# Social Media Analytics

Data Innovation Lab SS 2018

Abraham Duplaa(abraham.duplaa@tum.de)
Akshaya Muralidharan (akshayamuralidharan@gmail.com)
Jiho Yang(jiho.yang@tum.de)
Marie Delacourt(marie.delacourt@tum.de)

July 17, 2018

Project Lead : Dr. Ricardo Acevedo Cabra
Supervisor : Prof. Dr. Massimo Fornasier
Advisor : M.Sc. Matthias Wissel

**Abstract**

Throughout the last decade sentiment analysis, with the support of data flood and ever-improving modern computing power, has been served as a very effective method for analyzing people's opinions. In this report, we analyze over 32 million tweets related to 2016 United Kingdom European Union Membership Referendum (Brexit) via sentiment analysis in order to predict sentiment of each tweet towards this modern historical event of great importance. In conjunction, we develop an interactive & flexible pipeline of technology stacks for pre-processing & training the data, and predicting & visualizing the sentiment results. Over 300 GB of data is handled via Apache Spark on a cloud-based distributed system. We take double neural network approach for solving this supervised-learning problem, which combines Continuous Bag of Words method for pre-processing the tweets, and a two-layer neural network for training. We test, compare, and fully discuss various machine learning algorithms, including open-source pre-trained models, for this problem. Finally, different labeling strategies are compared and insights obtained from this data-driven analysis are discussed.

# Contents

# 1 Introduction

## 1.1 Project Description

Social media platforms more and more define how we consume news and also provide the place for political and social discourse. The forming of opinions on and the change of mass perception of certain topics occurs a lot faster than in the classical media setting. Therefore the analysis of social media before and after special events became crucial to understanding what drove the decisions that were made for example in a public poll. With this in mind, Capgemini proposed a social media analytics project for this semester related to the Brexit referendum. This project aims to carry on an analysis of opinions and relationships before, during and after the referendum using twitter data. The three main parts in this Brexit analysis project, timeline visualisation, sentiment analysis and network analysis, will be discussed below.

## 1.2 Project Scope

### 1.2.1 Timeline Analysis and Visualization

A first way to analyse the Brexit referendum situation is to represent Twitter activity over time according to different metrics (for different accounts, hashtags, etc). So a first objective is to create a pipeline that takes as an input the dataset (or a reduced dataset) and outputs a dashboard which gives the evolution of different Brexit-related metrics over time. The two main issues of this part will be selecting the pertient features from our data and deciding on which data visualisation tool we are going to use for the dashboard itself.

### 1.2.2 Sentiment Analysis

As the Brexit referendum was all about people's opinion on leaving the EU, a logical step in understanding the phenomenon would be to see whether it is possible to model the evolution of users' opinion. In this part of the project, we wil have to figure out a way of analysing the sentiment of people on Brexit on their tweets and find interesting metrics to reflect the results. We will obviously be using NLP (Natural Language Processing), and more particularly "sentiment analysis", which is a very popular approach to convert text into opinions. We will focus on modeling this using a neural network approach.
There are different challenges surrounding this topic, particularly concerning the model we are going to use to embed tweets as vectors in order to be able to precess them through a sentiment classification network. Moreover, our data is not labeled (i.e. there is no "sentiment" item for each tweet that tells us whether each it is "for Brexit" or "against Brexit") so we will have to find out whether we want to use the classical supervised approach or see whether it will be possible to find an unsupervised approach.

### 1.2.3 Network Analysis

The structure of Twitter as a social network can be represented as a graph with nodes as different users and edges as followers. Furthermore, based on choice of modeling, we may also choose weights for these edges such as number of re-tweets. From a cursory glance at such a graph we can visually and intuitively spot popular users - users with many edges or users with largest incoming weights. However, Twitter being such a vast pool of users, makes it impractical and difficult to visually depict all users and connections visually. Here, we aim to find metrics to identify the important nodes and influencers by filtering based on these importance metrics. Once we boil down the massive social network graph to the identified important users, we can identify patterns and influencers, either quantitatively, using the metrics or visually, by representing the same as graphs.

# 2 Methodology

## 2.1 Working Methodology

### 2.1.1 Agile Software Development

There is no doubt that agile software development is being considered as one of the most widely used and characteristic aspect of modern software development. Agile software development is rather a set of principles, than merely a methodology, that focuses on development, not planning, and interaction instead of processes. It aims to decompose the *final product*, that usually were developed after a lengthy planning, into smaller deliverable working software such that the product can be continuously reviewed and adjusted. Although the *agile process* is heavily used in software development, it would be a misunderstanding to consider it solely as a software development practice. For this project, we have adapted the agile development philosophy in order to improve our working efficiency, team interaction, and most importantly, to maximize the quality of deliverable outcome. Scrum was conducted as a mean to follow the agile process. Our scrum consisted of:

– Weekly scrum meetings: Once a week meeting with supervisor
– Weekly individual sprints: Individual work to complete the task for the week
– Weekly team member scrum meetings: Two to three days of additional meetings without the supervisor to update the sprint progress, continuously synchronize the current working stage, enhance team interaction, and (re)assign tasks accordingly

Our advisor, Matthias Wissel, acted as the scrum master to provide us both the weekly and overall template of the tasks. Each team member adjusted each weekly tasks such that they can be accomplished within a week and picked the tasks of their interest. The completed tasks were integrated after weekly scrum meetings to have a working pipeline/software, and the process was iterated.

#### 2.1.2 Communication Tools

Communication has to be one of the essential component of a successful group work that is often considered as an afterthought. In the perspective of both project management and agile software development, which highlights team interaction, several communication tools were used to maintain a good level of team communication. Other than standard messengers, we used Slack, and Trello board for scrum.

## 2.2 Tools

#### 2.2.1 LRZ Cloud

Leibniz-Rechenzentrum (LRZ) provides cloud services to students and faculty and these services were the foundation to the Social Analytics project. The LRZ cloud is built upon the OpenNebula framework which made creating, monitoring, and tuning virtual machines effortless. By having access to virtual machines, this allowed a cluster setup for distributed data computation as well as for modularity in running programs.

#### 2.2.2 Hadoop File System

In order for the virtual machines to have easy access to all the data, Hadoop File System (HDFS) was chosen for distributed file and data storage. HDFS allows all the data to be stored in a distributed fashion, while also ensuring fault-tolerance due to redundancy of the data. HDFS handles redundancy by creating data blocks, small chunks of data, and saving copies of these data blocks on other virtual machines in the cluster.
Furthermore, HDFS comes with a user-friendly GUI that facilitates management of HDFS. The UI provides insight into properties such as how the data is distributed, overall node health, access to each node log, and a very useful File browser so that files on HDFS can be viewed similar to on a native OS file browsers. Fortunately, the processing tools used (e.g. Apache Spark) work natively with HDFS.

#### 2.2.3 Apache Spark 2.3.0

Apache Spark is a well-known, open-source, and highly regarded general-purpose cluster computing system. Spark has been shown to achieve high performance when compared to Hadoop, rinning workloads around 100x faster. Aside from fast computation, Spark is also easy to use due to to extensive libraries in Java, Scala, Python, R, and SQL. For the purpose of this project, the PySpark library and SQL functionalities were used to create the applications which were then run as Spark applications.
Spark can be used locally or in cluster mode:

– **Local:** Also known as pseudo-cluster mode, the local mode only creates one Java Virtual Machine (JVM) and all the execution components are located within the same JVM. Local Spark still performs parallelism by using hardware threading. This mode is preferred for development, testing, and debugging, as it does not require the overhead of the cluster and can save computational resources.

– **Cluster:** When the code is ready for production, cluster mode is used for data processing of the whole dataset. This mode has a master-worker architecture, as depicted in our architecture diagram (Fig. 1). When a spark application is run, the SparkContext object coordinates the set of processes running on each node within the cluster. The SparkContext connects to the cluster manager and the resources for the processes are allocated by the cluster manager. For this project, we initially tried using Kubernetes as the cluster manager. Kubernetes is a container-orchestration system that automatically deploys containers across a cluster. By using Kubernetes, there would be all the advantages of containerized applications in the form of Docker and would make deployment of Spark and other applications painless. However, in order to focus more energy on the project and less energy on environment setup, it was decided to use the built-in Spark Standalone cluster manager, which worked flawlessly as well. Once SparkContext is connected to the cluster manager, the following steps occur:

1. Executors on each node are acquired.
2. The application code or python file is sent to the executors.
3. SparkContext sends tasks to the executors and the tasks are executed in a distributed fashion.

**Utilized Spark Packages**

1. **spark.sql:** With Spark 2+, the traditional basic Spark RDD API has been placed into maintenance mode and Spark SQL has become the recommended Spark module for data processing. The major difference to the user is that Spark SQL provides a more native syntax to users comfortable with SQL. For functions which are easier done in RDD, spark users can switch back and forth between Spark SQL and Spark RDD since the same execution engine is used. Furthermore, there are also performance gains to be considered when using Spark SQL. More information about the structure of the data and computation are provided to the Spark engine when using Spark SQL compared to when using Spark RDD, which can lead to extra optimizations. Spark.sql contains most of the functions used for the Preprocessing and the ETL step.

2. **spark.ml:** Similar to spark.sql, spark.ml is the primary machine learning package as of Spark 2+, with spark.mllib now in maintenance mode. This library is the main driving force behind tokenizing, removing stop words, Word2Vec, and many other aspects of the Sentiment Analysis section.

### 2.2.4 JupyterHub

In order to provide direct contact to the virtual machines, JupyterHub was chosen as the interface of choice. JupyterHub allows each user to have their own server and create Jupyter Notebooks, python scripts, and browse the user's directory. JupyterHub spawns multiple instances of the single-user Jupyter notebook server and makes logging in easy with user GitLab account login.
JupyterHub works in the following way [8]:

- The Hub launches a proxy
- The proxy forwards all requests to the Hub by default
- The Hub handles user login and spawns single-user servers on demand
- The Hub configures the proxy to forward URL prefixes to the single-user notebook servers

### 2.2.5 GitLab

Working on a same project in a group not only requires a version control tool, but a cloud-based code sharing platform. LRZ GitLab was chosen for these purposes. A private GitLab repository was created on top of which JupyterHub was setup.

### 2.2.6 Docker

In order to deploy the visualization agents as well as the databases painlessly, Docker was chosen as the container technology. Docker is a computer program that abstracts running applications from the core operating system level. Each application runs in its own operating-system-level virtual "container". This makes it very easy to run, stop, or deploy a software application in any operating system and platform (which supports Docker).

### 2.2.7 Kubernetes

When setting up the environment for this project, Kubernetes was initially attempted for resource management and ensuring easy scalability. Kubernetes is a container-orchestration system for automating deployment, scaling and management of containerized applications (usually containerized using Docker). Kubernetes would have made scaling easier if needed and would have prevented system conflicts due to using the technologies in containers. Recent Spark editions including Spark 2.3 natively support Kubernetes to scale and deploy Spark nodes. Furthermore, Kubernetes includes a very useful UI to make scaling of containers even easier. Unfortunately, Kubernetes installation was difficult and there were many issues with connectivity between nodes. However, Spark and our other applications ran without many issues.

#### 2.2.8 Keras

We used Keras with Tensorflow for implementing the neural network. TensorFlow is an open-source library for building large neural networks and Keras is a high level neural network API that can be used out of the box to construct different layers of the network. It also allows us to compile the entire network where we can customize the optimizers for learning rate, specify the loss function and the validation metrics.

#### 2.2.9 Databases

##### 2.2.9.1 PostgreSQL

Since the data being processed is in a structured and relational format, PostgreSQL was chosen to store the data. PostgreSQL is an Object-Relational database management system and was chosen as the database of choice due to the emphasis on standards compliance. This facilitates PostgreSQL use.

##### 2.2.9.2 MapD

One of the major challenges faced when working with large amounts of data is how to meaningfully and efficiently visualize data. Meaningful visualizations can fortunately be designed but the latter requires excellent pre-processing, sampling, hardware, and software. In choosing how to visualize the data effectively and in real-time, we first tried using python libraries such as matplotlib and plotly. Although this was useful in the early stages of discovering the data, this proved to be rather slow and cumbersome.
After trying out common python plotting libraries, 3rd party software options were tested. Tableau unfortunately could not provide the capabilities we needed, especially when compared to MapD. MapD is a database and visualization agent which is available as open-source, free community edition, and Enterprise edition. Created for optimized SQL queries by utilizing GPU power, MapD is currently the fastest open-source in-memory database. Since the virtual machines used for this project did not include GPUs, MapD was not used to its full potential but still was able to display the Brexit twitter data in real-time with interactive visualizations. The backend of MapD follows many of the conventions of traditional SQL databases.

### 2.3 Infrastructure Architecture

The distributed systems architecture is shown in Fig. 1. A distributed and modular architecture was chosen to ensure that concurrently running programs do not interfere with each other and compete for resources. This could lead to reduced performance or system crashes. The Database and Visualization node is completely removed from the HDFS and Spark environment to ensure that the node has maximum memory to perform complex queries and render real-time graphics. Spark computations are very memory intensieve which almost certainly would cause insufficient memory issues if the visualizations and queries were rendered on a Spark worker or Spark master.

### 2.4 Data Description

The data was extracted using the Twitter API. Fortunately, it was not necessary or in the project scope to mine the tweets from Twitter for this project. The Brexit data was provided by CapGemini in zipped directories filled with JSON files. In order to extract the data from the tar files, a Bash script was created to untar, reformat the JSON files, and place them in HDFS. After preliminary screenings of the data, it was found that the data had a timespan of April 2016 to December 2017 that had at least one of the following 15 hashtags:

- #TakeControl
- #VoteLeave
- #Brexit
- #BrexitWound
- #ProjectHope
- #LeaveEU
- #EUReferendum
- #OutOfEu
- #EURef
- #Remain
- #UKIP

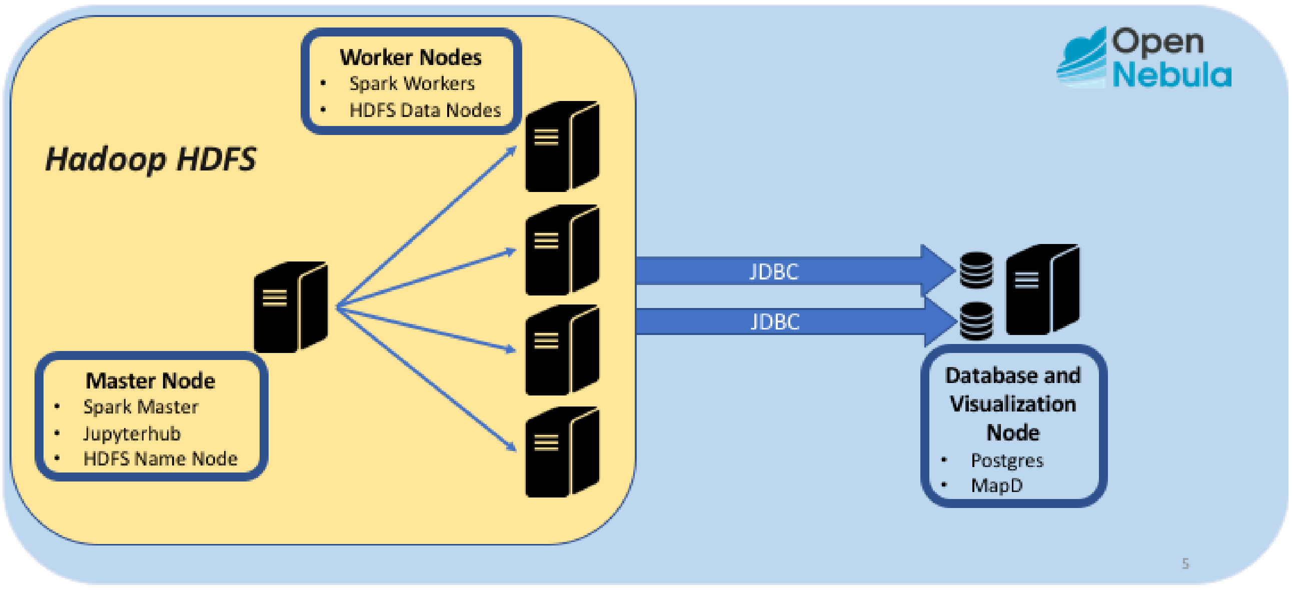


Figure 1: A schematic of the Architecture used for Computation and Visualization

- #StrongerIn
- #BetterTogether
- #LeaveChaos
- #VoteLeaveGetChaos

However, this is not the exact dataset we will be working with. Indeed, as a first preprocessing (or relevance checking) step, we classified the hashtags according to their orientation (against Brexit, neutral, for Brexit) and found out that two hashtags are ambiguous as they are often used for content that is not Brexit-related (see Appendix 6.1). From now on, we will be working on the original dataset excluding the tweets associated to those ambiguous hashtags.

