NATIONAL AND KAPODESTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSC THESIS

Just-in-time Sentiment Analysis for Multilingual Streams

Ioanna N. Karageorgou

Supervisor: **Alexis Delis,** Professor NKUA

ATHENS

JUNE 2020

ΕΘΝΙΚΟ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ

Just-in-time Sentiment Analysis for Multilingual Streams

Ιωάννα Ν. Καραγεώργου

Επιβλέπων: **Αλέξης Δελής**, Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2020

BSC THESIS

Just-in-time Sentiment Analysis for Multilingual Streams

Ioanna N. Karageorgou
S.N.: 1115201600057

SUPERVISOR: **Alexis Delis,** Professor NKUA

ΠΤΥΧΙΑΚΗ

Just-in-time Sentiment Analysis for Multilingual Streams

Ιωάννα Ν. Καραγεώργου
Α.Μ.: 1115201600057

ΕΠΙΒΛΕΠΩΝ: **Αλέξης Δελής**, Καθηγητής ΕΚΠΑ

ΠΕΡΙΛΗΨΗ

Στις μέρες μας οι πλατφόρμες κοινωνικής δικτύωσης έχουν σημειώσει τεράστια ανάπτυξη τόσο στο πλήθος των χρηστών που έχουν καταφέρει να προσελκύσουν όσο και στον όγκο των δεδομένων που παράγουν. Όλο και περισσότεροι άνθρωποι τείνουν να εκφράζουν ελεύθερα τις αντιλήψεις και τα συναισθήματά τους σε πλατφόρμες όπως το Twitter, καθιστώντας τες κυρίαρχα μέσα έγκαιρης ενημέρωσης. Για το λόγο αυτό, η ανάλυση συναισθήματος σε τέτοιες πλατφόρμες αποτελεί όργανο συλλογής μαζικών τάσεων ενώ παράλληλα δημιουργεί προκλήσεις για το χειρσμό του όγκου των πληροφοριών. Στην παρούσα πτυχιακή εργασία παρουσιάζουμε ένα μοντέλο ανάλυσης συναισθήματος βασισμένο στο *Apache Spark* για πολύγλωσσα δεδομένα πραγματικού χρόνου. Πιο συγκεκριμένα το σύστημά μας: i) χρησιμοποιεί την βιβλιοθήκη μηχανικής μάθησης του `Spark` με σκοπό να κατηγοριοποιήσει tweets στα Ελληνικά, Γαλλικά και Αγγλικά σε αμελητέο χρόνο, ii) διαχειρίζεται προσεγμένα τη ροή των δεδομένων χρησιμοποιώντας σύχρονες ουρές δρομολόγησης μηνυμάτων, και iii) αποφαίνεται με υψηλή ακρίβεια για το αν ένα tweet προέρχεται από αληθινό λογαριασμό.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ανάλυση συναισθήματος σε πραγματικό χρόνο μέσω Spark πάνω σε πολύγλωσσα δεδομένα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: apache spark, spark streaming, μηχανική μάθηση, ανάλυση συναισθήματος σε πραγματικό χρόνο, πολύγλωσσα δεδομένα

ABSTRACT

The growth of social-media platforms has been remarkable in terms of both number of users and volume of content generated. As citizens tend to freely express their sentiments on social platforms, Twitter has inherently become an indispensable source for the public discourse in a wide variety of topics. Carrying out sentiment analysis on a timely manner on streamed tweets is undoubtedly a demanding endeavor. In this thesis, we propose a Spark-based Twitter sentiment analysis software architecture that receives online multilingual streamed messages and compiles analytics. We outline the main elements of our proposal and discuss how they collectively help address the challenges involved in this big-data processing task. In particular, our framework: i) exploits the `Spark` machine-learning library to classify Greek, French and English tweets in a timely-manner, ii) manages streamed tweets in synergy with contemporary queuing and in-memory data systems, and iii) determines with high accuracy whether a sentiment is expressed by a genuine account.

SUBJECT AREA: Just-in-time Sentiment Analysis for Multilingual Streams

KEYWORDS: apache spark, spark streaming, machine learning, multilingual data

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Mr. Alexis Delis and Mr. Panagiotis Liakos for their excellent guidance and assistance in the preparation of this thesis.

CONTENTS

1. INTRODUCTION 8

2. SPARK COMPONENTS 10

2.1 What is Spark 10

2.2 Spark Core 10

2.3 Resilient Distributed Datasets (RDDs) 10

2.4 Spark SQL 11

2.5 Spark Machine Learning 11

2.5.1 Spark Machine Learning Pipeline 12

2.6 Spark Streaming 12

2.7 Spark GraphX 13

3. PROPOSED SYSTEM ARCHITECTURE 14

3.1 Fake Account Detection Service 16

3.2 Language Identification Service 17

4. EVALUATION 20

4.1 Assessing the SA-Models for the 3 Languages 20

4.2 Assessing Language Identification Model 21

4.3 Effectiveness In Detecting Fake Accounts 22

4.4 Monitoring Twitter’s Live-Feed 23

5. CONCLUSION 25

ABBREVIATIONS, ACRONYMS 26

REFERENCES 27

FIGURES LIST

Figure 1: Spark Ecosystem 10

Figure 2: Overall Workflow of the Suggested Architecture 14

Figure 3: Architectural Elements for multilingual Twitter SA 15

Figure 4: Structured Streaming with Kafka 16

Figure 5: Excerpt from the Multilingual Dataset 18

Figure 6: Prediction of with hashtags: #covid19 and #COVID19greece. 23

Figure 7: Prediction for tweets with hashtags: #trump and #presidenttrump. . . 24

1. INTRODUCTION

Nowadays, we live in a modern society in which the Internet has become an integral part of our lives. People of different ages and expertise fields use it for both personal and professional reasons. With millions of users accessing the ”Net” every hour, web services have evolved to be more reliable and higly available. Internet companies are growing at a massive scale and there is a deluge of data being collected everywhere. The complexity of data has also increased, requiring speed and sophisticated processing methods. Conventional databases cannot handle these large datasets and individual machines, with their current I/O and processing capabilities, do not support the efficient processing of aforementioned datasets. This is where Big Data has appeared[4].

Big Data analysis is a process of gathering data from different resources, organizing them in a meaningful way and then analyzing those sets of data to discover substantial and potentially interesting facts. Obtaining a deeper understanding of the trends in the public discourse taking shape in media such as the Twitter is now of paramount importance in numerous fields of financial activity. These areas entail accurately predicting sales, understanding developing social phenomena, ascertaining stock valuation swings, and even forecasting polls and electoral campaigns. In all such activities, individuals or organizations seek to predominantly establish a clear understanding of the expressed opinions or sentiments at large. In this process which has become collectively known as *Sentiment Analysis* (SA), there are a few key factors that have to be considered, namely:

1. *just-in-time* compilation of aggregations of opinions expressed on a specific topic,
2. management of the *voluminous* content generated by potentially large (i.e., tens of thousands or even millions of users) on a particular issue,
3. effective handling of the abbreviated and occasionally mal-formed statements expressed in the textual part of tweets.

