Book Analysis using pyspark.

A screenshot of a computer

Description automatically generated with medium confidence

Book analysis in Eclipse

A screenshot of a computer

Description automatically generated

A screenshot of a computer error

Description automatically generated with medium confidence

`localPath`: This string variable represents the local directory path where the downloaded book file will be saved. It specifies the location on the local machine where the file will be stored.

`bookText`: This Dataset of Row objects is used to store the book's text. It reads the contents of the locally downloaded file and selects the "value" column, which contains the text of each line.

`bookWords`: This Dataset of Row objects stores the individual words extracted from the book's text. It performs tokenization by splitting the text into words using whitespace as the delimiter. The `lower()` function is also applied to convert the words to lowercase.

`stopWords`: This array of strings contains common stop words that will be filtered out from the words. Stop words are frequently used words that do not carry much information and are typically excluded from text analysis.

`filteredWords`: This Dataset of Row objects stores the filtered words, excluding empty words and common stop words. It applies filtering conditions using the `notEqual()` and `array\_contains()` functions to remove empty words and stop words, respectively.

`wordCounts`: This Dataset of Row objects stores the word frequencies. It performs word counting by grouping the filtered words by the "word" column and calculating the count for each word using the `count()` function.

`topWords`: This Dataset of Row objects represents the top 5 most frequent words. It orders the `wordCounts` dataset by the "count" column in descending order using the `orderBy()` function and limits the result to the top 5 rows using the `limit()` function.

`numInstances`: This long variable holds the total number of instances (words) in the book. It is calculated by counting the rows in the `filteredWords` dataset using the `count()` function.

The code also includes a `downloadFile()` method that takes a URL and local path as parameters. It downloads the file from the specified URL and saves it to the local directory specified by the local path.

Dataset<Row> bookWords = bookText

.select(functions.*explode*(functions.*split*(functions.*lower*(**new** Column("value")), "\\W+")).alias("word"));

Certainly! Let's break down the expression step by step:

1. `functions.lower(new Column("value"))`: The `lower()` function from `functions` is applied to convert the "value" column to lowercase. It converts all characters in the column's values to lowercase.

2. `functions.split(..., "\\W+")`: The `split()` function from `functions` is applied to split the lowercase values of the "value" column into an array of words. The delimiter `\\W+` specifies that any non-word character (such as punctuation) will be used as the delimiter for splitting.

3. `functions.explode(...)`: The `explode()` function from `functions` is applied to explode the array of words into multiple rows, with each row containing a single word. This is necessary to transform the array into a dataset of rows.

4. `.alias("word")`: The `alias()` function is used to provide an alias (or name) for the resulting column. In this case, the column resulting from the explode operation is named "word".

5. `bookWords`: The resulting dataset from the above transformations is stored in the `bookWords` variable. It contains a single column named "word" that holds the individual words extracted from the "value" column of the `bookText` dataset.

Overall, the expression is performing a sequence of operations to tokenize the text from the "value" column. It converts the text to lowercase, splits it into words using non-word characters as delimiters, and explodes the resulting array of words into multiple rows, resulting in a dataset (`bookWords`) with a single column named "word" that contains the individual words.

String[] stopWords = {"the", "and", "of", "to", "in", "a", "is", "that", "it", "with", "as"};

Dataset<Row> filteredWords = bookWords

.filter(functions.*col*("word").notEqual("").and(functions.*not*(functions.*array\_contains*(functions.*lit*(stopWords), functions.*col*("word")))));

Certainly! Let's go through the code snippet step by step:

1. `String[] stopWords = {...}`: This line declares an array of strings called `stopWords` and initializes it with common stop words. These are words that are typically excluded from text analysis as they do not carry significant meaning or information.

2. `bookWords.filter(...)`: The `filter()` function is applied to the `bookWords` dataset to filter out unwanted words based on specified conditions.

3. `functions.col("word").notEqual("")`: This condition checks if the value of the "word" column is not equal to an empty string. It ensures that empty words are excluded from the filtered dataset.

4. `functions.array\_contains(functions.lit(stopWords), functions.col("word"))`: This condition checks if the value of the "word" column is contained within the `stopWords` array. The `array\_contains()` function checks if the array `stopWords` contains the specified column value.

5. `functions.not(...)`: This negates the result of the previous condition. It ensures that words which are not present in the `stopWords` array are included in the filtered dataset.

6. `filteredWords`: The resulting dataset from the filtering operation is stored in the `filteredWords` variable. It contains the rows from `bookWords` dataset where the "word" column is not an empty string and is not present in the `stopWords` array.

Overall, this code snippet filters out empty words and common stop words from the `bookWords` dataset, resulting in the `filteredWords` dataset that contains only the relevant words for further analysis.