It was decided to store the data in HDFS for several reasons. There are 3 main advantages of storing the JSON files in the HDFS directory:

1. **Fault Tolerance:** Although the LRZ Cloud is robust and reliable, virtual machines can fail. Fortunately, HDFS will shard the data and distribute data block copies throughout the cluster.
2. **Increased Storage Capacity:** With over 300 GB of raw data, it would be difficult to store and manage the data on a local directory. By using HDFS, the storage size is vastly increased since HDFS utilizes disk space from each virtual machine in the cluster. Furthermore, the master node should not have the sole burden of storing and providing access to the files.
3. **Facilitated Access for Distributed Computing:** By having the data on HDFS, Spark can read data in a much faster and optimized way over using NFS or the master's local directory.

### 2.4.1 Data Discovery

Once in HDFS, the data discovery phase could begin. As shown in Tab. 1, it's worth noting that the original schema had 75 fields. Having ample amounts of data is important but unnecessary or irrelevant data could hinder the project computationally and analytically. To understand what each field contained, a sample output for each field was created. Please refer to Tab. 14 in the Appendix. 6.2 for a description of each field.
Once the crucial columns were identified, the data was processed in the way shown in Figure 2. This allows us to then use the data for Data Visualization as well as for Sentiment Analysis.
Some useful insights from the data discovery are as described in Table 1 and in the Timeline section.

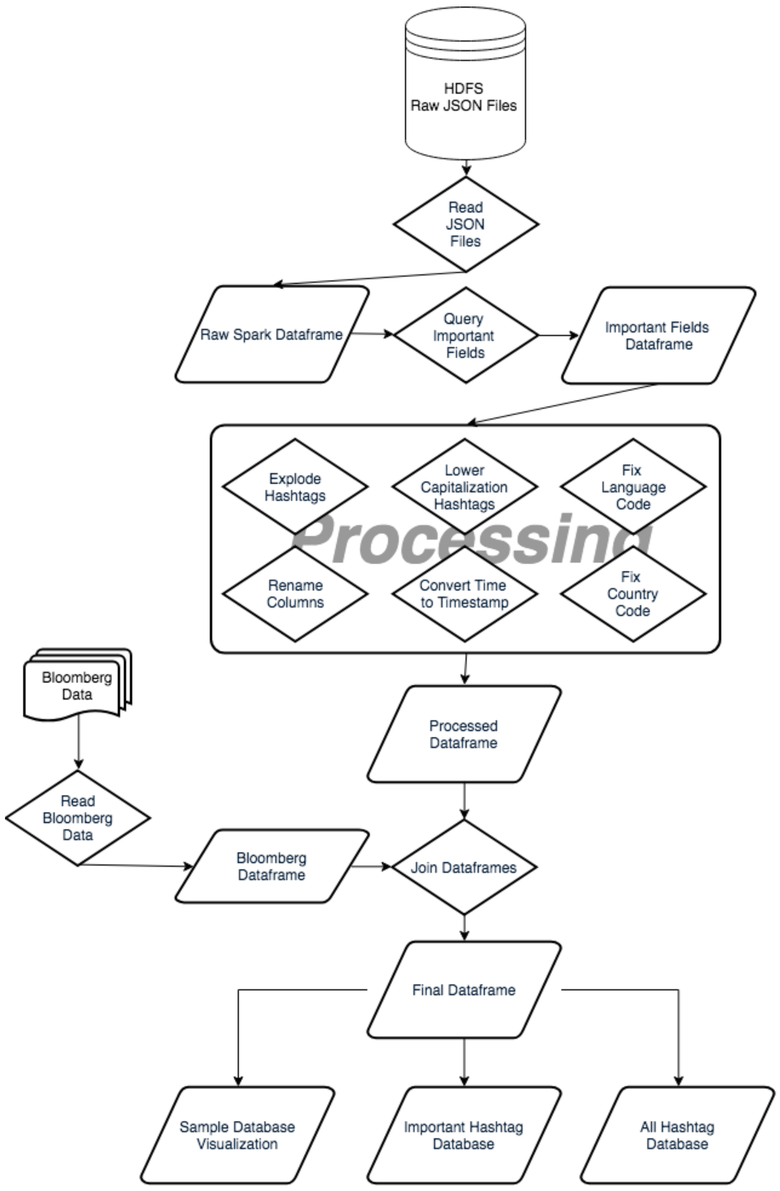


Figure 2: The processing pipeline to take JSON files and make meaningful data tables for analysis

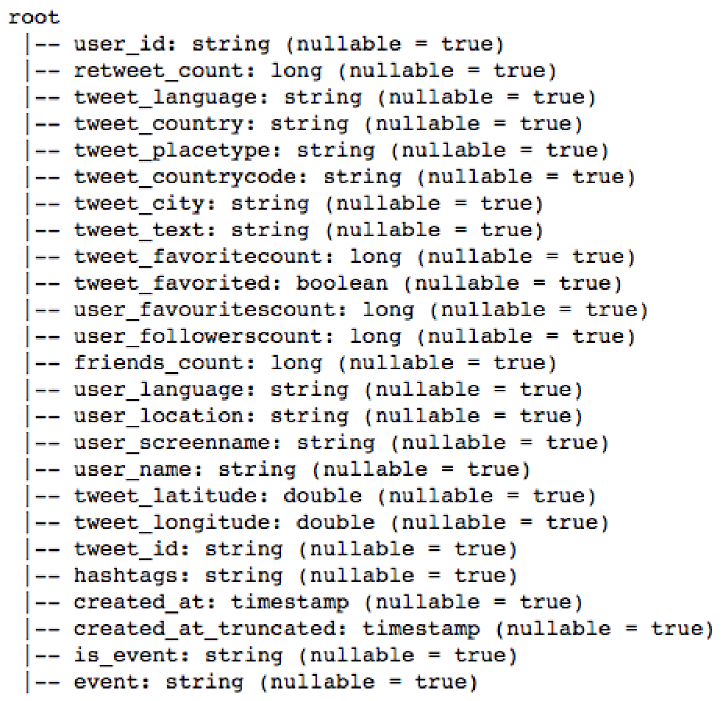
| Dataset Characteristics | Metric |
|---|---|
| Data Format | JSON |
| Physical Memory Size | $\pm 280GB$ |
| Number of Fields | 75 |
| Number of Important Fields | 25 |

Table 1: Important metrics regarding the full dataset

## 2.5 Timeline Visualization

### 2.5.1 Data Processing

We aim to represent Twitter activity surrounding Brexit and correlate it with real-life events and decisions on that topic. This (obviously) requires us to work with our tweets dataset but also to get data concerning chronology of the events before and after Brexit. For the latter, we used an online Bloomberg Brexit timeline [2] (June 2016 to March 2017) that we extracted using a web scraper and converted to a Json file (on 19.04.18). In order to manipulate and then visualize the data on a timeline, we first of all have to find a suitable size for our dataset. Indeed, the "reduced" dataset (full dataset without ambiguous hashtags) has 32 million tweets stored on Spark, which is equivalent to a 32 million rows of a Spark table. We decided not to work with so much data on a Jupyter notebook, which means we had to find a way of working on a smaller dataset that is still large enough to represent tendencies among tweets. We finally decided to randomly pick one third of the data (and we noted later that it's a sufficient proportion as the tendencies were still preserved on the graphical representation).We then selected the necessary columns from the dataset by picking only those that could be interesting to represent on a timeline.



```
root
 |-- user_id: string (nullable = true)
 |-- retweet_count: long (nullable = true)
 |-- tweet_language: string (nullable = true)
 |-- tweet_country: string (nullable = true)
 |-- tweet_placetype: string (nullable = true)
 |-- tweet_countrycode: string (nullable = true)
 |-- tweet_city: string (nullable = true)
 |-- tweet_text: string (nullable = true)
 |-- tweet_favoritecount: long (nullable = true)
 |-- tweet_favorited: boolean (nullable = true)
 |-- user_favouritescount: long (nullable = true)
 |-- user_followerscount: long (nullable = true)
 |-- friends_count: long (nullable = true)
 |-- user_language: string (nullable = true)
 |-- user_location: string (nullable = true)
 |-- user_screenname: string (nullable = true)
 |-- user_name: string (nullable = true)
 |-- tweet_latitude: double (nullable = true)
 |-- tweet_longitude: double (nullable = true)
 |-- tweet_id: string (nullable = true)
 |-- hashtags: string (nullable = true)
 |-- created_at: timestamp (nullable = true)
 |-- created_at_truncated: timestamp (nullable = true)
 |-- is_event: string (nullable = true)
 |-- event: string (nullable = true)
```

Figure 3: Selected columns for time line visualization

### 2.5.2 Data Visualization

We tried two different approaches for the visualization: one using Jupyter notebooks (with plotly library) and the other with MapD (an analytics platforms that has a solution to create dashboards). Our first attempt was using Jupyter notebooks with plotly, which is a library that is specially designed for creating graphs that can be shared and displayed online. A very strong advantage of this solution is that it is very interactive (graph range can be changed, hovering over the graph shows the data, zooming in is possible, etc). So it offers both a user-friendly interface that makes it very useful for a dashboard and freedom regarding what we want to represent and how we want to visualize it. On the other hand, a (huge) disadvantage is that it takes a rather long time to load the data and display it. For example, to display tweet frequency for only one week, it takes

around 30 seconds to 1 minute.
So what we tried as a second approach (and what we used for or final deliverable) is dashboard creation using MapD. For a more detailed description of MapD, please refer to Section 2.2.9.2.

## 2.6 Sentiment Analysis

### 2.6.1 A Double Neural Network Pipeline

We aimed to build a sentiment analysis tool that takes into input a tweet (a 280-characters message) and outputs a ”pro Brexit” or ”against Brexit” sentiment. Several approaches are possible for this problem, the most popular approach being training a neural network. To determine the best approach for our task, we did some research on the subject and met Michal Dura, a Capgemini consultant who did his master’s thesis on predicting the results of the Brexit referendum. He worked on similar problematics: sentiment analysis and times series analysis, and our datasets are very similar (our dataset is just over a bigger timespan, but was extracted using the same method - selection by hashtags). We finally decided to go for a deep neural network approach for sentiment analysis, which should give better accuracy than logistic regression or a naive Bayes classifier (respectively 75% and 64% accuracy for Michal).

But before predicting sentiments, we first had to find the best way to represent tweets, i.e. sequences of words, as input vectors for our neural network. Today, there are several models to encode words as vector, but the one that is by far the most successful and widely used today is Word2Vec [16] (a neural network developed by Mikolov et al. at Google in 2013). We will explain in Section 2.6.2 the specificities of this model in detail. In conclusion, our pipeline for sentiment analysis is the following (Fig. 4):

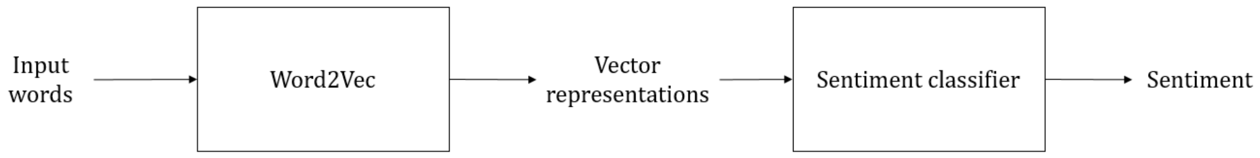
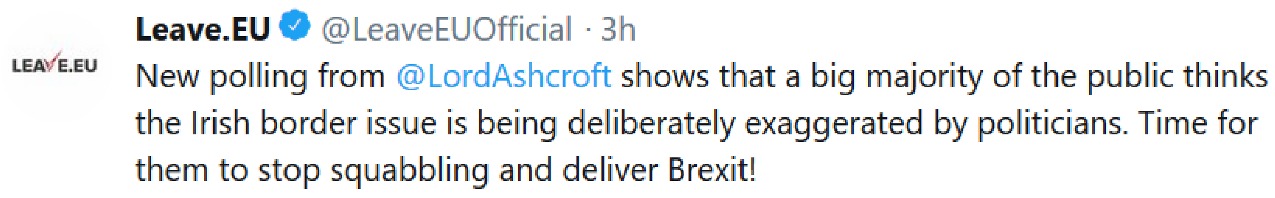


Figure 4: Pipeline for sentiment analysis using two neural networks

### 2.6.2 Word2Vec Continuous Bag of Words (CBoW)

#### 2.6.2.1 Introduction to Word Encoding

If we want to represent a tweet as a vector, the first step is to understand how to represent a word as a vector. Let us now take the following tweet as an example and compare different approaches for word encoding.



Leave.EU @LeaveEUOfficial · 3h
New polling from @LordAshcroft shows that a big majority of the public thinks the Irish border issue is being deliberately exaggerated by politicians. Time for them to stop squabbling and deliver Brexit!

Figure 5: Pro-Brexit tweet by @LeaveEUOfficial

The first step is to define a vocabulary of $V$ words that contains all the possible words in our document. A first, very naive way to tackle this problem would be to encode each word as a (one-hot) vector in $\mathbb{R}^V$. But this would give a very high number of orthogonal vectors and make it difficult to show similarity between words (measured as a distance between vectors).

