Spark Program

Chapter 10: Natural Language Processing

Chapter Objectives

In this chapter, we will:

- Explore unstructured text
- Use NLTK and Spark pipeline transformation to process natural language
- Visualize natural language with Word Clouds
- Explore how to monitor and optimize Spark

Chapter Concepts

Natural Language

Optimizations

Chapter Summary

Natural Language Processing

- Processing free-form text is not as simple as a formatted table-structured object
- You need to break the natural language up into pieces and fix variations in the wording to try to standardize it and extract meaning
- There are many different types of transformations you can do to text and they vary depending on the text and what you're trying to do with the results
- Generally though, it comes down to some common steps:
  - Break the text up into sentences and words
  - Fix the words by adjusting everything to the same case
  - Remove punctuation
  - Standardize word variations to the root word
  - Remove insignificant words
  - Find natural word groupings that make up a phrase
  - Determine the overall sentiment of the words

NLTK

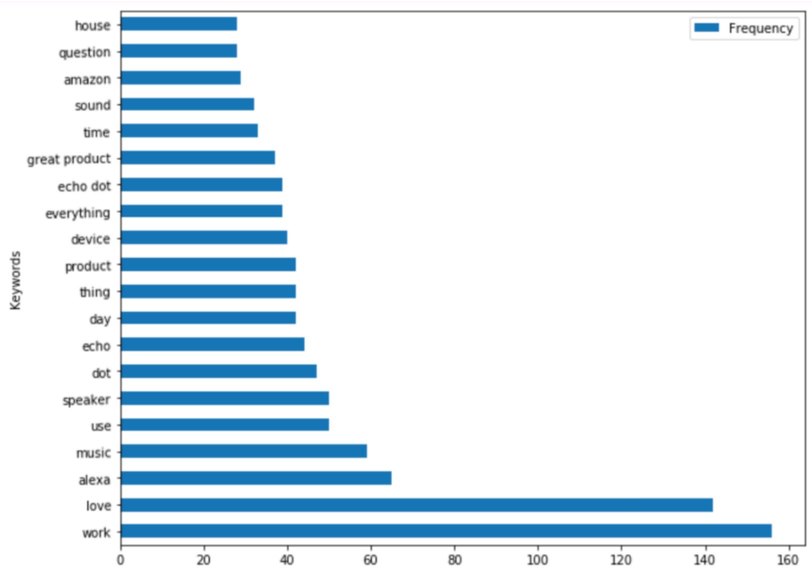
- NLTK stands for **N**atural **L**anguage **T**ool **K**it and is a comprehensive Python package with lots of functions to manipulate text
- It is a stand-alone Python package, but it works in a distributed mode on Spark RDDs
- Install it as normal: `pip install nltk`
- Import it as normal: `import nltk`
- Let's explore the features and examples in Jupyter

Spark Text Processing

- Spark also has some of its own text processing features
- Found in `pyspark.ml.features`
- `Tokenizer` breaks up a document into words
- `RegexTokenizer` allows more control
- `StopWordsRemover` removes insignificant words
- `HashingTF` transforms a set of words and vectorizes them
- `IDF` rescales the numbers in the vector to de-emphasize words that occur a lot in the entire corpus
- Usually, just follow a common recipe to fix up a DataFrame into the shape you need for ML operations by putting the various steps into a pipeline

Word Cloud

- Word Clouds and other charts can be made from the results of the text processing by bringing the small DataFrame results back to the driver



Chapter Concepts

Natural Language

Optimizations

Chapter Summary

Monitoring

- You can monitor the Spark processes by navigating in a browser to the address of the cluster and port 4040
  - `localhost:4040`
- You will see various tabs where you can:
  - Watch the progress of jobs
  - Watch the progress of stages
  - See what objects are cached
  - Get an overview of the environment
  - See the nodes that are executing

Caching

- Because an RDD is a lazy evaluation of a chain of transformations, each time you do an action on that RDD or DataFrame, it recalculates everything from the very beginning
- This can cause performance problems if you want to do several different actions to the same set of data
- Caching is the answer—it allows you to pin the results of an RDD in the cluster for the duration of the session or until you manually release it
- You can cache to either memory or disk or a combination of the two
- You also have control over how many redundant copies it stores and whether it should store it as:
  - Deserialized which takes more memory but less CPU
  - Serialized Java object which is more memory efficient but uses more CPU
- There are two methods, `persist` and `cache`, which is simply a shorthand for persist with the default option of memory only
- To remove a cached object, use `unpersist`

Broadcast Variables

- It is also sometimes desirable to keep a copy of a variable on each node instead of passing it around each time it receives instructions to do a new task
- This is useful if you need to pass around something like a reference table to each node
- Broadcast variables allow you to accomplish this

```
lookupTable = sc.broadcast([1, 2, 3])

lookupTable.value → [1, 2, 3]
```

Accumulators

- Sometimes you want to create a counter that is global to the entire process
- Using a regular Python variable would not scale out to the cluster level, so you need a special way to handle this so that each node can contribute to one global counter
- There is an `accumulator` method on the Spark context meant just for this

```
counter = sc.accumulator(0)
def fun2(x):
    global counter
    counter += x

x0.foreach(fun2)
print (counter.value)
```

Chapter Concepts

Natural Language

Optimizations

Chapter Summary

Chapter Summary

In this chapter, we have:

- Explored unstructured text
- Used NLTK and Spark pipeline transformation to process natural language
- Visualized natural language with Word Clouds
- Explored how to monitor and optimize Spark