JIGSAW ACADEMY
THE ONLINE SCHOOL OF ANALYTICS

Case study for Spark: ML-Lib

Most relevant terms in documents

Time: 5 hours.

Business Case Scenario: Wikipedia is an ocean of information and has hundreds of documents on many subjects. These documents can be downloaded and used for text mining to find lot of useful information. A manufacturing organization has collected the Wikipedia documents about manufacturing category and has lot text files that constitute the articles about manufacturing in Wikipedia. The organization wants to analyze these documents and extract the most relevant terms used in these documents so that they can use them in their web sites. This will use the text documents extracted from Wikipedia and use the TF/IDF based text processing algorithms from Spark-MLLib.

Data Set: The data set to be used for this case study is publicly available Wikipedia documents about manufacturing category. There are 251 documents that are downloaded and text extracted from them, stored as .txt files. The data is free form text and may have punctuation and other characters.

Problem: Analyze the text documents and for each document produce a list of ten most relevant terms in the document.

Output: List the terms per document like below:

```
|wikiout_54.txt |[branding, brand, signet, extrinsic, obm, anonymous, odm,
value, oem, differential]                              |
|wikiout_134.txt|[physics, factory, stocks, transformation, foundations,
flows, synchronize, book, sets, attendees]                  |
|wikiout_188.txt|[laser, metal, sintering, 3d, moulding, selective,
injection, melting, molding, moulded]                     |
|wikiout_142.txt|[seconds, factory, wafs, resold, factoryseconds, fault,
retailing, returned, retailer, item]               |
|wikiout_70.txt |[imts, show, comeback, emo, anonymous, numbered, sets,
attendees, exhibitors, strong]
```

Solution: Apache Spark can be used to provide a solution for this. Because of the distributed nature f Spark, the solution can scale to thousands of documents and can take any text data that can be from Wikipedia, twitter, facebook etc. Also the Spark uses distributed machine learning algorithms that can be used here. Spark provides built in algorithms for processing text and providing useful statistics as well as for tokenizing and removing stop words. Using the statistics, reverse processing is done extract the most relevant terms.

Text Processing and semantic analysis: Two factors are used extensively in analyzing text documents to understand the hidden semantics in documents. These are Term frequency and Inverse document frequency, popularly referred to as TF-IDF. These are explained below:

Term Frequency (TF) : This refers to how many times each word occurs in a document. This measure shows how important a term is to the document. For example if we take the 3 sentences below:

Spark is good at machine learning, machine learning predicts outcomes.

Spark provides distributed machine learning

hadoop stores distributed data

A term frequency matrix can be built like below:

	Spark	good	machine	learning	predicts	outcomes	provides	distributed	hadoop	stores	data
1	1	1	2	2	1	1	0	0	0	0	0
2	1	0	1	1	0	0	1	1	0	0	0
3	0	0	0	0	0	0	0	1	1	1	1

Normally stop words (is, was, were, at, a, an, the etc) can be removed before the analysis.

Inverse Document Frequency (IDF) : If a term is present in all the documents, then it may not be relevant to just one document. So inverse document frequency is used to reduce the effect of terms that are common in the language. It is calculated as:

$$\log\left[\text{ No.of documents + 1)} / \text{(No.of documents in which the term occurs + 1)}\right]$$

log (logarithm) is used above to reduce the effect of large number of occurrences and +1 is used to make sure division by zero does not occur.

A frequently used measure is the product of TF and IDF referred to as TF-IDF. For the above sentences, these can be as below:

IDF:

	Spark	good	machine	learning	predicts	outcomes	provides	distributed	hadoop	stores	data
	0.29	0.69	0.29	0.29	0.69	0.69	0.69	0.29	0.69	0.69	0.69

TF * IDF:

	Spark	good	machine	learning	predicts	outcomes	provides	distributed	hadoop	stores	data
1	0.29	0.69	0.58	0.58	0.69	0.69	0	0	0	0	0
2	0.29	0	0.29	0.29	0	0	0.69	0.29	0	0	0
3	0	0	0	0	0	0	0	0.29	0.69	0.69	0.69

The list of words forms the vocabulary for the documents.

Following ML library methods are available for use in the project. Explore these methods in the Spark documentation to learn more about them:

RegexTokenizer : This converts each text document into array of words. The pattern specified as a regular expression is extracted. For example the pattern \W extracts all words so that spaces and punctuation characters can be excluded.

StopWordsRemover : This removes the common stop words from the array of words. Stop words are words like is, are, a, an, the etc that do not add meaning to the document. The default setting removes the English language stop words.

CountVectorizerModel : This model builds a term frequency matrix by assigning a unique number to each distinct word. There is an alternative in using hashing TF model but CountVectorizerModel provides us a vocabulary that can be used to reverse map the indices back to unique words.

IDF : This model calculates the IDF and provides TF * IDF for each document and term, so that we don't have to calculate the product separately.

Tasks in the project.

Task 1. Get Wikipedia data : This is available as wikidocs.zip.

create a separate directory for this case study like casestudy4 and copy the wikidocs.zip to that directory and unzip. The unzipped files will be in wikidocs directory as .txt files.

Task 2. Load the data in Spark.

Start your spark shell and load the data in spark. You can load the data using the wholeTextFiles command. You can specify it as a regular file system file by using the prefix file://' and then the path of the files. For example the path with prefix can be "file:///home/hduser/casestudy4/wikidocs/*.txt.

You can do a count on the above RDD to check that the you have the file properly specified. You can also do a take(1) to check the number of fields in the data. The data will have two fields with the first one file name (full path) and second one the document as a string. Optionally you can use a map to remove the path from the file.

Task 3. Convert the data to dataframe.

As there are two fields in data, you can give the fields the names as (name , doc). Since the text is in free form, no need to format the data as in other case studies.

Task 4. use RegexTokenizer to convert the data into a bag of words.

Steps required are:

1. Create the RegexTokenizer model with input column (doc) and output column (allwords)
2. Fit the model with the documents dataframe
3. Transform the documents data with the fitted model

Task 5. Use StopWordsRemover to remove the stop words from the bag of words

Steps required are:

1. Create the StopWordsRemover model with input column (allwords) and output column (words)
2. Transform the words data with the model

Task 6. Build a subset of data by selecting only label and words columns

This can be done by select on the dataframe with label and words columns.

Task 7. Build a Term Frequency matrix by using CountVectorizer Model. Check the vocabulary built

Steps required are:

1. Create the CountVectorizer model with input column (words) and output column (rawFeatures)
2. Fit the model with the above subset dataframe
3. Check the number of words in the vocabulary of the model (You may see as high as 25500 words)
4. Transform the subset data with the fitted model

Task 8. Build a Inverse document frequency model using IDF.

Steps required are:

1. Create the IDF model with input column (rawFeatures) and output column (features)
2. Fit the model with the above dataframe
3. Transform the above data with the fitted model

Now we have the model built and applied to our data. Next we need to extract information from the model and show the relevant terms.

Task 9. Create a function to extract the top words given the vocabulary and the TF*IDF data.

Note that so far we have :

Vocabulary which is the list of all the distinct words from all the documents available in the CountVectorizerModel built in Task 7 above.

The TFIDF data for each document available in the dataframe as a sparse matrix in the features column. The sparse matrix has two arrays: an indices array that is an index to the vocabulary and a TFIDF array that has TFIDF number as a double value for each index within that document. For example if the vocabulary has 1000 words, then the indices can be from 0 to 999. If document 1 has two words “Spark” and “predicts” that map to index 4 and 25 in the vocabulary, then the indices array for that row in the dataframe will have two integers 4 and 25. The IFIDF array will have two double values (0.29 and 0.69 for example).

Task 10. Extract and print the top 10 most relevant terms for each document.

Use the print schema to check the schema of the dataframe created in task 9. Use a map to call the function created in task 9 for each row of the dataframe. Include the label from the dataframe also in the output so that you have the filename and the list of words.

Make sure your Row specification in the map matches the schema exactly. Otherwise you will get matching errors when you use show. For the words column that shows as Array of string, Spark actually uses an unwrapped array which gives problems in matching. So use the `Seq[String] instead of Array[String] for this column.` This may give a warning but will work properly.

You an use the show command on the resulting dataframe to check the words for each file. If you show only 5 rows, you may get output like below:

```
scala> strings.show(5,false)

+---------------+---------------------------------------------------------------------------------------------+

|_1             |_2                                                                                           |

+---------------+---------------------------------------------------------------------------------------------+

|wikiout_54.txt |[branding, brand, signet, extrinsic, obm, anonymous, odm, value, oem, differential]         |

|wikiout_134.txt|[physics, factory, stocks, 257, transformation, foundations, flows, synchronize, 256, book] |

|wikiout_188.txt|[laser, metal, sintering, 3d, moulding, selective, injection, melting, molding, moulded]    |

|wikiout_142.txt|[seconds, factory, wafs, resold, factoryseconds, fault, retailing, returned, retailer, item]|

|wikiout_70.txt |[imts, show, comeback, emo, 2010, numbered, sets, attendees, exhibitors, strong]            |

+---------------+---------------------------------------------------------------------------------------------+

only showing top 5 rows
```