu, n
eu, neu, neu, neu, fru, neu, fru, neu, fru, neu, fru, neu, neu, neu, fru, fru, neu, neu,
fru, neu, fru, neu, fru, neu, fru, neu, fru, neu, fru, ang, fru, ang, neu, ang, neu, ang
, f...
```

```
val df1 = df.toDF()
```

FINISHED

```
df1: org.apache.spark.sql.DataFrame = [value: string]
```

```
df1
```

FINISHED

```
res15: org.apache.spark.sql.DataFrame = [value: string]
```

```
df1.show()
```

FINISHED

```
+-----+
|value|
+-----+
|  neu|
|  fru|
|  neu|
|  fru|
|  neu|
|  fru|
|  fru|
|  neu|
|  fru|
|  fru|
|  fru|
|  fru|
|  fru|
|  fru|
|  fru|
|  fru|
```

```
val indexer = new StringIndexer().
        setInputCol("value").
        setOutputCol("index")
```

FINISHED

```
indexer: org.apache.spark.ml.feature.StringIndexer = strIdx_b0d2f4fac037
```

```
val model = indexer.fit(df1)
```

FINISHED

```
model: org.apache.spark.ml.feature.StringIndexerModel = strIdx_b0d2f4fac037
```

```
val indexed1 = model.transform(df1)
```

FINISHED

```
indexed1: org.apache.spark.sql.DataFrame = [value: string, index: double]
```

```
indexed1.show()
```

FINISHED

```
+-----+-----+
|value|index|
+-----+-----+
|  neu|  1.0|
|  fru|  0.0|
|  neu|  1.0|
|  fru|  0.0|
|  neu|  1.0|
|  fru|  0.0|
|  fru|  0.0|
|  neu|  1.0|
|  fru|  0.0|
|  fru|  0.0|
|  fru|  0.0|
|  fru|  0.0|
|  fru|  0.0|
|  fru|  0.0|
|  fru|  0.0|
|  fru|  0.0|
```

READY

Analysis to Text dataset

```
//show the number of words in the longest sentense
val lines = sc.textFile("file:///ilab/users/xl422/Downloads/text_output_new.txt")
var longest = lines.map(line => line.split(" ").size).reduce((a,b) => if (a>b) a else b)
print("length of the longest Sentense: "+longest)
```

```
lines: org.apache.spark.rdd.RDD[String] = file:///ilab/users/xl422/Downloads/text_output_new.txt MapPartitionsRDD[1] at textFile at <console>:27
longest: Int = 48
length of the longest Sentense: 48
```

Took 31 sec. Last updated by xl422 at December 10 2018, 10:37:07 PM. (outdated)

```
//show the number of sentences
val fileRdd = sc.textFile("file:///ilab/users/xl422/Downloads/text_output_new.txt")
println( "How much sentense included in the text file: " + fileRdd.count())
```

```
fileRdd: org.apache.spark.rdd.RDD[String] = file:///ilab/users/xl422/Downloads/text_output_new.txt MapPartitionsRDD[7] at textFile at <console>:28
How much sentense included in the text file: 7204
```

```
//print every sentense
val input = sc.textFile("file:///ilab/users/xl422/Downloads/text_output_new_no Punctuation.txt")
input.take(7204).foreach(line=>println(line))
```

```
Excuse me
Do you have your forms
Yeah
Let me see them
Is there a problem
Who told you to get in this line
Okay But I didn't tell you to get in this line if you are filling out this particular form
Well what's the problem Let me change it
This form is a Z X four
You can't This is not the line for Z X four If you're going to fill out the Z X four you need to have a different form of ID
What I'm getting an ID This is why I'm here My wallet was stolen
No I need another set of ID to prove this is actually you
How am I supposed to get an ID without an ID How does a person get an ID in the first place
I don't know But I need an ID to pass this form along I can't just send it along without an ID
I'm here to get an ID
No I need another ID a separate one
Like what Like a birth certificate
A birth certificate a passport a student ID didn't you go to school Anything
Yes but my wallet was stolen I don't have anything I don't have any credit cards I don't have my ID Don't you have things on file here
```