A key requirement in successfully addressing the above factors and produce reliable Twitter-Sentiment Analyses would be to create data-systems mostly based on contemporary and off-the-shelf software components such as `Apache Spark`.

In 2009 `Spark` was created at the Univ. of California, Berkeley and is a lightning-fast cluster computing technology, designed for fast computation. Since inception, `Spark` was optimized to run in memory, and so increased its processing speed of data and overtaking its competitors like `MapReduce` [2]. It extends the `MapReduce` model to efficiently use it for more types of computations, which include interactive queries and stream processing. `Spark` provides built-in APIs in `Java`, `Scala`, or `Python`, being flexible in developing applications in different languages. `Spark` also supports SQL queries, Streaming processing, Machine learning (ML), and graph algorithms.

In this thesis, we investigate the Twitter-sphere in 3 European Languages while considering streamed live tweets as they arrive from the platform’s API. Our main objective is to propose a software architecture that is flexible, efficient, and presents features that both enable the timely processing of SA-analyses and addresses our 3 above-mentioned must-have factors. Our core requirements for the realization of our service platform are to: i) develop highly-accurate machine-learning (ML) models that help reliably process and classify incoming tweets from all 3 languages, ii) deploy a bot-net functionality that can prevent posts originated from illegitimate accounts to be further considered for SA, iii) make extensive use of the `MLlib` library to classify in a timely manner our multilingual streamed input, and iv) take advantage of the scale-out that the multi-queue system `RabbitMQ` can offer so as to facilitate timely delivery of tweets for processing.

The rest of the thesis is organized as follows: Section 2 has a brief introduction to Spark and its components. A detailed description of our implementation is found in Section 3. In Section 4, we examine our implementation's performance. Finally, some general conclusions and comments about our sentiment analysis model are presented in Section 5.

2. SPARK COMPONENTS

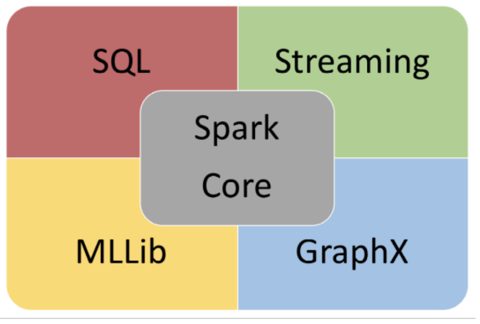
2.1 What is Spark

”Spark is a general-purpose distributed data processing engine”. This is where Spark’s empowerment lies. The term ”general-purpose” refers to the variety of domains that it has an application on. It supports Scala, Python, Java, R, and SQL. Apache Spark system consists of several main components including Spark Core, Spark’s MLlib for machine learning, GraphX for graph analysis, Spark Streaming for stream processing and Spark SQL for structured data processing. The last four elements are provided by upper-level libraries.

When datasets get too large, or when new data comes in too fast, it can become too much for a single computer to handle. Instead of trying to process a huge dataset on one computer, these tasks can be divided between multiple computers that communicate with each other to produce an output. Spark is designed to deal with very large datasets and to process them on distributed mode on a cluster. A cluster is the collection of multiple individual computers, known as nodes. Spark is deployed as a cluster consisting of a master server and potentially many worker servers. The master server accepts incoming jobs and breaks them down into smaller tasks that can be handled by workers.

2.2 Spark Core

Spark Core is the fundamental unit of the whole Spark project. It provides diverse functionalities including task dispatching, scheduling, and input-output operations. Spark Core is embedded with a special collection called RDD (Resilient Distributed Datasets). RDD is among the abstractions of Spark that handles partitioning data across all the nodes in a cluster.



Spark Ecosystem

Figure 1: Spark Ecosystem

Spark Core is a distributed execution engine with all the functionality attached atop. All the basic functionality of Apache Spark like in-memory computation, fault tolerance, memory management, monitoring, task scheduling is provided by Spark Core.

2.3 Resilient Distributed Datasets (RDDs)

The foundation of all high-level Spark objects is the concept of RDD. Spark increases processing efficiency by implementing Resilient Distributed Datasets (RDDs), which are immutable, fault-tolerant collections of objects. RDDs are essentially datasets, which are distributed across the cluster and are manipulated in parallel. They are fault-tolerant with the help of the lineage graph, called DAG, and so they can recompute missing or damaged partitions due to node failures. By default, when an RDD is created, it cannot be changed.

It can, however, be transformed into a new `RDD` by performing a set of transformations on it.

Two major operation types can be performed on an `RDD`. More specifically:

- Transformations: are operations that create a new dataset, as `RDDs` are immutable. They are used to transform data from one form to another, which could result in expansion/reduction of the data volume or a totally different shape altogether. These operations do not return any value back to the driver program, and hence lazily evaluated, which is one of the main benefits of Spark. The two most basic type of transformations are `map()` and `filter()`.
- Actions: are `RDD` operations that produce non-`RDD` values. They materialize a value in a Spark program and trigger execution of `RDD` transformations to return values. In other words, an action evaluates the `RDD` lineage graph. Some of Spark's actions are: `count()`,`collect()`,`reduce()` etc.

2.4 Spark SQL

`Apache Spark SQL` is the module for structured data processing in `Spark`. Using the interface provided by `Spark SQL`, we can get more information about the structure of the data and the computation performed, achieving additional optimization. In other words, `Spark SQL` is a `Spark` module to simplify working with structured data using `DataFrame` and `DataSet` abstractions in `Python`, `Java`, and `Scala`.

To be more specific, `DataFrame` is a distributed collection of data, ordered into named columns. They have all the features of `RDDs` but also have a schema. It is similar to a table in a relational database, but under the hood, it has much richer optimizations. Like `RDD`, `DataFrame` offers two type of operations: transformations and actions.

`DataSets` are similar to `DataFrames` but are strongly-typed, meaning that the type is specified upon the creation of the `DataSet` and is not inferred from the type of records stored in it. For instance, `DataSets` are not used in `PySpark` because `Python` is a dynamically-typed language. Similar to a `DataFrame`, the data in a `Dataset` is mapped to a defined schema. It is more about type safety and is object-oriented. A `Dataset` has helpers called encoders, which are smart and efficient encoding utilities that convert data inside each user-defined object into a compact binary format. This translates into a reduction of memory usage if and when a `Dataset` is cached in memory as well as a reduction in the number of bytes that Spark needs to transfer over a network during the shuffling process.

