12/12/2019

Twitter Sentiment Analysis

Using Apache PySpark and Apache Kafka

Name: Kevin Rodrigues

Professor: Jack Polnar

Course: MET CS 779 Advanced Databases

ABSTRACT

It is estimated that 2.5 quintillion bytes are generated each day.(Marr, 2019) Since data that is created in such huge amounts each day. It is becoming a necessity to analyze this data on the fly.

Data streaming is becoming more popular each day. Research within this area is booming. Data storage of such streaming data and analyzing them are very important for companies to help them design marketing strategies.

Sentiment analysis of such data is one of the techniques that can be used to design marketing strategies, product improvements, etc. since sentiment analysis can help identify the public opinion about the company or the product.

The streaming data can be captured using Apache Kafka and TweePy. And this captured data can be analyzed using Apache Spark.

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

Sentiment analysis is a Natural Language Processing technique of identifying sentiments within a sentence, allowing businesses to identify their customers emotions regarding the company, social media, stocks, etc.

TweePy is a package that is used in python to obtain live tweets.
Apache spark is a open source distributed cluster computing frame work. It is mainly used to perform analysis on Big Data.

Apache Kafka is an opensource stream processing software which was built by LinkedIn and is now taken over by Apache.

All these frameworks and models can be put together to perform sentiment analysis on live streaming data. This can help companies perform sentiment analysis on the fly allowing them to know the current emotions of their customers regarding their products. In turn the company can really pay attention on strategies for marketing, improvements, etc.

TECHNOLOGIES USED

APACHE SPARK FRAMEWORK

The spark paper describes how spark is an in memory distributed system. It is similar to Hadoop but it performs computations in the systems memory. (Zaharia, Chowdhury, Franklin, Shenker, & Stoica, n.d.)

Many high-volume data sources operate in real time such as IOT, log from mobile devices, etc. Since companies have over the time evolved the process of capturing data they also now have the need for processing this data. The spark structured streaming paper discusses the process of doing so.

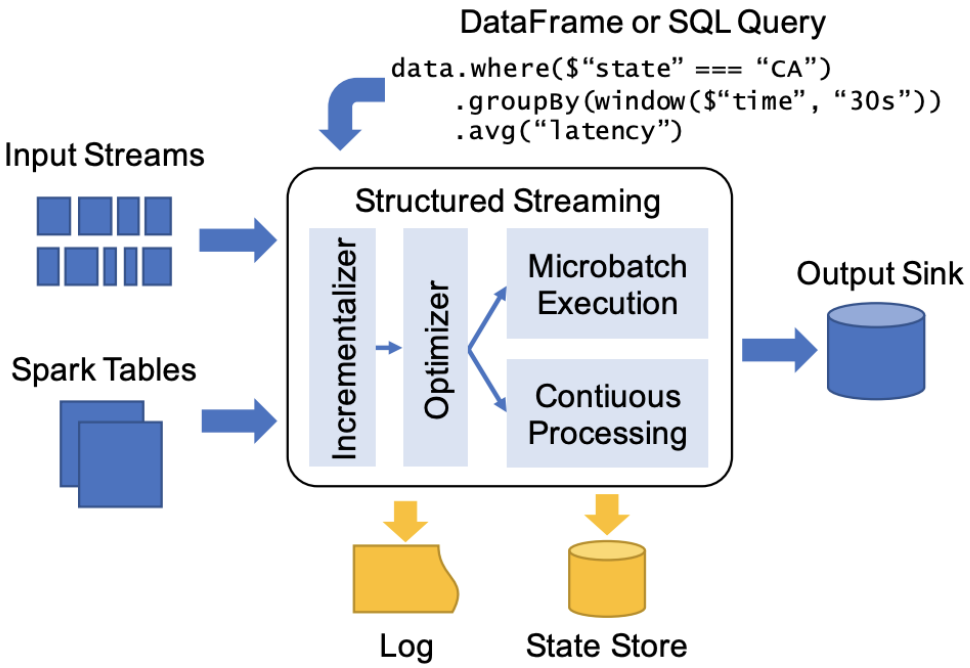

Figure 2 Spark Structured Streaming Components (Armbrust et al., 2018)

The paper discusses the architecture of structured streaming and how it is used. The structured stream uses the structured data API's defined in spark such as datasets, dataframes and SQL. It also uses the micro batch processing along with spark SQL to perform computations on the data.
The structured streaming needs a replayable input source inorder to re-read data in case if the node performing the computation crashes. Mainly it is used with a reliable message bus such as Kafka or Amazon kinesis.

The structured streaming adds to the Sparks SQL. It adds a trigger kind of function which controls how frequent the engine will try to perform computations. It also adds a feature of watermark policy along with marking a column as denoting event time to determine when to output when enough data is collected. Complex processes such as custom session-based windows can be implemented using stateful operators as they allow users to track and update mutable state by key. This is similar to spark streaming's updateStateByKey API.

The structured streaming once it receives the query optimizes it, incrementalizes it and then executes it. Structured streaming uses two kinds of durable storages. The first one is the write-ahead log which keeps track of which data has been computed and written to the output sink. The second, the system uses lager-scale state store this holds a snapshot of the states of operator for long running aggregations. On failure using this state store and the log the system can recompute the last updated state and continue from there.(Armbrust et al., 2018)

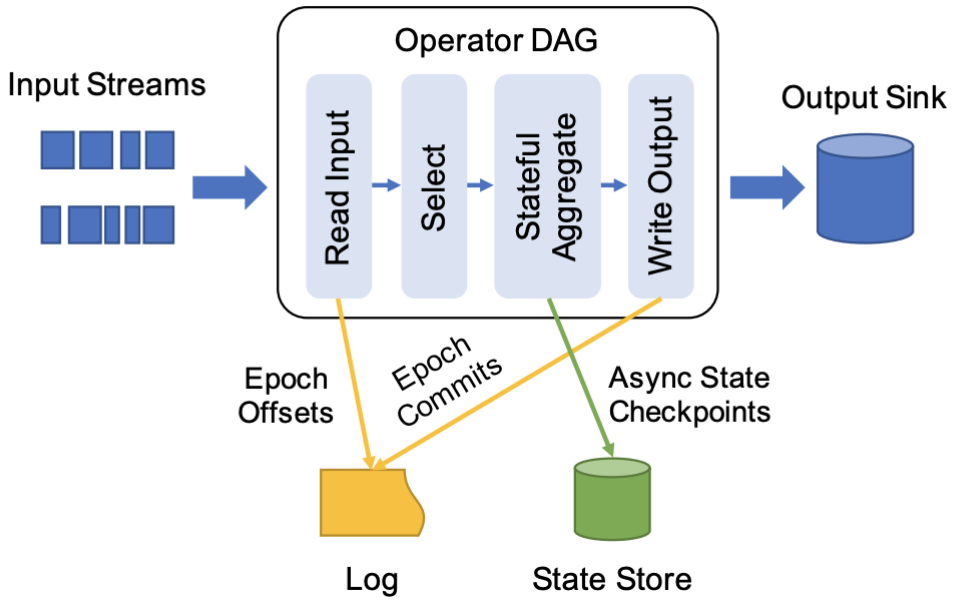

Figure 2 State Management During the Execution of Structured Streaming (Armbrust et al., 2018)

APACHE KAFKA FRAMEWORK

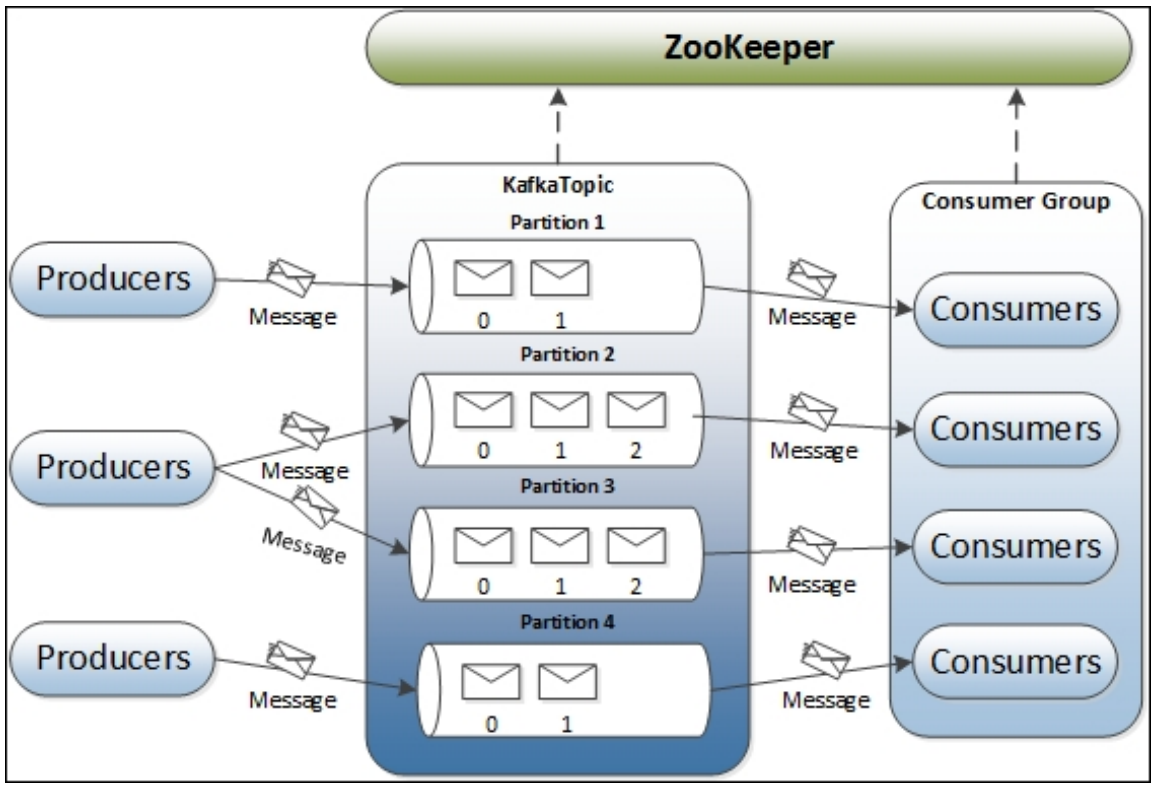

Figure 1 Kafka Ecosystem (*Kafka design fundamentals—Learning Apache Kafka—Second Edition*, 2015)

Kafka is a system that is used as a log processing system which has the benefits of various log aggregators and messaging systems. Kafka is scalable, distributed and provides high throughput and also provides an API like the traditional messaging systems.

The Kafka paper discusses various weaknesses of traditional enterprise messaging systems (such as IBM Websphere MQ) due to which they cannot be used at LinkedIn. It also discusses newly invented systems (such as Fume and Scribe).It also discusses why Pull model is more beneficial for LinkedIn than the Push model.
A stream of messages in kafka is called a topic. Multiple producers can post to a topic while multiple consumers can subscribe to a topic and consume the topic using the pull model. The paper shows an example of how the API can be used both by the producer and consumer. The paper also discusses how the topic is partitioned and stored at multiple brokers from where the consumer can pull and consume the data.

The kafka log is divided into segments and the system uses offsets instead of id's. The consumer makes asynchronous pull requests to consume messages using the offset and the number of bytes it wants to consume. It is assumed that a consumer always has consumed some amount of segments and only requires newer segments. It is explained kafka does not use in memory caching and relies on underlying file systems page cache so as to avoid the use of double buffer and to have the benefit of retaining warm cache even when a broker process is restarted. Kafka uses the LINUX/UNIX send file API so as to avoid 2 of the 4 copies needed to be done and 1 of the system call needed to be done to make the consuming process more efficient. The broker deletes messages that have stayed in the broker for longer than seven days. It is also explained as to how a consumer can rewind to a previous offset and consume data.

Kafka uses consumer groups. Wherein from each consumer group one or multiple consumers consume messages without coordination between these groups. Within a consumer group only one of the consumer is allowed to consume from a given partition. To facilitate this the Kafka uses a Zookeeper API. The zookeeper allows consumers to be notified in case there is a change. The same applies for the broker.

Kafka guarantees that a message is at least delivered once. Kafka guarantees that messages from a single partition are delivered to a consumer in order. However, there is no guarantee on the ordering of messages coming from different partitions. CRC is used to avoid log corruption. If a broker goes down, any message stored on it is not yet consumed becomes unavailable. If the storage system on a broker is permanently damaged, any unconsumed message is lost forever. (Kreps, Narkhede, & Rao, n.d.)

TWEEPY

TweePy is an API which you need to use in twitter to access data such as tweets, retweets, followers, DM's, etc. Tweepy gives you access to twitters API. First you need to create a twitter developer account and obtain the consumer_key, consumer_secret, access_token, access_secret. These credentials can be passed to the Twitter API using the Oauth handeler.

WORKFLOW

STREREAMING DATA PIPELINE

The following is the workflow of the Streaming Data Pipeline:

1. Obtain live twitter data for the Twitter API using Tweepy and Oauth
2. Preprocess the incoming tweets to remove the mentions, html links, etc.
3. Send preprocessed tweets to Kafka server as messages using Kafka producer
4. Access the messages from the Kafka server using spark structured streaming
5. Pass the structured streaming dataframe to the PySpark Model Pipeline.
6. Output the count of total positive and total negative tweets after every batch processing.
7.

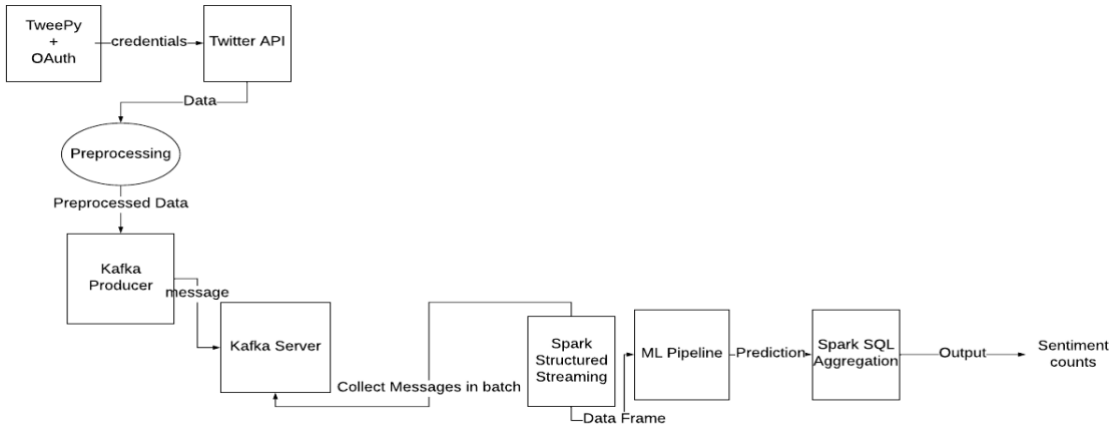

Figure 2 Kafka and Spark Pipeline WorkFlow

BUILDING A MACHINE LEARNING MODEL

The following are the steps for building a machine learning model in PySpark that can be used by the Streaming Data Pipeline.

1. Obtain the labelled dataset (I used the sentiment140 dataset that uses tweets which are labelled 0 for negative and 4 for positive)
2. Perform preprocessing on it to remove the mentions, html links, etc.
3. Use the PySpark Spark.ML Library to build the model.
4. Tokenize each sentence to split the tweets into a vector of words.
5. Perform ngrams on the tweets which means create feature columns with sequence of 1 word, 2 words and 3 words.
6. Perform TF-IDF of these ngrams using Count Vectorizor and IDF.
7. VectorSize hint is used to provide a hint regarding the size of vectors of the dataframes. This is important for streaming data since streaming data is dynamic and spark wont be able to compute the sizes of the vectors.
8. VectorAssembler then combines all necessary features into on feature column which will be used by the Logistic Regression Model to train along with the Labels column.

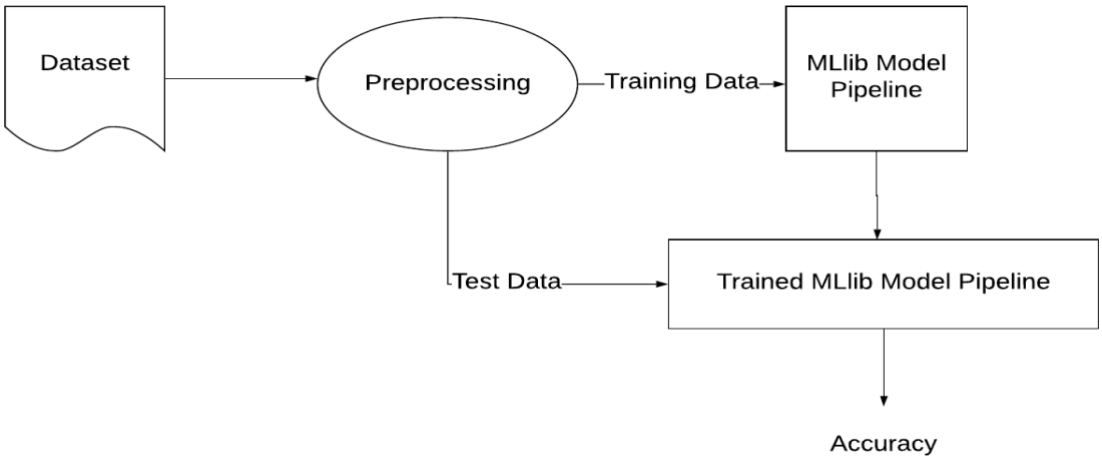

Figure 3 workflow for building Spark.ML Model Pipeline

STREAMING DATA PIPELINE

ACCESSING TWITTER API

The tweepy package along with OAuth gives us access to the Twitter API.

```
consumer_key = 'FsR9pvv2eZWqDtOexzrbPfw7i'
consumer_secret = '4p7uu6yQuIUxnbH5QyKXCLON1S3V7MkY5jIrXA3bMQ6Hl6Y4XY'
access_key = '120464862-8rG8LqZa1DeUw0sAXvp6MBGEfcAYtmjCPpkugIcu'
access_secret = 'aEmEWzq4cVZys7C3pUUJe6SUpKSP0QSxFge9eG9xQ717D'

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)
api = tweepy.API(auth)

stream = tweepy.Stream(auth, listener = KafkaStreamListener(api))


stream.filter(locations=[-180,-90,180,90], languages = ['en'])
```

Figure 4 TweePy Example

After this the data can be preprocessed which will be discussed when we discuss how the Spark.ML pipeline was built.("Working with Streaming Twitter Data Using Kafka – BMC Blogs," n.d.)

SENDING TWEETS TO KAFKA SERVER

The Tweepy stream accepts a listener class which has the code to be executed on the incoming tweets. This listener can now call the kafka producer. This kafka producer has a function to send messages to the server. This message is first preprocessed as required by the Spark.ML model. This message is converted to json format and encoded before pushing to the server.("Working with Streaming Twitter Data Using Kafka – BMC Blogs," n.d.)

```
class KafkaStreamListener(tweepy.StreamListener):

    def __init__(self, api):
        self.api = api
        super(tweepy.StreamListener, self).__init__()

        self.producer = kafka.KafkaProducer(bootstrap_servers="localhost:9092")

    def on_status(self, status):

        msg = tweet_preprocessing(status.text)


        try:
            self.producer.send("test",json.dumps({'tweet': msg, "Target": 0}).encode('UTF-8','ignore'))

        except Exception as e:
            print(e)
            return False
        return True


    def on_error(self, status_code):
        print("Error received in kafka producer")
        return True

    def on_timeout(self):
        return True
```

Figure 5 Sending Message to Kafka Sever using Kafka Producer

ACCESSING THE KAFKA SERVER MESSAGES USING SPARK STRUCTURED STREAMING

The spark structured streaming can be called initialized with the kafka server ip and port number. Now the spark streaming will access this data in batches and provide it as a dataset. ("Structured Streaming + Kafka Integration Guide (Kafka broker version 0.10.0 or higher)—Spark 2.4.4 Documentation," n.d.)

```
spark = SparkSession \
    .builder \
    .appName("StructuredNetworkWordCount") \
    .getOrCreate()



df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("maxFilesPerTrigger", 1)\
    .option("subscribe", "test") \
    .load()
```

Figure 6 Reading Messages from Kafka Sever Using Structured Streaming

Now this dataset can be converted to a dataframe table by providing the dataset a schema.("Spark Streaming—Consume & Produce Kafka message in JSON format," 2019) I have provided it a schema of Tweets, Target. The target is a place holder column since this is the required structure for the Spark.ML pipeline model that I have built.

```
Schema = StructType().add("tweet", StringType()).add("Target", IntegerType())

df1 = df.select(F.col("value")\
    .cast("string")\
    .alias("json"))\
    .select(from_json(F\
    .col("json"), Schema)\
    .alias("data"))

df2 = df1.select("data.*")
```

Figure 7 Coverting Structured Stream Dataset to Dataframe

```
model = PipelineModel.load("pipe1/")

df3 = model.transform(df2)
```

Figure 8 Using Pyspark ML Model with Structured Streaming

COUNTING THE TOTAL POSITIVE AND NEGATIVE TWEETS USING SPARK SQL

The Spark SQL allows you to query the data frame. We can write an aggregation query using Spark SQL to get the total counts of the positive and negative tweets prediction.

```
df3 = df3.select("prediction").groupBy("prediction").count()
```

Figure 9 Aggregation Query on Dataframe

The Spark structured dataframe accesses the Kafka server and obtains the tweets from it in batches and appends it to the dataframe. Hence the count that we see is the total count of all the tweets that the structured stream has received over the period of time.

Now this Spark SQL creates a new data frame which has the predictions and the current counts. This new dataframe can now be written to the sink such as console so that we can see the counts.

```
df3.writeStream \
    .outputMode("complete")  \
    .format("console") \
    .option("truncate", "false") \
    .start() \
    .awaitTermination()
```

Figure 10 Writing Output to Console Sink

```
-------------------------------------------
Batch: 8
-------------------------------------------
+----------+-----+
|prediction|count|
+----------+-----+
|0.0       |168  |
|1.0       |464  |
+----------+-----+
```

Figure 11 Prediction Counts as Output

BUILDING THE SPARK.ML PIPELINE MODEL

For sentiment analysis we need to build a model. Sentiment analysis is a supervised learning problem. Hence we require a opensource dataset that is labelled.

DATASET

I have used the sentiment140 dataset("For Academics—Sentiment140—A Twitter Sentiment Analysis Tool," n.d.) which is a opensource dataset that was collected by Stanford University for sentiment analysis.

The structure of this dataset is as follows.

	Target	id	date	query	user	tweet
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

Figure 12 Sentiment140 Dataset

PREPROCESSING

This sentiment140 dataset has tweets that are unclean. They posses references to other url links, other users, etc.. All these links will decrease the accuracy of the model. Hence it is important that we remove such mentions, links from the tweets.
The data cleaning follows the following steps(Kim, 2018a):

1. Firstly we remove all the mentions of other users using regex
2. We need to convert url links to normal text using beautiful soup.
3. Now we remove the Byte Order Mark(these are UTF-8 codes that help identify that the text is encoded as UTF-8)
4. Now we can remove the https links as well as the www links.
5. Converting the tweets to lower case
6. Now we carry out negation handling using the dictionary we built.
7. Remove all numbers and special characters
8. Tokenize all the words and join them again

```
token = WordPunctTokenizer()
patern1 = r'@[A-Za-z0-9]+'
patern2 = r'https?://[A-Za-z0-9./]+'
combinedpatern = r'|'.join((patern1, patern2))
www_patern = r'www.[^ ]+'
negativedic = {"isn't":"is not", "aren't":"are not", "wasn't":"was not", "weren't":"were not",
               "haven't":"have not","hasn't":"has not","hadn't":"had not","won't":"will not",
               "wouldn't":"would not", "don't":"do not", "doesn't":"does not","didn't":"did not",
               "can't":"can not","couldn't":"could not","shouldn't":"should not","mightn't":"might not",
               "mustn't":"must not"}
negativepatern = re.compile(r'\b(' + '|'.join(negativedic.keys()) + r')\b')


def tweet_preprocessing(tweet):
    soup = BeautifulSoup(tweet, 'lxml')
    soup = soup.get_text()
    try:
        clean = soup.decode("utf-8-sig").replace(u"\ufffd", "?")
    except:
        clean = soup
    rip = re.sub(combinedpatern, '', clean)
    rip = re.sub(www_patern, '', rip)
    lower = rip.lower()
    neg = negativepatern.sub(lambda x: negativedic[x.group()], lower)
    alpha= re.sub("[^a-zA-Z]", " ", neg)
    w = [w for w in token.tokenize(alpha) if len(w) > 1]
    return (" ".join(w)).strip()
```

Figure 13 Data Preprocessing Function

Now the preprocessed data looks like:

	tweet	Target
0	awww that bummer you shoulda got david carr of...	0
1	is upset that he can not update his facebook b...	0
2	dived many times for the ball managed to save ...	0
3	my whole body feels itchy and like its on fire	0
4	no it not behaving at all mad why am here beca...	0

Figure 14 Preprocessed Data

MACHINE LEARNING MODEL PIPELINE

There are various models that can be used to do sentiment analysis. I choose to apply Tokenization + N-grams +TF-IDF + Logistic Regression.
Spark has the Spark.ML library which can be used to train machine learning models. The Pipeline library within spark allows you to create a queue process using various processes from the machine learning library.
I created a pipeline which consists of the following steps.(Kim, 2018b)

1. **Tokenization** means creating a vector of individual words for a given sentence(“Tokenization,” n.d.)
2. ***N*-gram** is a continuous sequence of *n* words within a sentence.(“*N*-gram,” 2019)
3. **Term Frequency–Inverse Document Frequency of n-grams**, is a statistical number defining importance of a word or a sequence of words existing within a set of multiple documents.(Bornstein, 2019)
4. **Linear Regression** is a classifier that predicts a label using multiple features**.**(“The Regression Line | Boundless Statistics,” n.d.)

```
from pyspark.ml.feature import NGram, VectorAssembler,VectorSizeHint

def build_model(inputCol=["tweet","Target"], n=3):
    tokenizer = [Tokenizer(inputCol="tweet", outputCol="words")]
    ngrams = [
        NGram(n=i, inputCol="words", outputCol="{0}_grams".format(i))
        for i in range(1, n + 1)
    ]

    cv = [
        CountVectorizer(vocabSize=5460,inputCol="{0}_grams".format(i),
            outputCol="{0}_tf".format(i))
        for i in range(1, n + 1)
    ]
    idf = [IDF(inputCol="{0}_tf".format(i), outputCol="{0}_tfidf".format(i), minDocFreq=5) for i in range(1, n + 1)]
    hint1 = [VectorSizeHint(inputCol="1_tfidf",size=5460)]
    hint2 = [VectorSizeHint(inputCol="2_tfidf",size=5460)]
    hint3 = [VectorSizeHint(inputCol="3_tfidf",size=5460)]

    assembler = [VectorAssembler(
        inputCols=["{0}_tfidf".format(i) for i in range(1, n + 1)],
        outputCol="features"
    )]
    label_stringIdx = [StringIndexer(inputCol = "Target", outputCol = "label")]
    lr = [LogisticRegression(maxIter=100)]
    return Pipeline(stages=tokenizer + ngrams + cv + idf+ hint1 + hint2 + hint3 +  assembler + label_stringIdx + lr )
```

Figure 15 Sparl.ML Pipeline Model

Now this model can be saved and used along with the Kafka pipeline.

