

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**Clasificarea, clusterizarea si cartografierea datelor de pe Twitter
folosind Apache Spark**

propusă de

Rareș Bradea

Sesiunea: *Iulie, 2018*

Coordonator științific

Lect.dr. Cristian Frăsinaru

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ

Clasificarea, clusterizarea si cartografierea datelor de pe Twitter folosind Apache Spark

Rareș Bradea

Sesiunea: *Iulie, 2018*

Coordonator științific
Lect.dr. Cristian Frăsinaru

DECLARAȚIE PRIVIND ORIGINALITATE ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul „*Clasificarea, clusterizarea și cartografierea datelor de pe Twitter folosind Apache Spark si Folium*” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau din străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imaginile etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

lași, *data*

Absolvent *Rareș Bradea*

(semnătura în original)

Cuprins

Introducere	5
Motivatie si gradul de noutate.....	5
Obiectivele lucrarii.....	5
Metodologia folosita	6
Descrierea sumara a solutiei.....	9
1Colectarea datelor in timp real	11
2Clasificare.....	15
2.1Descrierea problemei.....	15
2.2Incerari ulterioare.....	16
2.3Solutia.....	17
3Clusterizare.....	20
3.1Definitie.....	20
3.2Tipuri de clustere.....	20
3.3Setul de date.....	22
3.4Preprocesare.....	22
3.5Modelul creat.....	23
LDA.....	23
Gaussian Mixture.....	23
K-Means.....	24
3.6Concluzie.....	25
4Cartografiere.....	26
4.1Folium.....	26
4.2Afisare pe harta.....	26
4.3Gasirea clusterului relevant.....	26
5Concluzii	27
5.1Directii de viitor.....	27
Bibliografie.....	28

Introducere

Motivatie si gradul de noutate

In ultimul timp, retelele sociale online s-au dezvoltat enorm, plecand de la firavele inceputuri lipsite de multe functionalitati si importanta si ajungand in ziua de azi sa reprezinte structuri sociale si chiar politico-economice de uriase proportii. O retea sociala online poate fi definita in contextul recentului fenomen numit Web 2.0 (care se refera la noua generatie de website-uri unde accentul se pune pe usurinta folosirii si crearea de continut de catre utilizatori) [4] ca fiind un site web unde utilizatorii fac parte dintr-o structura sociala si reprezinta „actori” sociali ce sunt conectati prin mai multe legaturi de tip „one-to-one”, creand astfel graful retelei sociale. Printre cele mai populare retele sociale online se numara Facebook, Instagram si Twitter, avand 2234, 813 si respectiv 330 de milioane de utilizatori, potrivit unui recensamant din Aprilie 2018 [5].

Prin popularitatea lor, aceste retele sociale devin foarte usor tinta utilizatorilor malitiosi ce au ca scop propagarea interactiunilor si fenomenelor de tip spam si phishing. Acestea au ca obiectiv convingerea utilizatorilor firesti, prin cai daunatoare, deranjante si de obicei greu de descoperit, sa isi expuna datele personale sau sa piarda bani in favoarea celor care recurg la aceste tactici.

Pe langa aceste practici malitioase, pe Twitter exista si foarte multe postari irelevante pentru multi din utilizatori. O data postat un tweet, acesta ajunge pe feed-ul oricarui utilizator ce urmareste persoana ce posteaza. Acest lucru face ca uneori sa existe o neconcordanza intre doleantele unui utilizator si ce citeste acesta pe propriul feed.

Obiectivele lucrarii

Unul dintre cele doua principale obiective ale lucrarii este acela de a studia in profunzime frameworkul de cluster computing numit 'Apache Spark'. Dintre componentele sale, cele mai relevante pentru implementarea unei aplicatii folosind frameworkul acesta au fost studiate mai in profunzime si descrise in capitolele relevante fiecarui modul din aplicatie. Spark Streaming este descris in primul capitol si Spark MLlib in capitolele 2 si 3.

Al doilea obiectiv este implementarea unei aplicatii ce foloseste tehnologii existente in frameworkul Apache Spark. Acest software are scopul de a filtra multe din aceste mesaje folosind

tehnici de clasificare din invatare automata, explicate in al doilea capitol. De asemenea, se doreste crearea unor grupuri de tweeturi ce abordeaza subiecte asemanatoare si sunt de inalta calitate, pentru a veni in ajutorul persoanelor ce sunt interesate de a 'lua pulsul' societatii in care se afla, precum jurnalistii, fara sa fie nevoiti sa sorteze printr-o multime de mesaje ce se pot dovedi a fi irelevante, astfel avand parte de o experienta sigura si utila.

Aceste grupuri de tweeturi vor fi afisate pe o harta si vor putea fi cautati termeni ce se doresc a fi gasiti in grupuri special create dupa acei termeni, astfel creandu-se anumite topicuri ce sunt relevante pentru inputul unui utilizator. Cartografierea acestor tweeturi este utila deoarece creeaza o perspectiva noua si ofera o imagine de ansamblu ce poate descrie foarte usor de unde a pornit un anumit fenomen social, fie el o stare, un zvon, o idee pentru o miscare sociala sau chiar un dezastru natural sau uman.

Metodologia folosita

Pentru a atinge aceste obiective, adica de a clasifica binar un tweet, fie intr-o categorie de continut de calitate inalta, fie o categorie de continut de calitate scazuta, si de a grupa (in termeni de invatare automata, de a clusteriza) o colectie de tweeturi in mai multe subcolectii ce sunt asemanatoare intre ele, iar apoi cartografia aceste tweeturi intr-un format usor de interpretat am apelat la biblioteca 'folium' din Python si frameworkul de cluster computing 'Apache Spark'. [Anexa1]

Lansat in anul 2014, avand ca autor initial pe Matei Zaharia, in cadrul proiectului AMPLab al universitatii Berkley, din California, acesta are la baza o abstractizare a datelor numit resilient distributed dataset (RDD). Si din denumire se poate infera ca acesta descrie un multiset (un set ce poate contine mai multe instante ale aceluiasi element) de date distribuite pe un cluster de computere.

In cadrul acestui framework exista mai multe componente ce ofera diverse functionalitati:

- pentru a utiliza comenzi SQL peste o abstractizare a datelor numita DataFrame, ne este pus la dispozitie Spark SQL

- componenta Spark Streaming se ocupa cu analiza stream-urilor de date. Datele sunt aduse in memorie in mini-batch-uri (grupuri mai mici de date) si se pot efectua transformari RDD pe acestea

- componenta Spark MLlib ofera implementari ale unor algoritmi de invatare automata, capabili de calcul computational distribuit

- componenta GraphX este un framework de procesare a grafurilor

Din toate acestea, cele utilizate in aplicatie sunt Spark SQL, Streaming si MLlib, pentru citirea datelor de pe Twitter, aplicarea unor algoritmi de invatare automata si manipularea a unor DataFrames.

Mai în profunzime, Apache Spark este un framework open source destinat procesării unor volume de date la scară mare. Vizează aplicațiile construite pe sisteme distribuite și API-ul expune funcționalități pentru limbajele Java, Scala, Python și R. Folosind Spark Application Frameworks, Spark, scris în Scala, simplifică accesul la algoritmi de machine learning și analiză predictivă. Spark Core, componenta de bază a frameworkului, se bazează pe o abstractizare a datelor numită “resilient distributed dataset” (RDD), ce reprezintă o mulțime, o colecție imutabilă de elemente distribuită pe un cluster de sisteme computaționale peste care se poate opera în paralel. Caracteristicile acestui tip de date sunt, după cum sugerează numele:

- Rezilient, există posibilitatea de a recomputa partiții cu probleme
- Distribuit, datele se află pe mai multe noduri într-un cluster
- Este un dataset cu valori primare sau valori de valori (tuple sau obiecte)

Spark, rulat în mod nelocal, are nevoie de un manager de clustere și un sistem distribuit de stocare a datelor. Pe lângă soluția nativă a managerului, există suport pentru Hadoop Yarn și Apache Mesos. Pentru stocarea datelor se poate utiliza Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu,

Spark mai poate fi descris prin capabilitățile din modulul Spark Streaming. Acesta permite utilizatorului să lucreze cu cantități mari de date ce sunt servite în timp real, prin deschiderea unui stream și “ascultând” date precum statusuri de Twitter sau, de pildă, streamuri custom construite în Kafka, Flume, Kinesis sau chiar socketuri TCP. Aceste date pot fi procesate utilizând algoritmi complecși de machine learning sau procesarea grafurilor cu funcții high-level precum map, reduce, join și window. Datele procesate pot fi exportate în fișiere, baze de date sau live dashboards. Spark Streaming lucrează astfel, primește un input live de data streams și le împarte în elemente numite batches, sunt apoi procesate și astfel rezultă streamul final de batches.



Illustration 1

Spark Streaming oferă o abstractizare numită discretized stream (DStream), ce reprezintă un stream continuu de date. Acestea pot fi create din surse precum Kafka, dar și prin aplicarea unor operațiuni high-level pe alte DStreams. Intern, un DStream este reprezentat de o serie continuă de

RDDs. Fiecare RDD dintr-un DStream conține date dintr-un anumit interval de timp.



Illustration 2

Orice operațiune aplicată pe un DStream este de fapt tradusă în operațiuni pe RDDs din spate. Aceste operațiuni sunt efectuate de Spark engine. Operațiunile pe DStream ascund multe din detalii și oferă un API high-level, pentru facilitarea utilizării.

Fiecărui DStream de input (în afara celor care fac streaming din fișiere) îi este asociat un Receiver, o componentă, un obiect ce primește datele de la o sursă și o stochează în memorie pentru procesare. Există două tipuri de receivers, în funcție de fiabilitatea acestora. Surse fiabile precum Kafka sau flume permit datelor transferate să fie recunoscute. Dacă sistemul ce primește datele de la aceste surse fiabile recunoaște corect datele primite, atunci există siguranța că nu vor exista pierderi de informație ca urmare oricăror tipuri de defecțiuni. Astfel, receivers pot fi de două tipuri:

- Reliable receiver; acesta trimite confirmarea către o sursă fiabilă când datele au fost primite și stocate.
- Unreliable receiver; acesta nu trimite niciun fel de confirmare către sursă. Se folosesc pentru surse care nu suportă acest sistem de acknowledgment (admitere și confirmare).

Procesarea DStreamurilor și, deci, a RDDurilor se face prin transformări. Acestea sunt niște funcții care modifică datele. De exemplu, funcția `map(myFunc)` returnează un nou DStream prin aplicarea funcției `myFunc` peste toate elementele din DStreamul inițial. Funcția `transform(myFunc)` permite apelarea unor funcții RDD-to-RDD, care nu sunt aplicabile direct pe DStreams, pe fiecare RDD dintr-un DStream. Un exemplu de astfel de funcție este joinul dintre batchurile dintr-un stream și alt dataset.

Există transformări ale căror apeluri sunt constrânse de timp. Prin aceste windowed transformations, se pot aplica modificări pe datele peste care trece o “fereastră glisantă”, ca în exemplul de jos.

Pe DStreams pot fi utilizate, de asemenea, DataFrames și operațiuni SQL. Fiecare RDD este convertit într-un DataFrame, înregistrat ca un tabel temporar peste care se pot face interogări SQL.

Algoritmii de streaming machine learning din MLib pot învăța din streamurile de date și, în același timp, să aplice aceste cunoștințe pe același stream de date. Menționez algoritmi capabili de

aceste lucruri: Streaming Linear Regression, Streaming KMeans etc. Pentru alți algoritmi, se pot folosi date istorice pentru learning, mai apoi aplicându-se modelul pe streamuri de date.

În concluzie, Apache Spark este o tehnologie foarte puternică și foarte rapidă, ce permite cluster computing pe dataseturi foarte mari, utilizând machine learning, graph processing, streamuri de date și alte metode.

Descrierea sumara a solutiei

Aplicatia se poate imparti in patru module ce lucreaza impreuna pentru a ajunge la rezultatul final, reprezentat de o colectie de tweeturi, impreuna cu locatia lor, ce au fost clasificate ca fiind de calitate inalta, clusterizate in grupuri relevante, dupa asemanarea dintre ele si apoi afisate pe o harta ca puncte ce contin textul tweetului, locatia acestuia si grupul semantic carui apartine.

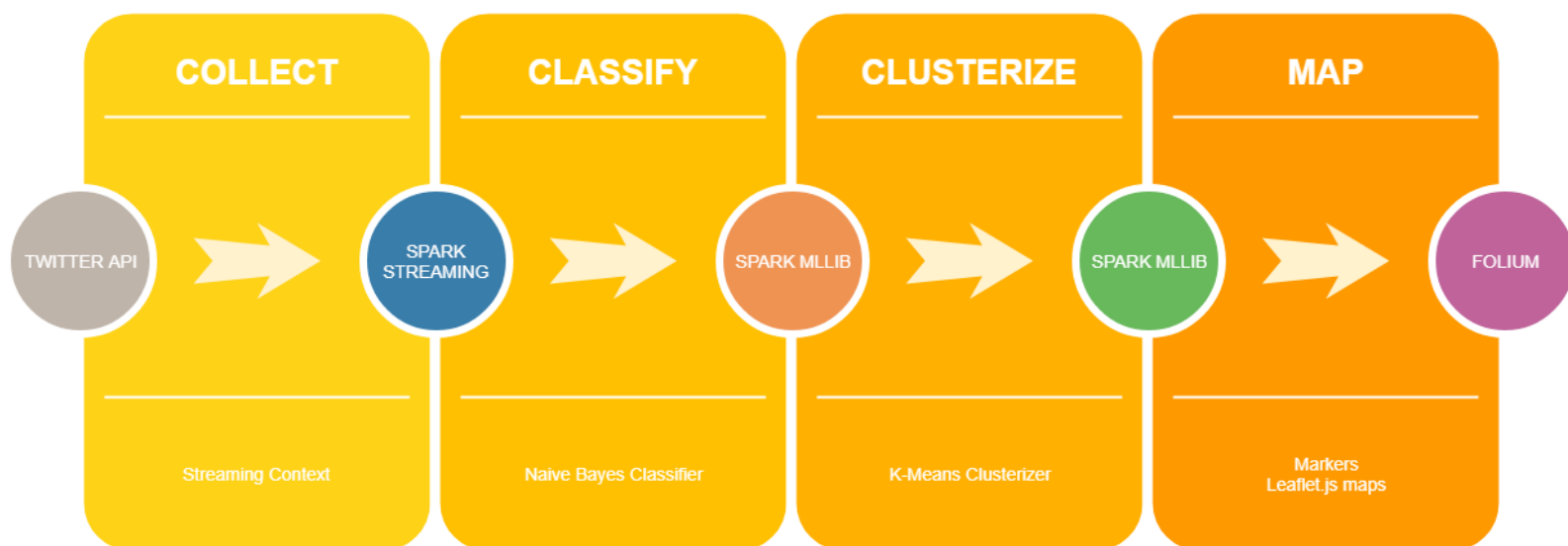


Illustration 3

Primul modul este compus din doua scripturi Python. Primul din ele apeleaza API-ul Twitter pentru a primi acces la un stream live de tweeturi. Acesta transmite prin TCP/IP tweeturi catre o instanta de Spark Streaming ce se afla pe un al doilea script. Acesta din urma preia tweeturi si le clasifica ca fiind de slaba sau inalta calitate, folosind un model de clasificare bazat pe algoritmul Bayes naiv.