2.5 Spark Machine Learning

Machine Learning helps to train systems to automatically learn and improve with experience. Standard implementations of machine learning algorithms require very powerful machines and using distributed computing engines we accomplish both a much faster learning phase and better model creation.

`Apache Spark ML` is a machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, and underlying optimization primitives. `Spark`'s `MLlib` is divided into two main packages: `spark.mllib` and `spark.ml`. While `spark.mllib` is built on top of `RDDs`, `spark.ml` is built on top of `DataFrames`. Both packages come with a variety of common machine learning tasks such as featurization, transformations, model training, model evaluation and optimization. `spark.ml` provides the pipelines API for building, debugging and tuning machine learning pipelines, whereas `spark.mllib` includes packages for linear algebra, statistics and other basic utilities for machine learning.

2.5.1 Spark Machine Learning Pipeline

Typically when running machine learning algorithms, the procedure employs involves a sequence of tasks including pre-processing, feature extraction, model fitting, and validation. For example, when classifying text documents might involve text segmentation , data cleaning and feature extraction. Most ML libraries are not designed for distributed computation or they do not provide native support for pipeline creation and tuning. The `ML Pipelines` is a High-Level API for `MLlib` under the `spark.ml` package. A pipeline consists of a sequence of stages. There are two basic types of pipeline stages:

- `Transformer` - A `Transformer` in Spark essentially takes a `DataFrame` as its input and generates a new `DataFrame`, usually by appending new columns to the original `DataFrame` as a result of read and map operations. For example, in a learning model, the input `DataFrame` may contain a column containing the `Feature Vectors`. The `Transformer` may then read this column and predict the outcome (label) for each `Feature Vector`, generating a new column containing the predicted labels.
- `Estimator` - An `Estimator` is essentially a learning algorithm that trains on data producing a model - where the model is a `Transformer`. For example the `Spark Estimator` *Naive Bayes* may be called to train on a training dataset, generating a `NaiveBayesModel` which is the resultant model and `Transformer`.

A `Pipeline` in `Spark` is an ordered series of stages where each stage is either a `Transformer` or an `Estimator`.

2.6 Spark Streaming

Data Streaming is a technique for transferring data between systems so that it can be processed as a steady and continuous stream. Spark's streaming processing system offers a scalable, high-throughput, fault-tolerant service on live data. `Apache Spark` provides us with two ways of working with streaming data:

- `Spark Streaming`
- `Structured Streaming`

`Spark Streaming` is a separate library in Spark to process continuously flowing streaming data. The basic programming abstraction in `Spark Streaming` is `Discretized Streams (DStreams)`. A `DStream` is a stream of data, chopped into series of `RDDs`. Most traditional stream processing systems are designed to process records one at a time. `Spark Streaming` uses a micro-batch architecture where a stream is treated as a sequence of small batches of data and the streaming computation is done through a continuous series of batch computations on these batches of data.

From the Spark 2.x release onwards, `Structured Streaming` came into the picture. We can imagine it as an unbounded table, which practically means that it hasn't got a fixed size but the table is growing with new incoming data. This model of streaming is based on `Dataframe` and `Dataset` APIs. Hence, with this library, we can easily apply any `SQL` query or `Scala` operations on streaming data.

2.7 Spark GraphX

Lastly, `Apache Spark` has a special API for for graphs and graph-parallel computation. `GraphX` includes a growing collection of graph algorithms and builders to simplify graph analytics tasks. The usage of graphs can be seen in Facebook's friends, LinkedIn's connections, internet's routers, relationships between galaxies and stars in astrophysics and Google's Maps. The applications of graphs are limitless in use cases such as disaster detection, banking, stock market, banking and geographical systems [6].

Extending the `RDD` API, with a `Resilient Distributed Property Graph`, `GraphX` offers an efficient abstraction for representing graph-oriented data. In essence, the property graph is a directed multigraph, called `RDG (Resilient Distributed Graph)`, which has multiple edges in parallel. Here, every vertex and edge have user-defined properties associated with it. Moreover, parallel edges allow multiple relationships between the same vertices. `GraphX` unifies data extraction, transformation and loading, exploratory analysis and iterative graph computation within a single system.

3. PROPOSED SYSTEM ARCHITECTURE

We aspired to realize a software architecture that carries out SA in near real-time while deploying on an aggregation of contemporary and off-the-shelve components. Our key constraints have been that processing multilingual posts in diverse topics should occur in near real-time and the collective outcome of the analyses performed should display high precision and reflect the discourse created by users reliably. Just-in-time processing of tweets is predominantly concerned with the effective management of snippet processing within their expected-deadline [8], successfully addressing overheads ensued [7, 9], and attaining low–latency via *easy-pass-through* of data frames in a pipelined data systems [10]. Fig. 2 depicts the workflow that our suggested architecture handles based on data-pipelines.

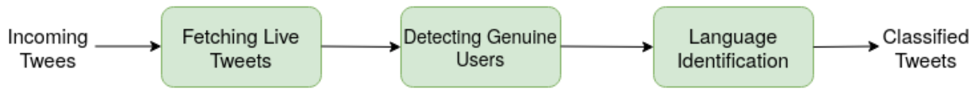


Figure 2: Overall Workflow of the Suggested Architecture

Our tweets collection occurs the time instance messages are uploaded by users via the `Twitter Streaming API`. At this stage, a fundamental challenge is to ascertain whether the initiator of the snippet is a legitimate user. As in any social networking environment, tweets does suffer from accounts that cannot be traced to real persons and so, they are deemed as fake accounts. The goal of such accounts is to manipulate public opinion about statements made or public figures, to exert influence for brands and services and in general to exercise unethical initiatives. To address this issue, we created an ML *fake detection service* with the intention to further improve the overall precision of our SA analysis. Once through the above check, snippets are directed to the ML-model we have created for Greek based on `Spark`'s `MLlib` so as to achieve sentiment prediction as either positive or negative.

Fig. 3 depicts the various components that our approach entails. To access tweets in real-time, we use the `Python3` library `Tweepy`[1], an open-source, `gitHub`-hosted project that allows for the communication with the tweets's API. This Streaming API allows us to retrieve snippets complying with stated criteria of interest and requires a persistent *HTTP*-connection and user authentication through the `OAuth` protocol. The latter helps users access the platform's API without sharing their credentials.