```
//generate a dictionary to convert every word into onehot encoding index sorted by the word frequency
import org.apache.spark.sql.functions.row_number
import org.apache.spark.sql.expressions.Window
val rdd = sc.textFile("file:///ilab/users/xl422/Downloads/text_output_new_no Punctuation.txt")
val wordcount = rdd.flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_).map(x => (x._2,x._1)).sortByKey(false).map(x => (x._2,x._1)).toDF("words","count")
val w = Window.orderBy(-wordcount("count"))// sort the words by the count in decrease order
val result = wordcount.withColumn("onhot", row_number().over(w))//generate the onehot encoding index to each words
result.show()
//Because this existing OneHotEncoder is a stateless transformer, it is not usable on new data where the number of categories may differ from the training data. In
    order to fix this, a new OneHotEncoderEstimator was created that produces an OneHotEncoderModel when fitting. For more detail, please see SPARK-13030.
    OneHotEncoder has been deprecated in 2.3.0 and will be removed in 3.0.0. Please use OneHotEncoderEstimator instead.
```

FINISHED

```
+-----+-----+-----+
|words|count|onhot|
+-----+-----+-----+
|    I| 4057|    1|
|  you| 2946|    2|
|   to| 2291|    3|
|  the| 1504|    4|
|    a| 1373|    5|
|  and| 1272|    6|
| know| 1260|    7|
|   it| 1218|    8|
| that| 1213|    9|
|   of|  885|   10|
|don't|  855|   11|
| just|  775|   12|
|   is|  767|   13|
| like|  750|   14|
|  I'm|  721|   15|
|   do|  681|   16|
```

```
//each line in text coloumn stands for the words in each sentense
import org.apache.spark.ml.feature.StandardScaler
import org.apache.spark.sql.SparkSession
import org.apache.spark.ml.feature.Word2Vec
val spark = SparkSession.builder().
            master("local").
            appName("my App Name").
            getOrCreate()
import spark.implicits._
val input = sc.textFile("file:///ilab/users/xl422/Downloads/text_output_new_no Punctuation.txt").map(line => line.split(" ").toSeq)
val documentDF = spark.createDataFrame(input.map(Tuple1.apply)).toDF("text") //split the sentenses to words and convert to dataframe format
documentDF.show()
```

```
+--------------------+
|                text|
+--------------------+
|        [Excuse, me]|
|[Do, you, have, y...|
|              [Yeah]|
|[Let, me, see, them]|
|[Is, there, a, pr...|
|[Who, told, you, ...|
|[Okay, But, I, di...|
|[Well, what's, th...|
|[This, form, is, ...|
|[You, can't, This...|
|[What, I'm, getti...|
|[No, I, need, ano...|
|[How, am, I, supp...|
|[I, don't, know, ...|
|[I'm, here, to, g...|
|[No, I, need, ano...|
```

```
//each line in text coloumn stands for the words in each sentense
//we remove the stop words in each sentense, and each line in filtered coloumn stand for the words of the sentense without the stop words
import org.apache.spark.ml.feature.StopWordsRemover
val remover = new StopWordsRemover()//define the remover
  .setInputCol("text")
  .setOutputCol("filtered")
val filteredDF = remover.transform(documentDF)//remove the stop words from the sentenses
filteredDF.show()
```

