FACULTEIT ECONOMIE
EN BEDRIJFSKUNDE

Predicting which tweets will be retweeted most.

Word count: 9468

Jolan Bourdeaud'hui

Student number: 01305763

Promotor/ Supervisor: Prof. dr. Dirk Van den Poel

Co-promotor/ Co-supervisor: Matthias Bogaert

Master's Dissertation submitted to obtain the degree of:

Master of Science in Business Engineering

Academic year: 2017 – 2018

UNIVERSITEIT
GENT

CONFIDENTIALITY AGREEMENT

PERMISSION

I declare that the content of this Master's Dissertation may be consulted and/or reproduced, provided that the source is referenced.

Name student
:……………………………………………………………………………………………

Signature

“To be retweeted, or not to be retweeted, that’s the question:”

A prediction of which tweets will be retweeted .

Inhoud

Preface

This master dissertation has been made possible by my promoter Prof. Dirk Van den Poel. First, I would like to thank twitter and their environment for letting tweets being easily extracted from their site. Special thanks go to my family and friends who have supported me during this long agony. Finally special thanks to my girlfriend Charlotte who supported me during this long journey. Social media is getting more and more attention, it is interesting to discover what an impact has on your social status and your social network. We all want to be retweeted. Twitter and the corresponding retweet behavior is an excellent example how you can spread your message. The major challenge of my research was to come up with extra parameters that influence the number of retweets. Another challenge was to find underlying reasons why certain parameters were performing significant worse/better than predicted. Our final conclusion is that the user specific information is superior to the specific text information. The size of your network is important; without followers you cannot be retweeted.

Abstract

In a time where the amount of likes and retweets is more important than ever, we want to investigate what the ideal recipe is for a tweet to be recognized and retweeted. The purpose of this paper is to predict which parameters of a tweet are most important to be retweeted and what the ideal recipe is for a tweet. We use different models such as Random Forest, Logistic Regression and Decision Tree to increase the AUC of the prediction model. Further, we want to discover if the content of the text or the user information is the most important. Therefore, we do analyses about user specific data (followers, friends, number of tweets,…), but also on text specific data (hashtags, text, URL, …). The global trend is that user specific information is more decisive than text content specific information to be retweeted. A few important new introduced parameters came to light such as the number of public lists that a user is a member of. To our knowledge, this study is the first that evaluates the parameters of retweeted tweets and the parameters of most retweeted tweets.

Dutch summary

Omdat likes en retweets altijd maar belangrijker worden, willen we onderzoeken wat het ideale recept is om een tweet te laten retweeten, Het doel van deze paper is om te voorspellen welke parameters van een tweet de meest belangrijke zijn om het retweeten te triggeren. We gebruiken verschillende modellen zoal Random Forest, Logical Regression en Decision Tree om de AUC van de voorspelling van het model te verbeteren. Verder willen we onderzoeken of de content van de tekst of eerder de gebruikersinformatie de belangrijkste parameters zijn. Daarom doen we analyse van gebruiker specifieke data (volgers, vrienden, aantal tweets, ...), maar ook tekst specifieke data (hashtags, tekst, URLs, ...). De globale trend is dat gebruiker specifieke informatie meer bepalend is voor retweets dan tekstinhoud. Een paar belangrijke nieuwe parameters kwamen aan het licht, zoals het aantal lijsten waar de gebruiker deel van uitmaakt. Voor zover wij konden nagaan, is deze studie de eerste die de parameters van de tweets die worden geretweet en de tweets die meest worden geretweet evalueert.

1. Introduction

1.1. An introduction to social media

Nowadays, online social networks such as Facebook, Instagram and Twitter, are very important and powerful communication tools. In our 21st society, people want more likes, more retweets and more followers. In this research paper we focus on the social network “Twitter”. Twitter is a very attractive tool for people who are looking to enlarge their network and influence. With more than 300 million active users, Twitter is an increasingly significant media context of empirical examination (Twitter, 2018). Twitter is unique among social media platforms as it is an intermediary for linking anonymous users to each other as well as a space to break and contextualize news (Hermida, 2010).

Social media platforms allow rapid information diffusion and serve as a source of information to many of the users. Therefore, for all kinds of individuals and organizations using Twitter as a communication channel, it is important to know what they can do to diffuse their tweets. In a time where status in the form of likes and retweets is more important than ever, individuals want to know what the ideal recipe is to be recognized or retweeted. In addition, for marketing investors and organizations it is important to know which messages or tweets are being spread and which not.

