Feature Extractors

TF-IDF

Term frequency-inverse document frequency (TF-IDF) is a feature vectorization method widely used in text mining to reflect the importance of a term to a document in the corpus.

In [10]:
```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer

import findspark
findspark.init()

import pyspark
import random

from pyspark.sql import SparkSession
from pyspark import SparkContext, SparkConf

spark = SparkSession.builder.appName('abc').getOrCreate()

sc = spark.sparkContext
```

In [11]:
```
sentenceData = spark.createDataFrame([
    (0.0, "Hi I heard about Spark"),
    (0.0, "I wish Java could use case classes"),
    (1.0, "Logistic regression models are neat")
], ["label", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
wordsData = tokenizer.transform(sentenceData)

hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures", numFeatures=20)
featurizedData = hashingTF.transform(wordsData)
# alternatively, CountVectorizer can also be used to get term frequency vectors

idf = IDF(inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
rescaledData = idfModel.transform(featurizedData)
print(sentenceData.show())
print(rescaledData.select("label", "features").show())
```

```
+-----+--------------------+
|label|            sentence|
+-----+--------------------+
|  0.0|Hi I heard about ...|
|  0.0|I wish Java could...|
|  1.0|Logistic regressi...|
+-----+--------------------+

None
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(20,[0,5,9,17],[0...|
|  0.0|(20,[2,7,9,13,15]...|
|  1.0|(20,[4,6,13,15,18...|
+-----+--------------------+

None
```

Word2Vec

Word2Vec is an Estimator which takes sequences of words representing documents and trains a Word2VecModel. The model maps each word to a unique fixed-size vector. The Word2VecModel transforms each document into a vector using the average of all words in the document; this vector can then be used as features for prediction, document similarity calculations, etc.

In [12]:
```
from pyspark.ml.feature import Word2Vec

# Input data: Each row is a bag of words from a sentence or document.
documentDF = spark.createDataFrame([
    ("Hi I heard about Spark".split(" "), ),
    ("I wish Java could use case classes".split(" "), ),
    ("Logistic regression models are neat".split(" "), )
], ["text"])

# Learn a mapping from words to Vectors.
word2Vec = Word2Vec(vectorSize=3, minCount=0, inputCol="text", outputCol="result
model = word2Vec.fit(documentDF)

result = model.transform(documentDF)
for row in result.collect():
    text, vector = row
    print("Text: [%s] => \nVector: %s\n" % (", ".join(text), str(vector)))
```

```
Text: [Hi, I, heard, about, Spark] =>
Vector: [0.06088668443262577,-0.03652646020054817,-0.07268779873847962]

Text: [I, wish, Java, could, use, case, classes] =>
Vector: [-0.03514155585850988,0.023327834754517034,-0.04929091329021113]

Text: [Logistic, regression, models, are, neat] =>
Vector: [0.03419528752565384,-0.038853179663419724,0.07270659841597081]
```

CountVectorizer

In [24]:
```
from pyspark.ml.feature import CountVectorizer

# Input data: Each row is a bag of words with a ID.
df = spark.createDataFrame([
    (0, "a b c d R R".split(" ")),
    (1, "a b b c a".split(" ")),
    (2, "a a b a a".split(" ")),
     (3, "a b c a d".split(" "))
], ["id", "words"])


print(df.show())
# fit a CountVectorizerModel from the corpus.
cv = CountVectorizer(inputCol="words", outputCol="features", vocabSize=4, minDF=

model = cv.fit(df)

result = model.transform(df)
result.show(truncate=False)
```

```
+---+------------------+
| id|             words|
+---+------------------+
|  0|[a, b, c, d, R, R]|
|  1|   [a, b, b, c, a]|
|  2|   [a, a, b, a, a]|
|  3|   [a, b, c, a, d]|
+---+------------------+

None
+---+------------------+-------------------------------+
|id |words             |features                       |
+---+------------------+-------------------------------+
|0  |[a, b, c, d, R, R]|(4,[0,1,2,3],[1.0,1.0,1.0,1.0])|
|1  |[a, b, b, c, a]   |(4,[0,1,2],[2.0,2.0,1.0])      |
|2  |[a, a, b, a, a]   |(4,[0,1],[4.0,1.0])            |
|3  |[a, b, c, a, d]   |(4,[0,1,2,3],[2.0,1.0,1.0,1.0])|
+---+------------------+-------------------------------+
```

FeatureHasher

Feature hashing projects a set of categorical or numerical features into a feature vector of specified dimension (typically substantially smaller than that of the original feature space). This is done using the hashing trick to map features to indices in the feature vector.

The FeatureHasher transformer operates on multiple columns. Each column may contain either numeric or categorical features.

In [25]:
```
from pyspark.ml.feature import FeatureHasher

dataset = spark.createDataFrame([
    (2.2, True, "1", "foo"),
    (3.3, False, "2", "bar"),
    (4.4, False, "3", "baz"),
    (5.5, False, "4", "foo")
], ["real", "bool", "stringNum", "string"])

hasher = FeatureHasher(inputCols=["real", "bool", "stringNum", "string"],
                       outputCol="features")

featurized = hasher.transform(dataset)
featurized.show(truncate=False)
```

```
+----+-----+---------+------+---------------------------------------------------
------+
|real|bool |stringNum|string|features
|
+----+-----+---------+------+---------------------------------------------------
------+
|2.2 |true |1        |foo   |(262144,[174475,247670,257907,262126],[2.2,1.0,1.
0,1.0])|
|3.3 |false|2        |bar   |(262144,[70644,89673,173866,174475],[1.0,1.0,1.0,
3.3])  |
|4.4 |false|3        |baz   |(262144,[22406,70644,174475,187923],[1.0,1.0,4.4,
1.0])  |
|5.5 |false|4        |foo   |(262144,[70644,101499,174475,257907],[1.0,1.0,5.5,
1.0]) |
+----+-----+---------+------+---------------------------------------------------
------+
```

PCA

PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. A PCA class trains a model to project vectors to a low-dimensional space using PCA. The example below shows how to project 5-dimensional feature vectors into 3-dimensional principal components.

In [27]:
```
from pyspark.ml.feature import PCA
from pyspark.ml.linalg import Vectors

data = [(Vectors.sparse(5, [(1, 1.0), (3, 7.0)]),),
        (Vectors.dense([2.0, 0.0, 3.0, 4.0, 5.0]),),
        (Vectors.dense([4.0, 0.0, 0.0, 6.0, 7.0]),)]
df = spark.createDataFrame(data, ["features"])

pca = PCA(k=1, inputCol="features", outputCol="pcaFeatures")
model = pca.fit(df)

result = model.transform(df).select("pcaFeatures")
result.show(truncate=False)
```

```
+---------------------+
|pcaFeatures          |
+---------------------+
|[1.6485728230883807] |
|[-4.645104331781534] |
|[-6.428880535676489] |
+---------------------+
```

In [ ]: