The University of Texas at Dallas
CS6350
Big Data Management and Analytics
Spring 2019
Project Report

Project Title: Large Scale Data Collection and Preprocessing in Spark

Harsh Verma, hxv180001@utdallas.edu
Prachi Rajendra Hagwane, pxh180000@utdallas.edu
Nirbhay Sibal, nxs180002@utdallas.edu
Swathy Priya Sathishbarani, sxs175832@utdallas.edu
Divya Tyagi, dxt180002@utdallas.edu

Introduction:

In the current era, big data systems collect complex data streams and give rise to 6 Vs of big data, namely – Volume, Velocity, Value, Variety, Variability and Veracity. The reduced and relevant data streams are more useful than collecting raw, redundant, inconsistent and noisy data. Another reason for big data reduction is that million variables cause curse of dimensionality which requires unbounded computational resources to uncover actionable knowledge patterns. The above said issues can be resolved by the process of deduplication which is one of the essential tasks in data preprocessing. This process results in data cleaning and replica-free repositories which allow retrieving increased high-quality information. The project aims at running such deduplication algorithm at content level and comparing two articles from different URLs to find out whether they cover the same story.

Our goal is to compare the contents of various Spanish news websites and ascertain whether the articles cover the same stories.

The objective of project covers the development/Design of spark based preprocessing tool that does the following steps :

1 . Collects data by crawling a set of Spanish news websites on a daily basis .

2 . Extracts the main content of the article and related metadata i. e headline¨ author¨ date Published.

3 . Process the extracted content with udpipe and generate universal dependency parse for each sentence within the content : use Apache Spark here.

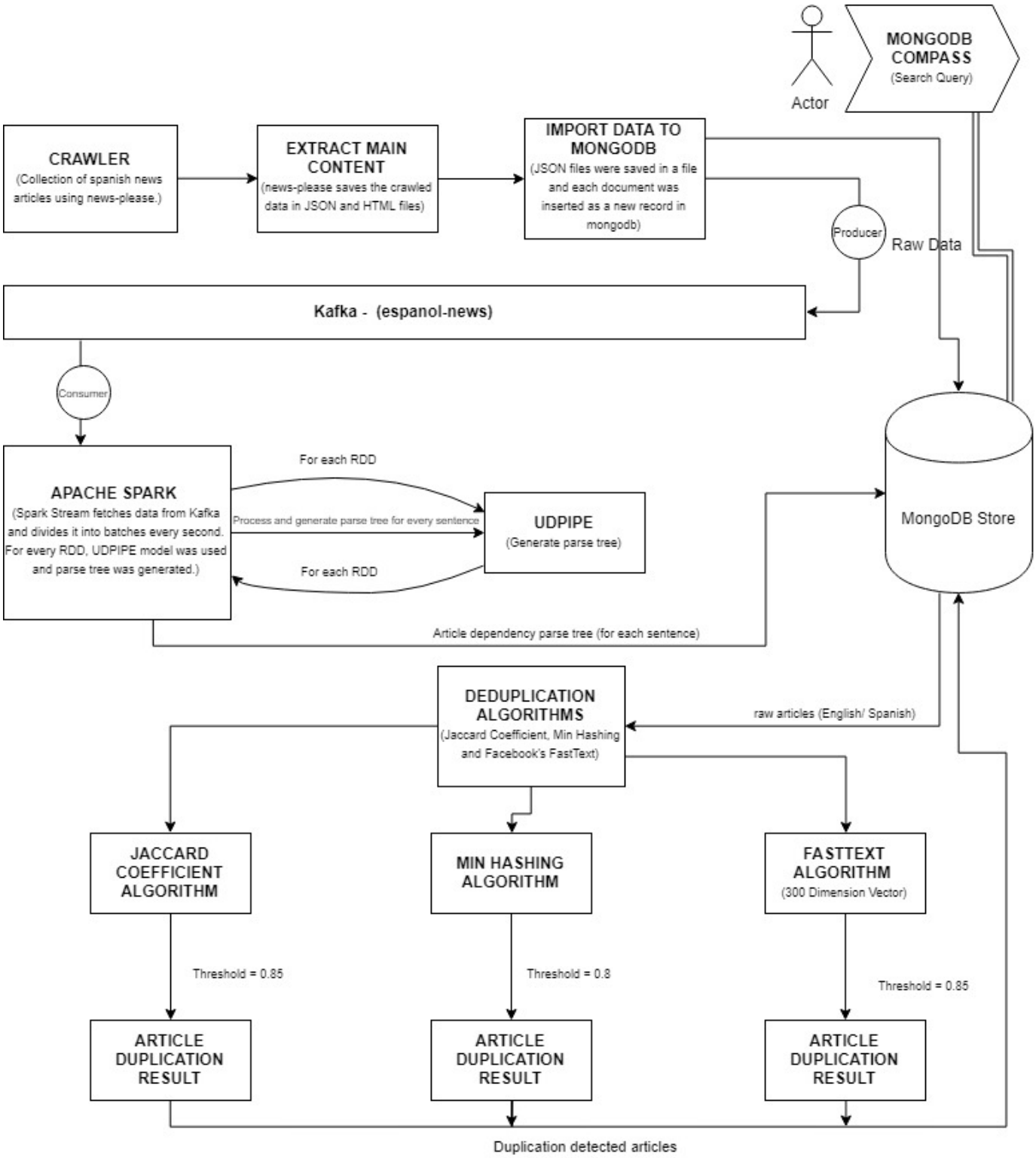
4 . Store the collected data and processed data in a way compatible with event coder's input format in MongoDB