The popularity of tweets has resulted in information propagation becoming a prominent fundamental function of online social networks. From the perspective of information propagation, retweeting is viewed as an atomic behavior (Boyd, Golder & Lotan 2010). Specifically, retweeting action diffuses information carried in the original message. If a tweet gets a lot of retweets, the potential audience of a tweet enlarges significantly. Prior information propagation models treat the retweeting behavior as having constant retweet probability, or as following a certain probability distribution. Previous research demonstrated that the composition of Twitter users indicated a highly skewed composition, where the majority of users (90%) only send out a very few tweets and the minority of users (10%) are highly active (Bruns & Stieglitz, 2012). Moreover, statistics show that 1% of Twitter users produce 20-50% of its content and control 25% of its information diffusion (Tang et al., 2015).

A lot of Twitter research focus on which tweets are retweeted the most. Most of these studies are looking for features of a tweet that give the highest probability of being retweeted. First, some papers show a deeper insight of which tweets are retweeted the most. For example Hong, Dan and Davison (2011) examined which factors influence the spreading of Twitter tweets via a classification algorithm. These factors can be the content of the message, the hashtags, the metadata (i.e. has the message been retweeted before?) or the structural properties of the user. These findings are in line with the study conducted by Suh, Hong, Pirolli and Chi (2011) who built a predictive retweet model. They found interesting content features in tweets that result in higher probability of being retweeted. Their most remarkable conclusion was that the number of past tweets does not predict retweet ability of a user's tweet.

To know how many times a tweet will be retweeted, it is important to investigate if the tweet will be retweeted in the first place. Earlier research don't focus on both questions. Therefore, in this study we will first try to solve the binary classification problem before counting how many times a tweet will be retweeted. This focus doesn't exist in earlier Twitter research. In addition, the retweeting research is stagnating over the last few years. Nevertheless, a lot has changed during the past few years and different extra tweet features are coming up like symbols, included media, extended URL's and user lists. The behavior of twitterers also has changed as social media is more available for everyone and not only for professionals. Therefore, in this paper we want to predict which state-of-the-art tweets are the ones most likely to be retweeted. To answer this question, we first need to investigate if a tweet gets retweeted or not. In a second stage, we focus on the expected retweet count.

2. Theoretical framework

2.1. Previous twitter research

Twitter tweets provide information over the users through the diffusion of retweets. Hence, being able to predict the retweet count of a given tweet is important for understanding and controlling information diffusion on Twitter. As the length of a tweet is limited to 280 characters, extracting relevant features to predict the retweet count is a challenging task. Different authors report about the sentiment analysis and their influence on the retweetability of tweets (Rosenthal, 2017; Stieglitz & Dang-Xuan, 2012). More precisely, tweets convey related information about their author's sentiment. Tweets can be classified through sentiment analyses in positive, negative or neutral tweets. For example, Stieglitz and Dang-Xuan (2012) examined whether sentiment occurring in politically relevant tweets has an effect on their retweetability. The result was a positive relationship between the quantity of words indicating affective dimensions, including positive and negative emotions in a tweet and its retweet rate. Rosenthal (2017) declares that the sentiment analysis of a tweet could be classified on a two-point or a five-point ordinal scale.

To analyze the retweetability of a tweet, there are two different kinds of features: *user specific features* and *text specific features* (Kupavskii, 2013). The category of *user specific features* includes all the variables in the user's profile (e.g. the number of followers, the number of tweets or the number of favorites). The category of *text specific features* includes all the variables derived from the actual text of the tweet (e.g. the number of hashtags, positive/negative sentiment of the tweet and number of URLs in the tweet) (Kupavskii, 2013).

In Table 1 you can find an overview of the different features used in our research in comparison to previous researches. Paper 1, the paper of Hong (2011) does not disclose the features used, so some features might be missing in the previous research binary column.

