

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**Clasificarea, clusterizarea și 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 și 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*” 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
Motivație și gradul de noutate.....	5
Obiectivele lucrării.....	5
Tehnologia folosită	6
Descrierea sumară a soluției.....	9
Contribuțiile autorului.....	10
1 Colectarea datelor în timp real	11
2 Clasificare.....	15
2.1 Descrierea problemei.....	15
2.2 Încercări ulterioare.....	16
2.3 Soluția.....	17
3 Clusterizare.....	20
3.1 Definiție.....	20
3.2 Tipuri de clustere.....	20
3.3 Setul de date.....	22
3.4 Preprocesare.....	22
3.5 Modelul creat.....	23
LDA.....	23
Gaussian Mixture.....	23
K-Means.....	24
3.6 Concluzie.....	25
4 Cartografiere.....	26
4.1 Folium.....	26
4.2 Afișare pe hartă.....	26
4.3 Găsirea clusterului relevant.....	26
5 Concluzii	27
5.1 Direcții de viitor.....	27
Bibliografie.....	28

Introducere

Motivație și gradul de noutate

În ultimul timp, rețelele sociale online s-au dezvoltat enorm, plecând de la firavele începuturi lipsite de multe funcționalități și importanta și ajungând în ziua de azi să reprezinte structuri sociale și chiar politico-economice de uriașe proporții. O rețea socială online poate fi definită în contextul recentului fenomen numit Web 2.0 (care se referă la noua generație de website-uri unde accentul se pune pe ușurința folosirii și crearea de conținut de către utilizatori) [2] ca fiind un site web unde utilizatorii fac parte dintr-o structură socială și reprezintă „actori” sociali ce sunt conectați prin mai multe legături de tip „one-to-one”, creând astfel graful rețelei sociale. Printre cele mai populare rețele sociale online se numără Facebook, Instagram și Twitter, având 2234, 813 și respectiv 330 de milioane de utilizatori, potrivit unui recensământ din Aprilie 2018 [1].

Prin popularitatea lor, aceste rețele sociale devin foarte ușor ținta utilizatorilor malițioși ce au ca scop propagarea interacțiunilor și fenomenelor de tip spam și phishing. Acestea au ca obiectiv convingerea utilizatorilor firești, prin cai dăunătoare, deranjante și de obicei greu de descoperit, să își expună datele personale sau să piardă bani în favoarea celor care recurg la aceste tactici.

Pe lângă aceste practici malițioase, pe Twitter există și foarte multe postări irelevante pentru mulți din utilizatori. O dată postat un tweet, acesta ajunge pe feed-ul oricărui utilizator ce urmărește persoana ce postează. Acest lucru face ca uneori să existe o neconcordanță între doleanțele unui utilizator și ce citește acesta pe propriul feed.

Obiectivele lucrării

Unul dintre cele două principale obiective ale lucrării este acela de a studia în profunzime frameworkul de cluster computing numit 'Apache Spark'. Dintre componentele sale, cele mai relevante pentru implementarea unei aplicații folosind frameworkul acesta au fost studiate mai în profunzime și descrise în capitolele relevante fiecărui modul din aplicație. Spark Streaming este descris în primul capitol și Spark MLlib în capitolele 2 și 3.

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

tehnici de clasificare din învățare automată, explicate în al doilea capitol. De asemenea, se dorește crearea unor grupuri de tweeturi ce abordează subiecte asemănătoare și sunt de înaltă calitate, pentru a veni în ajutorul persoanelor ce sunt interesate de a 'lua pulsul' societății în care se afla, precum jurnaliștii, fără să fie nevoiți să sorteze printr-o mulțime de mesaje ce se pot dovedi a fi irelevante, astfel având parte de o experiență sigură și utilă.

Aceste grupuri de tweeturi vor fi afișate pe o hartă și vor putea fi căutați termeni ce se doresc a fi găsiți în grupuri special create după acei termeni, astfel creându-se anumite tipicuri ce sunt relevante pentru inputul unui utilizator. Cartografierea acestor tweeturi este utilă deoarece creează o perspectivă nouă și oferă o imagine de ansamblu ce poate descrie foarte ușor de unde a pornit un anumit fenomen social, fie el o știre, un zvon, o idee pentru o mișcare socială sau chiar un dezastru natural sau uman.

Tehnologia folosită

Pentru a atinge aceste obiective, adică de a clasifica binar un tweet, fie într-o categorie de conținut de calitate înaltă, fie o categorie de conținut de calitate scăzută, și de a grupa (în termeni de învățare automată, de a clusteriza) o colecție de tweeturi în mai multe subcolecții ce sunt asemănătoare între ele, iar apoi cartografia acestor tweeturi într-un format ușor de interpretat am apelat la biblioteca 'folium' din Python și frameworkul de cluster computing 'Apache Spark'.

Lansat în anul 2014, având ca autor inițial pe Matei Zaharia, în cadrul proiectului AMPLab al universității Berkeley, din California, acesta are la bază o abstractizare a datelor numită resilient distributed dataset (RDD). Și din denumire se poate infera că acesta descrie un multiset (un set ce poate conține mai multe instanțe ale aceluiași element) de date distribuite pe un cluster de computere.

În cadrul acestui framework există mai multe componente ce oferă diverse funcționalități:

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

- componenta Spark Streaming se ocupă cu analiza stream-urilor de date. Datele sunt aduse în memorie în mini-batch-uri (grupuri mai mici de date) și se pot efectua transformări RDD pe acestea

- componenta Spark MLlib oferă implementări ale unor algoritmi de învățare automată, capabili de calcul computațional distribuit

- componenta GraphX este un framework de procesare a grafurilor

Din toate acestea, cele utilizate în aplicație sunt Spark SQL, Streaming și MLlib, pentru citirea datelor de pe Twitter, aplicarea unor algoritmi de învățare automată și manipularea a unor DataFrames.

Mai în profunzime, Apache Spark este un framework open source destinat procesării unor volume de date la scara 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 numita “resilient distributed dataset” (RDD), ce reprezintă o mulțime, o colecție imutabilă de elemente distribuita pe un cluster de sisteme computaționale peste care se poate opera în paralel. Caracteristicile acestui tip de date sunt, după cum sugerează numele:

- Rezilient, exista posibilitatea de a recomputa partiții cu probleme
- Distribuit, datele se află pe mai multe noduri într-un cluster
- Este un dataset cu valori primite 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, stream-uri 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.



Fig. 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.



Fig. 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 MLlib pot învăța din stream-urile 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 stream-uri 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, stream-uri de date și alte metode.

Descrierea sumara a soluției

Aplicația se poate împărți în patru module ce lucrează împreună pentru a ajunge la rezultatul final, reprezentat de o colecție de tweeturi, împreună cu locația lor, ce au fost clasificate ca fiind de calitate înaltă, clusterizate în grupuri relevante, după asemănarea dintre ele și apoi afișate pe o harta ca puncte ce conțin textul tweetului, locația acestuia și grupul semantic cărui aparține.