[1]`https://github.com/tweepy/tweepy`

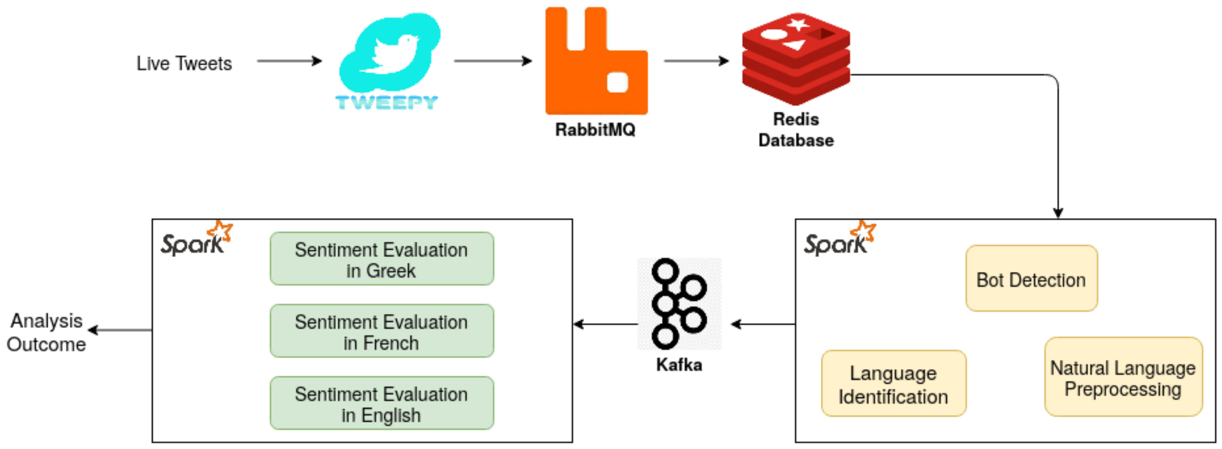

Live Tweets
TWEEPY
RabbitMQ
Redis Database
Spark
Bot Detection
Language Identification
Natural Language Preprocessing
Kafka
Sentiment Evaluation in Greek
Sentiment Evaluation in French
Sentiment Evaluation in English
Analysis Outcome

Figure 3: Architectural Elements for multilingual Twitter SA

Once fetched, the tweets are directed to a `RabbitMQ` configuration so as we can exploit a number of benefits this system offers including: 1) persistent, deadline-designated and acknowledgement delivered queuing, 2) routing and tracing of messages using exchanges, should we want to impose different classes of snippet at the input of our architecture, and 3) easy-to-use queue monitoring and management through a versatile Web-interface. Subsequently, tweets are directed to the `Redis` main-memory database for storage while `Spark` deploys its instances and operations for bot-detection as well as pre-processing, stemming and ultimately tweet–characterization. Directly pushing tweets from the API into `Redis` is not a plausible choice for it could create operational issues due to flush-crowds occasionally formed. In this case, the arrival message rate to `Redis` is greater than its rate to absorb incoming tweets resulting into an unreliable operation for the proposed architecture. It is worth pointing out that `RabbitMQ` guarantees the integrity of all the messages that it handles in transition and that its dispatched tweets to `Redis` in the same temporal order received. The cooperation of `RabbitMQ` with `Redis` is the basis for providing just-in-time Twitter–classification and SA analysis as our architecture properly places time-outs mechanisms for any potential blocking operation along its constituent elements [10].

The `Spark` instantiation on the bottom right of Fig. 3 undertakes 3 distinct processing tasks for every `Redis`-stored message that also contains respective user-data: 1) bot and fake user–account detection, 2) language identification, and 3) cleansing the text of natural language, filtering and stemming. Fake Account Detection and Language Identification services will be described in the next subsections. In Natural Language Processing step, tweets undergo a “cleansing” stage in which unhelpful and irrelevant portions of text are discarded. Moreover, missing parts or noisy segments are considered no further. In all languages, we apply “cleansing” actions where tags, URLs, hashtags and punctuation symbols are removed. It is equally important to indicate that several emoticons meant to highlight emotions are replaced with their actual corresponding word. For instance, “`:)`” is substituted by words “laugh”, “sourire”, and “χαμόγελο”, for English, French, and Greek, respectively. Lastly, for French and Greek we also use stemmers to remove unnecessary and likely confusing word endings due to their high inflection. With the assistance of a `Kafka` producer clean and stematized tweets are stored in 3-topics based on language.

`Kafka` is a messaging and integration module that offers decoupling and buffering for `Spark Streaming`. Along the pipeline, a `Kafka` consumer has subscribe to the corresponding in terms of language store to obtain new posts. This is the way filtered posts from the multilingual stream enter the final Sentiment Evaluation stage. This last stage consists of 3 `Spark` *contexts*, each dedicated to a single language. Every context features a `Kafka` consumer who receives cleansed tweets and organizes these messages in `DataFrame`s. Every language-based `Spark Context` uses the `ML Pipeline` to transform its input and yield its SA outcome. Each of the 3 pipelines consists of an `TF.IDF`-phase that normalizes textual data and a `MLlib`-based language-specific classifier. To cretae the training datasets for SA, we used [11] for French and English language and for Greek we created on our own a fully annotated $10K$ dataset of tweets; we make this dataset available in [12]. We finally opted for using `Naive Bayes` classifier for Greek and `Logistic Regression` for French and English. Although we considered a wide range of choices including *Logistic Regression* and *Decision Tree* classifiers, we opted for the above classifiers for each language as our classification approach turns out through experimentation to be both effective and efficient.

In our architecture, we opted for the use of `Structured Streaming` as it demonstrates a clear advantage as far as real-time handling is concerned. In general, `Spark Streaming` uses very low level operations which are applied to individual records found in `RDD`s. `Structured Streaming` on the other hand, operates on entire `Dataframes`; as a processing engine, `Structured Streaming` is built atop the`Spark SQL`. In our architecture, we opted for the use of `Structured Streaming` as it demonstrates clear processing advantage when real-time handling is necessary. Unlike its `Spark`-counterpart, `Structured Streaming` treats all arriving data as an *unbounded input table* in which every item in the Twitter-stream is a row appended [13]. As new rows enter the table, the ML-model gets dynamically updated, as depicted in Fig 4 [13].

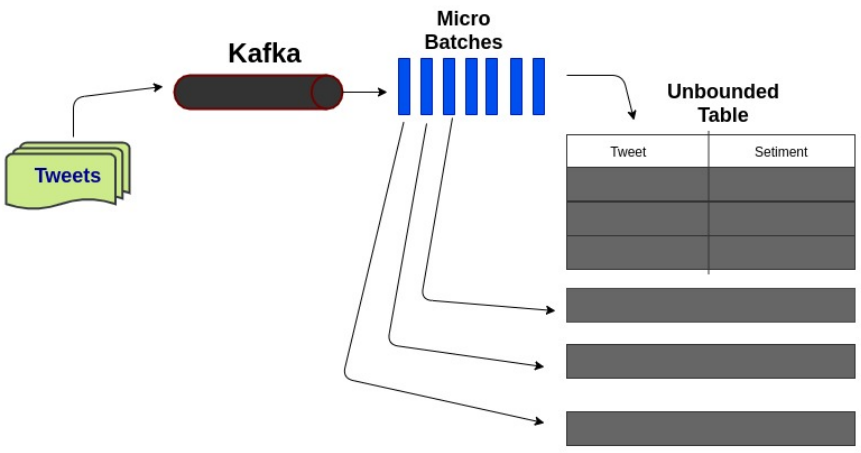


Figure 4: Structured Streaming with Kafka

3.1 Fake Account Detection Service

A bot sends out automated posts often at a rate impossible to be matched by a human, with the sole purpose of alerting individuals to imminent and/or noteworthy events and actions. Although the negative contribution of botnets to the social discourse is hard to refute [14, 15], botnets have been also used as alert-mechanisms in physical phenomena, emergencies and their aftermath including storms and forest-fires. Evidently, the type of SA analysis we investigate in this paper, would be biased if we also consider tweets by fake users. To address this issue, we have experimented with a Twitter bot-detection dataset from Kaggle [2] and created and adopted a corresponding *random forest model* [16].