5 . Running some deduplication algorithm at content level . Comparing two articles from different urls and find out whether they cover the same story.

Input: Raw unstructured text from Spanish news sources.

Output: Data from the articles, including raw text and processed information – in MongoDB and spark steaming processing the dependency parse tree using udpipe for each crawled news article.

Project Architecture Flow Diagram:



Architecture Diagram for Large Scale Data Collection, Preprocessing and Duplication Detection in Spark

Step 1:

Collection of data by crawling Spanish news websites on daily basis.

To collect the data by crawling we used a news-please library in python. News-please is an open source, news crawler. It extracts structured information from any news website by following internal hyperlinks and reads RSS feeds to fetch old archived and most recent articles. News-please makes use of multiple libraries such as scrapy, Newspaper and readability.

Steps for crawling using news-please: -

- Install news-please by running the following command. For this we prefer Python 3, but Python 2.7 is also supported.

  $ pip3 install news-please
- In the next step you need to provide the root URL of the news websites to crawl it completely.
- After this step we run news-please using CLI. Use the following command.

  $ news-please
- News-Please will then start crawling the pages in the **sitelist.hjson** file in the config folder inside your news-please repository. To terminate the process, press CTRL+C. news-please will then shut-down within 5-60 seconds.
- The results are stored by default in JSON files in the data folder in news-please repository. It also stores original HTML files.
- The JSON files stores information like headline, lead paragraph, main content, main image, author's name and publication date.
- This extracted data is then imported to the mongo database.

For command line processing to crawl Spanish news articles we wrote a shell script that would crawl the news websites for a specified amount of time.

We collected this data daily and ran various deduplication algorithms on the present days data.

The data extracted using news-please could have had articles which did not have the same published or modified date as the crawled date. Such articles were rejected while importing the JSON data into the database.


```
{
    "authors": ["Casa Editorial El País Cali"],
    "date_download": "2019-04-26 20:32:49",
    "date_modify": "2019-04-26 20:32:49",
    "date_publish": "2016-08-10 00:00:00",
    "description": "El Cuerpo de Bomberos de Cali inició removió las ramas que se encontraban sobre la Avenida Simón Bolívar con Carrera 11C. Un taxi resultó afectado.",
    "filename": "cali_caida-de-rama-de-arbol-genero-congestion-vehicular-en-oriente-de_1556310769.html",
    "image_url": "http://www.elpais.com.co/images/logos/logo_elpais.jpg?1555954977",
    "language": "es",
    "localpath": "C:\\Users\\Prachi
    .LAPTOP-TVE4M276/news-please-repo//data/2019/04/26/elpais.com.co/cali_caida-de-rama-de-arbol-genero-congestion-vehicular-en-oriente-de_1556310769.html",
    "title": "Caída de rama de árbol generó congestión vehicular en oriente de Cali",
    "title_page": "Caída de rama de árbol generó congestión vehicular en oriente de Cali",
    "title_rss": "NULL",
    "source_domain": "elpais.com.co",
    "text": "User Admin\nEl Cuerpo de Bomberos de Cali inició removió las ramas que se encontraban sobre la Avenida Simón Bolívar con Carrera 11C. Un taxi resultó afectado.\nLa caída
    de una rama de un árbol sobre un taxi en la Calle 70 Carrera 11C en el sentido sur - norte generó congestión vehicular en la mañana de este miércoles.\nDe acuerdo con el reporte
    del Cuerpo de Bomberos Voluntariosde Cali, la rama del árbol cayó hacias las 6:30 a.m. sobre un taxi de placas VCT 939 que pasaba por la vía pública.\n\"En el hecho no se
    presentaron personas lesionadas, pero fue necesario una máquina extintora con cuatro unidades\", indicó el organismo de socorro.",
    "url": "https://www.elpais.com.co/cali/caida-de-rama-de-arbol-genero-congestion-vehicular-en-oriente-de.html"
}
```