That is why, to get word representations that express semantic similarity, we make use of the context of the words (”You shall know a word by the company it keeps” - J.R. Firth, 1957). Predicting a word knowing its context is called the *Bag of Words* approach because it counts occurrences from the vocabulary in a window around our target word. As an example, for the above tweet, our vocabulary would be { *big, border, Brexit, deliberately, deliver, exaggerated, Irish, issue, LordAshcroft, majority, new, politicians, polling, public, show,* *squabble, stop, think* }. If we choose a window of width 4, then we could embed target words by counting occurrences in the word window surrounding them. We get the following *embedding matrix* :

$$\mathbf{E} = \begin{array}{cccccccccccc|l} \text{big} & \text{borders} & \text{Brexit} & \text{deliberately} & \text{deliver} & \text{exaggerated} & \text{Irish} & \dots & \text{show} & \text{squabble} & \text{stop} & \text{think} & \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 0 & 0 & 1 & \text{majority} \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & \cdots & 0 & 0 & 1 & 0 & \text{politicians} \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & 1 & \text{public} \end{array}$$

We can then retrieve the words we are interested in with the corresponding one-hot vectors:
$\mathbf{onehot_{public}} = [0\ 0\ 1]^T$ so $\mathbf{public} = \mathbf{onehot_{public}}^T\ \mathbf{E}$

#### 2.6.2.2 The General Idea Behind Embedding Learning

The general idea behind the process of embedding learning is to find the best embedding matrix (by maximising a score or minimising a loss). The general steps are:

- collect instances $\mathbf{x^i} \in instances(\mathbf{x})$ of a word $\mathbf{x}$ of vocabulary $V$ of size $v$
- for each instance, collect its context words $context(\mathbf{x^i})$
- define a score function $score(\mathbf{x}, context(\mathbf{x}), \theta, \mathbf{E})$, where $\theta, \mathbf{E}$ are parameters we want to estimate
- define a loss $L = -\sum_{\mathbf{x} \in V} \sum_{\mathbf{x^i} \in instances(\mathbf{x})} score(\mathbf{x^i}, c(\mathbf{x^i}), \theta, \mathbf{E})$
- estimate $\hat{\theta}, \hat{\mathbf{E}} = argmin_{(\theta, E)} L$
- use the estimated $\mathbf{E}$ as our embedding matrix

An important point is to design a useful scoring function. Such a function should produce a score that accounts of how well a context accounts for word $\mathbf{x}$ and make apparent that this word $x$ is better in the context $context(\mathbf{x})$ than any other word. Of course, it should also be differentiable w.r.t $\mathbf{E}$ and $\theta$.

#### 2.6.2.3 Continuous Bag of Words

The Word2Vec CBoW model consists of a neural network with one hidden layer. What we typically want to do once our model is trained is feed it the context $(\mathbf{x_1}, ..., \mathbf{x_{i-1}}, \mathbf{x_{i+1}}, ..., \mathbf{x_c})$ of a word $\mathbf{x_i}$ and get the best encoding vector as an output (and then build our embedding matrix that way). To train our network, we first have to define a good scoring function as defined in section 2.6.2.2. Let us consider a simple approach called the *bigram model* to explain this model.

In the *bigram model*, we consider that each word has only one word in its context. So we simply use a 1-hot encoded vector from $\mathbb{R}^V$ (where $V$ is the size of the vocabulary) as an input. We expect a probability that this word belongs to one among $V$ categories as an output, as we are in a 1-word-context approach. So this can be translated into a linear classification problem with $V$ classes $(C_1, ..., C_V)$, the 1-hot vector showing the class to which the word belongs. We use a *logistic regression model* with $V$ classes, so $\mathbf{y}|\mathbf{x} \sim \mathbf{multinomial}(\sigma(\mathbf{W'^T x}))$, where $\mathbf{y}$ is the class of the input word and $\mathbf{W'} = (\mathbf{w_1'}, ..., \mathbf{w_V'})$ the weights that we want to learn. Let us say that we train our model on $n$ words, then we typically want to maximize the likelihood of our data:

$$P(\mathbf{y}|\mathbf{W'}, \mathbf{x}) = \prod_{i=1}^{n} P(y_i|\mathbf{W'}, \mathbf{x_i}) = \prod_{i=1}^{n} \prod_{v=1}^{V} P(y_i = C_v|\mathbf{W'}, \mathbf{x_i})^{1_{y_i = C_v}}$$

Using Bayes' law and the formula of total probabilities, we get

$$P(y_i = C_v|\mathbf{W'}, \mathbf{x_i}) = \frac{P(\mathbf{x_i}|\mathbf{W'}, y_i = C_v)\, P(y_i = C_v)}{\sum_{j=1}^{V} P(\mathbf{x_i}|\mathbf{W'}, y_i = C_j)\, P(y_i = C_j)}$$

Which translates into, using our logistic regression approach:

$$P(y_i = C_v|\mathbf{W'}, \mathbf{x_i}) = \frac{exp\left(\mathbf{w'^T_v x_i}\right)}{\sum_{j=1}^{V} exp\left(\mathbf{w'^T_j x_i}\right)}$$

Where $\mathbf{w_j\prime}$ is the weight vector corresponding to class $C_j$, and the bias terms have been absorbed in the $\mathbf{w_j\prime}$'s by adding a 1 on top of vector $\mathbf{x_i}$. We can now define an objective function $E$ by minimizing the log-likelihood:

$$\begin{aligned} E(\mathbf{W\prime}) &= -ln\, P(y = C_v|\mathbf{w\prime}, \mathbf{x}) \\ &= -\sum_{i=1}^{n}\sum_{v=1}^{V} 1_{y_i=C_v}\, ln\, P(y_i = C_v|\mathbf{w\prime}, \mathbf{x_i}) \\ E(\mathbf{W\prime}) &= -\sum_{i=1}^{n}\sum_{v=1}^{V} 1_{y_i=C_v}\, ln\, \left( \frac{exp\left(\mathbf{w\prime_v^T x_n}\right)}{\sum_{j=1}^{V} exp\left(\mathbf{w\prime_j^T x_i}\right)} \right) \end{aligned}$$

$E$ is also called binary cross-entropy. Now we have our objective function and the expression of outputs $P(y = C_v|\mathbf{W\prime}, \mathbf{x})$ according to the inputs, let us describe the neural network we are going to use for this problem. It has 1 hidden layer, where the weights from matrix $\mathbf{W\prime}$ are used to connect the hidden to the output layer. Another set of weights $(\mathbf{w_1}, ..., \mathbf{w_N})$ are used to create a $N$-dimension representation of the input vector (N is the hyperparameter that corresponds to the number of neurons of the hidden layer). See illustration on Fig. 6.

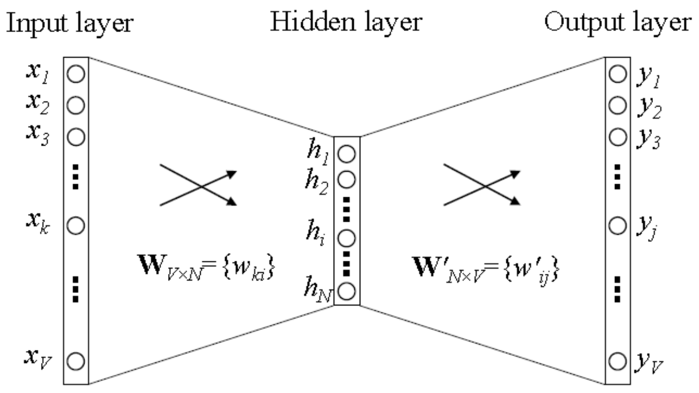


Figure 6: Illustration of the neural network used for the bigram model

The layers are fully connected. The weights between the input layer and the hidden layer can be represented by a $V \times N$ matrix $\mathbf{W}$, where each row of $\mathbf{W}$ is the $N$-dimension vector representation of the associated word of the input layer. For the hidden layer, we get $\mathbf{h} = \mathbf{W^T x}$, which is essentially copying the $N$-dimension vector representation of the context word to $\mathbf{h}$. We choose a linear activation function (directly passing the weighted sum of inputs to the next layer). From the hidden layer to the output layer, we use $\mathbf{W\prime} \in \mathbb{R}^{N\times V}$ as explained before. We then get an output $y_j$ for each word $y_j = (\mathbf{W\prime^T h})_\mathbf{j} = (\mathbf{W\prime^T W^T x})_\mathbf{j} = (\mathbf{E^T x})_\mathbf{j}$, where $\mathbf{E} = \mathbf{WW\prime}$, and then compute a score using softmax function.
We then use gradient descent and the backpropagation algorithm to update the weights until convergence.

Having explained the mechanism for the bigram model, we can now generalise easily to the n-gram model which is used in CBoW and uses the same type of network. The only difference is that we now consider that the word $\mathbf{\tilde{x}} = \mathbf{\tilde{x_i}}$ has c >1 words in its context $\mathbf{\tilde{x_1}}, ..., \mathbf{\tilde{x_{i-1}}}, \mathbf{\tilde{x_{i+1}}}, ..., \mathbf{\tilde{x_c}}$ where each $\mathbf{\tilde{x_k}}$ is a one-hot vector. In order to get an input vector that accounts for all the words in the context, the simplest way is to average and get the following vector $\mathbf{x}$ for our input word $\mathbf{\tilde{x}}$:

$$\mathbf{x} = \frac{1}{C}(\mathbf{\tilde{x_1}} + \mathbf{\tilde{x_{k_1}}} + ... + \mathbf{\tilde{x_{k+1}}} + ... + \mathbf{\tilde{x_{c+1}}}) = (\mathbf{x_1} ... \mathbf{x_V})^T$$

**Remark:** Averaging over the context is efficient, but a weighted average works better as it can reflect the importance of the words (for example according to the number of time they appear in a corpus of documents). One very efficient method is tf-idf, where weights increase proportionally to the number of times words appear in the document and is offset by the frequency of each word in the corpus

This gives us the following structure for our neural network, very similar to the result for the bigram model assumption (Fig. 7).

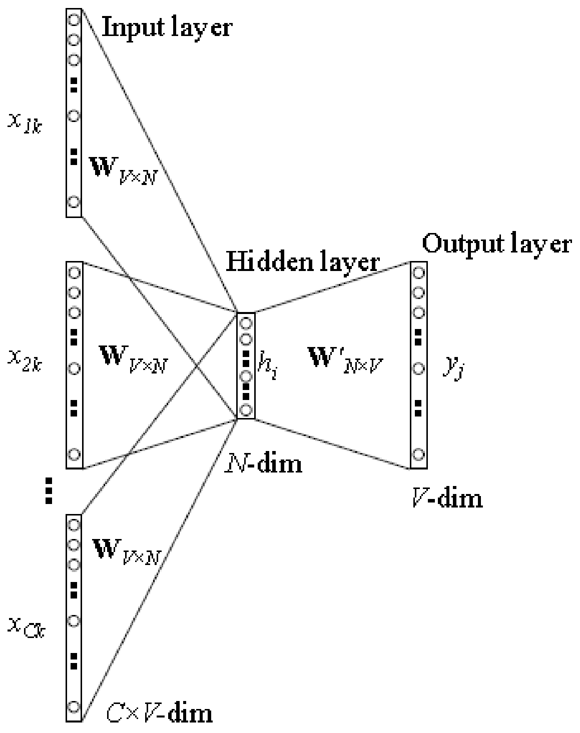


Figure 7: Illustration of the neural network used for the n-gram model

We define the hyperparameter N as the number of neurons on the hidden layer. We then use the matrix $\mathbf{W}$ to map the weights from the first to the hidden layer and $\mathbf{W}'$ for the weights from the hidden to the output layer, just like in the bigram model. So we get as output of the first layer $\mathbf{h} = \mathbf{W^T x}$. Similarly, for the next layer, we get an output $y_j$ for each word $y_j = (\mathbf{W\prime^T h})_\mathbf{j}$. We then use the softmax function to turn these scores into probabilities.

#### 2.6.2.4 Pre-trained Word2Vec Models

Before training our own Word2Vec model, we initially tried to use the available pre-trained models. Google, for instance, provides a pre-trained Word2Vec model based on texts from Google news [12, 16]. However, due to insufficient main memory on our LRZ compute cloud master node, we were not able to load it. On the other hand, we tested this model on our local machines to observe its accuracy but the word **Brexit** was not included in the model. This is most likely due to the model being trained before Brexit (its post on Google website [12] is dated back to 2013), meaning it will not be suitable for our purposes.
GloVe is another algorithm for obtaining vector representations for words [17] from which pre-trained models are available online (including the ones from twitter dataset) [7]. There are more pre-trained models for word embeddings from a big corpus [15], but they were all trained prior to Brexit and did not include the keywords related to this event. For the algorithm proposed in [15], for instance, it also provides an online demo of their pre-trained model [6] and it does not recognize the word **Brexit**. This imposes an important message. It is very difficult to both create and maintain a generic model for vector representations for words. Hence for an appropriate sentiment analysis on a particular subject, corresponding word embedding model must be trained accordingly.

### 2.6.3 Vector Representation Neural Network

#### 2.6.3.1 Tokenizing and Data Processing

As stated previously, in order for the sentiment analysis model to accurately understand the textual tweets, the tweets must be converted to vectors. This conversion from tweet to text heavily involves using Spark. The work-flow to change a tweet filled with symbols, links, many spaces, and other confusing parameters to a useful vector for training is shown in Figure 8. Please refer to Table 2 for an example of tokenization.

| id | tweet | tokenized tweet |
|---|---|---|
| #0 | ”I don’t like Brexit today #Brexit” | [I,don’t,like,Brexit,today,#Brexit] |

Table 2: An example of a tweet being tokenized by the spark.ml function, Tokenizer

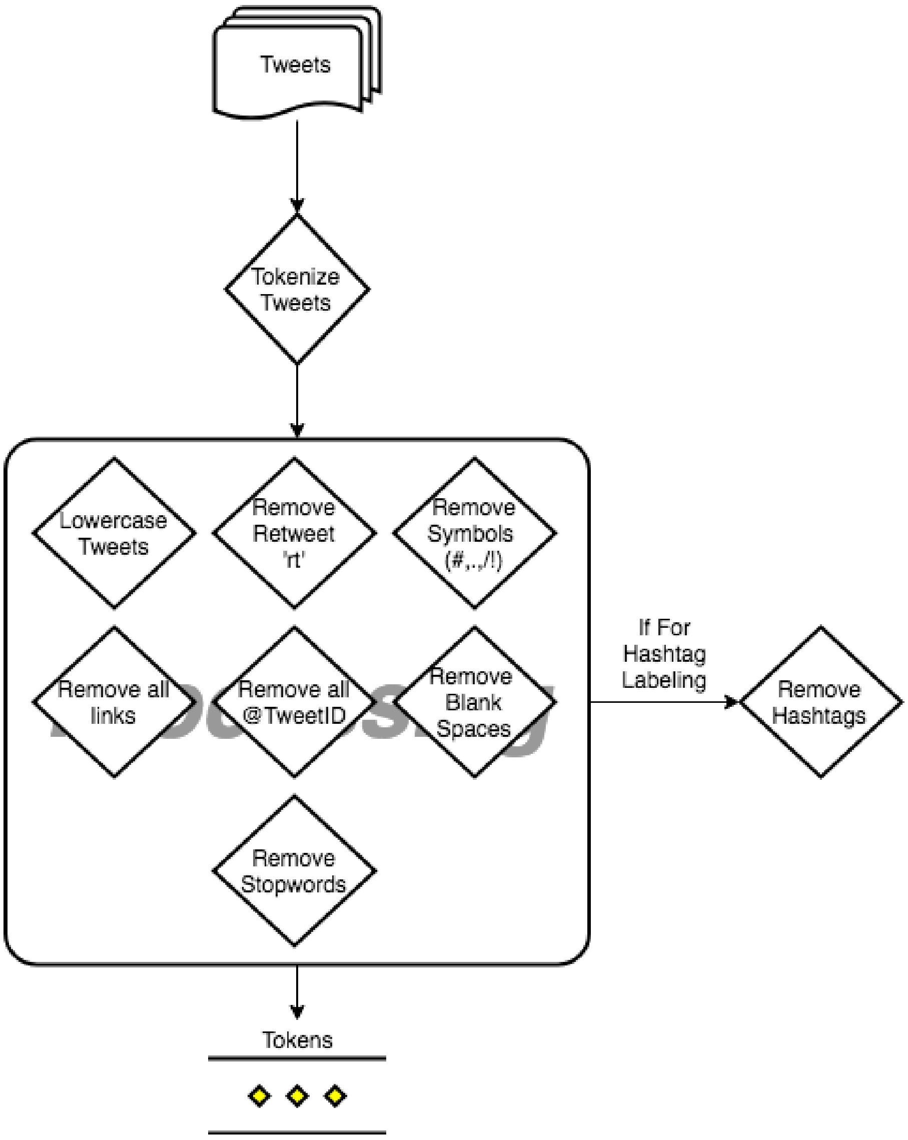


Figure 8: Work-flow for pre-processing tweets for conversion to vector representation

Tokenization works by splitting each tweet in to an array of ’tokens’. Each token is a word from a tweet. However, this requires a great deal of processing, As shown in Fig. 8. Processing the tokens and removing erroneous information is crucial for the Word2Vec Training because otherwise the Word2Vec model will be computationally slower and more importantly create inaccurate vectors for uninteresting/unreal words. The processing in Fig. 8 is described here:

– Lowercase the tweets: Tweets come in all shapes and sizes. Some twitter users will tweet with proper capitalization, while others may not use any capitalization. There are also occasional twitter users which tweet *iN ThIs FaShIoN*. Since Word2Vec is case-sensitive, it was decided to make all words lowercase.
– Remove retweet symbol: Some tweets are actually retweets from other users. This causes the tweet to have ”rt” at the beginning. These were removed as they have no significance.
– Remove symbols: Furthermore, words with symbols such as ”Brexit!!!” are considered a whole word and different from ”Brexit” when using Word2Vec. A user defined function was created in Spark to remove all symbols from tokens.
– Remove links: Links (e.g. https://www.tum.de) are also not words and were removed.
– Remove tweet IDs: Tweet IDs (e.g. @realDonaldTrump) were removed.

– Blank spaces were removed.

– Remove stop words: Spark.ml has a function called *StopWordsRemover* that removes all stop words. Stop words are words that are used frequently and don't carry much meaning such as 'the' or 'a'.

Once the tokens have been fully processed and created, the next step is to vectorize the tokens using Word2Vec.

#### 2.6.3.2 Vectorization of Tokens

In order to vectorize the tokens, Word2Vec from the spark.ml library was utilized. Table 3 describes the parameters chosen for the Word2Vec Model.

| Parameter | Description | Chosen Values |
|---|---|---|
| Vector Size | The size of the vector each word will have. | 10,20,100,200,400 |
| Minimum Count | The minimum amount of times a word must appear for consideration | 500 |
| Window Size | The amount of words around the target word taken to consideration | 7 |

Table 3: A description of the important parameters taken in by the Word2Vec model and the parameters we chose, as well as justification for their values

Since training of the Word2Vec model requires a great amount of time due to the amount of data being handled and hardware constraints, we chose one parameter to tune when creating our Word2Vec model. The chosen parameter was vector size. A larger vector space allows the word to have a complex definition and for each word to have a more succinct definition. We made the hypothesis that a larger vector space would correlate with better synonyms. A *qualitative* experiment was formulated in order to test a trained Word2Vec model's ability to associate words. Please refer to Section 3.2.1 for the results.

### 2.6.4 Classification into Pro/Against Brexit

The problem statement could be defined as separating the tweets into two classes - Tweets that support the clauses of the EU Referendum, hereby referred to as Brexit, and those that express sentiments against the same.

Our first and naive approach was to perform sentiment analysis using pre-trained libraries for sentiment analysis. There were several options considered for this approach - NLTK, TextBlob, Stanford CoreNLP, gensim etc. We considered each of the libraries and selected TextBlob. NLTK provides functions for various aspects of language processing such as stemmers and several text analysis algorithms for over 50 corpora, however, since our tweets are in English and TextBlob (which is built on top of NTLK and Pattern) combines the requires components of NLTK and the required functionalities in a clean package and provides a single function to perform Sentiment Analysis. Stanford Core-NLP requires a server-client architecture whereas TextBlob operates with out-of-box functions which can be fed to Spark. Since pySpark was our choice of platform for the task, TextBlob was the better option. While gensim has really powerful tools for topic modelling, there is no straightforward sentiment analysis API. Hence we chose TextBlob for its simplicity and compatibility with out problem statement. TextBlob uses a Naive Bayes analyser built on top a movie reviews dataset to classify sentiments.

To process our own dataset, a more specialized model, tuned to our dataset would give better results. For example, consider the following statement - "Staying in the EU is surely economically advantageous!" - this is a strongly positive sentiment. However, it is a strongly negative statement against Brexit. Hence a naive sentiment classifier is erroneous in this situation. Hence, we formulate different methods of labeling data as described in above sections and train several learning models and observe their accuracy. The learning algorithms we used are the following:

– Logistic Regression

– SVM with linear kernel

– Neural Network

Rationale behind picking each of these algorithms was based on some literature survey as well as based on interactions with a Capgemini employee who has done previous work on a very similar dataset. In his work, logistic regression was the best performing model with a 75% accuracy and hence, we decided to pick up logistic regression as the baseline model. Based on the survey of articles and blogs on text classification, we observed that SVMs generally perform well on text data. Hence SVM was the next learning model choice. Finally, for the volume of data we possessed, intuition suggests that a deep learning network would perform well. This was our third model of choice.

In this section we describe the input for classification and the different learning models used in short in context of our problem.

#### 2.6.4.1 Input into learning algorithm

The output of the first neural network, i.e. Word2Vec gives us a 2D matrix where each row and each column represents a vector corresponding to the top words. To transform a tweet into such a vector, we first tokenize it as described in above sections and create a word vector for each tweet by taking applying a tf-idf weight to each token vector.

$$tf - idf_{w,t} = (tf_{w,t}) \cdot \log(1 + \frac{1 + n_t}{1 + df_{w,t}}) \tag{1}$$

where $tf_{w,t}$ is the number of times a word w appears in a tweet t, $df_{w,t}$ is the number of tweets in which the word appears and $n_t$ is number of tweets. This is performed using TfidfVectorizer from sklearn library. We produce such a weighted token word vector for each word and sum them over all tokens to create an input vector $x$ for our learning algorithms.

#### 2.6.4.2 Learning Algorithms

**Logistic Regression**

A logistic function can be defined as given below

$$g(z) = \frac{1}{1 + exp(-z)} \tag{2}$$

where z is the output of a linear model computed with our features.

$$z = \beta^T x \tag{3}$$

Here x, is the feature vector which is the output of a tf-idf Word2Vec weighted model as described in the previous section. The $\beta$'s are obtained by minimizing the below loss function.

$$J(\beta) = \frac{1}{N}\sum_{i=1}^{N}[-y_i log(h_\beta(x_i) – (1–y_i) log(1 - h_\beta(x_i))] \tag{4}$$

where

$$h_\beta(x) = g(\beta^T x)$$

This is done by differentiating the function and setting it to 0.

$$\frac{dJ(\beta)}{d\beta} = \frac{1}{N}\sum_{i=1}^{N} x_i(h_\beta(x_i) - y_i) = 0 \tag{5}$$

The $\beta$'s obtained in this manner are then used to calculate z and the logistic function given in the beginning of the section is used to compute the odds of the tweet falling in category positive Brexit sentiment (value >0.5) or negative Brexit sentiment (value <0.5).

We use LogisticRegression from sklearn library to learn our data.

**SVM**

The basic idea of SVM is to find a hyperplane that separates the different classes of data. It can be formulated at an optimization problem as follows

$$\min_W \|W\|^2 \tag{6}$$

subject to

$$y_i(W^T x_i + b) \geq 1 \quad \forall i = 1...N$$

A kernel represents the kind of algebraic transformation we do to the data to identify the hyperplane. Here, we have used a linear kernel, which is most recommended for text classification tasks, i.e, the data is not transformed.

We use SVC from sklearn library to perform SVM on our data.

**Neural Networks**

The basic functioning and structure of a neural network are already described in the previous sections about Word2Vec. Here, we will describe the input layers, output layers, and other relevant parameters. We used Keras with Tensorflow for implementing the neural network. TensorFlow is an open-source library for building large neural networks and Keras is a high level neural network API that can be used out of the box to construct different layers of the network. It also allows us to compile the entire network where we can customize the optimizers for learning rate, specify the loss function and the validation metrics.
The input to the network is $x$, a weighted tf-idf vector of tokens as described above. It was partitioned into a ratio of 8:1:1 for training, validation and testing respectively. There is a single output neuron which provides the likelihood of the tweet being pro-Brexit. The sigmoid function

$$g(z) = \frac{1}{1 + exp(-z)} \tag{7}$$

is used for activation at the every layer to introduce non-linearity at the layer. The loss function used was a binary cross entropy function, also known as softmax loss given by

$$L = -\frac{1}{m}\sum_{i=1}^{N} y_i \, log(\hat{y_i}) + (1 - y_i) \, log(1 - \hat{y}) \tag{8}$$

where $\hat{y_i}$ is the predicted score for the data point i and $y_i$ is the target value.
Our initial and base network structure and parameters for the network were inspired by the work in [9]. We have a 2-layer neural network as shown in the figure where input layer $\in \mathbb{R}^{\mathbf{400}}$, hidden layers $\in \mathbb{R}^{\mathbf{32}}$ and output layer $\in \mathbb{R}^{\mathbf{1}}$

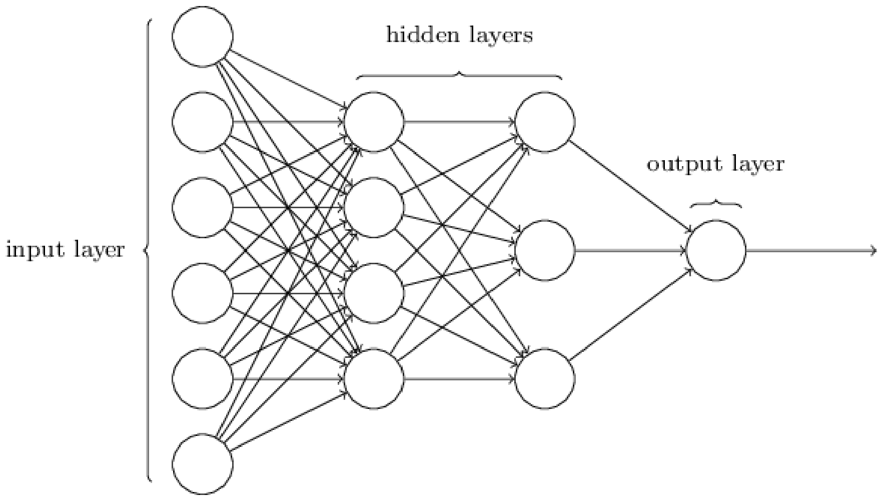


Figure 9: Two layer Neural Network

The optimizers used were:

- RMSProp - Here, the weights are updated during backpropagation using the following

$$s_{k+1} = \beta s_k + (1 - \beta)[\bigtriangledown_\theta L \odot \bigtriangledown_\theta L] \tag{9}$$

$$\theta_{k+1} = \theta_k + \alpha \frac{\bigtriangledown_\theta L}{\sqrt[2]{s_{k+1}} + \epsilon} \tag{10}$$

  where $\theta$'s are the weights, $\beta$ is a friction hyperparameter and $\bigtriangledown_\theta L$ is the gradient.

- Adam - Here, the weights for the network were updated using the following

$$s_{k+1} = \beta_1 s_k + (1 - \beta_1)[\bigtriangledown_\theta L \odot \bigtriangledown_\theta L] \tag{11}$$

$$v_{k+1} = \beta_2 v_k + (1 - \beta_2)(\bigtriangledown_\theta L) \tag{12}$$

$$\theta_{k+1} = \theta_k + \alpha \frac{v_{k+1}}{\sqrt[2]{s_{k+1}} + \epsilon} \tag{13}$$

  where $\theta$'s are the weights, $\beta_1$ is a friction hyperparameter, $\beta_2$ is a velocity hyperparameter and $\bigtriangledown_\theta L$ is the gradient.

We also tested the network with dropout probabilities of 1 and 0.5. The dropout probability is the probability with which every node is dropped, in a given iteration.
Other hyperparameters that were tuned are:

– Number of hidden layers - The number of hidden layers is an hyperparameter that is adjusted for optimal accuracy. The number of layers which were varied from 2 to 7.
– Number of neurons - The number of neurons were 32 or 64 in each hidden layer.
– Number of epochs - The number of epochs were varied from 5 as in the base model to 50 to identify the threshold of overfitting and adjust accordingly.

#### 2.6.5 Data Labeling

Despite the scope of our project being sentiment analysis, we are interested in predicting whether each tweet describes positive or negative sentiment towards Brexit. This makes the addressed problem to be a text classification problem via supervised learning. Supervised learning requires providing of *answers* of each data. The most straight-forward, possible the most accurate, yet time-consuming approach would be manually labeling each tweet as either pro-Brexit or against-Brexit. There are commercial services for tasks like this, such as Amazon Mechanical Turk [1], a web-service that provides on-demand human workforce. Such services may be useful, but certainly is not of consideration for projects like this. Other approach would be to automate this labeling process based on certain classification criteria. For this project, we took both of the approaches as follows:

– Manual labeling
– Automated labeling on influencers' tweets
– Automated labeling via Hashtags

##### 2.6.5.1 Manual Labeling

For manual labeling, we obtained 2000 tweets from **Gold Standard Brexit Tweets** [3]. These tweets are manually labeled, by a group of professionals, as pro-brexit, neutral, irrelevant, and against-brexit [13]. This dataset also provides strengths between 1 (very weak) and 5 (very strong). We ignored neutral and irrelevant labels, and strength due to accuracy reasons (see Section 3.2.2). After removing the tweets with neutral and irrelevant labels we were left with 1246 tweets for both training and testing (37.7% of the tweets were labeled as either irrelevant or neutral [13]). 80% of the tweets were used for training (997 tweets) and 20% for testing (249 tweets).