```
+--------------------+--------------------+
|                text|            filtered|
+--------------------+--------------------+
|        [Excuse, me]|            [Excuse]|
|[Do, you, have, y...|             [forms]|
|              [Yeah]|              [Yeah]|
|[Let, me, see, them]|          [Let, see]|
|[Is, there, a, pr...|           [problem]|
|[Who, told, you, ...|   [told, get, line]|
|[Okay, But, I, di...|[Okay, tell, get,...|
|[Well, what's, th...|[Well, problem, L...|
|[This, form, is, ...|  [form, Z, X, four]|
|[You, can't, This...|[line, Z, X, four...|
|[What, I'm, getti...|[getting, ID, wal...|
|[No, I, need, ano...|[need, another, s...|
|[How, am, I, supp...|[supposed, get, I...|
|[I, don't, know, ...|[know, need, ID, ...|
|[I'm, here, to, g...|           [get, ID]|
|[No, I, need, ano...|[need, another, I...|
```

```
//each line in text coloumn stands for the words in each sentense
//we remove the stop words in each sentense, and each line in filtered coloumn stand for the words of the sentense without the stop words
//each line in the result stands for the word2vec features of each word in each sentense
val word2Vec = new Word2Vec()
    .setInputCol("text")
    .setOutputCol("result")
    .setVectorSize(1000)
    .setMinCount(0)
val model = word2Vec.fit(filteredDF)// extract the word2vec features from the words
val result = model.transform(filteredDF)
result.show()
```

```
+--------------------+--------------------+--------------------+
|                text|            filtered|              result|
+--------------------+--------------------+--------------------+
|        [Excuse, me]|            [Excuse]|[-0.0256313392164...|
|[Do, you, have, y...|             [forms]|[-0.0247791818226...|
|              [Yeah]|              [Yeah]|[-0.0126148136332...|
|[Let, me, see, them]|          [Let, see]|[-0.0184083408094...|
|[Is, there, a, pr...|           [problem]|[0.01124024497403...|
|[Who, told, you, ...|   [told, get, line]|[-0.0133896946608...|
|[Okay, But, I, di...|[Okay, tell, get,...|[-0.0122802466450...|
|[Well, what's, th...|[Well, problem, L...|[-0.0108132692694...|
|[This, form, is, ...|  [form, Z, X, four]|[0.02174417074222...|
|[You, can't, This...|[line, Z, X, four...|[-0.0028138925338...|
|[What, I'm, getti...|[getting, ID, wal...|[0.00575942211019...|
|[No, I, need, ano...|[need, another, s...|[-0.0073616716657...|
|[How, am, I, supp...|[supposed, get, I...|[-0.0019508874287...|
|[I, don't, know, ...|[know, need, ID, ...|[-0.0107165386289...|
|[I'm, here, to, g...|           [get, ID]|[-0.0093275682108...|
|[No, I, need, ano...|[need, another, I...|[0.00148644893852...|
```

```
//each line in text coloumn stands for the words in each sentense
//we remove the stop words in each sentense, and each line in filtered coloumn stand for the words of the sentense without the stop words
//each line in the result stands for the word2vec features of each word in each sentense
//each line in scaledFeatures stands for the word2vec features after standard scaler of each word in each sentense
val scaler = new StandardScaler()//define the standscaler
  .setInputCol("result")
  .setOutputCol("scaledFeatures")
  .setWithStd(true)
  .setWithMean(false)
val scalerModel = scaler.fit(result)//stand scale the word2vec feature
val scaledDF = scalerModel.transform(result)
scaledDF.show()
```