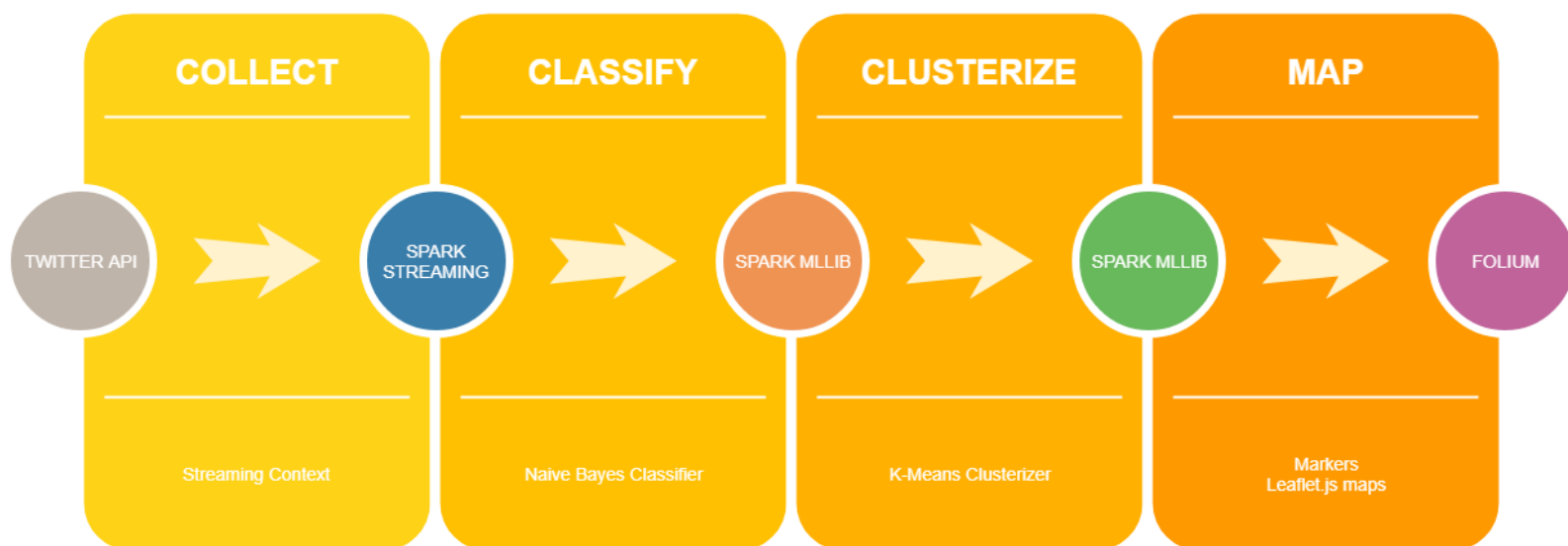


Fig. 3

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

Al doilea modul consta în scriptul Python ce a antrenat și testat diverși algoritmi de clasificare, implementați în Spark MLlib. Concluzia optimă după multiple încercări a fost utilizarea algoritmului Bayes naive pe un dataset cu aproximativ 1200 de instanțe de tweeturi de slabă calitate și aproximativ 10000 de instanțe de tweeturi de înaltă calitate. Acesta oferă o acuratețe de 93.5% la testare, una destul de apropiată de celelalte variante, cuprinse între 91% și 92%.

Al treilea modul este alcătuit din funcții ce pot clusteriza datele salvate în primele module și pot produce rezultate ce constau în asocierea fiecărui tweet cu un cluster.

Al patrulea modul se referă la randarea tweeturilor clusterizate în modulul precedent, afișându-le pe harta globului pământesc și oferind o imagine de ansamblu asupra modului de propagare și naștere a unor subiecte de interes major pe rețeaua de socializare Twitter.

Contribuțiile autorului

Cele patru module descrise mai sus au fost create în întregime de către autor folosind limbajul de programare Python și frameworkul Apache Spark și diverse alte biblioteci, cum ar fi Folium.

Descrierea tehnologiei Apache Spark s-a făcut ca urmare a studiului acesteia de către autor.

1 Colectarea datelor în timp real

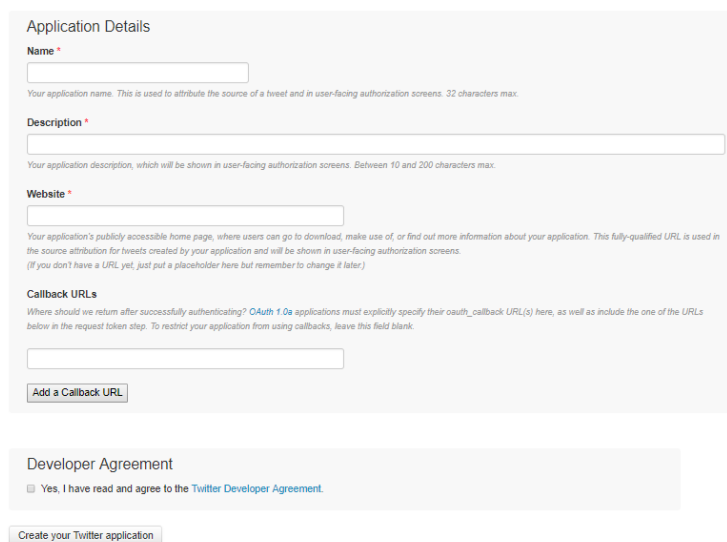
Orice aplicație ce include în implementarea ei și un modul de învățare automată are imperioasă nevoie sau cel puțin beneficiază foarte mult de pe urma unui dataset cât mai extins, dar și curat. Twitter, prin popularitatea sa imensă atinge cu excelență punctul referitor la cantitatea datelor, dar se îndepărtează 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.



The image shows a screenshot of the 'Create an application' form on the Twitter Developer Portal. The form is titled 'Application Details' and contains several input fields: 'Name' (with a 32-character limit), 'Description' (with a 10-200 character limit), 'Website' (with a note about OAuth 1.0a applications), and 'Callback URLs' (with a note about OAuth 1.0a applications). Below these fields is a button labeled 'Add a Callback URL'. At the bottom of the form is a 'Developer Agreement' section with a checkbox and a 'Create your Twitter application' button.

După completarea acestui formular, vom avea acces la un dashboard cu anumite informații. De acolo vom prelua patru coduri importante și necesare autorizării noastre la serviciul oferit.

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

```
ACCESS_TOKEN = 'sampleString1'
ACCESS_TOKEN_SECRET = 'sampleString2'
CONSUMER_KEY = 'sampleString3'
CONSUMER_SECRET = 'sampleString4'
auth_worker = requests_oauthlib.OAuth1(CONSUMER_KEY, CONSUMER_SECRET,
ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
```

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

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

Funcția 'stream_tweets' returnează un răspuns http ce conține, pentru fiecare linie, un tweet complet respectând parametrii dați în query. Mai exact, cautăm tweeturi scrise în limba engleză, iar parametrul locations astfel setat ne garantează că locația mesajelor provine din aproximativ partea continentală a Statelor Unite ale Americii.