##### 2.6.5.2 Automated Labeling Based on Influencers

We hand-picked 49 influencers from our tweet dataset and labeled their tweets (in total of 11000 tweets) based on influencers' attitude/opinion towards Brexit. The hand-picked influencers were categorized as pro/against brexit or undetermined. The influencers with undetermined or neutral view towards Brexit were removed due to accuracy reasons (see Section 3.2.2). Table 4 shows a sample set of influencers and their general opinion about Brexit. A full list of the influencer can be found in Appendix 6.3. Influencers' view towards Brexit were determined based on online research, and in total, 27 influencers were considered. Of these 27 influencers, we could obtain 11000 labeled tweets for training and testing. Same ratio of train-test tweets were used as manual labeling (80% for training and 20% for testing).

| **Account** | **Occupation** | **Pro/Against Brexit** |
|---|---|---|
| OwenJones84 | Author & Guardian columnist | Against |
| MhairiBlack | Scottish National Party MP | Against |
| tom_watson | Labour Party deputy leader | Against |
| GuidoFawkes | Right-wing political blogger | Pro |
| stellacreasy | Labour MP | Pro |
| JohnRentoul | Independent chief political commentator & author | Against |
| Kevin_Maguire | Daily Mirror associate editor & New Statesman columnist | Against |

Table 4: Sample list of influencers considered for automated labeling. Influencers with neutral/undetermined view towards Brexit were removed. Total of 27 influencers were considered from which 11000 tweets were labeled

#### 2.6.5.3 Automated Labeling Based on Hashtags

Another labeling method implemented was to label the tweets based on the hashtags used. The tweets were filtered to only incorporate tweets which included hashtags that are clearly either Pro or Against Brexit. Table 5 shows the hashtags chosen for this dataset. Pro Brexit and Against Brexit tweets were labeled with binary 1 and 0, respectively.
In total, this created a labeled dataset of 6 million tweets, by far our largest dataset for training/testing. Due to memory constraints, this limited the amount of data used to train the model to 200,000 tweets.

| Hashtag Used | Pro/Against Brexit |
|---|---|
| #voteleave | Pro |
| #leaveeu | Pro |
| #outofeu | Pro |
| #ukip | Pro |
| #remain | Against |
| #strongerin | Against |
| #bettertogether | Against |
| #leavechaos | Against |
| #voteleavegetchaos | Against |

Table 5: Hashtags used to automatic labeling. Each hashtag was chosen based on a review of a sample set of tweets for the respective hashtag

## 2.7 Network Analysis

### 2.7.1 Model Representation

Twitter's Social Network may be readily translated into a graph structure. Here, we define various aspects of the social network and correlate the semantic meaning from it to the different components of a directed graph.

- **Nodes**: Each node of the graph represents a user or a unique user id, also known as a twitter handle. In this text, we use the phrases 'user', 'unique user id', or 'unique twitter handle' interchangeably.
- **Edges**: Each directed edge $\mathbf{E_{i,j}}$ represents a directed edge from user i to user j, indicated that user i follows user j, i.e. user i subscribes to the tweets published by user j.
- **Edge Weights**: In our model, we assign a numeric weight to each edge $\mathbf{E_{i,j}}$ to indicate the amount of information flow through the edge. We assign the value as the number of re-tweets by person i of tweets written by person j. The rationale behind this assignment is that the number of re-tweets indicates the strength of influence, i.e. a user more likely to re-tweet is more influenced.
- **Node Weights/Metrics**: In this model, each node will be assigned a score indicative of the influence of the node and is calculated the based on graph structure and edge weights.

We define our graph to be an adjacency matrix $\mathbf{G} \in \mathbb{R}^{\mathbf{n \times n}}$ to represent a social network with n users. Each element $\mathbf{e_{i,j}}$ is the weight of corresponding edge.
However, storing such a data as a matrix is inefficient since it would be a sparse matrix with most of the values as 0. This is because, out of the millions of users, only few users would have a similar order of followers and most of the users would have a few hundred or thousand at the most. Hence, the most optimal way to store the followers would be as a map with each user as a key and the list of followers as a value. This would drastically reduce the size for loading all the followers and computing metrics from the data in-memory. However, the disadvantage of this is that conventional algorithms operate on a matrix and such algorithms may not function on such an adjacency list.

### 2.7.2 Metrics

In this section, we develop some common graph metrics to fit our context. We explore the intuition behind of each metric in this scenario of Twitter's network. We also discuss some challenges with computing these common metrics and propose an extended metric formulation.

- **Degree centrality**: This metric is defined by the number of followers a user has. If we have our graph $\mathbf{G} \in \mathbb{R}^{\mathbf{n \times n}}$ and a vector of 1's, $\mathbf{k} \in \mathbb{R}^{\mathbf{n}}$ , then degree of each user is given by the matrix $\mathbf{D} \in \mathbb{R}^{\mathbf{n}}$

$$\mathbf{D} = \mathbf{kG} \tag{14}$$

  This metric of influence indicates that users with higher number of direct followers are more influential.

– **Betweenness centrality**: This metric is based on the intuition that connecting nodes are influential, that is, a user who follows many publishers and has many subscribers is more influential as he is capable of distributing information to a wide audience. For example - a user who tweets exclusively about golf may have lesser followers than a user who re-tweets sports related tweets from multiple sources. This user hence is an influencer as he lies between the source of information and the sink. It is calculated as follows:

$$betweenness(X) = \frac{sp(X)}{sp} \tag{15}$$

where sp(X) denotes the number of shortest paths in the graph where X is a part of the path and sp is the total number of shortest paths.

– **Closeness centrality**: This metric tells us how close a user is to all the other users. This tells us the chance with which a message passed by a user reaches other users. If a user is very close to many users, best case distance 1, this becomes the degree measure and all his neighbors receive his message. We define it using

$$closeness(X) = \sum_{i \in n} \frac{1}{d(i, X)} \tag{16}$$

where d(i,X) is the shortest distance from user X to user i.

– **Page-rank inspired metric**: Page-rank is an algorithm used to measure popularity of webpages on the internet. The rank of the page is proportional to the number of incoming links from other pages and the rank of those pages. However, we can adapt the same to our model by replacing a page with a user and an outgoing link with a follower edge. Using this algorithm is beneficial over the other measures because of multiple reasons. It incorporates degree measure, since more incoming links implies higher rank. The closeness is also measured since the rank contributed by a particular link is propagated forward through each link. However, the biggest advantage is that the algorithm can be computed in a distributed manner without loading the entire adjacency matrix in memory. Regular updates can also be performed when new nodes or edges are added, without recomputing the whole graph.

### 2.7.3 Tools

Although one could hard-code a graph data structure in distributed fashion, there are numerous ready to use tools available within Apache Spark. The following list provides a summary of the tools we have considered:

– **GraphX**: This is Apache Spark's API for graphs and parallel graph manipulation/computation. Although GraphX is highly flexible, widely used, and native to Apache Spark, it only supports Scala

– **Neo4j**: Neo4j is a freemium graph database management system. Similar to GraphX, Neo4j can also be used within Apache Spark framework and supports .Net, Java, JavaScript, and Python. Unfortunately, Neo4j is freely available only for a single node application and requires commercial subscriptions for parallel computations

– **GraphFrames**: GraphFrames is a freely available package for graph structures in Apache Spark framework. Unlike GraphX, which is based on Spark RDD files, GraphFrames is based on Spark data frames. It supports high-level APIs in Scala, Java, and Python, and parallel computations on a Spark cluster

GraphX certainly was the first option to consider since it comes natively with Spark. However, all of our tasks were being conducted with Python and it did not seem sensible to necessarily use Scala just for graph processing. Neo4j, despite its Python support, was not freely available for parallel processing, making GraphFrames our pick for network analysis.

### 2.7.4 Challenges

Data required for constructing the adjacency matrix/list:

1. List of users in the network
2. List of followers for each user
3. Number of re-tweets between two users

We can extract the list of users in the Brexit network from the tweets. We can also count the number of re-tweets from the Brexit related tweets. However, we do not possess the follower data and cannot extract or compute the required data from the current data dump.

### 2.7.5 Steps Taken

We tried to gather the required data from Twitter. We tried the following to extract the data.

1. Registered for Twitter Developer Account
2. Mined the list of users from Brexit tweets
3. Developed a script using python-twitter library to extract followers for each user

However, due to limitations of API, only 5000 followers can be extracted per minute. Since the influential users have at least a few million followers each, it would take hours to extract the data of even one user. This made it infeasible for us to extract the data required for performing network analysis.

## 3 Results & Discussion

## 3.1 Timeline Visualization

### 3.1.1 Timeline analysis dashboard

As explained in the previous section, we chose to make a dashboard using Mapd, below is a description and justification of the metrics that were chosen. Please see Figure 16 in Appendix for the dashboard, where you can find the following representations:

– **Number of tweets per hashtag:** We included a bar graph representing the number of tweets per hashtag so the user can have at first glance an idea of the ratio of the number of tweets per hashtag. Not very surprisingly, # Brexit is the most used hashtag (more than 550% times more used than the second most used hashtag # EURef).

– **Correlation of tweet frequency and real-life events:** Our main aim was to see how well activity peaks on Twitter relate to real-life events. So as a second part of the dashboard, we superposed a line graph showing tweets frequency over time with a bar graph representing the number of events that happened on a given day. A table with an explanation of the event is provided on the dashboard for understanding of the data.

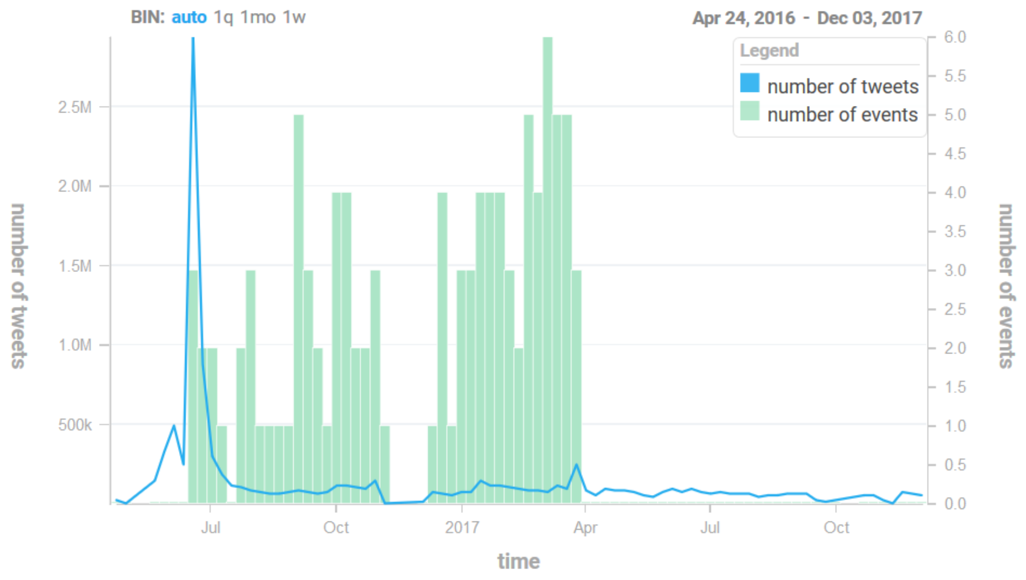


Figure 10: Tweet frequency - the bars represent the number of events happening per week

This shows us several interesting peaks of Twitter activity:

– **June 5th to June 11th, 2016:** There doesn't seem to be any major Brexit-related event during that week that could explain this peak - one explanation could be the campaigns of the "Pro" and "Against" sides on Twitter and the surrounding debates before the referendum on June 23rd.

– **June 19th to June 25th, 2016:** This is the week of the Brexit Referendum (June 23rd). It is also by far the highest peak of activity (approximately 6 times the second biggest peak two weeks before).

  - **October 30th to November 5th 2016:** The High Court rules U.K. must hold a vote in Parliament before starting Brexit process on November 3rd. Concretely, this means that UK ministers cannot start Brexit Process (triggered by Article 50) without consulting the Parliament as they had hoped. This event received a lot of public criticism, opposing the "Remain" and "Leave" supporters.
  - **March 26th to April 1th, 2017:** On March 29th in Brussels, Tim Barrow (UK's ambassador to the EU) hands Donald Tusk (President of the European Council) the letter which begins two years of talks. Theresay May tells lawmakers in London that "this is an historic moment from which there can be no turning back."

  We could of course argue that the bar graph event representation is not the best for that kind of visualisation (a display of the event when the mouse hovers over the peak was originally what he had aimed for), but we were restricted by the possibilities of MapD. That is the disadvantage of using a built-in solution and not build our dashboard from scratch with the plotly library on Jupyter notebooks... but it is largely compensated by the speed of MapD.
  Moreover, another disadvantage of this representation is linked to the available data for our graph: the Bloomberg timeline we are using to explain the events onl extends from June 23rd to March 29th, 2017 - which means that all explanations are not directly available on the dashboard.

- **Tweet frequency per hashtag:** Seeing how the use of hashtags changes along time, particularly before/after referendum, is also a good indicator of popular opinion. In our case, #Brexit was used more than all others, but there is no striking difference in the variation in use of a hashtag over time.
- **Tweets per country and user information:** One graph illustrate how tweets are split between different countries (not surprisingly, a huge majority comes from the UK, followed by the US and mostly European countries. An important information to bear in mind is that data regarding country of publication of the tweet was available for less than 25% of our dataset! Indeed, the user chooses whether his location can be available or not. So even if the results seem coherent with the real-life situation, they will not enable us to detect possible "manipulations". For example, data scientists at the University of Swansea and University of California, Berkeley found that over 150 000 accounts based in Russia posted content relating to Brexit in the days leading up to voting day on June 23, 2016 (according to this Cnet article [4]).
  Finally, a table giving the number of followers per user show who were the most influential users over this time period.

The results of this timeline representation coincide with real-life Brexit related events. However, individually they do not bring great insights or unexpected results regarding the Brexit referendum, and should be considered as a basis to perform a sentiment analysis over time to identify trends.

## 3.2 Sentiment Analysis

### 3.2.1 Word2Vec

We tested 4 different models with different vector sizes to determine how vector-size could affect performance and similarity between words. To compare each model, we chose 5 words that are highly related to Brexit and compared the words that came up as the 20 most similar words. 40% of the data was used to evaluate the models. The results are displayed in Tab. 7. The 20 most similar words were then placed in the categories **Similar**, **Related**, **Opposite**, and **Irrelevant**. An example of the output from Word2Vec which then had to be manually classified is shown in Tab. 6. From observing both tables, it is interesting to note how the similarity values decrease significantly with vector space, yet the amount of similar and related words increase with vector space. The results (7) clearly show that larger vector spaces increase similarity. It is also important to note that a vector space of 200 also performed well and could be used.
Once the best parameters were chosen, the Word2Vec model was trained on 80% of the data. This is the model used in the Sentiment Analysis Pipeline. The final Word2Vec model contained over 18,000 words in its dictionary and was especially suited for words pertaining to Brexit events.