[2] `https://www.kaggle.com/charvijain27/detecting-twitter-bot-data`

To train our model, we considered the following tweet features:

– number of followers an account has,
– number of users an account follows,
– number of public lists a user is a member of,
– number of tweets a user liked,
– number of tweets including re-tweets, posted by an account, and finally,
– whether a user has a verified account.

Posts that have cleared the bot-detection module, often present unhelpful, irrelevant portions of text and they may have missing parts or noisy segments. If remain untreated, all these aspects will contribute to additional noise which might in turn yield our SA to be unreliable and likely weaker than expected.

3.2 Language Identification Service

Language identification is the task of automatically detecting the language(s) present in a text by solely examining its content [17]. This is deemed a critical first step for applications and software systems necessitating language-specific modeling. Such service artifacts include multilingual information systems, indexing services, search engines, and social networking portals such as Twitter.

By and large, tweets are a special form of user-generated content whose key constraint is their limited size of less than 280 characters. Although short and unlike news articles, tweets writing style varies widely and they frequently harbor spelling and grammatical mistakes.

We are interested in Greek, French and English because they portray a number of unique characteristics. English is a widely spoken and written global language that is austere with simplicity of inflexion, extensive use of paraphrasing and numerous abbreviations in informal sentences. On the other hand, French and Greek have rather complex morphological features that entail high inflection and stressing rules. Unlike Greek, French and English use the Latin alphabet. When it comes to tweets, another rising challenge is that many posts contain some English words along with the language they are composed in. Occasionally, we may encounter English words written in Greek alphabet or Greek written in Latin broadly known as "*Greeklish*". We should also keep in mind the "*Franglais*" phenomenon, which refers to overuse and blending of English terms into French informal spoken language, and is on the rise [18].

We introduce a multilingual identification model based on supervised ML that is trained with the help of a language-annotated corpus. The independently labeled textual data that make up our training dataset collectively help the model determine which one of the 3 choices is the language of a tweet under consideration. The word abbreviations and the inadvertent change of spelling primarily done by user complying with to the tweet-length requirement, give rise to idiomatic forms of languages that are difficult to match even when using statics from external corpora. In this ever-changing operational context, we have assembled a training dataset consisting of $160{,}725$ entries. These entries are either tweet snippets or short formal sentences that we have individually labeled properly as far as the language used is concerned. More specifically, we have $53{,}575$ entries for every language out of which $26{,}787$ are tweet snippets and the remaining are small formal sentences.

Table 1: ML-Training Dataset Quantitative Characteristics

Textual Features	*Greek*	*French*	*English*
Avg. Number of Characters	55.24	69.64	78.62
Avg. Number of Words	9.16	10.62	13.81
Percentage of English Words	2.41%	7.86%	98.84%
Number of Entries per Language	$53{,}575$	$53{,}575$	$53{,}575$

Please feel free to make a suggestion.	#English
Choisi Pierre comme capitaine, quel dommage!	#French
Το να σηκώνεται κανείς νωρίς είναι καλό για την υγεία.	#Greek
credit to great save well done lads its coming home threelions	#English
έχει πλάκα το πόσο σοβαρα παιρνετε τα σοσιαλ παιδιά	#Greek
L'invention du transistor a introduit une nouvelle ère.	#French

Figure 5: Excerpt from the Multilingual Dataset

Table 1 depicts key characteristics for our dataset: avr. number of characters per piece of text for all 3 languages, avr. number of words found in snippets, percentage of English words encountered in Greek and French texts using `enchant Python3` library, and finally, the number of tweets and formal short texts used from the 3 languages. We opted to have both formal and informal texts contribute to the training corpus for 2 reasons as the dataset: i) had to follow the pattern of informal snippets as our trilingual model is to operate with Twitter live-streaming, and ii) should be enriched as far as its vocabulary is concerned so as the cover as wide range as possible in all languages.

To be more effective, we applied a "gentle" cleansing step to the used tweets to remove unnecessary –for our purposes– tags, *url*s and emoticons. For example, the tweet "What a beautiful view you have @Arnold001 <3 #inlove #malibou", ultimately becomes: "What a beautiful view you have". We purposely included a balanced number of tweets that contain some English words in French and Greek texts in order to help train our model to recognize such appearances. We did not remove punctuation from Greek/French as our stipulation is that this would ultimately help the language discrimination task. It is worth pointing out that informal versions of French/Greek terms do not display stresses and consequently, we sought to provide balanced representation of words with stresses in our dataset. In creating our multilingual dataset, we resorted to both open datasets from `github` [12, 11] as well as formal sentences from the 3 languages [19]. Although these sources are not specialized for language identification, we believe they serve our objective in a pragmatic manner for they yield worthy experimental outcomes. Fig. 5 depicts excerpts of entries from all languages.

To empower our multilingual classification model we: i) transform raw text into vectors of numbers that can be exploited by ML algorithms, and ii) use *Term Frequency-Inverse Document Frequency (TF.IDF)* to score the relative importance of words [20]. We considered a range of multinomial supervised classifiers including *Naive Bayes* (NB), *Random Forest* (RF) and *SVM*, and *Logistic Regression* (lr). The latter provided the most promising results and hence, we opted to realize our ML-classifier for language recognition using the respective approach from `Spark`'s `MLlib` library [21].

Our language identification problem is essentially a multi-label classification task and so, we resort to address it with *multinomial Logistic (Softmax) Regression* under the assumption that labeled classes are mutually exclusive. *Softmax* performs analysis when the dependent variable is nominally of more than 2 levels. As a predictive technique, *Softmax* helps explain the relationship between one nominal dependent variable and one or more independent variables. It also normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1. *Softmax* output values are in the interval $[0, 1]$ and so, it can accommodate the 3-sought classes in our case.