Am ales această locație deoarece Twitter este foarte popular și utilizat în respectiva țară, fapt ce denotă o posibilă diversitate mai mare decât în alte locuri. Subiectele discutate sunt extrem de variate și ne pot oferi multe șanse de a obține date interesante și din prisma geolocației, un atribut ce îl are orice tweet al cărui utilizator permite accesul serviciilor de locație 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)
            tweet_text = full_tweet['text']

            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)
```

Functia '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, accepta și trimite prin conexiune, folosind funcțiile descrise mai sus, tweeturi către entitatea care face cerere.

A doua parte a procesului se referă la aceasta entitate. Ea este un DataStream creat de un StreamingContext din biblioteca pyspark.streaming. DataStreamul 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 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.

```
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()
```

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

Functia '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 împărțim stringul în mai multe substring-uri.

Acest proces se numește tokenizare, iar pentru acest task se poate folosi foarte utila bibliotecă creată pentru exact acest scop. Obiectul de tip TweetTokenizer din biblioteca nltk.tokenize împarte un string în tokenuri având în vedere și structura unui tweet. Parametrii 'strip_handles' și 'reduce_len' folosiți în constructorul acestei clase îndeamnă tokenizerul să reducă dimensiunea tweetului dacă este posibil. Acest lucru are loc dacă se repeta foarte multe litere într-un cuvânt. Acest lucru este destul de des

folosit, deoarece utilizatorii ar putea accentua anumite cuvinte prin repetarea unor litere.

De asemenea, referințele la alți utilizatori nu sunt incluse. Spre exemplu, textul '@remy: This is waaaaayyyy too much for you!!!!!!' este tokenizat și returnat ca într-o lista astfel: ['.', 'This', 'is', 'waaayyy', 'too', 'much', 'for', 'you', '!', '!', '!'].
Măi departe, predicția nu poate fi făcută pe niste simple cuvinte, astfel ca apelăm la un procedeu numit feature hashing. Pentru fiecare term, însemnând cuvânt, se calculează hashul acestuia și numărul de apariții al cuvântului în text și aceste rezultate sunt păstrate într-un SparseVector. Modelul de clasificare are nevoie de acest vector rar pentru a face predicția. Acest procedeu este detaliat în capitolul referitor la Clasificare.

Cat timp serverul detaliat în prima parte rămâne pornit, acest al doilea script primește tweeturi, le clasifică și le scrie într-un fișier pentru a fi apoi folosit de celelalte scripturi.

În final, vom avea date care arată în felul următor și pe care se poate lucra foarte ușor spre a fi clusterizate după subiect și cartografiate estetic, scoțând în evidență tweeturile care au în componență 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/RchjUbfHV #se	0	[-98.3420118, 40.9263957]

Fig. 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 și apoi se testează pe datele de testare. Comparând acuratețea rezultată pentru fiecare algoritm, îl alegem pe cel cu acuratețea cea mai bună.

Problema găsirii unui dataset relevant este ceva mai dificilă, deoarece avem nevoie de date care să fie deja etichetate corect. Acest lucru poate fi făcut manual, desigur, dar pentru foarte multe instanțe acest lucru devine impracticabil.

2.2 Încercări ulterioare

Inițial, doream să rezolv problema propagandei și mișcărilor cu tente de instigare la instabilitate în Statele Unite ale Americii creată de utilizatori cu conturi false provenind din Rusia. Acest lucru părea facil la început, deoarece exista un dataset publicat de NBCNews [5] foarte interesant cu 200.000 de tweeturi din 2016 aparținând unor utilizatori malițioși ce doreau să creeze instabilitate politică și socială în rândul cetățenilor, pentru a scădea popularitatea capitalismului și eventual pentru a împinge balanța șanselor câștigării alegerilor prezidențiale în favoarea unui participant sau altul.

Având atât de multe date cu tweeturi fake, aveam nevoie să găsim un dataset cu tweeturi ale unor utilizatori de bună credință și cu conținut relevant și curat. Acest lucru s-a dovedit a fi fiind destul de dificil deoarece majoritatea dataseturilor urmăresc ori un subiect anumit, ori tweeturi cu conținut negativ, astfel că dataseturile cu tweeturi ce discută subiecte aleatoare în mod non-spam sunt puține. De asemenea, folosind un dataset non-spam cu subiecte non-politice ar fi dus la o falsă foarte bună acuratețe la testare, deoarece, antrenând algoritmul pe două dataseturi cu topicuri diferite, adică unul cu materiale politice spam și unul cu materiale non-politice non-spam se ajunge la o clasificare a subiectului tweetului și nu neapărat a apartenenței la o categorie spam sau non-spam. Acest lucru s-a și întâmplat de altfel cu un dataset de genul acesta. În încercarea de a folosi un alt dataset de tweeturi non-spam, dar politice, am ajuns la concluzia că în cazurile reale, clasificatorul dădea dovadă de un comportament de *underfitting*, clasificând toate tweeturile non-politice ca fiind non-spam, iar cele câteva tweeturi politice întâlnite ca fiind aleatoriu spam sau non-spam. Acest lucru se datorează naturii datasetului propus de NBCNews, tweeturile conținute în acesta fiind aproape imperceptibil de asemănătoare cu tweeturile politice și non-spam obținute pe parcurs.

Astfel, a trebuit să renunț la încercarea de a rezolva problema clasificării tweeturilor de propagandă sau instigare la instabilitate politică deoarece textul din acele tweeturi nu oferă destule informații relevante. În acest impas se afla și mari organizații, guvernamentale sau nu, și deci rămâne o preocupare deschisă pentru viitor.

2.3 Soluția

Reluând analiza imaginii de ansamblu, am ajuns la concluzia ca soluția ideală este folosirea unui dataset cu tweeturi cu subiecte aleatoare, etichetate cu spam sau non-spam. Twitter este o platforma în care oricine poate avea o voce referitoare la orice, acest lucru ducând la o impresionanta diversitate a subiectelor abordate. Pentru domeniul de lucru al acestei aplicații, care este găsirea unor subiecte bine definite în aceasta mare de tweeturi aleatoare, clasificatorul nostru trebuie sa ne permită se renunțam la acele tweeturi care nu ar avea nicio relevanta pentru niciun subiect. Ne referim aici la tweeturi fără sens, cu caractere iligibile, enunțuri incorrigibile, vulgaritate fără menire, reclame și vânzări de factura malițioasă, vouchere, phishing, spam.

Un studiu făcut pe acest domeniu, de analiza a detecției tweeturilor cu conținut de slaba calitate [4] pune la dispoziție un dataset cu 100.000 de instanțe etichetate, tweeturi de conținut aleatoriu, fie de slaba sau înalta calitate. Acest fișier de tip CSV conține doar ID-ul tweetului și eticheta acestuia, încât, în mod oficial, dataseturile mari de tweeturi nu pot fi distribuite în mod public, cu textul și celelalte atribute în plaintext.

Pentru a extrage tweetul folosind API-ul Twitter, având la dispoziție ID-ul tweetului, avem nevoie de o funcție care utilizează key-urile descrise în 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 parsează CSV-ul oferit în articolul [4] și se apelează aceasta funcție pentru fiecare ID de acolo. Pentru fiecare tweet care este încă valabil, adică a căror conținut returnat nu începe cu 'error', se apelează o funcție care adaugă un JSON într-un CSV, pentru ușurința folosirii ulterioare.

Valabilitatea tweeturilor depinde de șansă; articolul de unde provine datasetul fiind publicat în 2017, exista posibilitatea ca unele din acestea sa fi fost șterse 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 în număr de 15942. Pentru un dataset balansat, se vor folosi aproximativ același număr de instanțe pentru fiecare categorie, fiind destule observațiile în număr de aproximativ 1200.

În final, distribuția datasetului se face în două fișiere, câte unul pentru fiecare categorie. Deoarece există diferențe între encodingul acestor fișiere și encodingul acceptat de interpretorul Python, fără ca să existe caractere iligibile, trebuie folosită o funcție care curăță datasetul.

De asemenea, pe Twitter există conceptul de retweeting care permite utilizatorilor să distribuie pe propriul cont anumite postări ale altor persoane. Pentru a semnaliza acest lucru, Twitter adaugă un substring de forma “RT @utilizator_cu_postarea_originala: ” respectivului tweet. Avem nevoie să eliminăm acest tip de substring din orice tweet ce îl conține.