### 3.2.2 Brexit Gold Standard Dataset

On the initial Brexit Gold standard dataset [3] where strength of a tweet was rated between -5 (strongly against Brexit) and 5 (strongly pro Brexit), we obtained 31% accuracy with neural network approach (10 epochs).
During testing phase, most of the tweets (>90%) were classified as neutral. This can mostly be attributed to the fact that a major chuck of the training data was neutral (37%) and the rest were split between the corresponding strengths (1 to 5 in each polarity). This greatly diluted the data available for the network to learn the features of each strength.

| VectorSpace: 10 | Similarity Values | VectorSpace: 400 | Similarity Values |
|---|---|---|---|
| based | 0.9370 | control | 0.4102 |
| continuously | 0.9350 | migration | 0.3486 |
| peril | 0.9289 | govern | 0.3141 |
| machine | 0.9279 | doesnt | 0.2922 |
| injustice | 0.9273 | nation | 0.2868 |
| enhanced | 0.9258 | wants | 0.2849 |
| commit | 0.9214 | immig | 0.2737 |
| privatising | 0.9197 | oreillyfactor | 0.2732 |
| swamped | 0.9179 | bilateral | 0.2710 |
| native | 0.9173 | immigrants | 0.2693 |
| vanity | 0.9161 | trea | 0.2670 |
| massively | 0.9153 | screaming | 0.2649 |
| petershore | 0.9135 | borders | 0.2640 |
| hurts | 0.9131 | racist | 0.2627 |
| avoid | 0.9113 | brightest | 0.2617 |
| lie | 0.9107 | 333000 | 0.2607 |
| bbcaq | 0.9096 | amp | 0.2599 |
| wasteful | 0.9095 | skilled | 0.2562 |
| discriminate | 0.9095 | based | 0.2525 |
| falsehood | 0.9091 | antiimmigration | 0.2505 |

Table 6: Word2Vec Example for the word **Immigration**. The results on the left are for a Word2Vec model trained with a vector of size 10 and the results on the right are with a vector size of 400.

Hence we transformed the training set into a binary classification problem, that is, pro-Brexit tweets as 1 and against as 0. Table 8 show the corresponding results.
Other than the training methods described in Section 2.6.4 we also tested TextBlob, a generic pre-trained library for Natural Language Processing (NLP) [11]. As observed, TextBlob gives a poor result. This could be due to the fact that it is trained on a very generic dataset that does not contain the vocabulary necessary for this classification task. Hence, it seems crucial to learn a word embedding using the use-case specific data.
Of the learning models, we observe a particularly poor result for SVM with RBF kernel. This may be because, linear kernels perform the best for text classification using SVM as there is a high dimensional input space and document vectors are sparse [14]. Neural network approach gave the best performance, although not satisfactorily, among the methods used giving 76% accuracy.

### 3.2.3 Automated Labeling Based on Influencers

Given the amount of data we have, there is no way we can manually label all the datasets and an automated labeling strategy is a must, unless small subset of data with manual labeling gives excellent performance (though is not the case here as we can see from Section 8). Table 9 shows the accuracy of different trained models.
Regardless of the model used, influencer based automated labeling showed highly inaccurate performance. The best accuracy results, obtained from neural network approach, only gave 50% accuracy. We must note this automated labeling approach based on influencers is a very naive method. From the results, we can observe that automated labeling on its own is a very important task that needs careful thought. Depending on the labeling strategy, be this any sophisticated or simple as influencer or hashtag based, the results can be very poor as it is here or even outperform manually labeled datasets (see Section 3.2.4) by hiding the lack of label accuracy with big set of train data. Such bad performance could be due to influencers' tweets containing too much neutral/irrelevant information degrading the quality of labeled data. It is also very likely that, for instance, the tweets from pro-Brexit influencers contain both pro and against Brexit hashtags, and vice-versa generating an ambiguous dataset. Considering the high impact of hashtags for classification accuracy (Section 3.2.4), this mix use of hashtags can significantly degrade the results.

### 3.2.4 Automated Labeling via Hashtags

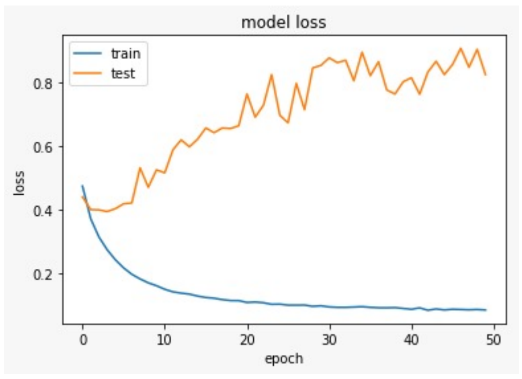
This automated labeling strategy along with neural network gave us the best performing model with a preliminary accuracy of 78%. Due to its highest accuracy we decided to use this model for hyper-parameter tunning and training on bigger datasets to improve the accuracy. We performed hyperparameter tuning on this dataset and obtained accuracies as given in Tab. 10.
The best accuracy was noted for a 5-layer network with 64 neurons using no dropout and Adam to update the gradient. We trained this network for 50 epochs to observe the validation losses and accuracy over time.

| Word | VectorSpace | Similar | Related | Opposite | Irrelevant |
|---|---|---|---|---|---|
| Immigration | 10 | 0 | 1 | 0 | 19 |
| | 20 | 1 | 3 | 0 | 16 |
| | 100 | 3 | 1 | 0 | 16 |
| | 200 | 2 | 5 | 0 | 13 |
| | 400 | 4 | 3 | 0 | 13 |
| Independence | 10 | 1 | 5 | 0 | 14 |
| | 20 | 2 | 7 | 1 | 10 |
| | 100 | 4 | 6 | 0 | 10 |
| | 200 | 5 | 10 | 0 | 5 |
| | 400 | 3 | 10 | 1 | 7 |
| Referendum | 10 | 2 | 1 | 0 | 17 |
| | 20 | 3 | 3 | 0 | 15 |
| | 100 | 5 | 4 | 0 | 11 |
| | 200 | 5 | 5 | 0 | 10 |
| | 400 | 7 | 3 | 0 | 10 |
| Vote | 10 | 7 | 1 | 0 | 12 |
| | 20 | 3 | 9 | 0 | 8 |
| | 100 | 4 | 2 | 0 | 4 |
| | 200 | 4 | 11 | 0 | 5 |
| | 400 | 4 | 11 | 0 | 5 |
| Economy | 10 | 1 | 11 | 0 | 8 |
| | 20 | 0 | 15 | 0 | 5 |
| | 100 | 1 | 13 | 0 | 6 |
| | 200 | 3 | 15 | 0 | 2 |
| | 400 | 4 | 13 | 0 | 3 |

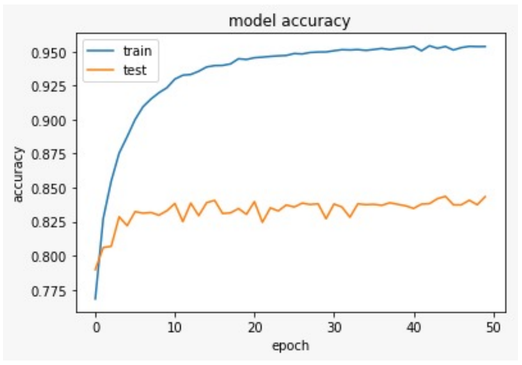
Table 7: Results from the qualitative analysis of the 20 nearest synonyms, output by 5 different Word2Vec models for 5 words.

| Method | Accuracy |
|---|---|
| TextBlob | 36% |
| Neural Network with dropout (15 epochs) | 76% |
| SVM (linear kernel) | 75% |
| SVM (RBC kernel) | 62% |
| Linear Regression | 72.5% |

Table 8: Training results from manually labeled Brexit Gold Standard tweets **excluding** neutral labels and sentiment strengths



(a) Model Loss



(b) Model Accuracy

Figure 11: Loss and Accuracy for 5-layer network with 64 neurons using no dropout and Adam trained on Hashtag-based labeled data

We observe that the validation accuracy saturates after 10 epochs at a much lower value than training accuracy. While this is normal since higher training accuracy may be due to overfitting, we attempted to include learning rate decays of $\epsilon^{-6}$ and $\epsilon^{-4}$ to gauge whether the accuracy was saturated below optimum due to large learning

| **Method** | **Accuracy** |
|---|---|
| Neural Network (10 epochs) | 50% |
| SVM (linear kernel) | 30% |
| SVM (RBC kernel) | 12% |
| Linear Regression | 30% |

Table 9: Training results from tweet dataset with automated labeling based on influencers

|  | Optimiser | SGD |  | RMSProp |  | Adam |  |
|---|---|---|---|---|---|---|---|
|  | Dropout | 0 | 0.5 | 0 | 0.5 | 0 | 0.5 |
| Layers | Neurons |  |  |  |  |  |  |
| 2 | 32 | 0.79 | 0.79 | 0.82 | 0.82 | 0.82 | 0.82 |
| 2 | 64 | 0.79 | 0.79 | 0.83 | 0.82 | 0.82 | 0.82 |
| 3 | 32 | 0.80 | 0.77 | 0.82 | 0.81 | 0.81 | 0.81 |
| 3 | 64 | 0.81 | 0.79 | 0.82 | 0.82 | 0.82 | 0.82 |
| 4 | 32 | 0.81 | 0.78 | 0.83 | 0.80 | 0.82 | 0.80 |
| 4 | 64 | 0.82 | 0.79 | 0.81 | 0.82 | 0.82 | 0.82 |
| 5 | 32 | 0.80 | 0.78 | 0.81 | 0.80 | 0.81 | 0.80 |
| 5 | 64 | 0.82 | 0.78 | 0.83 | 0.81 | 0.83 | 0.82 |
| 6 | 32 | 0.82 | 0.74 | 0.81 | 0.79 | 0.82 | 0.80 |
| 6 | 64 | 0.81 | 0.77 | 0.82 | 0.81 | 0.82 | 0.80 |
| 7 | 32 | 0.81 | 0.57 | 0.81 | 0.80 | 0.82 | 0.78 |
| 7 | 64 | 0.82 | 0.75 | 0.81 | 0.81 | 0.83 | 0.78 |

Table 10: Accuracy with different hyperparameters from tweet dataset with automated labeling based on hashtags

rate. However we did not observe a notable difference in the accuracy in both the above cases. It was concluded that the accuracy may not necessarily be saturated due to high learning rate. We record the classification scores and the confusion matrix for the selected model in Tab. 11 and Tab. 12, respectively.

|  | **Precision** | **Recall** | **f1-score** | **Support** |
|---|---|---|---|---|
| 0 | 0.84 | 0.83 | 0.84 | 5486 |
| 1 | 0.80 | 0.81 | 0.80 | 4513 |
| Avg/Total | 0.82 | 0.82 | 0.82 | 9999 |

Table 11: Classification Scores (5 layer NN with 64 neurons, Adam, no dropout, Hashtag based training data)

Here we observe the division in the misclassification of tweets. The misclassification rates seem uniform for false-positives and false-negatives. Since the text available is uniform, i.e. for every tweet that expresses a positive sentiment about "Stay in EU", there may be a positive sentiment about "Leave EU". Similarly, for every tweet that expresses a negative sentiment about "Stay in EU", there may be a negative sentiment about "Leave EU". Hence it follows that the false positive and false-negative rates must have a similar distribution. This is also reflected in the Receiver Operating Characteristic (ROC) curve (Fig. 12).

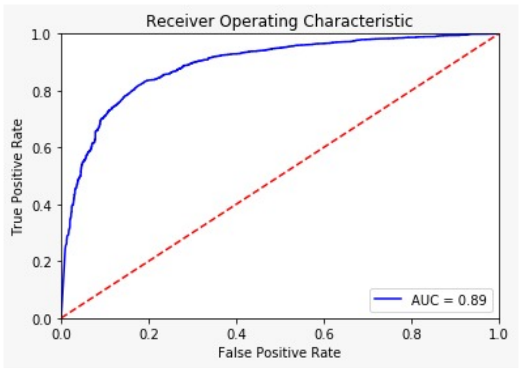


Figure 12: Receiver Operating Characterstics (5 layer NN with 64 neurons, Adam, no dropout, Hashtag based training data)

|  |  | Pro-Brexit |  |  |
|---|---|---|---|---|
|  |  | Pro | Against | Total |
| Against-Brexit | Pro | 4564 | 922 | 5486 |
|  | Against | 879 | 3634 | 4513 |
|  | Total | 5443 | 4556 | 9999 |

Table 12: Confusion Matrix (5 layer NN with 64 neurons, Adam, no dropout, Hashtag based training data)

Here we observe that we obtain the maximum true positive rate 0.8 for a false positive rate of 0.2 which also matches with our accuracy measures of 82%.
In general, we observe that we obtain a larger classification accuracy for hashtag based labeling than manual labeling or influencer based labeling. This may be explained by the fact that while a single person may express varying degree of sentiments about a topic, the user uses a strong polarized hashtag only when he expresses a strong opinion about the topic. However, we believe that the higher accuracy over manually labeled data is due to the difference in volume of training data.

## 3.3 Sentiment Analysis over Time

The resulting dashboard (Fig. 17 in Appendix 6.4) is particularly interesting as it brings insights both from the time-line analysis through tweet frequency and our sentiment classification model. The aim is to draw conclusion from the association of these two models. To do so, we represented the following graphs:

– **Tweet frequency with sentiment over time** (Fig. 13): This representation shows the repartition of pro and against Brexit tweets. Over time, the proportion of pro Brexit tweets diminishes. They are predominant until June 2016 (approximately 61%), roughly the same as the number of against Brexit tweets during the week of the referendum, and account for only 39% of all tweets in mid July 2017.

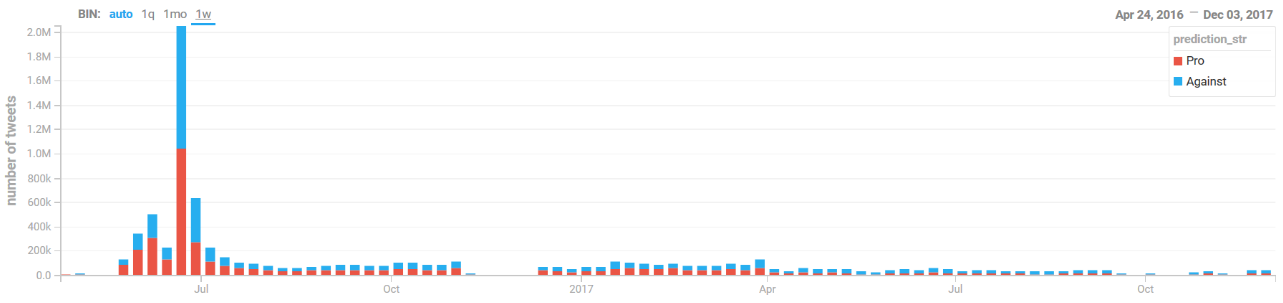


Figure 13: Tweet frequency with sentiment over time

It would seem that people are progressively changing their mind on Brexit from the moment the vote took place. Let us confirm this fact by correlating changes in sentiment with the events that took place (Fig. 14).

– **Number of tweets and average sentiment over time** (Fig. 14): Here, we will analyse the graph we represented the average sentiment score over time, 1 being completely pro Brexit and 0 being completely against Brexit. So an alternative way to view this score is a percentage of *how strong pro Brexit tweets are* over time.