Features	**Previous research binary (1)**	**Previous research regression (2)**	**Our research (binary + regression)**	**Our research (binary + regression final model**
Number of followers	x	x	x	x
Number of following	x	x	x	x
Number of times the user was listed	x	x	x	x
Favorites user	x	x	x	x
Favorites tweet	x	x	x	
PageRank		x		
URLs	x	x	x	x
Positive and negative smileys,		x		
Number of posts	x	x	x	x
The date of account creation		x		
Number of mentions	x	x	x	x
Number of tweet words		x	x	
Hashtags	x	x	x	x
Exclamation and question marks		x		
Number of symbols			x	x
Number of media			x	x
Positive words used		~	x	x
Negative words used		~	x	x

Table 1: Overview of different features

(1) Predicting Popular Messages in Twitter (Hong, 2011)

(2) Predicting the Audience Size of a Tweet (Kupavskii, 2013)

2.2 Program description

Research shows that JSONs are mostly downloaded in a python script and that the data cleaning and further analytics were done in *databricks PySpark*. Different authors such as Lekha and Sujala (2015) declared the use of *Spark* with twitter data. These authors examined the trendy Big Data processing engine which have gained tremendous interests over the past few years and offers faster solutions compared to a traditional Java-based programming framework like Hadoop. Lekha and Sujala (2015) experienced that *Spark* can be effectively utilized in finding patterns for companies who are advertising their job vacancies through tweets. In their study the numerous job advertisements were classified into various job categories, using *Spark*. *Spark* has many advantages:

1. Spark works with *"Resilient Distributed Datasets,"* or RDDs. Using this simple extension, Spark can capture a wide range of processing workloads that previously needed separate engines, including SQL, streaming, machine learning, and graph processing (Zaharia et al. 2016)
2. *Dataframes* are a common abstraction for tabular data in R and Python, with programmatic methods for filtering, computing new columns, and aggregation. In Spark, these operations map down to the Spark SQL engine and receive all its optimizations. A data frame is a set of records with a known schema, essentially equivalent to a database table that supports operations like filtering and aggregation using a restricted "expression" API. Unlike working in the SQL language, however, *dataframe* operations are invoked as function calls in a more general programming language (such as Python and R), allowing developers to easily structure their program using abstractions in the host language (such as functions and classes) (Zaharia et al. 2016)
3. In the literature we see that, the big advantage of Spark is that it is a *cluster computing system*. In comparison with older programs, Spark is remarkable faster at both running as writing programs. Shoro and Soomro (2015) quoted: "It is really a big challenge to analyze the bulk amount of twits to get relevance and different patterns of information on timely manner." Their paper explored the concept of Big Data Analysis and recognized interesting information from tweets, using one of industries emerging tool, known as Spark by Apache.

2.3. Algorithm and model description

During the modeling phase, we will make use of the following algorithms: *logistic regression, decision tree learning* and *random forest*. First, *Logistic regression* is a statistical model. Generally, *logistic regression* is well suited for describing and testing hypotheses about relationships between a categorical outcome variable and one or more categorical or continuous predictor variables. (Peng, 2002). The two possible dependent variable values are often labelled as "0" and "1". In this paper, these values will represent outcomes as respectively "not retweeted" and "retweeted".

Second, classification and regression trees are machine-learning methods for constructing prediction models from data. The models are obtained by recursively partitioning the data space and fitting a simple prediction model within each partition. As a result, the partitioning can be represented graphically as a decision tree. Classification trees are designed for dependent variables that take a finite number of unordered values, with prediction error measured in terms of misclassification cost. Regression trees are for dependent variables that take continuous or ordered discrete values, with prediction error typically measured by the squared difference between the observed and predicted values. (Wiley et al. 2011)

Third, *Random forests* cope with the limited robustness and sub-optimal performance (Dudoit et al., 2002) of decision trees by building an ensemble of trees (e.g., a thousand trees) (Breiman, 2001). Each individual tree is grown on a bootstrap sample using Binary Recursive Partitioning (BRP) (Breiman et al., 1984). In random forests, the BRP algorithm starts by randomly selecting a subset of candidate variables (Breiman, 2001) and evaluating all possible splits of all candidate variables. A binary partitioning of the data is then created using the best split. The algorithm proceeds recursively by, within each parent partition, again randomly selecting a subset of variables, evaluating all possible splits, and creating two child partitions based on the best split. In sum, random forests uses random feature selection at each node of each tree, and each tree is built on a bootstrap sample.