```
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
                newfile.write(' '.join(cut)) if tokenized[0] == "rt" else
newfile.write(' '.join(full))
            newfile.write('\n')
```

Pe lângă acest lucru, funcția “printAndSaveTweetTextFromCsv” tokenizează tweeturile folosind biblioteca `nlTK.tokenize`, procedeu descris în primul capitol, transformă orice caracter din tweet în echivalentul lowercase și codifică totul din format latin-1 în format ASCII normalizând Unicode în format NFKD cu funcția ‘normalize’ din biblioteca `unicodedata`, pentru a elimina caracterele iligibile datorate codificării utilizate de Twitter. Noile tweeturi sunt salvate într-un nou fișier de tip text cu un anumit sufix.

Fișierele rezultate sunt citite și încărcate în memorie cu funcția `textFile` a unui obiect de tip `SparkContext`.

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

Fiecare tweet este transformat într-o listă de cuvinte folosind funcția ‘map’ din Python.

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

Se hash-uieste fiecare term și pentru fiecare tweet va rezulta un SparseVector, un vector rar, ce conține numărul de feature-uri (în fiecare caz, 2*18), hash-ul și numărul de aparatii al fiecărui term.

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

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

```
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 și 0 altfel.

Folosind funcția randomSplit, împărțim aleatoriu setul de date în set de date de antrenare și 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 și testa un model astfel.