Figure: Sample output of JSON file

Following are the seed points of the Spanish news websites that were crawled using news-please:
-

https://www.am.com.mx/leon
https://www.am.com.mx/guanajuato
https://www.am.com.mx/lapiedad
https://www.am.com.mx/celaya
https://www.am.com.mx/sanfranciscodelrincon
https://www.elpais.com.co/
http://www.lapatria.com/
https://www.diariodequeretaro.com.mx/
https://www.eloccidental.com.mx/
https://www.elsoldemexico.com.mx/
https://www.elsudcalifornia no.com.mx/
https://www.lavozdelafrontera.com.mx/
https://elpais.com/cultura/
https://www.proceso.com.mx/

Step 2:

Extract the main content of the article and related metadata (i.e. headline, author, date published)

The articles were saved in HTML and JSON formats. We put the content of all the JSON files in one plain-text file. Then we read that file and pushed each line as a document in a MongoDB collection.

The raw data in the MongoDB collection looks like this:


```
_id: ObjectId("5cc0f6a35986e321b71b0d8f")
date_download: "2019-04-21 02:57:48"
date_publish: "2017-12-30 00:18:43"
image_url: "https://rt00.epimg.net/retina/imagenes/2017/12/27/innovacion/151436766..."
localpath: "C:\Users\Prachi .LAPTOP-TVE4M276/news-please-repo//data/2019/04/20/ret..."
source_domain: "elpais.com"
description: "Un bitcoin representa un bitcoin. Su precio elevado depende de su esca..."
language: "es"
title: "Pues no, el bitcoin no est caro"
url: "https://retina.elpais.com/retina/2017/12/27/innovacion/1514367669_7919..."
filename: "retina_2017_12_27_innovacion_1514367669_791954_1555815468.html"
title_rss: "NULL"
date_modify: "2019-04-21 02:57:48"
text: "El bitcoin roz 20.000 d lares la semana pasada. En verano estaba a 3...."
title_page: " Pues no, el bitcoin no est caro | Innovaci n | EL PA S Retina"
authors: Array
  0: "Nu o Rodrigo Palacios"
  1: "Carlos G mez Abajo"
  2: " ngel L. Mart nez"
  3: "Guillermo Vega"
  4: "Olivia L pez Bueno"
  5: "V deo"
  6: "Manuel G. Pascual"
  7: "Esther Paniagua"
  8: "El Pa s Retina"
  9: "M. Victoria S. Nadal"
```

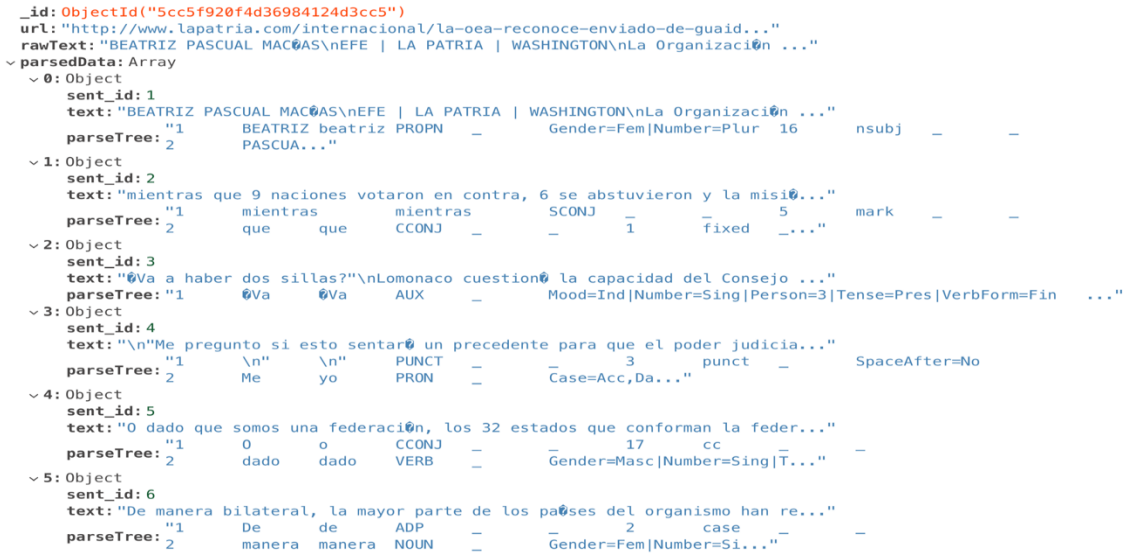

We created a Kafka topic and used the Kafka Producer to publish the documents to the topic, with their URLs as the keys and Text as the values.