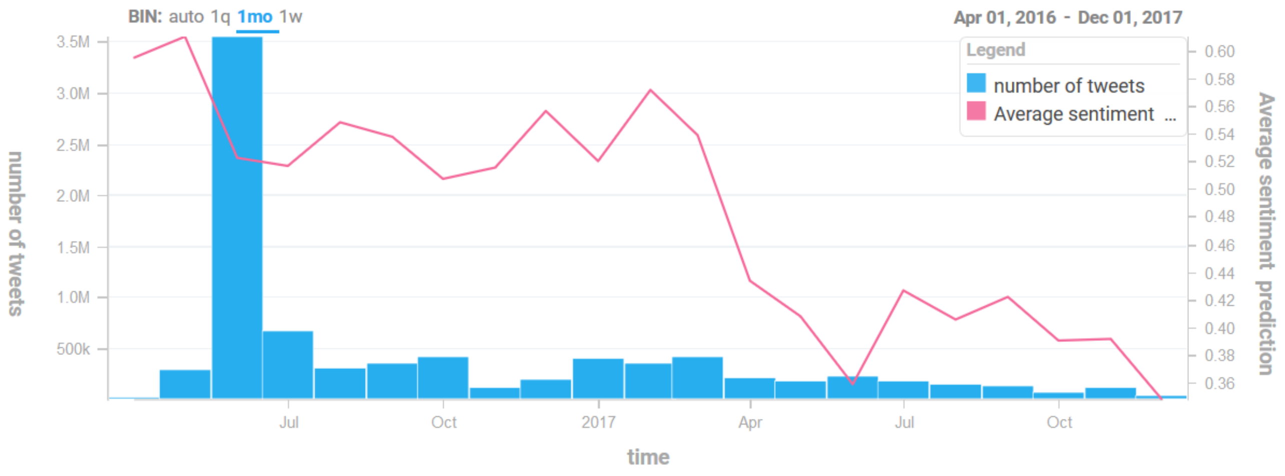


Figure 14: Number of tweets and sentiment average over time. Value of 1 and 0 corresponds to all the tweets being pro and against Brexit, respectively.

We chose to bin the time axis in months as the main peaks and general variation of the average sentiment are well visible but do not vary too much (there are too many peaks and creases to see a general evolution when we bin it into weeks). First of all, we see a global decline of Pro-Brexit sentiment over time (from 61% Pro-Brexit in May 2016 to only 31% Pro-Brexit in June 2017). We could say that in roughly one year, *the faith of people in Brexit has been halved.* While sentiment is relatively stable between June 2016 and March 2017, let us explain the outstanding points (by spotting the peak in the month-binned axis and then switching to week bins to refine the time interval and spot the matching event):

- **May 1st to May 7th 2016:** Highest global pro Brexit sentiment (0.64 sentiment score, i.e. tweets that are in average 64% pro Brexit) ever recorded over the considered period.
- **June 19th to June25nd, 2016:** This corresponds to the week of the Brexit referendum. In our analysis, during that week, the tweets are 51% pro Brexit (global sentiment score of 0.51), which actually reflects very well the vote: 51.9% of voters voted leave [5].
- **December 4th to December 10th, 2016:** Another predominance of a desire to remain in the EU (tweets are in average only 44% pro Brexit): this sentiment is also coherent with real-life events, as it corresponds to a Supreme Court Hearing (December 5th) to decide whether the Prime Minister has the right to invoke Article 50 unilaterally (in other words, whether the government can start triggering the Brexit process without the Parliament). The event provoked a lot of negative reactions from the media towards Brexit.
- **February 18th to February 25th, 2017:** There doesn't seem to be any decisive event during that period, but it does coincide with the time at which the government was fighting for amendments of the Brexit bill and could have triggered a revival of pro Brexit sentiment.
- **June 4th to June 10th, 2017:** The general sentiment reaches its second lowest score of 0.34 after the snap election requested by Theresa May on June 8th. It was a great failure for Conservatives, who now have 36.9% of the total seats (lost 13) and a success for Labour and now have 30% of the total seats (won 29). These election resuts express an overall negative view of Brexit as it was being proposed by the conservatives, and expresses the fact that more voters leaned towards the propositions from Labour (who are globally against Brexit). So our sentiment analysis results are coherent with this event.

- **Sentiment by hashtag & Sentiment evolution over time by hashtag** (Fig. 15): We plot those two graphs together to spot en eventual correlation between hashtags and sentiments. unsurprisingly, #ukip, #LeaveEU, #EURef, #voteLeave are labelled as pro Brexit in average, #Brexit as neutral and #remain and #bettertogether as anti-Brexit. However, #strongerin is supposed to be against Brexit but is labeled as pro in average. A more detailed analysis gives us the following (Fig 15):

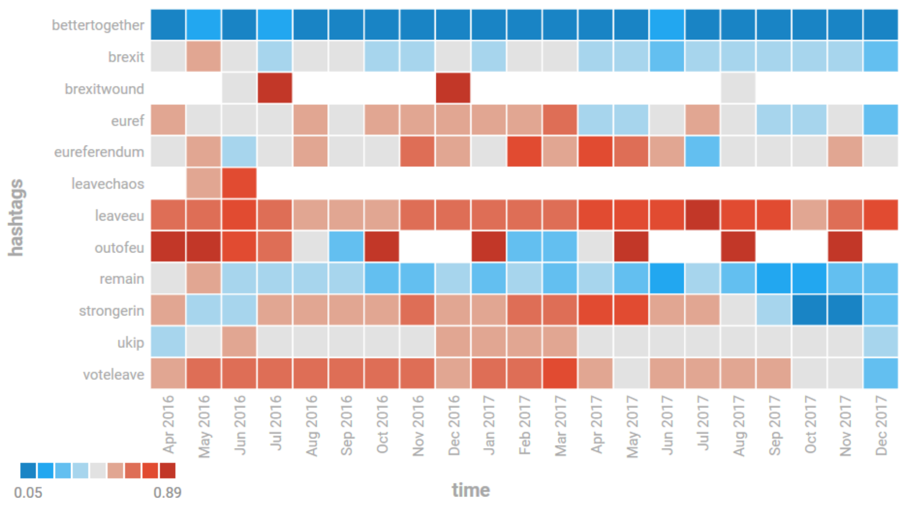


Figure 15: Sentiment evolution over time per hashtag

We notice that #strongerin is in fact mainly used with pro Brexit tweets until July 2017 (this correspond to the snap election failure for the Conservatives). There are several possible explanations. One possibility is that #strongerin was often associated with pro Brexit hashtags like #leaveEU for example, which augments its chances of being present on a pro Brexit tweet. Another one is that the hashtag changed its meaning along time and people started associating it to an against Brexit meaning.
Concerning the other hashtags, the small variations can be explained by the fact they will not necessarily be used in a pro Brexit tweet if they are pro Brexit (for example in a sentence like "I hate the #leaveEU movement").

Our Twitter data shows us that perception of Brexit globally deteriorates along time, with its peak of popularity being just at the time of the referendum. It then losing supporters brutally after the vote and the General Election in June 2017.

## 4 Conclusion & Future Works

The association of a sentiment classification model and a timeline representation gave us insights about the referendum and showed a progressive loss of faith in Brexit. However, we also showed that the sentiment is tightly linked to political events perceived as successes or failures for the independence of the UK, so a change in public opinion stays a possibility. Our final prediction on sentiment analysis over time gave surprisingly accurate results that both correlate with the occurring event and even poll results. Making (accurate) political predictions, based on social medial data is certainly nothing new as we've seen from 2016 Donald Trump's U.S presidential campaign [10], and many more. From the results obtained from this work, we can once more remind ourselves how powerful social media & sentiment analysis can be, and how we, the engineers, ought not forget the potential negative consequences an irresponsible misuse of big data may bring.

The world of social media analytics is a vast one with many domains to explore. In our particular problem statement, the network analysis section remains incomplete due to lack of data. Implementing the researched metrics may give us a much better insight into the real Twitter influencers involved. We may also be interested in judging the power of Twitter as a media of influence by measuring the opinions of followers of an influencer before and after the influencer's tweets. In the area of sentiment analysis, we may attempt to obtain a higher classification accuracy by including features other than word embeddings itself, such as users who retweeted the tweet or location of tweet origin. While these are not direct indicators of sentiment, they may tell us something about the prior probability of the tweet being pro or against an opinion. Apart from this, we could also try various other labeling strategies such as topic extraction with basic sentiment detection to label or even directly classify the data. For example, a topic such as "Leave the EU" can be given a pro-Brexit label. Hence any tweet with general a positive sentiment and a pro-Brexit label may be classified as a pro-Brexit tweet. Latent Dirichlet Allocation for Topic Modeling may be explored as an approach for the above. However, an exhaustive list of possibilities for future work in this area is beyond the scope of this report.

## References

[1] Amazon mechanical turk. https://docs.aws.amazon.com/AWSMechTurk/latest/RequesterUI/OverviewofMturk.html. Accessed: 2018-07-13.

[2] Bloomberg timeline. https://www.bloomberg.com/news/features/2017-03-20/brexit-timeline-from-eu-referendum-to-theresa-may-and-article-50. Accessed: 2018-07-13.

[3] Brexit gold standard data. https://bitbucket.org/ssix-project/brexit-gold-standard. Accessed: 2018-07-13.

[4] cnet. https://www.cnet.com/news/russian-accounts-tried-to-influence-brexit-vote-says-twitter/. Accessed: 2018-07-13.

[5] Eu referendum results. https://www.bbc.com/news/politics/eu_referendum/results. Accessed: 2018-07-13.

[6] Finnish internet parsebank word embedding demo. http://bionlp-www.utu.fi/wv_demo/. Accessed: 2018-07-13.

[7] Glove: Global vectors for word representation. https://code.google.com/archive/p/word2vec/. Accessed: 2018-07-13.

[8] Jupyterhub documentation. http://jupyterhub.readthedocs.io/en/latest/index.html. Accessed: 2018-07-13.

[9] Sentiment analysis on twitter using word2vec and keras. https://ahmedbesbes.com/sentiment-analysis-on-twitter-using-word2vec-and-keras.html. Accessed: 2018-07-13.

[10] Social media analysis predicted trump win. https://www.itweb.co.za/content/2j5alrvQA19vpYQk. Accessed: 2018-07-13.

[11] Textblob: Generic library for natural language processing. https://textblob.readthedocs.io/en/dev/#. Accessed: 2018-07-13.

[12] Word2vec google code archive. https://code.google.com/archive/p/word2vec/. Accessed: 2018-07-13.

[13] Manuela Hürlimann, Brian Davis, Keith Cortis, André Freitas, Siegfried Handschuh, and Sergio Fernández. A twitter sentiment gold standard for the brexit referendum. In *Proceedings of the 12th International Conference on Semantic Systems*, pages 193–196. ACM, 2016.

[14] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Machine Learning: ECML-98*, pages 137–142, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[15] Veronika LAIPPALA and Filip GINTER. Syntactic n-gram collection from a large-scale corpus of internet finnish. In *Human Language Technologies-The Baltic Perspective: Proceedings of the Sixth International Conference Baltic HLT 2014*, volume 268, page 184. IOS Press, 2014.

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[17] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

# 5 Author Contribution

## 5.1 Project Contribution

– **A. Duplaa**:
Time Visualization pipeline, Word2Vec model (Training and Testing),Tokenization pipeline, Automated Labeling (Hashtags), Data ETL for models, Sentiment Analysis pipeline,Environment Setup, Environment monitoring/maintenance,Code Refactoring

– **A. Muralidharan**:
Network Analysis (Data representation, Data collection, Metrics for influence measure), Brexit Gold Standard classification using TextBlob, Classification using learning algorithms (SVM, Logistic Regression, Neural Networks), Hyperparameter tuning, Metrics for measuring classification scores

– **J. Yang**:
Environment Setup, Network Analysis Research (Tools), Sentiment Analysis (Application), Word2Vec (Training and Applications), Brexit Gold Standard Dataset (pre-processing), Automated Labeling (Influencers), Structuring of the final report

– **M. Delacourt**:
Methodology for timeline visualization, Methodology for sentiment analysis (Word2Vec), Visualization dashboards, Insight discovery from timeline & sentiment analysis results

## 5.2 Report Writing Contribution

**Abstract**: J.Y

1. **Introduction**
   - **1.1. Project Description**: M.D.
   - **1.2. Project Scope**
     - **1.2.1. Timeline Analysis and Visualization**: M.D.
     - **1.2.2. Sentiment Analysis**: M.D.
     - **1.2.3. Network Analysis**: A.M

2. **Methodology**
   - **2.1. Working Methodology**
     - **2.1.1. Agile Software Development**: J.Y
     - **2.1.2. Communication Tools**: J.Y
   - **2.2. Tools**
     - **2.2.1. LRZ Cloud**: A.D
     - **2.2.2. Hadoop File System**: A.D.
     - **2.2.3. Apache Spark 2.3.0**: A.D.
     - **2.2.4. JupyterHub**: A.D.
     - **2.2.5. GitLab**: J.Y
     - **2.2.6. Docker**: A.D
     - **2.2.7. Kubernetes**: A.D.
     - **2.2.8. Keras**: A.M
     - **2.2.9. Databases**
       - **2.2.9.1. PostgreSQL**: A.D.
       - **2.2.9.2. MapD**: A.D.
   - **2.3. Infrastructure Architecture**: A.D.
   - **2.4. Data Description**: A.D.
     - **2.4.1. Data Discovery**: A.D
   - **2.5. Timeline Visualization**
     - **2.5.1. Data Processing**: M.D.
     - **2.5.2. Data Visualization**: M.D.
   - **2.6. Sentiment Analysis**
     - **2.6.1. Our Approach: A Double Neural Network Pipeline**: M.D.
     - **2.6.2. Word2Vec Continuous Bag of Words (CBoW)**
       - **2.6.2.1. Introduction to Word Encoding**: M.D.
       - **2.6.2.2. The General Idea Behind Embedding Learning**: M.D.
       - **2.6.2.3. Continuous Bag of Words**: M.D.
       - **2.6.2.4. Pre-trained Word2Vec Models**: J.Y

- **2.6.3. Vector Representation Neural Network** A.D
  - **2.6.3.1. Tokenizing and Data Processing**: A.D
  - **2.6.3.2. Vectorization of Tokens**: A.D
- **2.6.4. Classification into Pro/Against Brexit**: A.M
  - **2.6.4.1. Input to Learning Algorithm**: A.M
  - **2.6.4.2. Learning Algorithms**: A.M
- **2.6.5. Data Labeling**: J.Y
  - **2.6.5.1. Manual Labeling**: J.Y
  - **2.6.5.2. Automated Labeling Based on Influencers**: J.Y
  - **2.6.5.3. Automated Labeling Based on Hashtags**: A.D
- **2.7. Network Analysis**
  - **2.7.1. Model Representation**: A.M
  - **2.7.2. Metrics**: A.M
  - **2.7.3. Tools**: J.Y
  - **2.7.4. Challenges**: A.M
  - **2.7.5. Steps Taken**: A.M

**3. Results & Discussion**
- **3.1. Timeline Visualization**: M.D.
- **3.2. Sentiment Analysis**
  - **3.2.1. Word2Vec**: A.D
  - **3.2.2. Brexit Gold Standard Dataset**: J.Y, A.M
  - **3.2.3. Automated Labeling Based on Influencers**: J.Y
  - **3.2.4. Automated Labeling via Hashtags**: A.M
- **3.3. Sentiment Analysis over Time**: M.D

**4. Conclusion & Future Works**: A.M, M.D, J.Y

# 6 Appendix

## 6.1 Appendix 1: Hashtag classification

| Hashtag | Sentiment | Comment |
|---|---|---|
| #TakeControl | Pro | Many tweets with this hashtag concerns health/fitness/inspiration |
| #VoteLeave | Pro | |
| #Brexit | Neutral | General info regarding Brexit (future of UK, conferences, etc) |
| #BrexitWound | Against | |
| #ProjectHope | Against | Many tweets with this hashtag concern a charity |
| #LeaveEU | Pro | |
| #EUreferendum | Neutral | |
| #OutofEU | Pro | |
| #EUref | Neutral | |
| #Remain | Against | |
| #UKIP | Pro & author | UK Independence Party |
| #StrongerIn | Against | |
| #BetterTogether | Against | |
| #LeaveChaos | Against | |
| #VoteLeaveGetChaos | Against | |

Table 13: Full list of Hashtags considered for automated labeling.