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

```
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))
```

Funcția 'score' calculează acuratețea modelului cu următoarea formula. [3]

Accuracy (ACC)

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

Aceasta formula reprezintă raportul dintre instanțele True Positives + True Negatives și Positives + Negatives, adică numărul de instanțe a căror predicție a fost corectă supra totalul instanțelor. O predicție true positive descrie o instanță pozitivă a cărei predicție a fost calculată ca fiind pozitivă. O predicție true negative descrie o instanță negativă a cărei predicție a fost calculată ca fiind negativă.

În total au fost testate 6 modele, dintre care 2, decision tree și random forests aveau nevoie de prea multă memorie datorită numărului de features prea mare. Celelalte patru au oferit rezultate interesante și 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

Surprinzător, algoritmul naiv al lui Bayes oferă acuratețea cea mai mare dintre cele 4, deci acesta rămânând a fi folosit pentru a clasifica binar apartenența unui tweet la una dintre categoriile 'low-quality' și 'non-low-quality'. Clasificatorul este folosit în cadrul scriptului ce se ocupă cu colectarea datelor. Fiecare tweet este adăugat într-un fișier CSV ce conține textul, locația de unde a fost trimis tweetul și eticheta pusă de clasificator.

3 Clusterizare

3.1 Definiție

Clusterizarea este metoda de învățare nesupervizată ce are ca scop gruparea unor obiecte astfel încât instanțele din același grup, numit cluster, să fie mai asemănătoare între ele decât față de alte instanțe din alte clustere.

“Este unul din obiectivele principale ale minării de date și o tehnică comună în analiza statistică a datelor. Se folosește în multe domenii, cum ar fi machine learning, recunoașterea pattern-elor, analiza imaginilor, bioinformatică, compresia datelor și grafică pe calculator.” [8]

3.2 Tipuri de clustere

Există mulți algoritmi ce se ocupă cu clusterizarea unor date, deoarece există multe interpretări a ceea ce poate însemna un cluster sau cum se poate crea și modela un cluster în cadrul implementărilor

Modelele de cluster pot fi următoarele:

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

Pentru fiecare din aceste modele exista numeroase exemple de algoritmi. De exemplu, pentru modelele de clustere bazate pe conectivitate, exista clusterizare ierarhica, ce urmărește sa clădească o ierarhie de clustere. De obicei, este un algoritm greedy ce poate fi de tip aglomerativ, “bottom up”, (unde orice observație pornește în propriul cluster și perechi de cluster se îmbina o data ce un cluster urca în ierarhie) sau diviziv, “top down” (unde toate observațiile pornesc într-un singur cluster și se efectuează splituri începând cu acest cluster), iar rezultatele clusterizarii, adică ierarhia sunt descrise într-o dendrograma.

Un algoritm ce lucrează 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 pozițiilor fiecărui element.

În modelul de distribuție, clusterelor sunt descrise folosind distribuții statistice, cum ar fi distribuții normale multivariate, în cadrul algoritmului EM (expectation-maximization), care este o metoda iterativa de a găsi parametrii unui model statistic.

Modelele de densitate caracterizează clusterelor ca fiind niște zone dense de regiune în spațiul datelor și sunt utilizate în algoritmii DBSCAN și OPTICS.

Modelele subspatiu sunt folosite în biclusterizare, unde clusterelor sunt modelate și cu membrii clusterelor, și cu attributele relevante.

Algoritmii ce folosesc modelele cu grupuri nu produc doar informația despre cum se face gruparea, și 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 îndeplinit de diferiți algoritmi ce difera destul de mult între ei, prin prisma faptului ca clusterelor pot fi create și interpretate foarte diferit, în funcție de implementare.

Anumite interpretări ale clusterelor include grupuri cu distante mici între membrii clusterelor, arii dense în spațiul datelor, intervale sau distribuții statistice particulare. Deci, metoda de clustering poate fi formulată ca o problemă de optimizare cu mai multe obiective. Algoritmii potriviți și parametrii aleși depind de datasetul problemei și utilizarea rezultatelor. Clusterizarea nu este deci o sarcină automată, ci un proces iterativ de knowledge discovery (procesul automatizat de căutare a tiparelor în volume mari de date) sau optimizare interactivă cu mai multe obiective ce implică mai multe încercări.”

3.3 Setul de date

Pentru a înțelege cum clusterizează un anumit model, am avut nevoie de un anumit set de date cu anumite subiecte. S-a ajuns la un set de date ce conține 4 subiecte diferite, unul referitor la imigrarea în cadrul Canadei, unul la dezastre naturale sau umane, unul la produse din sfera tehnologiei și unul referitor la un produs medical numit Claritin. Există în jur de 1500 de instanțe pentru fiecare din acest subiect.