We then used Spark Streaming to receive data from Kafka, using the Direct (No Receivers) approach. We created an input stream that directly pulled messages from a Kafka Broker. We divided the streaming data into batches every second. Then, for every RDD, we used UDPipe model on its text, and generated a parse tree. We saved the parsed data in another collection in MongoDB, in the following fashion:

```
{
  "url" : <url of the document>,
  "rawText" : <body text of the document>,
  "parsedData" : [ <array of parse trees of each sentence>
    {
      "sent_id" : <sentence id>,
      "text" : <raw text of the sentence>,
      "parseTree" : <corresponding parse tree of that sentence>
    }
  ]


}
```

The screenshot below shows the format of the parsed data in the MongoDB collection:



We need spark-submit to deploy our Spark application. To run the python code, we have to follow the steps mentioned henceforth:

- Open terminal, and cd to the spark directory.
- Type the following command:

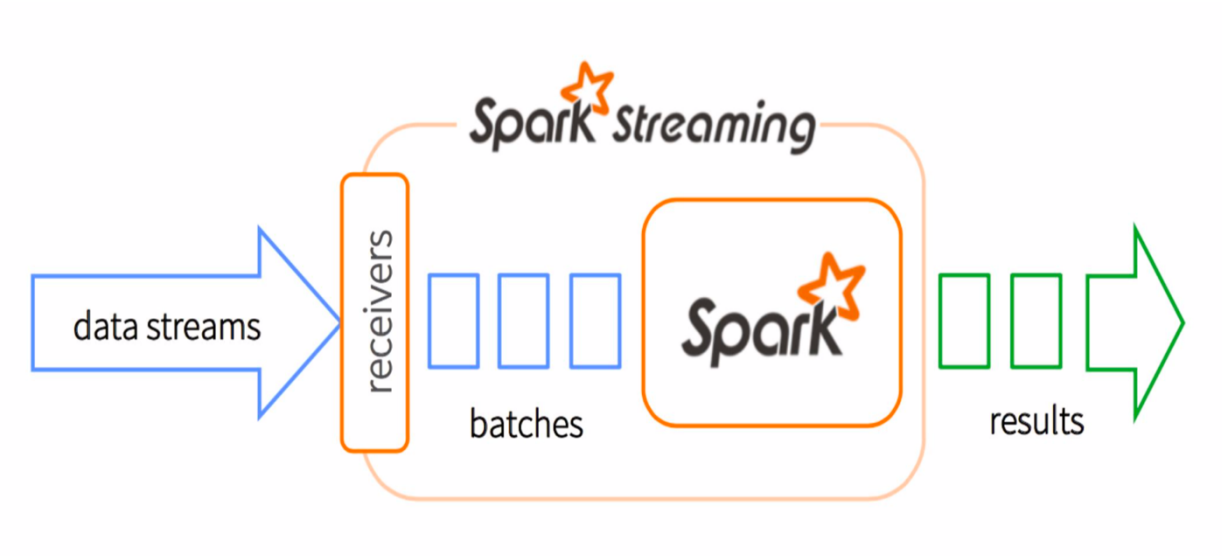
  ./bin/spark-submit –-jars <path to the JAR of the Maven artifact> <path to the python code>
- Run the python script which publishes the articles from MongoDB to the Kafka topic.

Step 3 & 4:

Process the extracted content with and generate universal dependency parse for each sentences within the content (use Apache Spark here)

Dependency parsing represents the structure of a sentence as a graph or tree, where the nodes represents the words and edges represents the dependencies between them. UDPipe is a language-agnostic trainable pipeline for tokenization, tagging, lemmatization and dependency parsing of data.

We used UDPipe to generate dependency trees for each sentence in a news article. Thus, each news article is represented by a list of dependency trees. These trees are pushed into mongo database and used later for analyzing the duplication of news articles.



Streaming Structure used for UDPIPE Processing and Generating Dependency Parse Tree:

Step 5:

Running some deduplication algorithm at content level.

- Parsed data from news link is fetched from Mongo database
- Fetched news article's contents are analyzed for deduplication using the following two approaches
- The following trials were carried out for already clustered English articles provided by TA (3.json and 4.json) and comparing the performances of our predicted clustering against the actual clustering (True Values)

Approach 1 – Jaccard Distance

- Building Shingles for the news articles – Shingle is a sequence of tokens from a given text data. The set of unique shingles from a document is known as w-shingling, where we denotes the length of shingles
- Computing Jaccard distance with the shingles generated to find out the level of similarity between any two pairs of news articles

  $JS(A, B) = |A \cap B| / |A \cup B|$

  Given a set A, the cardinality of A denoted |A| counts how many elements are in A. The intersection between two sets A and B is denoted $A \cap B$ and reveals all items which are in both sets. The union between two sets A and B is denoted $A \cup B$ and reveals all items which are in either set. Jaccard distance gives single numeric score to compare similarity between two articles. Higher the JS value, higher is the similarity between documents. The news id pairs are identified as duplicate of each other, if the JS value is above a threshold value. In our case, we have fixed it to be 0.85. For all possible news id pairs, if similarity value is above 0.85, they are classified as duplicate of each other and these results are used for clustering

Report from one of the trails:

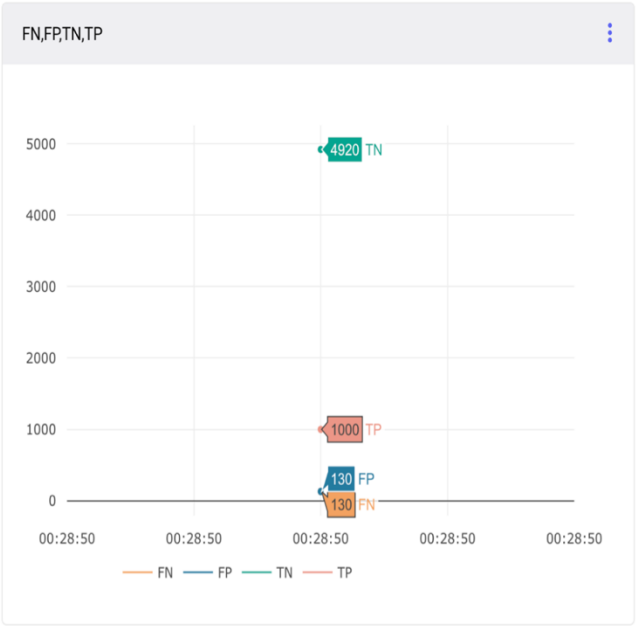
Accuracy: 0.9272727272727272
TP - 21, ***FP*** - 0, ***TN*** - 30, ***FN*** – 4

	Predicted - True	**Predicted - False**
Actual - True	TP (21)	FN (4)
Actual - False	FP (0)	TN (30)

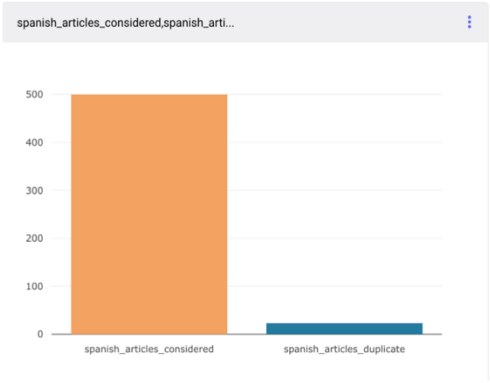
Results of Jaccard Similarity Approach:

- ***Takes less computation time and memory***
- ***Fast results***
- ***Accuracy Percentage : 70-96%***

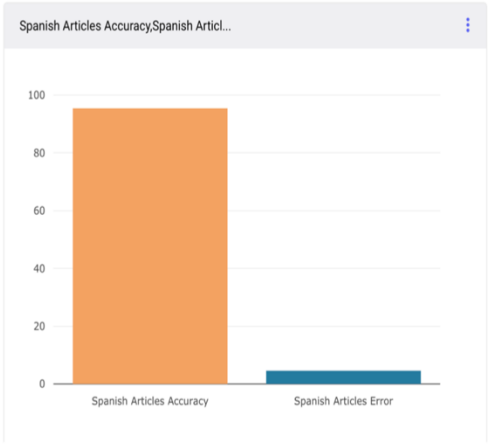
Result Jaccard Similarity:



Comparison Chart of Performance Evaluation metric : True Positive , True Negative, False Positive and False Negative Results



