Twitter sentiment analysis using Spark and Kafka

Sentiment analysis tasks has great impacts on organizations plans to make decisions about future

In this article we will implement an Apache Spark program to make a machine learning model to analysis Twitter tweets about some topics/terms

Project Architecture

The architecture of our project is split to three parts

Machine learning model

Streaming model

The machine model is responsible for taking input from streaming model and making some prediction about incoming stream input

Streaming model is responsible for taking input from Twitter API filtered by some terms and putting them in another stream of useful data for the machine learning model

Streaming model will put its output in a Kafka topic which the machine learning model will use it to read from and then put its output in another Kafka topic

Machine Learning

To build a machine learning model we need some training data to make our model knows what it's going to do, in sentiment analysis with supervised learning we need dataset that has both features and labels. There is some open-source datasets on the internet we can use, we will use this dataset which comes from Twitter which is good for our program that's going to analyze twitter tweets

Loading Data

First start by loading data to our program, we are going to use Spark with Scala to help us loading, analyzing and build model

First, we initiate spark

```
val inputfile = "training.1600000.processed.noemoticon.csv"
val conf = new SparkConf
if (!conf.contains("spark.master"))
  conf.setMaster("local[*]")
println(s"Using Spark master '${conf.get("spark.master")}'")

val spark = SparkSession
  .builder()
  .appName("TwitterSentimentAnalsis")
  .config(conf)
  .getOrCreate()
```

Then we read our data, but first we need to know what is the schema for our data, the easiest way to do it is to open our dataset file and see some samples manually

```
"0","1467810672","Mon Apr 06 22:19:49 PDT
2009","NO_QUERY","scotthamilton","is upset that he can't
update his Facebook by texting it… and might cry as a result
School today also. Blah!"
```

we our datasets starts with a label, tweet id, then date , some string and the tweet text and our dataset doesn't start with a header row.

```
//Reading data


val input = spark.read
  .option("header","false")
  .option("delimiter",",")
  .option("inferschema",value = true)
  .csv(inputfile)
  //Rename columns

.toDF("label","tweet_id","tweet_datetime","query","user_name
","tweet_text")
  //Drop unwated columns
  .drop("tweet_id")
  .drop("tweet_datetime")
  .drop("query")
  .drop("user_name")
input.printSchema()
input.show(5)
```

The first and last column are what we need for now, so we will drop remaining columns

you can use print schema to see what we have as schema and show method to display first n rows, using these methods shows us this

```
root
 |-- result: integer (nullable = true)
 |-- tweet_text: string (nullable = true)
```

```
+------+--------------------+
|label |          tweet_text|
+------+--------------------+
|     0|@switchfoot http:...|
|     0|is upset that he ...|
|     0|@Kenichan I dived...|
|     0|my whole body fee...|
|     0|@nationwideclass ...|
+------+--------------------+
```

now we need to know more about our data, let's see what we have in result column. To do so, we will group data be result column and then count the number of each result

```
input.groupBy("label").count().show()
--------------------------------------
+------+------+
|label | count|
+------+------+
|     4|800000|
|     0|800000|
+------+------+
```

We have 800,000 positive example with label 4 and 800,000 negative example with label 0

Building model

machine learning models only deals with numeric values, but for our case the tweet text is not a numeric value, so we need a way to convert it to some numeric value that is useful for our model

The ML package needs the label and feature vector to be added as columns to the input `DataFrame` . We set up a pipeline to pass the data through transformers in order to extract the features and label.

TF-IDF (Term Frequency—Inverse Document Frequency)

One way to convert words to is to numeric value is calculate the importance of some words for some topic. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. Read more about it here

Before using TF-IDF we need to split our text to words,The `RegexTokenizer` takes an input text column and returns a `DataFrame` with an additional column of the text split into an array of words by using the provided regex pattern. The `StopWordsRemover` filters out words which should be excluded, because the words appear frequently and don't carry as much meaning – for example, 'I,' 'is,' 'the.'

In the code below, the `RegexTokenizer` will split up the column with the review and summary text into a column with an array of words, which will then be filtered by the `StopWordsRemover` :

```
//Convert text to tokens of words
val tokenizer = new RegexTokenizer()
  .setInputCol("tweet_text")
  .setOutputCol("tweet_tokens_u")
  .setPattern("\\s+|[,.()\"]")

//Remove stop words (I , a ,an the,..etc)
val remover = new StopWordsRemover()
  .setStopWords(StopWordsRemover
    .loadDefaultStopWords("english"))
  .setInputCol("tweet_tokens_u")
  .setOutputCol("tweet_tokens")
```

A `CountVectorizer` is used to convert the array of word tokens from the previous step to vectors of word token counts. The `CountVectorizer` is performing the TF part of TF-IDF feature extraction.

```
//Convert array of words tokens to to vector of words count
//CountVectorizer perform TF
val cv = new CountVectorizer()
  .setInputCol("tweet_tokens")
  .setOutputCol("cv")
  .setVocabSize(200000)
```

Below the `cv` column created by the `CountVectorizer` (the TF part of TF-IDF feature extraction) is the input for IDF. IDF takes feature vectors created from the `CountVectorizer` and down-weights features which appear frequently in a collection of texts (the IDF part of TF-IDF feature extraction). The output `features` column is the TF-IDF features vector, which the logistic regression function will use.

```
//TF-IDF
val idf = new IDF()
  .setInputCol("cv")
  .setOutputCol("features")
```

The final element in our pipeline is an estimator, a logistic regression classifier, which will train on the vector of labels and features and return a (transformer) model.

```
val lr = new LogisticRegression()
```

Before building our model , the logistic regression needs some parameters , choosing these parameters manually can be time consuming so we will build a parameters grid which will take a list of parameters for each parameters and trying a combination of all of them and get the best result

```
val paramGrid = new ParamGridBuilder()
  .addGrid(lr.regParam,Array(0.01,0.02,0.1))
  .addGrid(lr.elasticNetParam,Array(0,0.3,0.5,0.8,1))
  .addGrid(lr.maxIter,Array(10,30,50,100))
  .build()
```

Finally we build our pipeline with all these stages

```
val stages =  Array( tokenizer, remover, cv, idf,lr)
val pipeline = new Pipeline().setStages(stages)
```

To train our model we will use train/validation split model which will split our data to train set and validation set

The estimator is our pipeline we built , Evaluator since we build a binary classification model we will us *BinaryClassificationEvaluator() and put our parameters grid*

Parallelism number indicates how many models Spark will try to build in the same time (not same time typically), If you're using the program on cluster of machines it'd be better to set parallelism to some higher number but for our program we will set only for 2

```
val crossValidator = new TrainValidationSplit()
  .setEstimator(pipeline)
  .setEvaluator(new BinaryClassificationEvaluator())
  .setEstimatorParamMaps(paramGrid)
  .setTrainRatio(0.8)
  .setParallelism(2)
```

We will split our data to training data and test data for ratio 80% and 20% each

```
val seed = 1234L
val Array(trainingData,testData)
=input.randomSplit(Array(0.8,0.2),seed)
```

finally we will fit training data to our crossValidator , this part will take some time

```
val model = crossValidator.fit(trainingData)
```

We need to evaluate our model so will use test data to evaluate using ROC

```
val predictions = model.transform(testData)
val evaluator = new BinaryClassificationEvaluator()
val areaUnderROC = evaluator.evaluate(predictions)
println(s"evaluator result ${areaUnderROC}")
-------
evaluator result 0.8415681
```

Possible enhancement

Trying different parameters

Trying different models (NaiveBayes , Decision Tree, SVM , Random Forest..)

Applying different languages to model by trying another word embedding with different languages

References

Streaming Machine learning pipeline for Sentiment Analysis using Apache APIs: Kafka, Spark and Drill—Part 1