3.4 Preprocesare

Pentru a preprocesa și apoi a clusteriza datele se folosește un pipeline de învățare automată. Acesta reprezintă o serie de transformări ce pot fi aplicate pe instanțe folosind diverși algoritmi.

Există 3 serii de transformări ce sunt aplicate pe setul de date înainte ca acestea să fie clusterizate. Acestea sunt tokenizarea, eliminarea cuvintelor foarte comune, crearea unui vector de trăsături cu mărime fixă și apoi calcularea frecvenței inverse per document, adică o măsură numerică ce descrie câtă informație oferă un termen (un cuvânt).

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

Tokenizarea se face folosind obiectul `Tokenizer` și împarte un string într-o listă de termi.

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

`HashingTF` a fost descris în capitolul referitor la clusterizare. Frecvența inversă a documentelor (IDF), împreună cu frecvența termenilor (TF prin funcția `hashingTF`) alcătuiesc o metodă de vectorizare a trăsăturilor foarte des folosită în minarea datelor text pentru a reflecta importanța unui termen într-un document în setul de date. Definim un termen ca fiind t , un document ca fiind d , și datasetul ca fiind D . Frecvența termenilor, $TF(t,d)$, este numărul de apariții a unui termen t în documentul d , în timp ce frecvența documentelor, $DF(t,D)$ este numărul de documente ce conțin termenul t . Dacă se folosește doar frecvența termenilor pentru a măsura importanța, este foarte ușor să se exagereze importanța unor cuvinte ce apar foarte des în dataset. IDF este o măsură numerică pentru a descrie câtă informație oferă un anumit termen.

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

$|D|$ este numărul total de documente din corpus (dataset). Din moment ce se folosește logaritmul, atunci dacă un termen apare în toate documentele, valoarea sa IDF va fi 0. TF-IDF este produsul dintre TF și IDF.

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

Un pipeline de preprocesare si procesare se construiește si folosește 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 opțiuni încercate au fost bazate pe Latent Dirichlet allocation si Gaussian Mixture.

LDA

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

- subiectele corespund centrelor de clustere si documentele corespund liniilor din datase
- subiectele si documentele exist in spațiul trăsăturilor, unde vectorii de trăsături sunt vectori de numărul de apariții a cuvintelor
- LDA nu clusterizeaza folosind o distanta in sensul uzual, ci folosește o funcție bazata pe un model statistic ce descrie cum sunt generate documentele”[6]

Modelul bazat pe LDA nu a oferit rezultate satisfăcătoare, deoarece instanțele din cele patru grupuri de subiecte din datasetul folosit erau răspândite neuniform in clustere. Clusterul de care aparține fiecare instanta poate fi calculat din distribuția subiectelor a respectivei instante. Distribuția subiectelor reprezinta un vector de probabilități a unei instante de a aparține unui anumit cluster. Am atribuit fiecărei instante clusterul unde instanta avea cea mai mare probabilitate de a apartenenta. Rezultatele sunt de asa natura încât un cluster nu conține doar instante dintr-un singur grup de subiecte din cele 4. Un alt rezultat dezirabil ar fi fost acela unde un cluster poate conține instante din mai multe grupuri, dar toate instanțele unui grup sa se afle într-un singur cluster. Nici acesta nu a fost atins.

Gaussian Mixture

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

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

Modelul a eșuat in a fi testat deoarece la antrenare acesta rămâne foarte repede fără destulă memorie, spațiul de memorie heap al Java umplându-se. Încercările au fost efectuate si local, unde spațiul heap a fost mărit, si folosind un serviciu cloud, Azure, unde acesta a fost mărit si mai mult. Ambele încercări au dus la eșec.

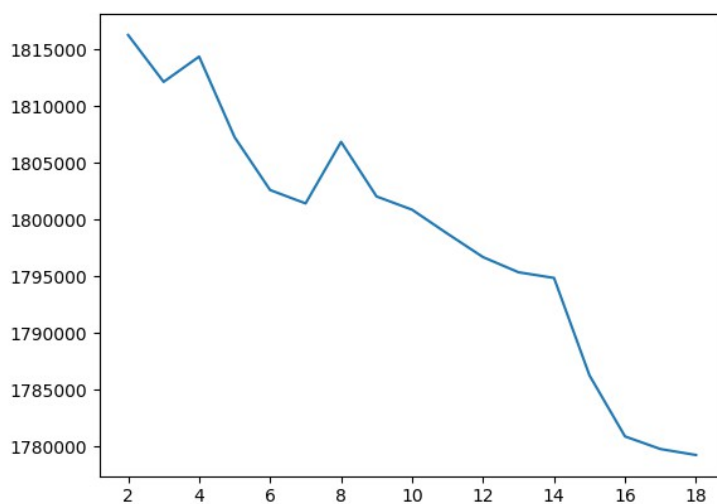
K-Means

“Clusterizarea K-Means este o metoda de cuantizare vector ce are ca scop partiționarea a n observații in k clustere, unde fiecare observație aparține clusterului cu media cea mai apropiata.”

