DADS 6002 / CI 7301
Big Data Analytics

Text Analytics

- The objective of text analytics is to uncover patterns and themes in the text data so we can derive contextual meaning and relationships.
- Text analytics applications includes :

1. Text Classification, e.g. document categorization
2. Sentimental Analysis, e.g. analyzing the political opinions of people on Facebook, Twitter or other social media.
3. Text Clustering, e.g. organize a collection of documents into groups, detection of duplicate or very similar documents

Text Analytics

4. Named Entity Recognition (NER), it detects the usages of words and nouns in sentences to extract information about person, organizations, locations and so on. Example “Mark Zuckerberg is one of the founders of Facebook, a company from the United States” we can identify three types of entities:
“Person”: Mark Zuckerberg, “Company”: Facebook
“Location”: United States. The information can give important contextual information on the actual content of the documents.
5. Event Extraction, it expands on the NER establishing relationships around the entities detected. It can help understand the actual content of documents.

Text Analytics

Feature Extraction Techniques

- TF-IDF (Term Frequency – Inverse Document Frequency), it measures how frequently words appear in documents and the relative frequency across the set of documents. This information can be used in building classifiers and predictive models.

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF
Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

For Spark, the following formula for IDF is used,
$idf_x = \log((N + 1) / (df_x + 1))$

Text Analytics

- Dimension Reduction, Text or document vector representation, e.g. TF-IDF or one-hot encoding is usually very long and sparse (containing a lot of zero entries) since there are ten thousand or more possible distinct words that can appear inside a document in a corpus. The representation dimension need to be reduced. Some dimension reduction techniques, such as SVD, LDA, Doc2Vec, can be used for this task.
- Topic Modeling, a technique for detecting the topics or themes in a corpus of documents. It is an unsupervised algorithm which can find themes in a set of documents.

Text Analytics

- In Spark ML, a transformer is a function object that transforms one dataset to another by applying the transformation logic (function) to the input dataset yielding an output dataset.
- For text analytics, there are two types of transformers, the Standard transformer and the Estimator transformer.

Standard Transformer

- A standard transformer transforms the input dataset into output dataset.
- This transformer is invoked as follows,

outputDF = transformer.transform(inputDF)

Tokenizer

- Tokenizer, a transformer which converts the input string (representing a document) in a given column of inputDF into lowercase and then splits it into a sequence of tokens or words using whitespaces as the delimiter and write the sequence into the given output column

```
import org.apache.spark.ml.feature.Tokeniser
import org.apache.spark.ml.feature.RegexTokenizer

val tokenizer = new
Tokenizer().setInputCol(“sentence”).setOutputCol(“words”)
val wordDF = tokenizer.transform(sentenceDF)
```

StopWordsRemover

- StopWordsRemover, It takes a string array of words in a given input column of inputDF and removes stop words in the array and writes the output string array into the given output column of the outputDF. The examples of stopwords are I, you, my, and, or etc.

```
import org.apache.spark.ml.feature.StopwordsRemover
val remover = new
StopwordsRemover().setInputCol(“words”).setOutputCol(“fil
teredWords”)
val noStopwordDF = remover.transform(wordDF)
```

NGrams

- Ngrams, it extracts N-grams from a given sentence stored in a given column of input DF and store the list of N-grams in the given column of the output DF.
- So we need to initializes the Ngram generator specifying the value of N, the input Column and the output column.

```
import org.apache.spark.ml.feature.NGram
val ngram = new
Ngram().setN(2),setInputCol(“filteredWords”).setOutputCol(
“ngrams”)
val nGramDF = ngram.transform(noStopWordsDF)
```

TF-IDF

- TF-IDF measures how important a word is to a document in a collection of documents.
- TF-IDF = TF * IDF
- HashingTF is a transformer which takes a set of terms (words) and converts them into vectors of fixed length by hashing each term using a hash function to generate an index for each term, then count the term frequencies TF using the indices of the hash table.