Al doilea modul consta in scriptul Python ce a antrenat si testat diversi algoritmi de clasificare, implementati in Spark Mllib. Concluzia optima dupa multiple incercari a fost utilizarea algoritmul Bayes naive pe un dataset cu aproximativ 1200 de instante de tweeturi de slaba calitate si aproximativ 10000 de instante de tweeturi de inalta calitate. Acesta ofera o acuratete de 93.5% la testare, una destul de apropiata de celalte variante, cuprinse intre 91% si 92%.

Al treilea modul este alcatuit din functii ce pot clusteriza datele salvate in primele module si pot produce rezultate ce constau in asocierea fiecarui tweet cu un cluster.

Al patrulea modul se refera la randarea tweeturilor clusterizate in modulul precedent, afisandu-le pe harta globului pamantesc si oferind o imagine de ansamblu asupra modului de propagare si nastere a unor subiecte de interes major pe retea de socializare Twitter.

1 Colectarea datelor in timp real

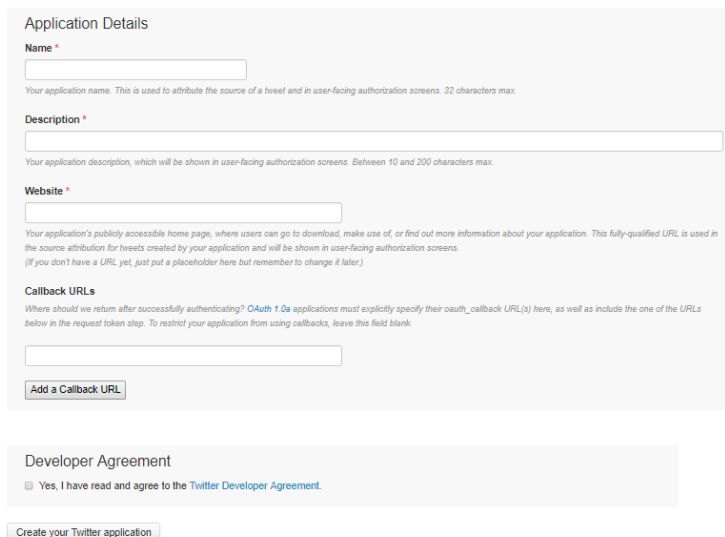
Orice aplicație ce include în implementarea ei și un modul de învățare automată are imperioasă nevoie sau cel puțin beneficiază foarte mult de pe urma unui dataset cât mai extins, dar și curat. Twitter, prin popularitatea sa imensă atinge cu excelență punctul referitor la cantitatea datelor, dar se îndepărtează de un ideal al datelor relevante și curate. Dat fiind faptul că oricine poate să își facă un cont unde poate să exprime idei în limita a 280 de caractere, nu este o surpriză faptul că relevanța multor tweeturi este minimă pentru multe persoane. Partea de clasificare a aplicației se va ocupa de etichetarea acelor tweeturi ce conțin enunțuri fără sens, cuvinte deosebit de vulgare, încercări de comercializare sau spam. Înainte de a ajunge acolo, trebuie să clădim un dataset cât mai mare pentru a micșora impactul postărilor cu relevanță scăzută.

Acestea fiind spuse, Twitter este un ecosistem foarte complex și activ, iar pentru a facilita munca dezvoltatorilor de software interesați de datele ce rezidă în interiorul aplicației lor, s-a creat un API ce expune accesul la tweeturi în timp real. Aplicația mea urmărește să clasifice, clusterizeze și apoi să afișeze pe hartă tweeturi care au o vechime scurtă, astfel că este foarte utilă utilizarea API-ului de streaming oferit. Alături de acesta, voi folosi și o instanță de Apache Spark cu un context de Streaming ce permite procesarea datelor stream-uite live.

O altă variantă ar fi fost folosirea unor dataseturi deja existente, însă relevanța lor referitoare la vârstă și noutate ar fi fost discutabilă. De altfel, multe dataseturi urmăresc un singur subiect, cum ar fi tweeturi ce discută situația imigrării în Canada. Acest lucru se dovedește a fi util pentru partea de testare a clusterizării, dar scopul aplicației este să fie semi-realtime și să dispună de o varietate a subiectelor vastă și necontrolată a priori.

În cele ce urmează, voi descrie succint pașii parcurși de mine pentru a ajunge la un modul ce accesează API-ul de tweet streaming și trimite date către o instanță stream-ready de Spark ce le procesează.

În primul rând, pentru a avea acces la API-ul Twitter este nevoie un cont simplu de Twitter și mai apoi de înregistrarea unei aplicații ce dorește accesul la API, accesând pagina



The screenshot shows the 'Create an application' form on the Twitter Developer Portal. The form is titled 'Application Details' and contains several input fields: 'Name' (required), 'Description' (required), 'Website' (required), and 'Callback URLs'. Below these fields are instructions for each. At the bottom of the form is a 'Developer Agreement' section with a checkbox and a 'Create your Twitter application' button.

Application Details

Name *

Description *

Website *

Callback URLs

Developer Agreement

Create your Twitter application

Illustration 4

<https://apps.twitter.com/>.

Dupa completarea acestui formular, vom avea acces la un dashboard cu anumite informatii. De acolo vom prelua patru coduri importante si necesare autorizarii noastre la serviciul oferit.

Acestea sunt un access token, un access token secret, consumer key si consumer key secret. Le voi folosi pentru a face autorizarea printr-un request OAuth1 folosind biblioteca 'requests_oauthlib' din Python, astfel.

```
ACCESS_TOKEN = 'sample'
ACCESS_TOKEN_SECRET = 'sample2'
CONSUMER_KEY = 'sample3'
CONSUMER_SECRET = 'sample4'
auth_worker = requests_oauthlib.OAuth1(
    CONSUMER_KEY,
    CONSUMER_SECRET,
    ACCESS_TOKEN,
    ACCESS_TOKEN_SECRET)
```

Mai departe, pornind un server care asteapta cereri pe localhost la un port oarecare (aici 9999), voi face accesul unui stream de date de la Twitter catre o instanta de Apache Spark.

```
def stream_tweets():
    query_url = 'https://stream.twitter.com/1.1/statuses/filter.json?language=en&locations=-136,15,-45,55&track=#'
    response = requests.get(query_url, auth=auth_worker, stream=True)
    print(query_url, response)
    return response
```

Functia 'stream_tweets' returneaza un raspuns http ce contine, pentru fiecare linie, un tweet complet respectand parametrii dati in query. Mai exact, cautam tweeturi scrise in limba engleza, iar parametrul locations astfel setat ne garanteaza ca locatia mesajelor provin din aproximativ partea continentală a Statelor Unite ale Americii.

Am ales aceasta locatie deoarece Twitter este foarte popular si utilizat in respectiva tara, fapt ce denota o posibila diversitate mai mare decat in alte locuri. Subiectele discutate sunt extrem de variate si ne pot oferi multe sanse de a obtine date interesante si din prisma geolocatiei, un atribut ce il are orice tweet al carui utilizator permite accesul serviciilor de locatie asupra contului.

```
def send_tweets_to_spark_with_location(http_response, connection):
    for line in http_response.iter_lines():
        try:
            full_tweet = json.loads(line)
            my_dict = dict([('text', full_tweet['text']), ('coordinates', full_tweet['coordinates'])])
            if my_dict['coordinates'] is not None:
                str_my_dict = json.dumps(my_dict)
                connection.send(bytearray(str(str_my_dict) + '\n', 'utf8'))
        except Exception as e:
            print(e)
```

Funcția 'send_tweets_to_spark_with_location' primește un răspuns http (în cazul acesta, cel returnat de funcția 'stream_tweets'), o conexiune și trimite prin aceasta un vector de bytes ce reprezintă o parte dintr-un tweet. Domeniul aplicației ne permite să folosim doar o parte din nenumăratele atribute ale unui tweet. Mai exact, avem nevoie doar de text și de coordonatele geografice reprezentant punctul de unde a fost trimis acel tweet.

Aceste funcții fac parte dintr-un script Python care la execuție face bind și listen unui socket pe localhost, port 9999. Când primește o cerere, acceptă și trimite prin conexiune, folosind funcțiile descrise mai sus, tweeturi către entitatea care face cerere.

A doua parte a procesului se referă la această entitate. Ea este un `DataStream` creat de un `StreamingContext` din biblioteca `pyspark.streaming`. `DataStream`ul este descris de un `socketTextStream` ce face cereri la serverul numit mai sus. Pe stream aplicăm o funcție pe datele live pentru a face split

```
ssc = StreamingContext(sc, 2)
dataStream = ssc.socketTextStream("localhost", 9999)
tweets = dataStream.flatMap(lambda line: line.split("\n"))
tweets.foreachRDD(print_with_location_rdd_with_prediction)
ssc.start()
ssc.awaitTermination()
```

între tweeturi. Apoi pentru fiecare tweet afișăm și salvăm într-un fișier, pentru fiecare tweet, textul, locația precum și predicția făcută de clasificatorul de spam.

Am folosit partea de Streaming din Apache spark pentru a avea acces la utilizarea în timp real a unor funcții peste niste date. Aici folosesc funcția 'foreachRDD' care, după cum reiese și din denumire, apelează o anumită funcție pe fiecare RDD din datastream.

Funcția 'print_with_location_rdd_with_prediction' afișează și salvează tweeturile astfel: folosind pe întregul tweet funcția 'loads' din biblioteca json, putem încărca un dicționar dintr-un string. Pentru a putea clasifica textul unui tweet ca fiind ori spam, ori non-spam, avem nevoie să împartim

stringul in mai multe substring-uri.

Acest proces se numeste tokenizare, iar pentru acest task se poate folosi foarte utila biblioteca creata pentru exact acest scop. Obiectul de tip TweetTokenizer din biblioteca nltk.tokenize imparte un string in tokenuri avand in vedere si structura unui tweet. Parametrii 'strip_handles' si 'reduce_len' folositi in constructorul acestei clase indeamna tokenizerul sa reduca dimensiunea tweetului daca este posibil. Acest lucru are loc daca se repeta foarte multe litere intr-un cuvant. Acest lucru este destul de des folosit, deoarece utilizatorii ar putea accentua anumite cuvinte prin repetarea unor litere.

De asemenea, referintele la alti utilizatori nu sunt incluse. Spre exemplu, textul '@remy: This is waaaaayyyy too much for you!!!!!!' este tokenizat si returnat ca intr-o lista astfel: [':', 'This', 'is', 'waaayyy', 'too', 'much', 'for', 'you', '!', '!', '!'].

Mai departe, predictia nu poate fi facuta pe niste simple cuvinte, astfel ca apelam la un procedeu numit feature hashing. Pentru fiecare term, insemnand cuvant, se calculeaza hashul acestuia si numarul de aparitii a cuvintului in text si aceste rezultate sunt pastrate intr-un SparseVector. Modelul de clasificare are nevoie de acest vector rar pentru a face predictia. Acest procedeu este detaliat in capitolul referitor la Clasificare.

Cat timp serverul detaliat in prima parte ramane pornit, acest al doilea script primeste tweeturi, le clasifica si le scrie intr-un fisier pentru a fi apoi folosit de celelalte scripturi.

In final, vom avea date care arata in felul urmator si pe care se poate lucra foarte usor spre a fi clusterizate dupa subiect si cartografiate estetic, scotand in evidenta tweeturile care au in componenta un anumit termen dat ca input.

	A	B	C
1	text	label	location
2	Heavy traffic in #Hillsborough on I-4 WB from Branch Forbes Rd to McIntosh Rd, incident	0	[-82.187, 28.02682]
3			
4	We're #hiring! Read about our latest #job opening here: Environmental Protection Assist	0	[-85.7584557, 38.2526647]
5			
6	Can you recommend anyone for this #job? Police Officer - https://t.co/RchjmUbfHV #se	0	[-98.3420118, 40.9263957]

Illustration 5

2 Clasificare

2.1 Descrierea problemei

În statistica și învățare automată, clasificarea este problema identificării unei categorii din care face parte o nouă observație, pe baza unui set de date de antrenare, unde se cunoaște categoria pentru fiecare observație. În învățare automată, aceste observații se numesc instanțe, clasificarea fiind un procedeu de învățare supervizată. Un algoritm de învățare supervizată (supervised learning) analizează datele din setul de antrenare și inferează o funcție care poate fi folosită pentru maparea noilor instanțe. Un algoritm care implementează acest lucru se numește clasificator și deseori în statistica se folosește regresia logistică.

Această parte a aplicației are menirea de a clasifica binar un tweet, fiind categorisit fie ca fiind spam sau non-spam. O foarte mare importanță pentru a crea cu succes un clasificator de spam o are utilizarea unui dataset relevant și destul de extins. Cealaltă necesitate este aceea de a alege metoda cea mai potrivită pentru datele obținute. Câteva din metodele cele mai populare de clasificare binară sunt:

- arbori de decizie
- random forests
- support vector machines
- regresie logistică
- rețele bayesiene

Din cele enumerate mai sus, Apache Spark are implementări pentru toate, dar o parte din metode nu sunt potrivite pentru arhitectura taskului necesar de îndeplinit. Implementarea algoritmilor de arbori de decizie și random forests au nevoie de prea multă memorie, deoarece spațiul problemei este foarte extins. Numărul de features de care se folosesc acești algoritmi este de 2^{18} . Acest număr survine din necesitatea de a nu avea coliziuni când se face Hashing pe termi. Vocabularul limbii engleze fiind foarte bogat, acest lucru trebuie reflectat și în numărul maxim de features folosit în Hashing, deoarece fiecare cuvânt să aibă un hash unic și, deci, clasificarea și calculul erorii la clasificare să fie unul relevant.

Problema alegerii unui algoritm potrivit este destul de trivială, deoarece aceștia sunt foarte ușor de folosit, implementările fiind disponibile în framework. Multe din aceste clasificatoare se utilizează după un tipar asemănător. Se instantiază un obiect de tipul algoritmului, se antrenează acest algoritm pe

datele de antrenare si apoi se testeaza pe datele de testare. Comparand acuratetea rezultata pentru fiecare algoritm, il alegem pe cel cu acuratetea cea mai buna.

Problema gasirii unui dataset relevant este ceva mai dificila, deoarece avem nevoie de date care sa fie deja etichetate corect. Acest lucru poate fi facut manual, desigur, dar pentru foarte multe instante acest lucru devine impracticabil.

2.2 Incercari ulterioare

Initial, doream sa rezolv problema propagandei si miscarilor cu tente de instigare la instabilitate in Statele Unite ale Americii creata de utilizatori cu conturi false provenind din Rusia. Acest lucru parea facil la inceput, deoarece exista un dataset publicat de NBCNews [1] foarte interesant cu 200.000 de tweeturi din 2016 aparinand unor useri malitiosi ce doreau sa creeze instabilitate politica si sociala in randul cetatenilor, pentru a descreste popularitatea capitalismului si eventual pentru a impinge balanta sanselor castigarii alegerilor prezidentiale in favoarea unui participant sau altul.

Avand atat de multe date cu tweeturi fake, aveam nevoie sa gasesc un dataset cu tweeturi ale unor useri de buna credinta si cu continut relevant si curat. Acest lucru s-a dovedit a fi fiind destul de dificil deoarece majoritatea dataseturilor urmaresc ori un subiect anume, ori tweeturi cu continut negativ, astfel ca dataseturile cu tweeturi ce discuta subiecte aleatoare in mod non-spam sunt putine. De asemenea, folosind un dataset non-spam cu subiecte non-politice ar fi dus la o falsă foarte buna acuratete la testare, deoarece, antrenand algoritmul pe doua dataseturi cu topicuri diferite, adica unul cu materiale politice spam si unul cu materiale non-politice non-spam se ajunge la o clasificare a subiectului tweetului si nu neaparat a apartenentei la o categorie spam sau non-spam. Acest lucru s-a si intamplat de altfel cu un dataset de genul acesta. In incercarea de a folosi un alt dataset de tweeturi non-spam, dar politice, am ajuns la concluzia ca in cazurile reale, clasificatorul dadea dovada de un comportament de underfitting, clasificand toate tweeturile non-politice ca fiind non-spam, iar cele cateva tweeturi politice intalnite ca fiind aleatoriu spam sau non-spam. Acesta lucru se datoreaza naturii datasetului propus de NBCNews, tweeturile continute in acesta fiind aproape imperceptibil de asemanatoare cu tweeturile politice si non-spam obtinute pe parcurs.

Astfel, a trebuit sa renunt la incercarea de a rezolva problema clasificarii tweeturilor de propaganda sau instigare la instabilitate politica deoarece textul din acele tweeturi nu ofera destule informatii relevante. In acest impas se afla si mari organizatii, guvernamentale sau nu, si deci ramane o preocupare deschisa pentru viitor.

2.3 Solutia

Reluand analiza imaginii de ansamblu, am ajuns la concluzia ca solutia ideala este folosirea unui dataset cu tweeturi cu subiecte aleatoare, etichetate cu spam sau non-spam. Twitter este o platforma in care oricine poate avea o voce referitoare la orice, acest lucru ducand la o impresionanta diversitate a subiectelor abordate. Pentru domeniul de lucru al acestei aplicatii, care este gasirea unor subiecte bine definite in aceasta mare de tweeturi aleatoare, clasificatorul nostru trebuie sa ne permita se renuntam la acele tweeturi care nu ar avea nicio relevanta pentru niciun subiect. Ne referim aici la tweeturi fara sens, cu caractere iligibile, enunturi incorrigibile, vulgaritate fara menire, reclame si vanzari de factura malitioasa, vouchere, phishing, spam.

Un studiu facut pe acest domeniu, de analiza a detectiei tweeturilor cu continut de slaba calitate [2] pune la dispozitie un dataset cu 100.000 de instante etichetate, tweeturi de continut aleatoriu, fie de slaba sau inalta calitate. Acest fisier de tip CSV contine doar ID-ul tweetului si eticheta acestuia, incat, in mod oficial, dataseturile mari de tweeturi nu pot fi distribuite in mod public, cu textul si celelalte attribute in plaintext.

Pentru a extrage tweetul folosind API-ul Twitter, avand la dispozitie ID-ul tweetului, avem nevoie de o functie care utilizeaza key-urile descrise in primul capitol.

```
def get_tweet_from_id(id):  
    url = "https://api.twitter.com/1.1/statuses/show.json?id=" + str(id)  
    response = requests.get(url, auth=auth_worker, stream=True)  
    for line in response.iter_lines():  
        my_full_tweet = json.loads(line)  
    return my_full_tweet
```

Mai departe, se parseaza CSV-ul oferit in articolul [2] si se apeleaza aceasta functie pentru fiecare ID de acolo. Pentru fiecare tweet care este inca valabil, adica a caror continut returnat nu incepe cu 'error', se apeleaza o functie care adauga un JSON intr-un CSV, pentru usurinta folosirii ulterioare.

Valabilitatea tweeturilor depinde de sansa; articolul de unde provine datasetul fiind publicat in 2017, exista posibilitatea ca unele din acestea sa fi fost sterse de pe Twitter. Acest fapt este unul foarte extins, dar din fericire nu unul complet. Din 100.000 de tweeturi totale, circa 1214 tweeturi cu label-ul 'low-quality' sunt valabile, iar cele cu label-ul 'not low-quality' sunt in numar de 15942. Pentru un dataset balansat, se vor folosi aproximativ acelasi numar de instante pentru fiecare categorie, fiind destule observatiile in numar de aproximativ 1200.

In final, distributia datasetului se face in doua fisiere , cate unul pentru fiecare categorie. Deoarece exista diferente intre encodingul acestor fisiere si encodingul acceptat de interpretorul Python, fara ca sa existe caractere iligibile, trebuie folosita o functie care curata datasetul.

De asemenea, pe Twitter exista conceptul de retweeting care permite utilizatorilor sa distribuie pe propriul cont anumite postari ale altor persoane. Pentru a semnala acest lucru, Twitter adauga un substring de forma “RT @utilizator_cu_postarea_originala: “ respectivului tweet. Avem nevoie sa eliminam acest tip de substring din orice tweet ce il contine.

```
def printAndSaveTweetTextFromCsv(file):
    newfile = open(os.path.splitext(file)[0] + "_cleanLOWERCASE.txt", 'w', encoding='ascii')
    with open(file, encoding='latin-1') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            normal = [x.lower() for x in row['text'].split()]
            normal_stringed = ' '.join(map(str, normal))
            cleaned = unicodedata.normalize('NFKD', normal_stringed).encode('ascii', 'ignore')
            tokenized = tokenizer.tokenize(cleaned)
            if len(tokenized) > 0:
                cut = tokenized[2:]
                full = tokenized
                #print('cut if tokenized[0] == "RT" else full')
                newfile.write(' '.join(cut)) if tokenized[0] == "rt" else newfile.write(' '.join(full))
            newfile.write('\n')
```

Pe langa acest lucru, functia “printAndSaveTweetTextFromCsv” tokenizeaza tweeturile folosind biblioteca nltk.tokenize, procedeu descris in primul capitol, transpune orice caracter din tweet in echivalentul lowercase si codifica totul din format latin-1 in format ascii normalizand unicode in format NFKD cu funtia 'normalize' din biblioteca unicodedata, pentru a elimina caracterele iligibile datorata codificarii utilizate de Twitter. Noile tweeturi sunt salvate intr-un nou fisier de tip text cu un anumit sufix.

