Assignment 3

Spring 2019

Due Date: April 26, 2019

Instructions

- This assignment will involve writing code in the form of Scala classes. The code for the 2 parts should be in different classes in a single Scala Spark project. The project should be compiled into a jar file that can run on AWS cluster. You should include build.sbt for dependency management.
- All instructions for compiling and running your code must be placed in the README file.
- You should use a cover sheet, which can be downloaded from here
- You are allowed to work in pairs i.e. a group of two students is allowed. Please write the names of the group members on the cover page.
- **You have a total of 4 free late days for the entire semester. You can use at most 2 days for any one assignment. After four days have been used up, there will be a penalty of 10% for each late day. The submission for this assignment will be closed 2 days after the due date.**
- Please ask all questions on Piazza, not via email.

1 Spark Streaming with Twitter and Kafka

In this part, you will create a Spark Streaming application that will continuously read data from Twitter about a topic. These Twitter feeds will be analyzed for their sentiment, and then analyzed using ElasticSearch. To exchange data between these two, you will use Kafka as a broker. Everything will be done locally on your computer. Below are the steps to get started:

Setting up your development environment

You will need to perform the following steps to get your environment set up.

- You will need to create a Twitter app and get credentials. It can be done at
  https://apps.twitter.com/
- Download Apache Kafka and go through the quickstart steps:
  https://kafka.apache.org/quickstart
- Windows users might want to consider WSL, which gives a Unix-like environment on Windows:
  https://docs.microsoft.com/en-us/windows/wsl/install-win10
- I have provided a Spark Scala project that gives an example of how to connect Spark Structured Streaming to a Kafka broker. The project can be downloaded from
  http://www.utdallas.edu/~axn112530/cs6350/kafka.zip.
  The next section gives you the instructions on how to compile and run the project.
- Later, you will also need to set up Elasticsearch and Kibana environment to visualize data. You will need to download Elasticsearch, Kibana, and Logstash from
  https://www.elastic.co/downloads/

Running the Example

The example project can be downloaded from:

It illustrates a simple way to communicate between Structured Streaming and Kafka. Below are the steps:

- Make sure that you first start Zookeeper and then Apache Kafka. After ensuring that these two services are running, create two topics, which we will call topicA and topicB. After this, use the producer utility to generate some messages on topicA, and start a consumer on topicB.
- For building this project, you need to have Scala Build Tool (SBT) installed. It can be downloaded from
  https://www.scala-sbt.org
  After downloading, you should be able to go to root of your project and build by the following commands.

  ```
  sbt
  > assembly
  ```

This will create a fat jar (i.e. a jar containing all its dependencies) under **target/scala-2.11** directory.

- Run the Spark project by using the following command:

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.0
--class ClassName PathToJarFile topicA topicB
```

The above will accept data from topicA, perform word count, and display the results on the console as well as forward the message to topicB. This is accomplished by the Spark Structured Streaming application.

Starting the Project

For this project, you will need to perform the following steps:

- Create a Twitter application using Spark and Scala. An example of how to do this is available here:
  http://bahir.apache.org/docs/spark/current/spark-streaming-twitter/
- The application should perform search about a topic and gather Tweets related to it. The next step is sentiment evaluation of the Tweets. Various libraries are available for this task:
  - CoreNLP Scala Examples
  - Databricks example - Note: you can't use Databricks for this assignment.
  - Sentiment Analyis in Scala
  - Sentiment Analysis of Social Media Posts using Spark

  The sentiment evaluation should happen continuously using a stream approach. At the end of every window, a message containing the **sentiment** should be sent to Kafka through a topic.
- You then have to write
- In the next step, you will configure Logstash, Elasticsearch, and Kibana to read from the topic and set up visualization of sentiment.

Visualizing the data using Elasticsearch and Kibana

You were able to read the data using a console consumer in a previous step. Let's now move forward and visualize the data using Elastcsearch. To send data from Kafka to Elasticsearch, you need a processing pipeline that is provided by Logstash. Download Elasticsearch, Kibana, and Logstash from https://www.elastic.co/downloads.

After downloading the data, you need to go to the appropriate directories, and start the services in the following order:

1. **Elasticsearch** using the following command in the elasticsearch-5.6.8/bin directory:
   `./elasticsearch`

2. **Kibana** using the following command in the kibana-5.6.8-darwin-x86_64/bin directory:
   `./kibana`

3. **Logstash**: Go to the logstash-5.6.8 directory and create a file logstash-simple.conf with following content:

```
input {
kafka {
bootstrap_servers => "localhost:9092"
topics => ["YourTopic"]
}
}

output {
elasticsearch {
hosts => ["localhost:9200"]
index => "YourTopic-index"
}
}
```

   Then run the following command
   `bin/logstash -f logstash-simple.conf`

   This sets up the right pipeline between Kafka and Elasticsearch.

If everything is set up properly, you should be able to go to http://localhost:5601 and use Kibana to visualize your data in real-time. You will have to search for the appropriate topic index, which is **YourTopic-index** in the example shown above. You can read more about Kibana here.

Note: If you have trouble setting up all of this on your local computer, please use online sources for help with troubleshooting.

What to Submit

You are required to submit the following:

- Your project in a compressed zip format. Please delete the .jar file from the target folder as it will be quite large.
- A graphical plot showing how the sentiment for your topic varied over a time interval. You are free to choose the time interval - but it should be at least a few hours.
- A brief summary of how the sentiment varied over the time frame, and what insights does it give you.
- A README file indicating how to run your code. Please do not use any hard coded paths - all paths should be specified as arguments.

2 Tweet Processing & Classification using Pipelines

In this part, we will work with a set of Tweets about US airlines and examine their sentiment polarity. More details about thedataset can be found on the website Kaggle competition on Twitter US Airline Sentiment. You can also download the dataset from this site. Our aim is to learn to classify Tweets as either "positive", "neutral", or "negative" by using two classifiers and pipelines for pre-processing and model building.

All this has to be done in a Scala class, which has to be part of a Scala SBT project. Make sure you have all your dependencies and the class can be run on AWS. The class will have 2 parameters - one that represents the path of the input file and the second one that represents the output path where the output will be stored.

Below are the steps of the project:

1. **Loading**: First step is to define an input argument that defines the path from which to load the dataset. After that, you will need to remove rows where the *text* field is *null*.
2. **Pre-Processing**: You will start by creating a pre-processing pipeline with the following stages:
   - **Tokenizer**: Transform the *text* column by breaking down the sentence into words
   - **Stop Word Remover**: Remove stop-words from the words column
     Hint: Use the import org.apache.spark.ml.feature.StopWordsRemover class.
   - **Term Hashing**: Convert words to term-frequency vectors
     Hint: Use the import org.apache.spark.ml.feature.HashingTF class
   - **Label Conversion**: The label is a string e.g. "Positive", which you need to convert to numeric format
     Hint: Use the import org.apache.spark.ml.feature.StringIndexer class

   Remember that you need to create a pipeline of the above steps and then transform the raw input dataset to a pre-processed dataset.
3. **Model Creation** - You will need to create two classification models that you can select from the MLlib classification library. You will have to create a **ParameterGridBuilder** for parameter tuning and then use the **CrossValidator** object for finding the best model. An example of this can be seen here: `https://spark.apache.org/docs/2.2.0/api/scala/index.html#org.apache.spark.ml.tuning.CrossValidator`
4. **Model Testing & Evaluation**: Next, you will create a random sample of the dataset and apply your model on it and output classification evaluation metrics, such as accuracy, etc. You can see details of multi-class evaluation metrics at `https://spark.apache.org/docs/2.2.0/mllib-evaluation-metrics.html`.
5. **Output**: Finally, you have to write the output the classification metrics to a file whose location is specified by the second argument to the class.

Remember that you have to write your code in the form of a Scala class that should run on AWS. You can specify paths on AWS S3.