Comparison Chart of total to duplicate detected articles



Comparison Chart of Accuracy and Error Percent duplicate detected articles

Approach 2 – Min Hashing

- Min hashing technique is randomized algorithm to quickly estimate Jaccard similarity between two sets
- Min hash signatures are generated for each document. Min hash signatures will have fixed length independent of the size of the set. To approximate similarity between two sets, count the number of similar components in each Min hash signature. Dividing this count by signature length gives a good approximation of Jaccard similarity between two sets
- Min hash signatures are generated using hash function

Report from one of the trails:

Accuracy: `0.4727272727272727`

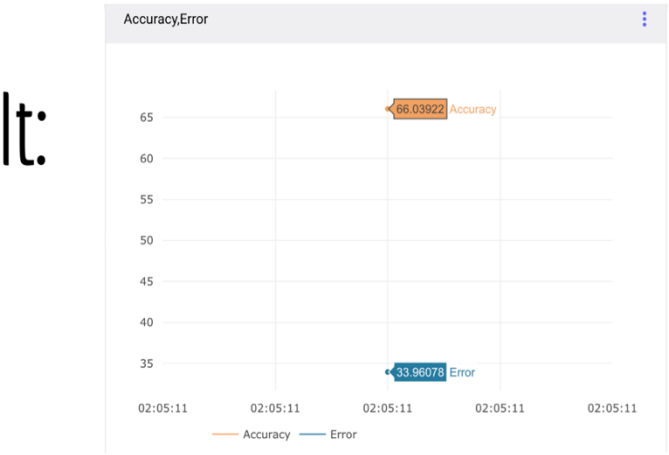
TP `- 21,` ***FP*** `- 25,` ***TN*** `- 5,` ***FN*** `- 4`

	Predicted - True	**Predicted - False**
Actual - True	TP (21)	FN (4)
Actual - False	FP (25)	TN (5)

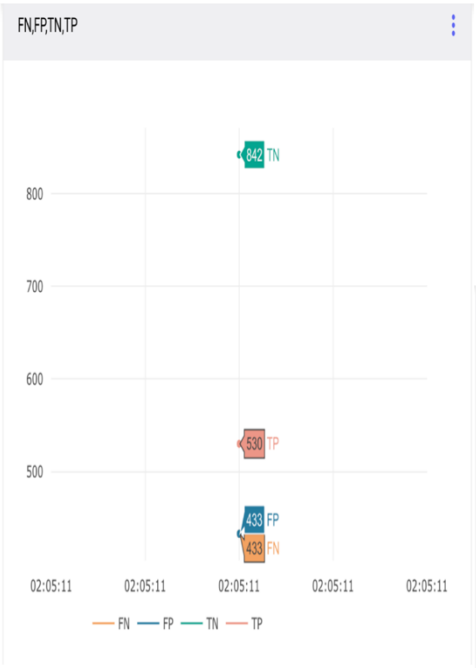
Results of Min Hashing Approach:

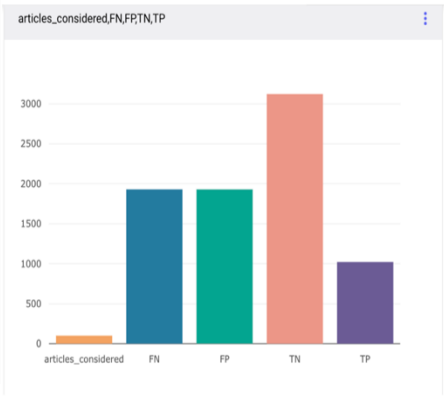
- ***Takes overhead time of building hash table of shingles, more computation time and memory***
- ***Fast results, but depended on choice of hash function***
- ***Accuracy Percentage : 70-96%***

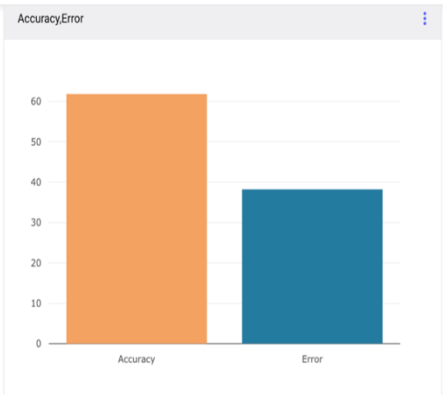
Min-Hash Technique Result:



Min-hash Technique Comparison Chart of Accuracy and Error Percent duplicate detected articles







Min-Hash technique : Comparison Chart of Performance Evaluation metric : True Positive , True Negative, False Positive and False Negative Results