Fisierele rezultate sunt citite si incarcate in memorie cu functia textFile a unui obiect de tip SparkContext.

```
fake = sc.textFile("a1newCSVFullTweets_cleanLOWERCASE.txt")
real = sc.textFile("a0newCSVFullTweets_cleanLOWERCASE.txt")
```

Fiecare tweet este transformat intr-o lista de cuvinte folosind functia 'map' din Python.

```
fake_words = fake.map(lambda sentence: sentence.split())
real_words = real.map(lambda sentence: sentence.split())
```

```
[['same', 'https://t.co/1ghdvqvzrc']]
[['heads', 'low', 'hopes', 'high', '~']]
```

Exemplu de tweeturi low-quality (sus) si non-low-quality (jos)

Se hash-uieste fiecare term si pentru fiecare tweet va rezulta un SparseVector, un vector rar, ce contine numarul de feature-uri (in fiecare caz, 2*18), hash-ul fiecarui term si numarul de aparitii a acestuia.

```
tf = HashingTF(numFeatures=2**18)
fake_features = tf.transform(fake_words)
real_features = tf.transform(real_words)
```

```
[SparseVector(262144, {179060: 1.0, 232159: 1.0})]
[SparseVector(262144, {5995: 1.0, 18426: 1.0, 100779: 1.0, 162531: 1.0, 170314: 1.0})]
```

Vectorul de features pentru tweeturile de mai sus.

Folosind LabeledPoint din pyspark.mllib.regression, putem adauga eticheta pentru fiecare astfel de SparseVector, respectiv eticheta 1 pentru low-quality si 0 altfel.

Folosind functia randomSplit, impartim aleatoriu setul de date in set de date de antrenare si set de date de testare. Datele de antrenare vor reprezenta 80% din total, iar datele de testare vor reprezenta 20% din total.

Mai departe, putem deja antrena si testa un model astfel.

```
algorithm = LogisticRegressionWithLBFGS()
model = algorithm.train(training_data)
print('logistic regression with lbfgs:', score(model))
```

Functia 'score' calculeaza acuratetea modelului cu urmatoarea formula. [3]

Accuracy (ACC)

$$ACC = (TP + TN) / (P + N)$$

```
def score(model):
    features = []
    for element in test_data:
        features.append(element.features)

    predictions = model.predict(features)

    labels = []
    for element in test_data:
        labels.append(element.label)

    labels_with_predictions = zip(labels, predictions)

    elements_gotten_right = []
    for element in labels_with_predictions:
        if element[0] == element[1]:
            elements_gotten_right.append(element)
    return len(elements_gotten_right) / float(len(test_data))
```

Aceasta formula reprezinta raportul dintre instantele True Positives + True Negatives si Positives + Negatives, adica numarul de instante a caror predictie a fost corecta supra totalul instantelor. O predictie true positive descrie o instanta pozitiva a carei predictie a fost calculata ca fiind pozitiva. O predictie true negative descrie o instanta negativa a carei predictie a fost calculata ca fiind negativa.

In total au fost testate 6 modele, dintre care 2, decision tree si random forests aveau nevoie de prea multa memorie datorita numarului de features prea mare. Celelalte patru au oferit rezultate interesante si destul de apropiate.

```
logistic regression sgd: 0.9186079953983319
logistic regression with lbfgs: 0.913718723037101
naive bayes: 0.9350014380212827
svm with sgd: 0.9194708081679609
```

Acuratetea modelelor

Surprinzator, algoritmul naiv al lui Bayes ofera acuratetea cea mai mare dintre cele 4, deci acesta ramanand a fi folosit pentru a clasifica binar apartenenta unui tweet la una dintre categoriile 'low-quality' si 'non-low-quality'. Clasificatorul este folosit in cadrul scriptului ce se ocupa cu colectarea datelor. Fiecare tweet este adaugat intr-un fisier CSV ce contine textul, locatia de unde a fost trimis tweetul si eticheta pusa de clasificator.

3 Clusterizare

3.1 Definitie

Clusterizarea este metoda de invatare nesupervizata ce are ca scop gruparea unor obiecte astfel incat instantele din acelasi grup, numit cluster, sa fie mai asemanatoare intre ele decat fata de alte instante din alte clustere.

“Este unul din obiectivele principale ale minării de date si o tehnica comuna in analiza statistica datelor. Se foloseste in multe domenii, cum ar fi machine learning, recunoasterea pattern-elor, analiza imaginilor, bioinformatica, compresia datelor si grafica pe calculator.” [4]

3.2 Tipuri de clustere

Exista multi algoritmi ce se ocupa cu clusterizarea unor date, deoarece exista multe interpretari a ceea ce poate insemna un cluster sau cum se poate crea si modela un cluster in cadrul implementarilor

Modelele de cluster pot fi urmatoarele:

- modele de conectivitate
- modele bazate pe centroizi
- modele de distributie
- modele de densitate
- modele subspatiu
- modele de grup
- modele bazate pe grafuri
- modele neurale

Pentru fiecare din aceste modele exista numeroare exemple de algoritmi. De exemplu, pentru modelele de clustere bazate pe conectivitate, exista clusterizare ierarhica, ce urmareste sa cladeasca o ierarhie de clustere. De obicei, este un algoritm greedy ce poate fi de tip aglomerativ, “bottom up”, (unde orice observatie porneste in propriul cluster si perechi de cluster se imbina o data ce un cluster urca in ierarhie) sau diviziv, “top down” (unde toate observatiile pornesc intr-un singur cluster si se efectueaza splituri incepand cu acest cluster), iar rezultatele clusterizarii, adica ierarhia sunt descrise intr-o dendrograma.

Un algoritm ce lucreaza cu clusteri din modelul bazat pe centroizi este algoritmul k-means, ce reprezinta un cluster ca fiind un vector de elemente ce au un centroid, centrul acelor instante, descris ca medie a pozitiilor fiecarui element.

In modelul de distributie, clusterelor sunt descrise folosind distributii statistice, cum ar fi distributii normale multivariate, in cadrul algoritmului EM (expectation-maximization), care este o metoda iterativa de a gasi parametrii unui model statistic.

Modelele de densitate caracterizeaza clusterelor ca fiind niste zone dense de regiune in spatiul datelor si sunt utilizate in algoritmii DBSCAN si OPTICS.

Modelele subspatiu sunt folosite in biclusterizare, unde clusterelor sunt modelate si cu membrii clusterelor, si cu attributele relevante.

Algoritmii ce folosesc modelele cu grupuri nu produc doar informatia despre cum se face gruparea, si nu un model rafinat pentru rezultate.

„Clusterizarea, sau analiza de tip cluster, nu se refera la un algoritm specific, ci la obiectivul general ce trebuie atins. Acesta poate fi indeplinit de diferiti algoritmi ce difera destul de mult intre ei, prin prisma faptului ca clusterelor pot fi create si interpretate foarte diferit, in functie de implementare.

Anumite interpretari ale clusterelor include grupuri cu distante mici intre membrii clusterelor, arii dense in spatiul datelor, intervale sau distributii statistice particulare. Deci, metoda de clustering poate fi formulata ca o problema de optimizare cu mai multe obiective. Algoritmii potriviti si parametrii alesi depind de datasetul problemei si utilizarea rezultatelor. Clusterizarea nu este deci o sarcina automata, ci un proces iterativ de knowledge discovery (procesul automatizat de cautare a tiparelor in volume mari de date) sau optimizare interactiva cu mai multe obiective ce implica mai multe incercari.”

3.3 Setul de date

Pentru a intelege cum clusterizeaza un anumit model, am avut nevoie de un anumit set de date cu anumite subiecte. S-a ajuns la un set de date ce contine 4 subiecte diferite, unul referitor la imigrarea in cadrul Canadei, unul la dezastre naturale sau umane, unul la produse din sfera tehnologiei si unul referitor la un produs medical numit Claritin. Exista in jur de 1500 de instante pentru fiecare din acest subiect.

3.4 Preprocesare

Pentru a preprocesa si apoi a clusteriza datele se foloseste un pipeline de invatare automata. Acesta reprezinta o serie de transformari ce pot fi aplicate pe instante folosind diversi algoritmi.