4. EVALUATION

We have based the implementation of our multilingual SA framework on `Apache Spark` $v2.3.4$ using `Python3's API PySpark` $v2.3.4$ which supports `Spark`'s programming model. We deployed `Tweepy` $v3.8.0$ to live-access Twitter and developed our messaging component based on `RabbitMQ` $3.8.3$ in conjunction with the `Pika`-client. Moreover, we used `Kafka` $v2.11.2$ to offer the pipeline to `Spark`'s `Structured Streaming` module. The entire system is hosted on an *Ubuntu 18.04* server with 8 cores on which `Spark` runs locally with 8 worker threads, one for each logical machine core.

In this section, we present the findings of our experimental evaluation with our system prototype. To this end, our specific goals have been to:

1. assess the quality of our SA in the 3 examined languages while experimenting with a range of `Spark` ML classifiers,
2. evaluate the performance of our Language Identification component,
3. investigate the effectiveness of our Twitter fake-account detection component and its contribution to the overall performance of our system prototype, and
4. outline some of the findings from recent platform deployments in order to evaluate sentiments expressed by users writing in *EN* and *GR*.

4.1 Assessing the SA-Models for the 3 Languages

It is imperative that the accuracy of the models we have ultimately adopted for English, French, and Greek be examined for the accuracy and reliability in their predictions. Although the training of any ML-model is of key importance, the behavior of any such model under new workloads is deemed a more critical aspect in any ML-pipeline formed to address a specific task. In our experimentation, we initially sought to establish the accuracy of the adopted models and demonstrate that we can trust their predictions. To accomplish this, we used *cross-validation*, a technique that entails partitioning of the original observation dataset into a training and an evaluation set; the former is used to train the model and the latter to show the validity of experimental outcomes.

With the help of cross-validation, we have investigated the effectiveness of our SA-model for the Greek language under 4 different classifiers provided by `Spark-MLlib`: 1) *Naive Bayes*, 2) *Logistic Regression*, 3) *Decision Tree*, and 4) *Random Forest*. The datasets *ratio* for our cross-validation was set to $70{:}30$, that is, we trained our models with 70% of the data while using the remaining 30% for testing validation.

Provided that for the datasets used in the experiment we have compiled the ground truth, we measure the performance of the above 4 classifiers in our assessment using the following metrics: α) *Accuracy*, β) *F1*-score, and γ) *Area Under the ROC Curve*(*AUC*). Accuracy measures how many observations, both positive and negative, were correctly classified. It is the ration of correct predictions made over all predictions made. *F1*-score is based on *Precision* and *Recall* and combines them into one metric by calculating the harmonic mean. *Precision* is the number of correct positive results divided by the total predicted positive observations. *Recall* is the number of correct positive results divided by the number of all relevant samples (total actual positives). *F1*-score yields an enhance measure of the incorrectly classified cases. *AUC* provides an aggregate measure of performance across all possible classification thresholds and is a probability curve that designates the degree or measure of *separability*. It essentially "states" how much a model is capable of distinguishing between classes. Thus, a successful model has *AUC* near 1 that means excellent separability. On the other hand, a *AUC* value near 0 reveals an unsuccessful model of poor separability.

Table 1 shows all the measurements that each of the 3 models has attained while carrying out our cross validation. We also depict the total *training time* required as well as the *average response time* to process a tweet from the dataset.

Table 2: Measurements for the 3 languages across the 4 Classifiers for SA.

Language	Metrics	Naive Bayes	Logistic Regression	Decision Tree	Random Forest
English	*Accuracy*	86.78	89.90	80.79	81.29
	AUC	69.14	95.51	78.32	79.92
	F1-score	86.74	89.47	80.02	82.01
	Training Time	8.74 sec	$9,99$ sec	13.15 sec	17.21 sec
	Avg. Resp. Time per tweet	$5.4\ 10^{-6}$ sec	$5.6\ 10^{-6}$ sec	$4.9\ 10^{-4}$ sec	$2.36\ 10^{-2}$ sec
French	*Accuracy*	83.52	85.39	75.39	72.71
	AUC	81.02	96.38	78.20	68.80
	F1-score	88.25	90.12	69.24	70.04
	Training Time	5.81 sec	6.52 sec	9.54 sec	13.42 sec
	Avg. Resp. Time per tweet	$1.3\ 10^{-5}$ sec	$5.0\ 10^{-6}$ sec	$4.1\ 10^{-4}$ sec	$4.02\ 10^{-3}$ sec
Greek	*Accuracy*	80.23	81.43	77.36	73.01
	AUC	87.35	75.00	72.47	69.51
	F1-score	75.28	79.32	71.09	65.33
	Training Time	1.15 sec	6.23 sec	14.87 sec	17.55 sec
	Avg. Resp. Time per tweet	$1.3\ 10^{-5}$ sec	$5.8\ 10^{-6}$ sec	$4.5\ 10^{-3}$ sec	$2.8\ 10^{-2}$ sec

We can easily discern that the *Random Forest* classifier yields consistently inferior results for all languages. Among the remaining 3 classifiers, *Naive Bayes* and *Logistic Regression* show better outcomes if compared with *Decision Tree*. The ultimately adopted SA-model we select is the classifier that presents high accuracy and as short a training phase as possible. Despite the fact that in English, the *Naive Bayes* demonstrated the shortest training time, we finally opted for *Logistic Regression*. This is done for we anticipate the majority of posts in the live feed to be authored in English and so we wanted to have the highest possible accuracy. In Greek, we traded off a slightly less accuracy for the impressive training time of the *Naive Bayes* classifier. Finally, the French SA-model overall worked with much better accuracy under the *Logistic Regression* classifier. Consequently, *Logistic Regression* was our choice in French.

4.2 Assessing Language Identification Model

Following the same pattern as in Section 4.1 we opted to evaluate our Language Identification model considering a range of multinomial supervised classifiers including *Naive Bayes*, *Random Forest*, *SVM*, and *Logistic Regression*. *Accuracy*, *F1*-score and *AUC* provide insights in the quality of the predictions and at the same time highlight the degree the model we experiment with can distinguish the 3 classes: English, French, and Greek. We created a model based on *Softmax* that yields the highest $89,67\%$ accuracy rate with *cross-validation* in ratio $70{:}30$.

To ascertain the correctness of the selected ML-classifier in real-life, we tested our model with the large number of tweets that presented no equitable numbers in the Greek, French and English posts. This workload was indeed asymmetric as far as the use of languages is concerned and as expected English was much more heavily used, filling 45% of the dataset. Despite the above constraint, the *Softmax* approach offered high-accuracy results for the live-feed as well. More specifically, 89% of the classified tweets were accurately

Table 3: Measurements Across the 4 Classifiers for Language Identification.