CONCLUSION

1. Spark goes hand in hand with Kafka streaming for Big Data problems, since Spark provides speedy processing of huge data.
2. Data must be stored after performing preprocessing. Analysis on unclean data may deviate the accuracy.
3. Kafka is a great tool for on the fly analysis of data.
4. Such pipelines can be used by companies to strategize their marketing, product updates, etc.
5. Structured streaming is a great tool in spark to access Kafka messages due to its better capabilities over Discrete Streaming (DStream).
6. Structured streaming uses Spark SQL to Query the data frames.
7. Structured streaming allows aggregation operations on the dataframes.
8. Structured streaming uses micro batch processing to access data.

REFERENCES

1. Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., … Zaharia, M. (2018). Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. *Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18*, 601–613. https://doi.org/10.1145/3183713.3190664
2. For Academics—Sentiment140—A Twitter Sentiment Analysis Tool. (n.d.). Retrieved December 15, 2019, from http://help.sentiment140.com/for-students
3. *Kafka design fundamentals—Learning Apache Kafka—Second Edition*. (2015). Retrieved from

https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781784393090/3/ch03lvl1sec18/kafka-design-fundamentals

4. Kim, R. (2018a, January 10). Another Twitter sentiment analysis with Python-Part 2. Retrieved December 14, 2019, from Medium website: https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-2-333514854913
5. Kim, R. (2018b, March 14). Sentiment Analysis with PySpark. Retrieved December 14, 2019, from Medium website: https://towardsdatascience.com/sentiment-analysis-with-pyspark-bc8e83f80c35
6. Kreps, J., Narkhede, N., & Rao, J. (n.d.). *Kafka: A Distributed Messaging System for Log Processing*. 7.
7. Marr, B. (2019, December 15). How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read. Retrieved December 14, 2019, from Forbes website: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/
8. Packt Subscription | More Tech, More Choice, More Value. (n.d.-a). Retrieved December 15, 2019, from https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781784393090/3/ch03lvl1sec18/kafka-design-fundamentals
9. Packt Subscription | More Tech, More Choice, More Value. (n.d.-b). Retrieved December 15, 2019, from https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781784393090/3/ch03lvl1sec18/kafka-design-fundamentals

10. Spark Streaming—Consume & Produce Kafka message in JSON format. (2019, March 17). Retrieved December 15, 2019, from Spark by {Examples} website: https://sparkbyexamples.com/spark/spark-streaming-kafka-consumer-example-in-json-format/
11. Structured Streaming + Kafka Integration Guide (Kafka broker version 0.10.0 or higher)—Spark 2.4.4 Documentation. (n.d.). Retrieved December 15, 2019, from https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html
12. Working with Streaming Twitter Data Using Kafka – BMC Blogs. (n.d.). Retrieved December 14, 2019, from https://www.bmc.com/blogs/working-streaming-twitter-data-using-kafka/
13. Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (n.d.). *Spark: Cluster Computing with Working Sets*. 7.
14. Beyond Word Embeddings Part 2—Towards Data Science. (n.d.). Retrieved December 15, 2019, from https://towardsdatascience.com/beyond-word-embeddings-part-2-word-vectors-nlp-modeling-from-bow-to-bert-4ebd4711d0ec
15. *N*-gram. (2019). In *Wikipedia*. Retrieved from https://en.wikipedia.org/w/index.php?title=N-gram&oldid=927473508
16. The Regression Line | Boundless Statistics. (n.d.). Retrieved December 15, 2019, from https://courses.lumenlearning.com/boundless-statistics/chapter/the-regression-line/
17. Tokenization. (n.d.). Retrieved December 15, 2019, from https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html