Exista 3 serii de transformari ce sunt aplicate pe setul de date inainte ca acestea sa fie clusterizate. Acestea sunt tokenizarea, eliminarea cuvintelor foarte comune, crearea unui vector de trasaturi cu marime fixa si apoi calcularea frecventa inversa per document, adica o masura numerica ce descrie cata informatie ofera un term (un cuvant).

Toate aceste obiecte se importa din biblioteca `pyspark.ml.feature`.

Tokenizarea se face folosind obiectul `Tokenizer` si imparte un string intr-o lista de termi.

Eliminarea cuvintelor foarte comune se face folosind `StopWordsRemover` [7]. Un exemplu a efectului folosirii acestei functii este transformarea listei `[I, saw, the, red, balloon]` in lista `[saw, red, balloon]`.

HashingTF a fost descris in capitolul referitor la clusterizare. Frecventa inversa a documentelor (IDF), impreuna cu frecventa termenilor (TF prin functia `hashingTF`) alcatuiesc o metoda de vectorizare a trasaturilor foarte des folosit in minarea datelor text pentru a reflecta importanta unui term intr-un document in setul de date. Definim un term ca fiind t , un document ca fiind d , si datasetul ca fiind D . Frecventa termilor, $TF(t,d)$, este numarul de aparitii a unui term t in in documentul d , in timp ce frecventa document, $DF(t,D)$ este numarul de documente ce contin termul t . Daca se foloseste doar frecventa termenilor pentru a masura importanta, este foarte usor sa se exagereze importanta unor cuvinte ce apar foarte des in dataset. IDF este o masura numerica pentru a descrie cata informatie ofera un anumit term.

$$IDF(t,D) = \log \frac{|D| + 1}{DF(t,D) + 1}$$

$|D|$ este numarul total de documente din corpus (dataset). Din moment ce se foloseste logaritmul, atunci daca un term apare in toate documentele, valoarea sa IDF va fi 0. TF-IDF este produsul dintre TF si IDF.

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D).$$

Un pipeline de preprocesare si procesare se construiesc si foloseste astfel:

```
#preprocesare
tokenizer = Tokenizer(inputCol="text", outputCol="tokens")
remover = StopWordsRemover(inputCol="tokens", outputCol="swrTokens")
hashingTF = HashingTF(inputCol="swrTokens", outputCol="rawFeatures", numFeatures=
2 **18)
idf = IDF(inputCol="rawFeatures", outputCol="features")
#modelul de clustering
kmeans = KMeans(k=8, seed=1 ,featuresCol='features' ,maxIter=10
,initMode='random')
#crearea pipeline-ului
pipeline = Pipeline(stages=[tokenizer, remover, hashingTF, idf])
#folosirea acestuia
model = pipeline.fit(dataframe)
results = model.transform(dataframe)
```

3.5 Modelul creat

Pentru a rezolva problema clusterizarii tweeturilor dupa asemanare a fost creat un model bazat pe algoritmul K-Means. Acesta a oferit cele mai bune rezultate. Celelalte optiuni incercate au fost bazate pe Latent Dirichlet allocation si Gaussian Mixture.

LDA

“Latent Dirichlet allocation (LDA) este un model de topic ce infereaza subiecte din colectii de documente text. LDA poate fi privit ca un algoritm de clustering astfel:

- subiectele corespund centrelor de clustere si documentele corespund liniilor din date
- subiectele si documentele exist in spatiul trasaturilor, unde vectorii de trasaturi sunt vectori de numarul de aparitii a cuvintelor
- LDA nu clusterizeaza folosind o distanta in sensul uzual, ci foloseste o functie bazata pe un model statistic ce descrie cum sunt generate documentele”[6]

Modelul bazat pe LDA nu a oferit rezultate satisfacatoare, deoarece instantele din cele patru grupuri de subiecte din datasetul folosit erau raspandite neuniform in clustere. Clusterul de care apartine fiecare instanta poate fi calculat din distributia subiectelor a respectivei instante. Distributia subiectelor reprezinta un vector de probabilitati a unei instante de a apartine unui anumit cluster. Am atribuit fiecarei instante clusterul unde instanta avea cea mai mare probabilitate de a apartenenta. Rezultatele sunt de asa natura incat un cluster nu contine doar instante dintr-un singur grup de subiecte din cele 4. Un alt rezultat dezirabil ar fi fost acela unde un cluster poate contine instante din mai multe grupuri, dar toate instantele unui grup sa se afle intr-un singur cluster. Nici acesta nu a fost atins.

Gaussian Mixture

“Un model Gaussian Mixture reprezinta o distributie compusa unde instantele sunt create dintr-una din cele k subdistributii Gaussiane, fiecare cu propria probabilitate. Implementarea din framework

foloseste algoritmul Expectation-Maximization (EM) pentru a induce modelul cu probabilitatea maxima fiind dat un set esantion.”[6]

Modelul a esuat in a fi testat deoarece la antrenare acesta ramane foarte repede fara destula memorie, spatiul de memorie heap al Java umplandu-se. Incercarile au fost efectuate si local, unde spatiul heap a fost marit, si folosind un serviciu cloud, Azure, unde acesta a fost marit si mai mult. Ambele incercari au dus la esec.

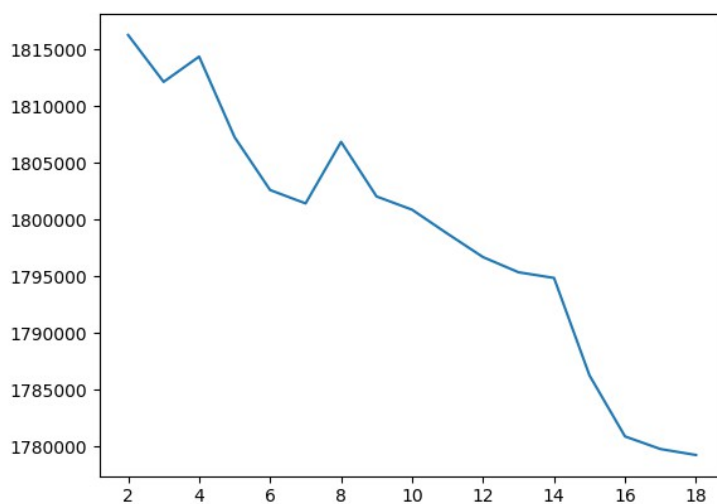
K-Means

“Clusterizarea K-Means este o metoda de cuantizare vector ce are ca scop partitionarea a n observatii in k clustere, unde fiecare observatie apartine clusterului cu media cea mai apropiata.”

O importanta problema in a folosi K-Means este deciderea numarului de clustere ce va fi folosit. Pentru aceasta, s-a folosit o metrica numita Within Set Sum of Squared Errors (WSSSE). Aceasta calculeaza suma, pentru intregul set, a distantelor dintre o instanta si centroidul clusterului de care apartine astfel:

```
for k in range(2, 19):  
    kmeans = KMeans().setK(k).setSeed(1)  
    model = kmeans.fit(results)  
    wssse = model.computeCost(results)
```

Pe datele initiale, graficul acestei functii, avand numarul de clustere pe abscisa si WSSSE pe ordonata, este descris in Grafic 1.