import org.apache.spark.ml.feature.HashingTF

val hashingTF = new HashingTF().setInputCol(“filteredWords”).setOutputCol(“rawFeatures”).setNumFeatures(100)

val rawFeatureDF = hashingTF.transform(noStopWordsDF)

TF-IDF

- Inverse Document Frequency (IDF) is an estimator, which is fit onto a given collection of documents then compute IDF features. IDF works on an input which is the output of a hashingTF transformer.

import org.apache.spark.ml.feature.IDF

val idf = new IDF().setInputCol(“rawFeatures”).setOutputCol(“features”).setNumFeatures(100)

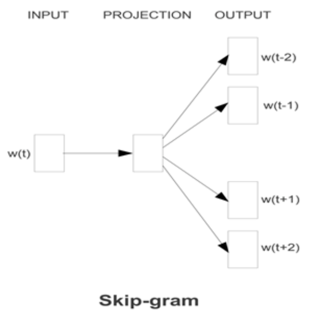
val idfModel = idf.fit(rawFeaturesDF)

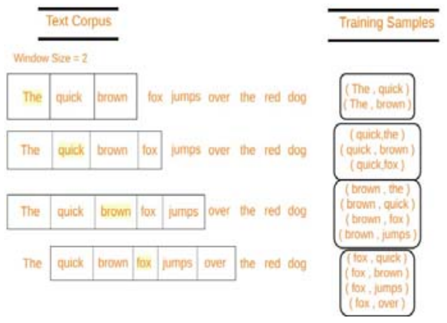
val featureDF = idfModel.transform(rawFeaturesDF)

Word2Vec

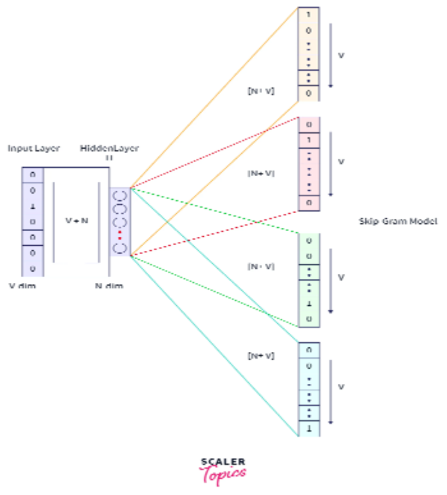
- Word2Vec is an estimator, a neural network that uses a technique called skip-grams to convert words in sentences of corpus into an embedded vector representation called neural word embeddings.
- Example, given a collection of sentences (corpus) about animals:
  - A dog was barking
  - Some cows were grazing the grass
  - Dogs usually bark randomly
  - The cow likes grass
- Using neural network with a hidden layer (unsupervised learning), we can learn (with enough examples) that dog and barking are related, cow and grass are related that appear close to each other a lot.

Word2Vec





Word2Vec



Source : https://www.scaler.com/topics/nlp/skip-gram-model/

Word2Vec

- The output of word2vec is a vector of double features for each word in a sentence. Words that are similar or related have similar feature vectors, e.g. based on cosine similarity

```
import org.apache.spark.ml.feature.Word2Vec
val word2vec = new
Word2vec().setInputCol(“words”).setOutputCol(“wordvector
”).setVectorsize(3).setMinCount(0)
val word2VecModel = word2vec.fit(noStopWordDF)
val word2VectorDF =
        word2VecModel.transform(noStopWordsDF)
```