## 6.2 Appendix 2: Description of schema fields

| Fields | Data Type | Description |
|---|---|---|
| user_id | string | User id |
| retweet_count | long | Number of retweets |
| tweet_language | string | Language of the tweet |
| tweet_countrycode | string | Country code |
| tweet_city | string | City where the tweet is posted |
| tweet_text | string | Tweet text |
| tweet_favoritecount | long | Number of times tweet was favorited |
| tweet_favorited | boolean | Whether tweet was favorited or not |
| user_favouritescount | long | Whether user was favorited or not |
| friends_count | long | Number of friends |
| user_language | string | User's language |
| user_location | string | User's location |
| user_screenname | string | Screen name of mentioned user |
| user_name | string | Name of the user |
| tweet_latitude | double | Latitude where tweet is posted |
| tweet_longitude | double | Longitude where tweet is posted |
| tweet_id | string | id of the tweet |
| hashtags | string | Hashtags |

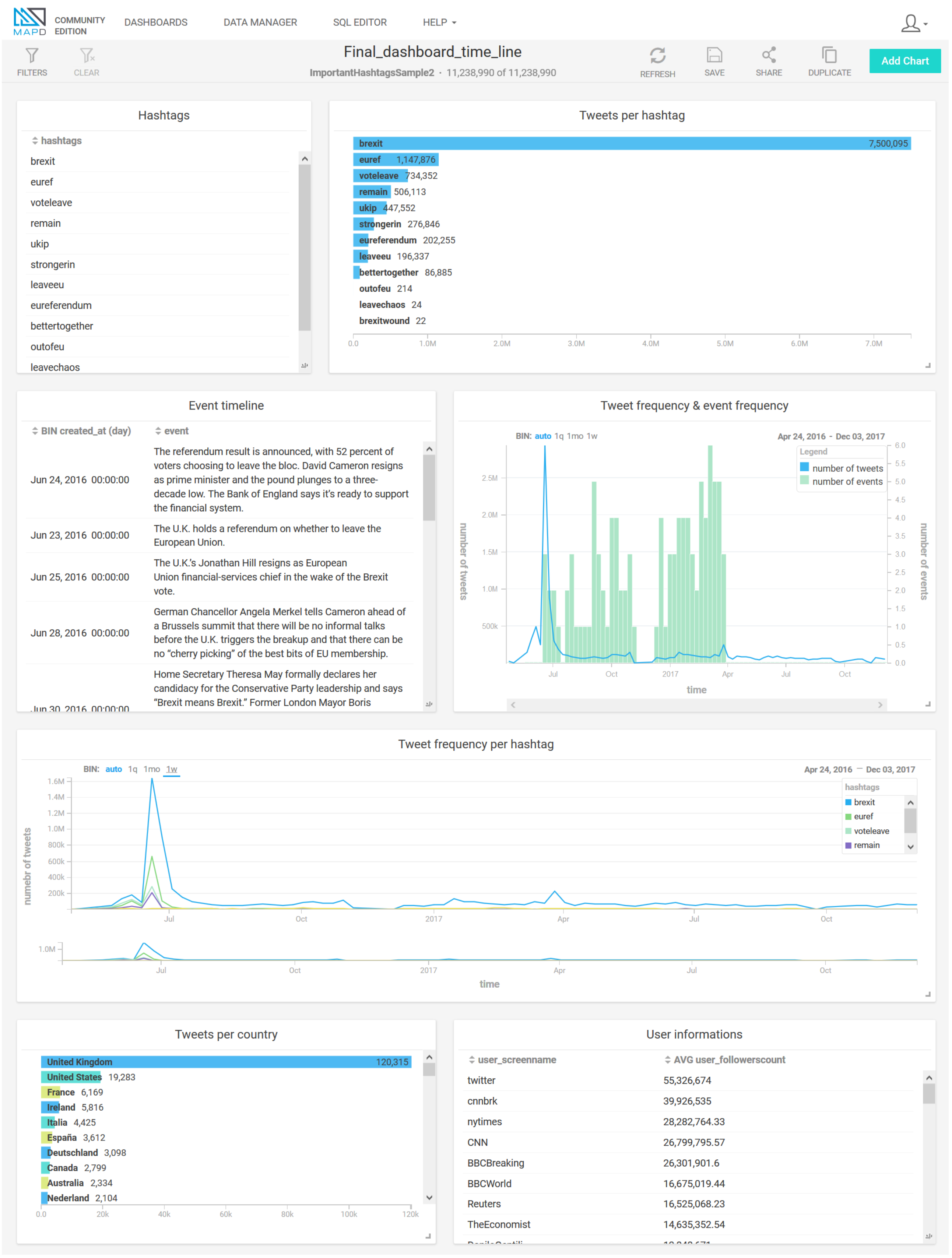
Table 14: List of selected schema fields and their descriptions

## 6.3 Appendix 3: List of influencers considered for automated labeling

| Account | Occupation | Pro/Against Brexit |
|---|---|---|
| OwenJones84 | Author & Guardian columnist | Against |
| MhairiBlack | Scottish National Party MP | Against |
| tom_watson | Labour Party deputy leader | Against |
| GuidoFawkes | Right-wing political blogger | Pro |
| stellacreasy | Labour MP | Pro |
| JohnRentoul | Independent chief political commentator & author | Against |
| Kevin_Maguire | Daily Mirror associate editor & New Statesman columnist | Against |
| MrHarryCole | The Sun Westminster correspondent | Pro |
| sunny_hundal | Author & journalist | Against |
| DouglasCarswell | UK Independence Party MP | Pro |
| ShippersUnbound | Sunday Times Political Editor & author | Pro |
| montie | The Times columnist | Against |
| LouiseMensch | Author & former Conservative MP | Pro |
| DavidLammy | Labour MP | Against |
| RuthDavidsonMSP | Scottish Conservative Party leader | Against |
| georgeeaton | New Statesman Political Editor | Against |
| SamCoatesTimes | The Times Deputy Political Editor | Against |
| jessphillips | Labour MP | Against |
| AbiWilks | Freelance political journalist | Against |
| stephenkb | New Statesman special correspondent | Against |
| jreedmp | Labour MP | Against |
| NadineDorriesMP | Conservative MP | Against |
| PatrickStrud | BuzzFeed UK LGBT editor. | Against |
| JolyonMaugham | Tax expert & political and legal commentator. | Against |
| PaulbernalUK | Law lecturer, political commentator, and author. | Against |
| robfordmancs | Politics professor & author | Against |
| Tom_Slater_ | Spiked Deputy Editor | Pro |
| asabenn | The Telegraph comment editor | Pro |

Table 15: Full list of influencers considered for automated labeling.

## 6.4 Appendix 4: An overview of the MapD dashboards



Final_dashboard_time_line

ImportantHashtagsSample2 · 11,238,990 of 11,238,990

**Hashtags**

| hashtags |
|---|
| brexit |
| euref |
| voteleave |
| remain |
| ukip |
| strongerin |
| leaveeu |
| eureferendum |
| bettertogether |
| outofeu |
| leavechaos |

**Tweets per hashtag**

| | |
|---|---|
| brexit | 7,500,095 |
| euref | 1,147,876 |
| voteleave | 734,352 |
| remain | 506,113 |
| ukip | 447,552 |
| strongerin | 276,846 |
| eureferendum | 202,255 |
| leaveeu | 196,337 |
| bettertogether | 86,885 |
| outofeu | 214 |
| leavechaos | 24 |
| brexitwound | 22 |

**Event timeline**

| BIN created_at (day) | event |
|---|---|
| Jun 24, 2016 00:00:00 | The referendum result is announced, with 52 percent of voters choosing to leave the bloc. David Cameron resigns as prime minister and the pound plunges to a three-decade low. The Bank of England says it's ready to support the financial system. |
| Jun 23, 2016 00:00:00 | The U.K. holds a referendum on whether to leave the European Union. |
| Jun 25, 2016 00:00:00 | The U.K.'s Jonathan Hill resigns as European Union financial-services chief in the wake of the Brexit vote. |
| Jun 28, 2016 00:00:00 | German Chancellor Angela Merkel tells Cameron ahead of a Brussels summit that there will be no informal talks before the U.K. triggers the breakup and that there can be no "cherry picking" of the best bits of EU membership. |
| Jun 30, 2016 00:00:00 | Home Secretary Theresa May formally declares her candidacy for the Conservative Party leadership and says "Brexit means Brexit." Former London Mayor Boris |

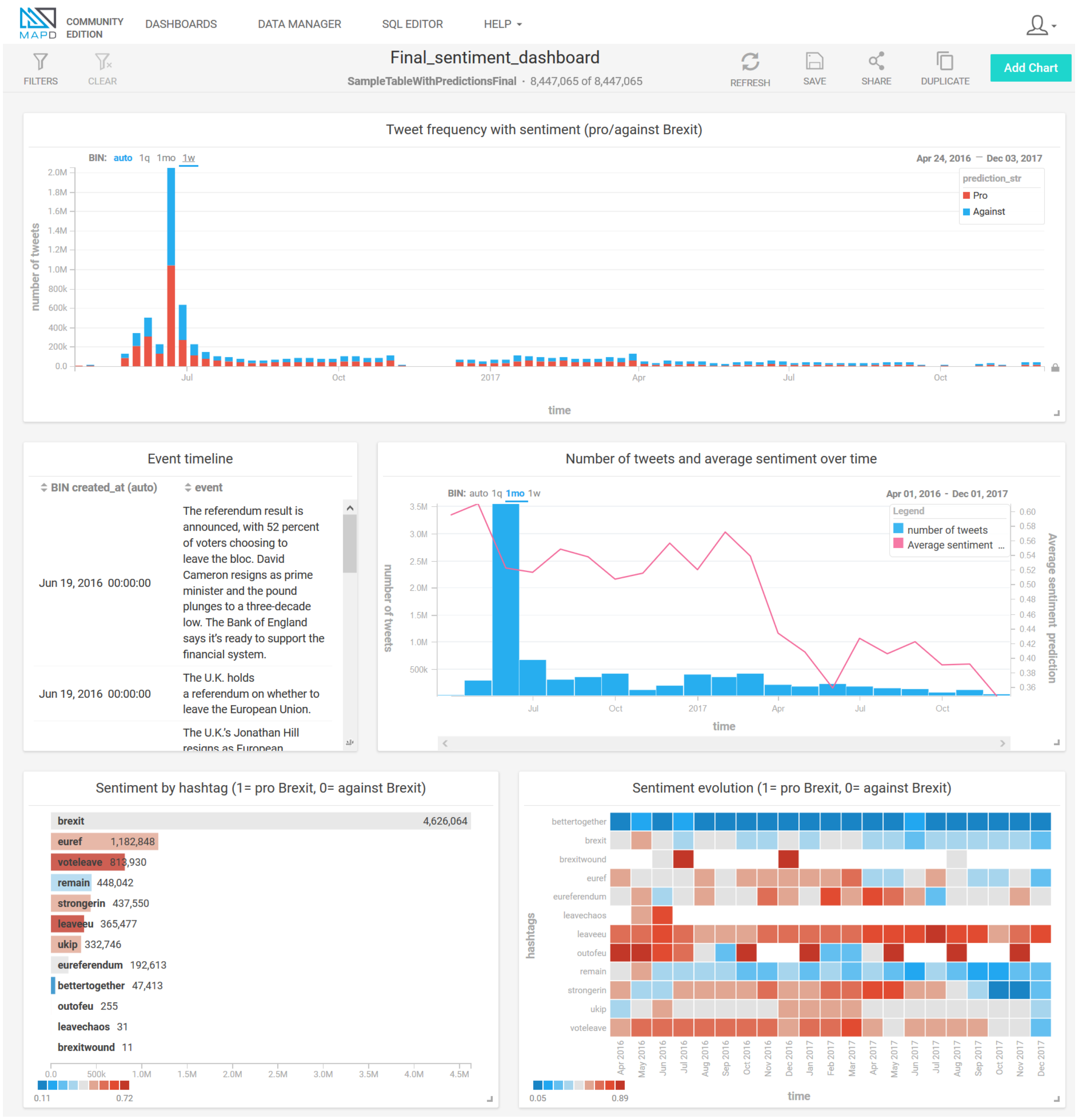
**Tweets per country**

| | |
|---|---|
| United Kingdom | 120,315 |
| United States | 19,283 |
| France | 6,169 |
| Ireland | 5,816 |
| Italia | 4,425 |
| España | 3,612 |
| Deutschland | 3,098 |
| Canada | 2,799 |
| Australia | 2,334 |
| Nederland | 2,104 |

**User informations**

| user_screenname | AVG user_followerscount |
|---|---|
| twitter | 55,326,674 |
| cnnbrk | 39,926,535 |
| nytimes | 28,282,764.33 |
| CNN | 26,799,795.57 |
| BBCBreaking | 26,301,901.6 |
| BBCWorld | 16,675,019.44 |
| Reuters | 16,525,068.23 |
| TheEconomist | 14,635,352.54 |

Figure 16: Screenshot of the time line analysis dashboard


MAPD COMMUNITY EDITION
DASHBOARDS
DATA MANAGER
SQL EDITOR
HELP
Final_sentiment_dashboard
SampleTableWithPredictionsFinal · 8,447,065 of 8,447,065
FILTERS
CLEAR
REFRESH
SAVE
SHARE
DUPLICATE
Add Chart
Tweet frequency with sentiment (pro/against Brexit)
BIN: auto 1q 1mo 1w
Apr 24, 2016 — Dec 03, 2017
prediction_str
Pro
Against
number of tweets
time
Event timeline
Number of tweets and average sentiment over time
BIN: auto 1q 1mo 1w
Apr 01, 2016 - Dec 01, 2017
Legend
number of tweets
Average sentiment ...
Average sentiment_prediction
Sentiment by hashtag (1= pro Brexit, 0= against Brexit)
0.11
0.72
Sentiment evolution (1= pro Brexit, 0= against Brexit)
hashtags
time
0.05
0.89

| BIN created_at (auto) | event |
|---|---|
| Jun 19, 2016 00:00:00 | The referendum result is announced, with 52 percent of voters choosing to leave the bloc. David Cameron resigns as prime minister and the pound plunges to a three-decade low. The Bank of England says it's ready to support the financial system. |
| Jun 19, 2016 00:00:00 | The U.K. holds a referendum on whether to leave the European Union. |
| | The U.K.'s Jonathan Hill resigns as European |

| | |
|---|---|
| brexit | 4,626,064 |
| euref | 1,182,848 |
| voteleave | 813,930 |
| remain | 448,042 |
| strongerin | 437,550 |
| leaveeu | 365,477 |
| ukip | 332,746 |
| eureferendum | 192,613 |
| bettertogether | 47,413 |
| outofeu | 255 |
| leavechaos | 31 |
| brexitwound | 11 |

Figure 17: Screenshot of the sentiment over time dashboard