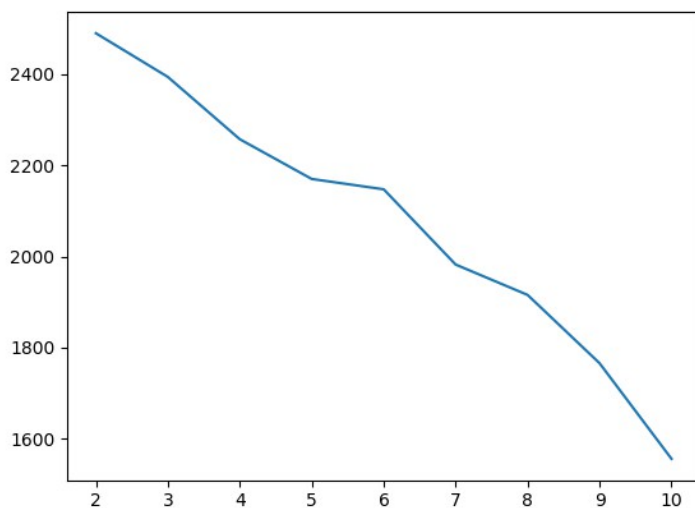
Grafic 1

Numarul de clustere se alege folosind metoda “cotului”. Punctul in care pe grafic se creeaza un contur ce se aseamana cu un “cot”, adica ce formeaza un unghi ascutit, si de unde mai departe nu scade drastic eroarea este punctul ce desemneaza numarul optim de clustere.

Cum in acest grafic nu exista un astfel de punct, numarul de clustere s-a ales dupa mai multe incercari si dupa ce s-a observat ca 8 clustere ofera o grupare logica a celor 4 subiecte din setul de date. De exemplu, un cluster contine toate instantele ce se refera la Canada si Claritin, un cluster contine instantele referitoare la produse din sfera tehnologiei si un cluster contine instante referitoare la dezastre. Celealte clustere contin extrem de putine instante (sub 10 fiecare) ce pot fi considerate neglijabile.

Pentru un alt dataset, colectat in cadrul primului modul al aplicatiei, cu valori foarte recente si ce nu contin un numar cunoscut de subiecte, graficul nu contine un punct ce sa descrie numarul de clustere potrivite, fapt ce sugereaza ca aceasta metoda de calcul al costului nu este perfecta, dar ofera informatii cu privire la eroarea modelului pentru un anumit set de date.

Tot pentru 8 clustere, modelul ofera o clusterizare acceptabila, ce situeaza majoritatea tweeturilor ce se refera la locatia curenta a unui utilizator (cele ce incep cu textul „I'm at”) intr-un singur cluster.



Grafic 2
WSSSE si K pentru date recente

3.6 Concluzie

Acest modul se ocupa cu preprocesarea datelor si clusterizarea acestora folosind un pipeline de invatare automata. Preprocesarea tweeturilor consta in tokenizarea, eliminarea cuvintelor foarte comune in vocabularul limbii engleze si aplicarea metode de vectorizare a trasaturilor TF-IDF. Modelul de clustering este bazat pe algoritmul K-Means.

4 Cartografiere

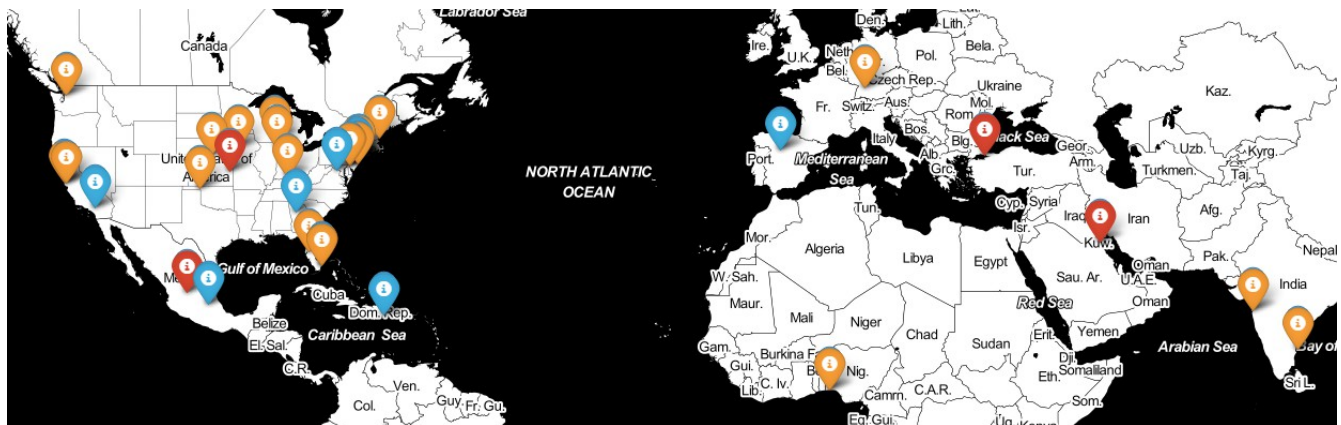
4.1 Folium

Vizualizarea datelor adunate in celelalte module se face cu ajutorul bibliotecii 'folium'. Aceasta isi propune sa foloseasca abilitatile de mapare a datelor din biblioteca Leaflet.js pentru a afisa date pe harta.

4.2 Afisare pe harta

Acest modul isi propune sa afiseze markere din biblioteca folium pe harta. Fiecare marker reprezenta un tweet si va fi insotit de clusterul de care apartine, fiind desenat pe harta in concordanta cu locatia de unde a fost trimis tweetul.

Se ofera un input ce reprezinta termenul de interes al utilizatorului. Se cauta clusterul cu cele mai multe aparitii ale acelui termen si se coloreaza in mod evident si special toate instantele din respectivul cluster ce contin in text termenul cautat. Daca este cazul, celelalte instante din cluster care nu contin termenul cautat vor fi colorati asemanator, dar totusi diferit de instantele ce contin respectivul termen. Toate celelalte instante sunt colorate diferit, in functie de cluster.



Instantele colorate in rosu contin termenul "I'm", iar cele portocalii fac parte din acelasi cluster

4.3 Gasirea clusterului relevant

Pentru a gasi clusterul cu cele mai multe aparitii a unui termen se foloseste urmatoarea secventa de cod:

```
def getMaxCluster(keyword, k):
    clusters_and_count = {}
    for i in range(0, k):
        clusters_and_count[i] = 0
    with open('liveTweetsLocationKmeans.csv') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
```

```
if keyword in row['text']:
    clusters_and_count[int(row['prediction'])] += 1
max_cluster = max(clusters_and_count, key=clusters_and_count.get)
return max_cluster
```

5 Concluzii

Aplicatia de fata reuseste sa ajunga la rezultate interesante, putand fi considerata un clasificator de spam si un clusterizator de date. Aceste lucruri se realizeaza cu o acuratete satisfacatoare folosind implementari de algoritmi de invatare automata din frameworkul Apache Spark. Rezultatele finale pot fi folosite, spre exemplu, de jurnalisti pentru a infera stiri in functie de activitatea ecosistemului Twitter.

5.1 Directii de viitor

Pentru a dezvolta aceasta aplicatie se pot face mai multe lucruri. In primul rand se poate imbunatati acuratetea clasificarii prin folosirea unui dataset diferit, eventual cu mai multe instante si subiecte.

Pentru a imbunatati performanta si a profita pe deplin de capabilitatile de cluster computing din cadrul Apache Spark, aplicatia poate trece de la un mediu pseudodistribuit, cel local, la unul bazat complet in cloud. Se poate extinde utilizarea serviciilor oferite de Azure, limitata acum la rularea in cloud a unor module separate.

Bibliografie

- [1] <https://www.nbcnews.com/tech/social-media/now-available-more-200-000-deleted-russian-troll-tweets-n844731>
- [2] <http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0182487>
- [3] https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers
- [4] https://en.wikipedia.org/wiki/Social_network
- [5] <https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/>
- [6] <https://spark.apache.org/docs/2.2.0/mllib-clustering.html>
- [7] <https://spark.apache.org/docs/2.2.0/ml-features.html#stopwordsremover>