CountVectorizer

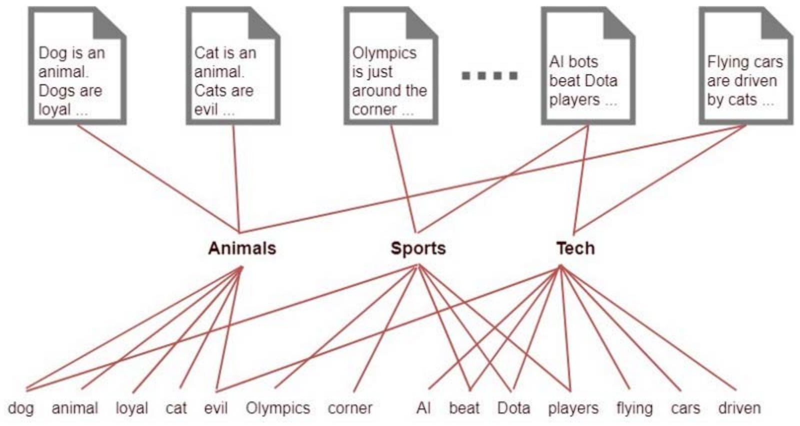
- CountVectorizer, an estimator that creates a model (a transformer) by extracting a vocabulary from the given dataset.
- The transformer can then be used to convert a collection of text documents to vectors of token counts. The output count vectors can be passed to other algorithms for futher analysis.

```
import org.apache.spark.ml.feature.CountVectorizer
val countVectorizer = new
CountVectorizer().setInputCol(“filteredWords”).setOutputCol(“fea
tures”)
val countVectorizerModel = countVectorizer.fit(noStopWordDF)
val countVectorizerDF =
        countVectorizerModel.transform(noStopWordsDF)
```

Topic Modeling using LDA

- LDA (Latent Dirichlet Allocation), an estimator, infers topics from a collection of text documents.
- LDA can be thought of as an unsupervised clustering algorithm. LDA imagines a fixed set of topics. Each topic represents a set of words.
- Each document can be described by a distribution of topics and each topic can be described by a distribution of words.
- The goal of LDA is to map all the documents to the topics in a way, such that the words in each document are mostly captured by those imaginary topics.

Topic Modeling using LDA

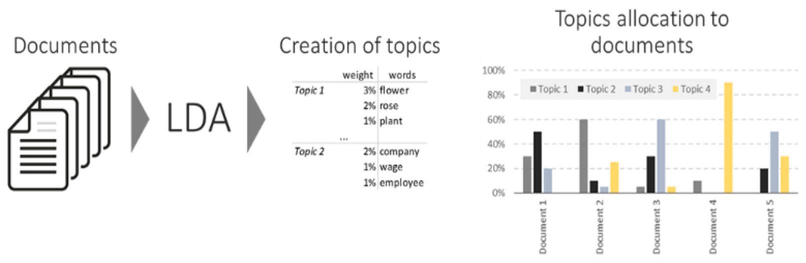

Dog is an animal. Dogs are loyal ...
Cat is an animal. Cats are evil ...
Olympics is just around the corner ...
....
AI bots beat Dota players ...
Flying cars are driven by cats ...
Animals
Sports
Tech
dog
animal
loyal
cat
evil
Olympics
corner
AI
beat
Dota
players
flying
cars
driven

Topic Modeling using LDA

- Rather than estimating a clustering of documents using a traditional distance, e.g. Euclidean or cosine similarity, LDA uses a function based on a statistical model of how text documents are generated from topics then words in the topics.

```
import org.apache.spark.ml.clustering.LDA
val lda = new LDA().setK(10).setMaxIter(10)
val ldaModel = lda.fit(counterVectorizerDF)
val topics = ldaModel.describeTopics(10)
# each topic in topics comprises of topic id, term Indices and their term weights
val ldaDF = ldaModel.transform(counterVectorizerDF)
# output column in ldaDF is TopicDistribution
```

Topic Modeling using LDA



Text Classification

- Example of text classification using Logistics Regression Model

```
import
org.apache.aprk.spark.ml.classification.LogisticsRegression
val inputData =
        countVectorizerDF.select(“label”,”features”).withColumn
        (“label”, col(“label”).cast(“double”))
val Array(trainingData, testData) =
        inputData.randomSplit(Array(0.8,0.2))
val lr = new LogisticRegression()
val lrmodel = lr.fit(trainingData)
```

Text Classification

```
lrModel.coefficients
lrModel.intercept
val training = lrModel.transform(trainingData)
val test = lrModel.transform(testData)
training.filter( “label == prediction” ).count
training.filter( “label != prediction” ).count
test.filter( “label == prediction” ).count
test.filter( “label != prediction” ).count
```