Last but not least, we are aware that statistics can be used in a misleading way. We refer to John D. Cook (PhD and consultant in applied mathematics, statistics and technical computing) who states that "With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.", therefore we opt to try not to overfit our data.

2.4. Privacy policy

One of the reasons General Data Protection Regulation (GDPR) gets a lot of attention is the recent scandal around user data leaks at Facebook. GDPR came into effect on May 25th, 2018. The GDPR builds upon and modernizes existing EU Data Protection and Privacy rules and will replace them with one single set of rules that govern how personal information is collected and processed. It is a law that helps to protect and defend user privacy and informs the user of what is being done with their personal details. For future research there will become more restrictions about user streaming. It is not completely clear what will change on the Twitter API platform.

3. Research questions

RO1: Which tweets will be retweeted?

Before we want to know, how many times a tweet will be retweeted, we need to investigate if a tweet gets retweeted or not. To reach this goal, we make a difference between a retweeted and a non-retweeted tweet and look at the differences between both.

- Hypothesis 1: We predict that user related features would be more important than text related features.
- Hypothesis 2: We predict that the most determinative parameter of the user is the number of followers.
- Hypothesis 3: We predict that messages that score high on sentiment, positive or negative, will be retweeted more.

RO2: Which tweets will be retweeted most?

After figuring out what differentiate a retweeted tweet from a not retweeted tweet, we can formulate hypotheses 1 as in our first research question.

- Hypothesis 1: The importance of the individual features for the binary classification model are similar but not the same as the importance of the features for the regression model.

4. Methodology

In the following section we are deciding which features we are selecting for which problem. We used three different algorithms Logistic Regression, Decision Tree and Random forest to compare the impact of features within the different algorithms.

4.1. Project overview

The project starts from the assumption that certain determinants could influence the chance of being retweeted, either positive or negative. To answer our research questions we follow the different steps of the Cross-Industry Standard Process for Data Mining (CRISP-DM). The CRISP-DM is a data mining process model that describes commonly used approaches that data mining experts use to tackle problems (Chapman, 2000). Figure 1 illustrates the six phases of CRISP-DM.

1. **Business Understanding:** Finding the features that influence retweetability as described in the problem statement and theoretical framework.
2. **Data Understanding:** Downloading tweets in the form of JSON through the twitter API to the local hard disk. And examining this JSON structure with a JSON viewer is elaborated in section 4.2.
3. **Data Preparation:** JSON structure manipulation in Python and Databricks as described in chapter 4.3 (Data Understanding and Data Preparation).
4. **Modeling:** Selecting and building the most appropriate models in Databricks is elaborated in chapter 4.4 (Data Analysis).
5. **Evaluation:** Using performance measurements as AUC to compare results can be found in addendum 2.
6. **Deployment:** Deploying and summarize our final findings. Method and strategy can be found in addendum 2.