```
+--------------------+--------------------+--------------------+--------------------+
|                text|            filtered|              result|      scaledFeatures|
+--------------------+--------------------+--------------------+--------------------+
|        [Excuse, me]|            [Excuse]|[-0.0256313392164...|[-2.0893965122283...|
|[Do, you, have, y...|             [forms]|[-0.0247791818226...|[-2.0199309774204...|
|              [Yeah]|              [Yeah]|[-0.0126148136332...|[-1.0283250276206...|
|[Let, me, see, them]|          [Let, see]|[-0.0184083408094...|[-1.5005974817863...|
|[Is, there, a, pr...|           [problem]|[0.01124024497403...|[0.91627395849176...|
|[Who, told, you, ...|   [told, get, line]|[-0.0133896946608...|[-1.0914912048853...|
|[Okay, But, I, di...|[Okay, tell, get,...|[-0.0122802466450...|[-1.0010520438590...|
|[Well, what's, th...|[Well, problem, L...|[-0.0108132692694...|[-0.8814680694787...|
|[This, form, is, ...|  [form, Z, X, four]|[0.02174417074222...|[1.77252519372407...|
|[You, can't, This...|[line, Z, X, four...|[-0.0028138925338...|[-0.2293808059082...|
|[What, I'm, getti...|[getting, ID, wal...|[0.00575942211019...|[0.46949230267879...|
|[No, I, need, ano...|[need, another, s...|[-0.0073616716657...|[-0.6001032943527...|
|[How, am, I, supp...|[supposed, get, I...|[-0.0019508874287...|[-0.1590309954113...|
|[I, don't, know, ...|[know, need, ID, ...|[-0.0107165386289...|[-0.8735828528228...|
|[I'm, here, to, g...|           [get, ID]|[-0.0093275682108...|[-0.7603577917893...|
|[No, I, need, ano...|[need, another, I...|[0.00148644893852...|[0.12117124281024...|
```

```
//each line in text coloumn stands for the words in each sentense
//we remove the stop words in each sentense, and each line in filtered coloumn stand for the words of the sentense without the stop words
//each line in the result stands for the word2vec features of each word in each sentense
//each line in scaledFeatures stands for the word2vec features after normalization of each word in each sentense
import org.apache.spark.ml.feature.Normalizer
val normalizer = new Normalizer()//define the normalizer
  .setInputCol("result")
  .setOutputCol("normFeatures")
  .setP(1.0)
val l1NormData = normalizer.transform(result)//normalize the word2vec features
val lInfNormData = normalizer.transform(result, normalizer.p -> Double.PositiveInfinity)
lInfNormData.show()
```

```
+--------------------+--------------------+--------------------+--------------------+
|                text|            filtered|              result|        normFeatures|
+--------------------+--------------------+--------------------+--------------------+
|        [Excuse, me]|            [Excuse]|[-0.0256313392164...|[-0.5341623616430...|
|[Do, you, have, y...|             [forms]|[-0.0247791818226...|[-0.4514962125027...|
|              [Yeah]|              [Yeah]|[-0.0126148136332...|[-0.2095667942535...|
|[Let, me, see, them]|          [Let, see]|[-0.0184083408094...|[-0.3489927528776...|
|[Is, there, a, pr...|           [problem]|[0.01124024497403...|[0.26634487840900...|
|[Who, told, you, ...|   [told, get, line]|[-0.0133896946608...|[-0.2750060863810...|
|[Okay, But, I, di...|[Okay, tell, get,...|[-0.0122802466450...|[-0.3634637306113...|
|[Well, what's, th...|[Well, problem, L...|[-0.0108132692694...|[-0.3634483632999...|
|[This, form, is, ...|  [form, Z, X, four]|[0.02174417074222...|[0.49854279525463...|
|[You, can't, This...|[line, Z, X, four...|[-0.0028138925338...|[-0.0849654014370...|
|[What, I'm, getti...|[getting, ID, wal...|[0.00575942211019...|[0.23740342086656...|
|[No, I, need, ano...|[need, another, s...|[-0.0073616716657...|[-0.2486897361675...|
|[How, am, I, supp...|[supposed, get, I...|[-0.0019508874287...|[-0.0550740241330...|
|[I, don't, know, ...|[know, need, ID, ...|[-0.0107165386289...|[-0.2604255395372...|
|[I'm, here, to, g...|           [get, ID]|[-0.0093275682108...|[-0.1594797662695...|
|[No, I, need, ano...|[need, another, I...|[0.00148644893852...|[0.04688953835197...|
```