Example of Detected Duplicate URLs : Similarity Index : 92%

https://www.bloomberg.com/news/articles/2015-05-27/fifa-sponsors-adidas-coke-seen-looking-at-ties-after-indictment
https://www.bangkokpost.com/news/world/575439/football-top-sponsors-pressure-fifa-to-clean-up

Analysis of Min-Hash and Jaccard Similarity approaches:

Given the list of similar English articles,

- We computed True positive, False positive, True negative, False negative values for both deduplication approaches
- Receiver Operating Characteristic curve is plotted
- Area under ROC curve and accuracy is been computed
  (Sample Trials ran are given along with the project)
- With the help of these metrics on various trials carried out, duplication analysis approach using Jaccard similarity gave us better results compared to using Min Hash technique. The main reason behind unsatisfiable results from Min hashing technique is due to the approximation of random samples it takes to predict to Jaccard similarity
- Thus, to get reliable results, we have used Jaccard Similarity technique for detecting duplicate pairs and clustering against the scrapped Spanish articles, even though it's bit time consuming than Min hash technique.

Fast-Text Approach:

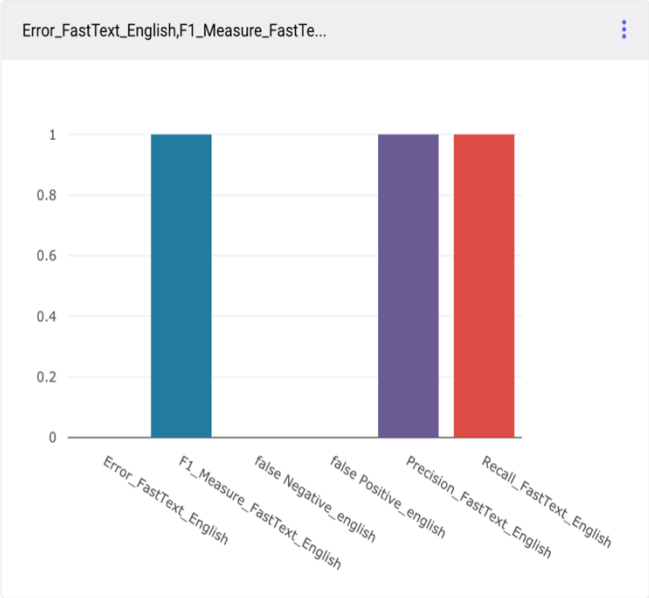
- Fast Text is an open source library created by Facebook which has pre-trained vectors on various languages.
- These pre-trained vectors can be used to create dictionaries which can directly be used for word embeddings. The vectors have dimension of →
  1 x 300.
- The advantage of using FastText library is that it is trained on an enormous wikipedia dataset & hence the vectors can have better learning of the semantics for given text.

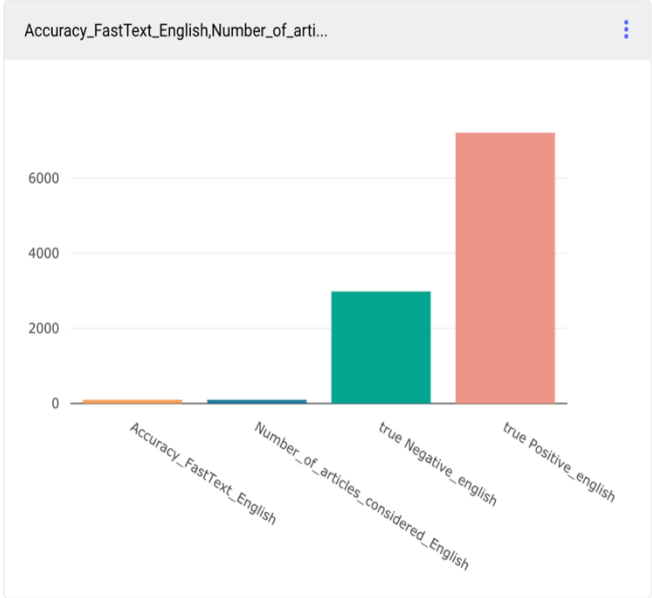
- To find similarity we used the following approach:-
- **Cosine Similarity:-**
- To find cosine similarity, the document vectors obtained the in the previous steps are reduced to dimensions of 1x 300 by taking average values of all columns.
- These vectors of the two documents whose similarity is to be determined are passed to cosine similarity function.
- Cosine similarity of >0 indicates partial similarity of documents. Cosine similarity of 1 indicates strong similarity & -1 indicates strong dissimilarity. Negative value indicates dis-similarity.