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

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

Pe datele inițiale, graficul acestei funcții, având numărul de clustere pe abscisa si WSSSE pe ordonata, este descris in Grafic 1.



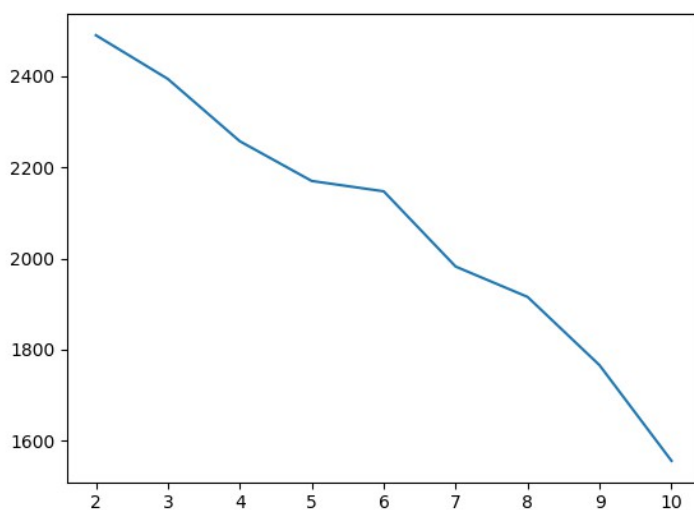
Grafic 1

Numarul de clustere se alege folosind metoda “cotului”. Punctul in care pe grafic se creează un contur ce se aseamăna cu un “cot”, adică ce formează un unghi ascuțit, si de unde mai departe nu scade drastic eroarea este punctul ce desemnează numărul optim de clustere.

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

Pentru un alt dataset, colectat in cadrul primului modul al aplicației, cu valori foarte recente si ce nu conțin un număr cunoscut de subiecte, graficul nu conține un punct ce sa descrie numărul de clustere potrivite, fapt ce sugerează ca aceasta metoda de calcul al costului nu este perfecta, dar oferă informații cu privire la eroarea modelului pentru un anumit set de date.

Tot pentru 8 clustere, modelul oferă o clusterizare acceptabila, ce situează majoritatea tweeturilor ce se refera la locația curenta a unui utilizator (cele ce încep cu textul „I'm at”) într-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 învățare automata. Preprocesarea tweeturilor consta in tokenizarea, eliminarea cuvintelor foarte comune in vocabularul limbii engleze si aplicarea metode de vectorizare a trăsăturilor 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 își propune sa folosească abilitățile de mapare a datelor din biblioteca Leaflet.js pentru a afișa date pe harta.

4.2 Afișare pe harta

Acest modul își propune sa afișeze markere din biblioteca folium pe harta. Fiecare marker reprezenta un tweet si va fi însoțit de clusterul de care aparține, fiind desenat pe harta in concordanta cu locația de unde a fost trimis tweetul.

Se oferă un input ce reprezinta termenul de interes al utilizatorului. Se cauta clusterul cu cele mai multe apariții ale acelui termen si se colorează in mod evident si special toate instancele din respectivul cluster ce conțin in text termenul cautat. Daca este cazul, celelalte instance din cluster care nu conțin termenul cautat vor fi colorați asemănător, dar totuși diferit de instancele ce conțin respectivul termen. Toate celelalte instance sunt colorate diferit, in funcție de cluster.



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

4.3 Găsirea clusterului relevant

Pentru a găsi clusterul cu cele mai multe apariții a unui termen se folosește secvența 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

Aplicația de fata reușește sa ajungă la rezultate interesante, putând fi considerata un clasificator de spam si un clusterizator de date. Aceste lucruri se realizează cu o acuratețe satisfăcătoare folosind implementări de algoritmi de învățare automata din frameworkul Apache Spark. Rezultatele finale pot fi folosite, spre exemplu, de jurnaliști pentru a infera știri in funcție de activitatea ecosistemului Twitter.

5.1 Direcții de viitor

Pentru a dezvolta aceasta aplicație se pot face mai multe lucruri. In primul rand se poate îmbunătăți acuratețea clasificării prin folosirea unui dataset diferit, eventual cu mai multe instante si subiecte.

Pentru a îmbunătăți performanta si a profita pe deplin de capabilitățile de cluster computing din cadrul Apache Spark, aplicația 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.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/>
- [2] https://en.wikipedia.org/wiki/Social_network
- [3] https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers
- [4] <http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0182487>
- [5] <https://www.nbcnews.com/tech/social-media/now-available-more-200-000-deleted-russian-troll-tweets-n844731>
- [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>
- [8] https://en.wikipedia.org/wiki/Cluster_analysis