Metrics	*Naive Bayes*	*Softmax*	*Random Forest*	*SVM*
Accuracy	45.29	89.67	55.39	77.44
AUC	38.41	86.24	49.65	77.33
F1-score	41.12	81.60	51.87	76.01
Training Time	6.30 sec	$27,49$ sec	10.79 sec	19.84 sec
Avg. Resp. Time per Account	$4.67\ 10^{-6}$ sec	$2.51\ 10^{-1}$ sec	$3.11\ 10^{-4}$ sec	$4.15\ 10^{-2}$ sec

identified in terms of language with the 11% of loss affecting mostly the French language with a rate of 5.78%. Greek and English saw losses at rates of 1.12% and 4.10% respectively.

We should also point out that *Softmax* attains a high-accuracy while taking the longest time to compute its corresponding model as Table 4 shows. While *Naive-Bayes*, *SVM* and *Random Forest* take a fraction of *Softmax* execution time, they all produce inferior accuracy rates. To prevent unnecessary re-computation of our *Softmax* model, we persist it and we can reuse it within `Spark` when the need for new prediction arises without creating CPU overheads.

4.3 Effectiveness In Detecting Fake Accounts

In order to facilitate as much as possible the just-in-time classification, we had to ensure that bot activity was detected early on so as respective processing for fake tweet-users can be avoided. To achieve this and be able to discard malicious users, we used the *Kaggle* bot detection dataset [22] to experiment with various algorithmic ML-options. Due to the limited number of datasets related to Twitter fake accounts, we tried to investigate the performance of our dataset in light of a wide range of `MLlib`-classifiers. Among the possible examined choices, the more promising on the *Kaggle* dataset were those of *Random Forest*, *Naive Bayes*, *Logistic Regression* and *Decision Tree*. As mentioned earlier, we used cross-validation with ratio $70{:}30$ to ascertain the effectiveness of our choices and used scores: *Accuracy*, *F1*-score, and *AUC*. We also consider the training time required of every candidacy and the measured average response time required per account. Table 4 depicts

Table 4: Measurements across 4 Classifiers for Detecting Bot Accounts.

Metrics	*Decision Tree*	*Naive Bayes*	*Random Forest*	*Logistic Regression*
Accuracy	85.27	75.35	87.17	86.02
AUC	87.21	75.01	93.54	89.14
F1-score	84.25	74.47	87.81	81.33
Training Time	$3.41\ 10^{-2}$ sec	$6.01\ 10^{-2}$ sec	$3.11\ 10^{-2}$ sec	$1.96\ 10^{-1}$
Avg. Resp. Time per Account	$6.67\ 10^{-7}$ sec	$2.45\ 10^{-6}$ sec	$3.33\ 10^{-7}$ sec	$4.82\ 10^{-5}$ sec

our results while experimenting with the 3 more promising classifiers. Overall, *Random Forest*, *Logistic Regression* and *Decision Tree* yielded higher accuracy than *Naive Bayes*. We find also the same to be the case as far as *AUC* and *F1*-score metrics is concerned. *Random Forest* and *Decision Tree* feature a very short training phase around a fraction of millisecond. We observe similar trends in the average response time. Hence, we opted for deploying a *Random Forest* classifier for our prototype that correctly identifies 87% of all Twitter fake accounts.

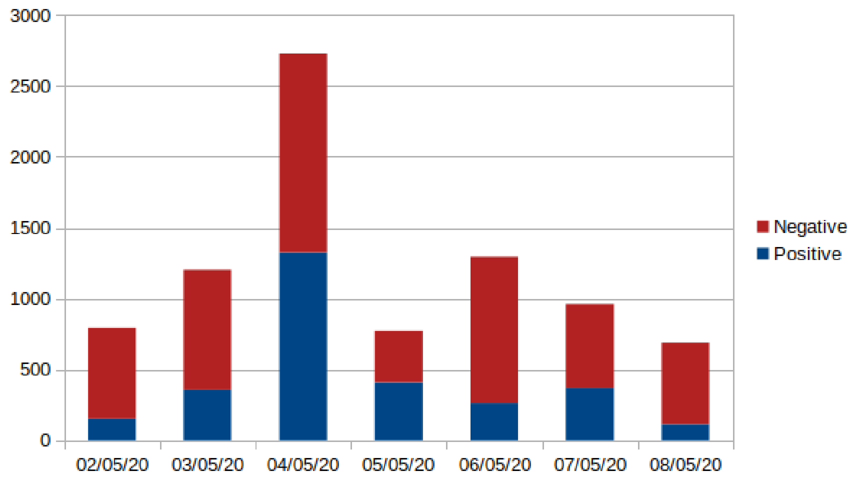


Figure 6: Prediction of with hashtags: #covid19 and #COVID19greece.

4.4 Monitoring Twitter's Live-Feed

We have sought to explore the value of our prototype using Twitter's live feed. For this, we fetched tweets posted live during the period of *April 14th, 2020* to *May 25th, 2020*. The messages were processed through our prototype that designated every valid tweet as either positive or negative. We should note that during the entire observation period the global Covid-19 pandemic was unfolding. Our goal was to monitor the behavior of our multilingual SA–service in real–life conditions and ascertain if the produced results reflected reality. Although we witnessed interested trends, mostly negative reactions, in all languages, we report 2 specific results monitored in Greek and English Twitter.

In early May 2020, we tracked on the average $42,592$ tweets per-day in Greek. From this corpus, the posts with hashtags #covid19 and #COVID19greece displayed an interesting behavior as the timeline of Fig. 6 shows. The trend displayed here is specific to the efforts undertaken to address the Covid-19 outbreak in Greece. Fig. 6 clearly shows an uptake in pertinent tweets on the 4^{th} of May when the number of negative posts is about *equal* to those characterized as positives. In all other days of observation the number of tweets remains on the low side in terms of numbers for the hashtags in discussion. The spike of May 4^{th} is, we believe, directly related to the fact that in this day Greece entered the initial phase of lifting the shelter-in-place order which was in effect since March, 16^{th}. It is also worth noting that the percentage of tweets classified as positive on May 4^{th} was the highest during the entire period of observation.

While working with English, we tracked on the average $767{,}268$ tweets per-day. Justifiably, we focused on major trending topics of the day as this allowed to capture large numbers of tweets. Clearly, developments in North American and tweets with hashtags related to #trump and #presidenttrump did receive heavy traffic during our observation period. Fig. 7 depicts a significant rise in the number of processed tweets related to the week-end sporting activities of D.J. Trump on May 23^{th} and 24^{th}. There is a major increase in the number of negative tweets as well. We stipulate that this finding is a direct outcome of 2 pieces of news reported nearly at the same time: 1) how resting time was spent by the president, and 2) the unfortunate development of the U.S. reaching $100,000$ fatalities in the Covid-19 pandemic.