```
//each line in words coloumn stands for the words in each sentense
//each line in features coloumn stands for the CountVectorizer features for each words in each sentense
import org.apache.spark.sql.SparkSession
import spark.implicits._
import org.apache.spark.ml.feature.{CountVectorizer, CountVectorizerModel}
val spark = SparkSession.builder().
            master("local").
            appName("my App Name").
            getOrCreate()
val df = sc.textFile("file:///ilab/users/xl422/Downloads/text_output_new_no Punctuation.txt").map(line => line.split(" ").toSeq).toDF("words")
val cvModel: CountVectorizerModel = new CountVectorizer().//define CountVectorizer
          setInputCol("words").
          setOutputCol("features").
          setVocabSize(50).
          setMinDF(2).
          fit(df)
var result = cvModel.transform(df)// extract its CountVectorizer features
result.show()
```

```
+--------------------+--------------------+
|               words|            features|
+--------------------+--------------------+
|        [Excuse, me]|     (50,[16],[1.0])|
|[Do, you, have, y...|(50,[1,17,48],[1....|
|              [Yeah]|     (50,[44],[1.0])|
|[Let, me, see, them]|     (50,[16],[1.0])|
|[Is, there, a, pr...|      (50,[4],[1.0])|
|[Who, told, you, ...|(50,[1,2,20,21,37...|
|[Okay, But, I, di...|(50,[0,1,2,20,21,...|
|[Well, what's, th...|(50,[3,7,16,33],[...|
|[This, form, is, ...|(50,[4,12],[1.0,1...|
|[You, can't, This...|(50,[1,2,3,4,9,12...|
|[What, I'm, getti...|(50,[12,14,32,46,...|
|[No, I, need, ano...|(50,[0,1,2,9,12,2...|
|[How, am, I, supp...|(50,[0,2,3,4,21,3...|
|[I, don't, know, ...|(50,[0,2,6,7,10,1...|
|[I'm, here, to, g...|(50,[2,14,37,46],...|
|[No, I, need, ano...|(50,[0,1,4,40],[1.0...|
```

```
//each line in sentense is the word in each sentense
//each line in words coloumn stands for the words vectors in each sentense
//each line in rawFeatures coloumn stands for TF features for each word in word vectors in each sentense
//each line in features coloumn stands for the TF-IDF features for each words in each sentense
import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().
            master("local").
            appName("my App Name").
            getOrCreate()
import spark.implicits._
val sentenceData = sc.textFile("file:///ilab/users/xl422/Downloads/text_output_new_no Punctuation.txt").toDF("sentence")
val tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words")
val wordsData = tokenizer.transform(sentenceData)
val hashingTF = new HashingTF().setInputCol("words").setOutputCol("rawFeatures").setNumFeatures(2000)//set the nums of the TF features
val featurizedData = hashingTF.transform(wordsData)//extract the TF features
val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features")
val idfModel = idf.fit(featurizedData)//extract the idf features
val rescaledData = idfModel.transform(featurizedData)
rescaledData.show()
```

```
+--------------------+
|            sentence|
+--------------------+
|          Excuse me |
|Do you have your ...|
|               Yeah |
|    Let me see them |
| Is there a problem |
|Who told you to g...|
|Okay But I didn't...|
|Well what's the p...|
|This form is a Z ...|
|You can't This is...|
|What I'm getting ...|
|No I need another...|
|How am I supposed...|
```

Because the audio data has saved as .mat files, they can not use spark to analyze.

READY

Audio branch data:

The format of the audio data is the index.mat files(index from 0 to 7203). And each index.mat file corresponds to the features of the dialog audios from the actors. And the index.mat files correspond to the sentences with the same line index in text_output.txt. For each index.mat file, it contains a 2-D array with floating-point number. The audio data has been preprocessed by the dataset provider. And the contents of it are the features of the audio files extracted with Mel-Frequency Cepstral Coefficients(MFCC) methods after the process of noise reduction and acoustic signal purification. Each matrix in the index.mat file has the same X dimension as 64 lines, which is split into the same size for the purpose of easy data-loading. The Y dimension of the matrix reveals different features of the sentences. And the length of features is proportional to the length of the dialog.