Results of Fast-Text Approach:

- ***Takes overhead time of creating vectors, more computation time and memory***
- ***Better results and high accuracy up to 95%-99%***

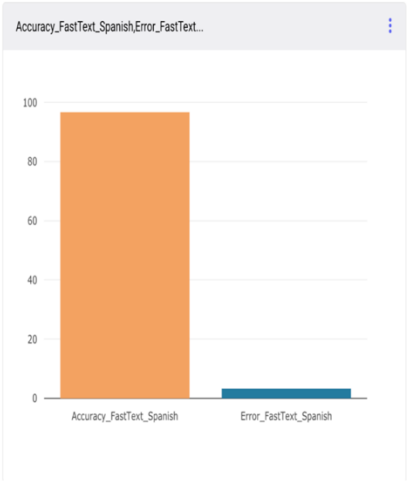
Result Fast Text English Articles: (Improved Accuracy, More Time)

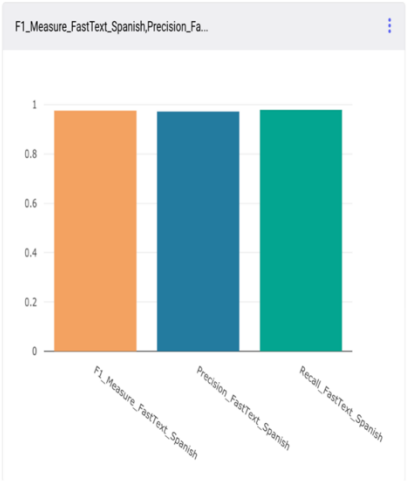


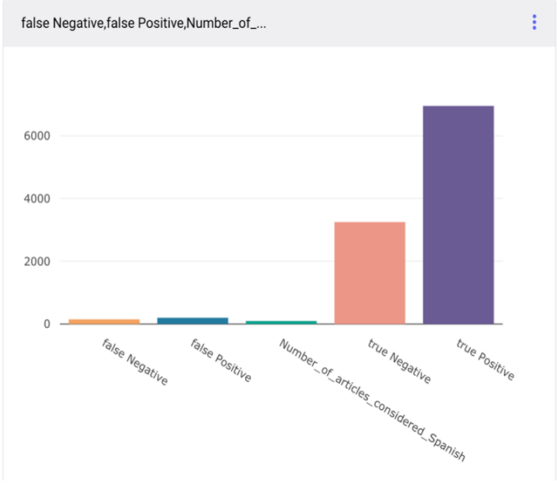


Fast-Text Vectorization technique : Comparison Chart of Performance Evaluation metric : True Positive , True Negative, False Positive and False Negative Results

Performance Result Fast Text : Spanish Articles







English Data Fast-Text Vectorization technique : Comparison Chart of Performance Evaluation metric : True Positive , True Negative, False Positive and False Negative Results along with F1 Score, Accuracy, Recall

Performance Metrics Evaluations of Fast-Text Approach

Challenges Faced during Project and Resolution :

Challenge 1: Improving the accuracy of the deduplication algorithms (min hashing and Jaccard similarity), which we were able to improve using fast text approach.

Solution: We can observe from the above results that the accuracy we achieved for Jaccard distance was about 92.72% and for Min Hashing was about 47.27%. In order to improve the accuracy, we used fast text library, that created vectors of the documents and then used cosine similarity to get the duplicated articles. We could improve the accuracy significantly using this due to the vectorization done by it. But a disadvantage of this approach was that vectorization was time consuming.

Challenge 2: Handle crawling of articles that were written or modified on the same day as they were crawled.

Solution: We crawled the documents using news-please but if an article had a link that was written few days or months ago. News-please crawled such documents as well. So in order to get only the most recent articles while importing the documents to the database we checked the **date_modified and date_published** in the JSON file and only the documents that were published on the current date were inserted into the MONGO database.

Challenge 3: Finding the folder structure and the pattern of data storage in news-please repository.

Solution: While importing the data into the database the JSON files were all written to a text file and then imported into the database. But the location where the JSON files were stored was a challenge faced while importing the data into the database. For this we wrote a shell script that had the paths of the folder that consisted of all the crawled files and imported only those documents that were crawled on the current date.

References:

- http://mccormickml.com/2015/06/12/minhash-tutorial-with-python-code/
- http://github.com/fhamborg/news-please
- https://scrapy.org/
- https://kafka.apache.org/
- https://ieeexplore.ieee.org/document/7474330
- https://universaldependencies.org/
- ufal-udpipe python package