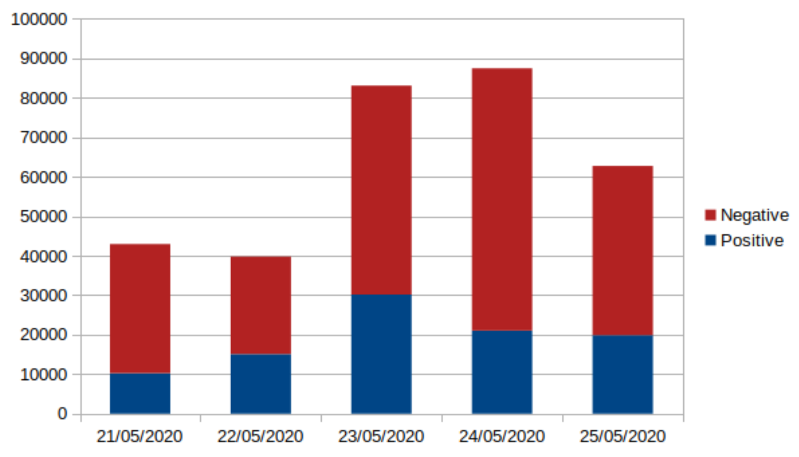


Figure 7: Prediction for tweets with hashtags: #trump and #presidenttrump.

5. CONCLUSION

Apache Spark is a cluster computing technology, designed for fast computation. It can cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. The main feature of Spark is its in-memory computing that increases the processing speed of an application.

In this thesis, we investigate the problem of *multilingual sentiment analysis* in streams of tweets, using the powerful `Apache Spark` engine. We propose a service platform to facilitate the timely classification and enable processing of incoming posts written in 3 languages: English, French, and Greek. We built ML-models that can accurately: i) identify the language of the streamed tweets, ii) prevent bot-originated posts to influence the result of analyses, and iii) help establish sentiment trends in the 3 languages we consider. Moreover, we outline the design choices made and show how we map all required functionalities of our platform to off-the-shelf components that include `Spark`, `RabbitMQ`, `Kafka`, as well as libraries such as `Tweepy` and `MLlib`. Through experimentation, we establish both the effectiveness and efficiency of our suggested platform and report on trends observed regarding the social discourse due to contemporary community events and breaking news.

In conclusion, the implementation offers the potential to be further extended and provide a wider range of analysis when combined with other social media data. The framework may be further developed to make streaming predictions through a unified API that searches Twitter, Tumblr, and Reddit concurrently to obtain synthesis of sentiments with an information fusion approach.

ABBREVIATIONS, ACRONYMS

DAG	Directed Acyclic Graph
RDDs	Resilient Distributed Datasets
NLP	Natural Language Processing
SA	Sentiment Analysis
ML	Machine Learning
MLlib	Spark Machine Learning Library
NB	Naive Bayes
LR	Logistic Regression
DT	Decision Tree
RF	Random Forest

REFERENCES

[1] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker ans Ion Stoica. ”Apache Spark: A Unified Engine for Big Data Processing” , Communications of the ACM - November 2016

[2] Shi J, Qiu Y, Minhas UF, et al.” Clash of the titans: MapReduce vs. Spark for large scale data analytics”. Proc VLDB Endow. 2015;8(13):2110-2121.

[3] Haripriya Ayyalasomayajula. ”An evaluation of the Spark programming model for Big Data Analytics”. University of Houston - May 2015 2015;8(13):2110-2121.

[4] Raghavendra Kune1, Pramod Kumar Konugurthi, Arun Agarwal, Raghavendra Rao Chillarige and Rajkumar Buyya. ”The anatomy of big data computing”. Softw. Pract. Exper. 2016; 46:79–105 - 9 October 2015

[5] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng and Joshua Zhexue Huang. “Big data analytics on Apache Spark” International Journal of Data Science and Analytics. November 2016, Volume 1, Issue 3–4, pp 145–164

[6] Apache Spark Use Cases in Real Time. Available: `https://data-flair.training/blogs/spark-use-cases/`

[7] S. Biyabani and J. Stankovic and K. Ramamritham. The Integration of Deadline and Criticalness in Hard Real–Time Scheduling. Proc. of RT Symp. December 1998

[8] A. Bestavros and S. Braoudakis. Timeliness via Speculation for Real–Time Databases. Proc. of the IEEE Real–Time Systems Symp. December 1994

[9] V. Kanitkar and A. Delis. Real-Time Processing in Client-Server Databases. IEEE Transactions on Computers, March 2002. Volume 51, Issue 3, pp 269-287

[10] . Stonebraker and U. Cetintemel and S. Zdonik. The 8 Requirements of Real-time Stream Processing. ACM SIGMOD-Record, December 2005. Volume 34, pp 42-47

[11] C. Malafosse. Open Dataset for Sentiment Analysis. Available at: www.github.com/charlesmalafosse/ (Accessed on May 19th, 2020)

[12] I. Karageorgou. Stemmed and Classified Tweet Dataset in Greek. Available at: www.github.com/ioannakarageorgou/grtweetsdataset (Accessed on May 9th, 2020)

[13] S. Salloum and R. Dautov and X. Chen and P.X. Peng and J.Z. Huang. Big Data Analytics on Apache Spark. Int. Journal of Data Science & Analytics, October 2016. Volume 1, pp 145-164

[14] P.N. Howard. How Political Campaigns Weaponize Social Media Bots. IEEE Spectrum, October 2020

[15] E. Ferrara and O. Varol and C. Davis and F. Menczer. The Rise of Social Bots. Communications of the ACM, July 2016. Volume 7, pp 96-104

[16] P. Efthimion and S. Payne and N. Proferes. Supervised Machine Learning Bot Detection Techniques to Identify Social Twitter Bots. SMU Data Science Review 2018

[17] T. Vatanen and J. Vayrynen and S. Virpioja. Language Identification of Short Text Segments with N-gram Models. Proc. of Int. Conf. on Language Resources and Evaluation, May 2020

[18] R.J. Blackwood and S. Tufi. The Linguistic Landscape of the Mediterranean, Palgrave-Macmillan.

[19] A Collection of Sentences and Translations. Available at: https://tatoeba.org/eng/downloads (Accessed on May 19th, 2020)

[20] Rijsbergen, C. J. Van. Information Retrieval, 2nd Edition, Butterworth-Heinemann

[21] X. Meng and J. Bradley and B, Yavuz and E. Sparks and S. Venkataraman and D. Liu and J. Freeman and DB Tsai and M. Amde and S. Owen and D. Xin and R. Xin and M. Franklin and R. Zadeh and M. Zaharia and A. Talwalkar. MLlib: Machine Learning in Apache Spark. Machine Learning Research, April 2016. Volue 17, pp 1-7

[22] Bot Detection Dataset. Available at: https://www.kaggle.com/charvijain27/detecting-twitter-bot-data (Accessed on April 3rd, 2020)