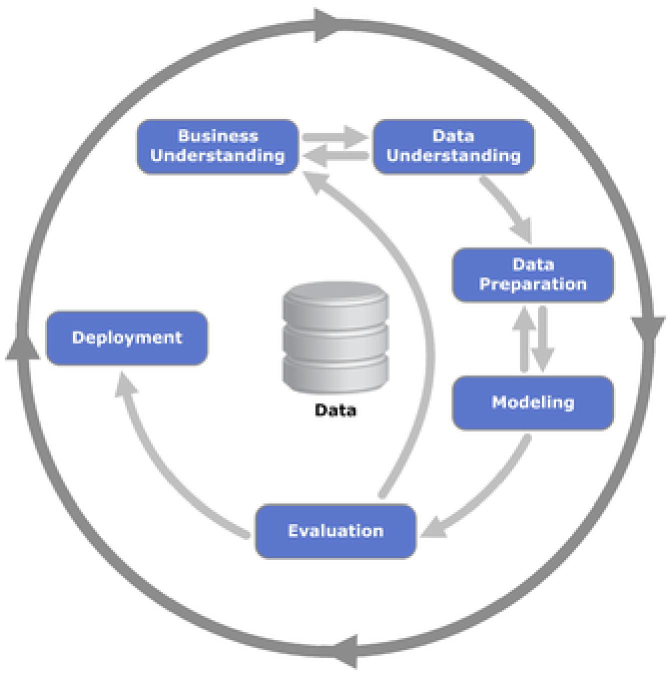


Figure 1: CRISP-DM

4.2. Data Understanding

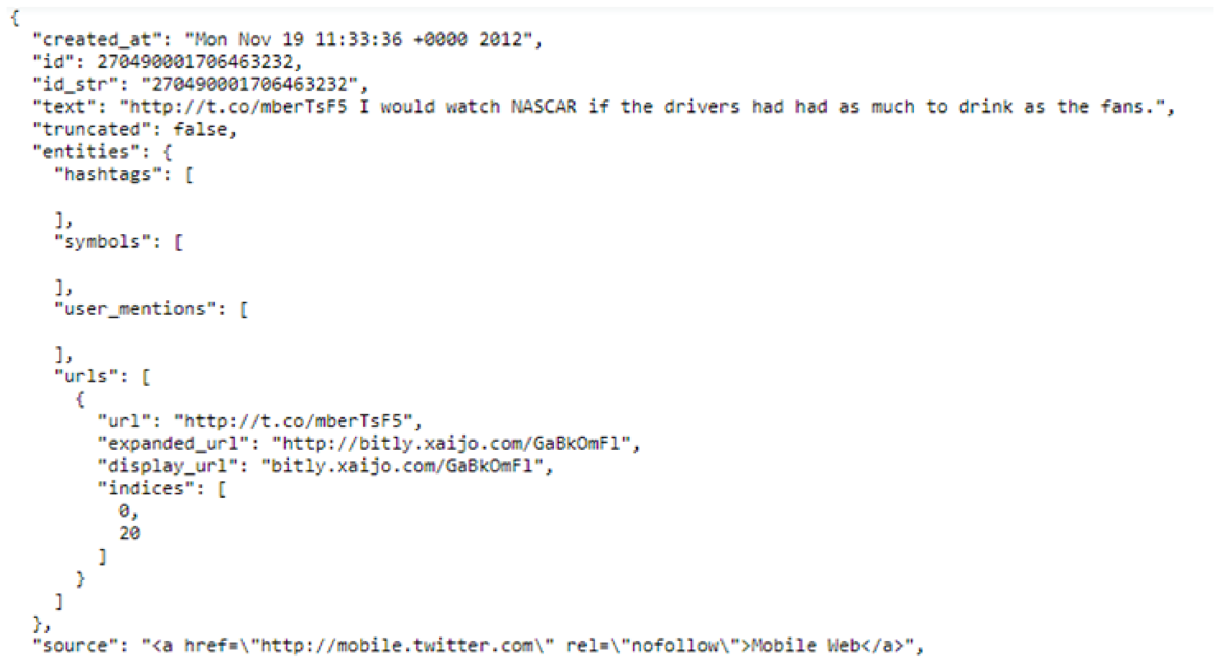
We extracted our social media data from the website Twitter. Earlier research shows that social media data is often more subjective and personal. To answer our research question "Which tweets will be retweeted most?", we need to filter this raw data. The first problem is to decide from whom we are we going to collect our data. As we want to create a dataset with both important and non-important twitterers, a logical solution is to select the twitterers using random Twitter id's. The code that is used for downloading the dataset is available in addendum 1. Most important is that we use randomly generated account numbers (nine numbers). To select a tweet from an account, two criteria are added. First, the tweets should be in English so we don't have to take translations into account for the sensitivity analysis. We note that the language of the retweeted tweet is in English (not the original tweet that is retweeted). Therefore, it is possible that there are still non-English tweets in the dataset before data preparation. Second, the tweets should contribute to an equally divided dataset. More specifically, we want to have 50% tweets that are retweeted and 50% that are not retweeted, because literature shows that the results are best with equally divided samples.

We used the twitter *API_user_timeline* command and the package *tweepy* for downloading tweets. Furthermore, we made use of personal keys: a consumer key, an access key and a consumer access. These items made it possible to set up a personal connection with twitter. In accordance with those keys we can extract tweets and user information for free. Nevertheless, twitter can track who is downloading their data and there is an upper limit to the amount of calls you can make in a period of time. For this reasons you need to build in sleeping periods in your program when extracting data. Pay attention that we downloaded this data locally.

After running the tweet downloader for many nights on the computer, we accumulated a dataset of 560 000 tweets with 280 000 retweeted and 280 000 not retweeted. Due to the limitations of the download API, we never had more than 200 tweets per user. The number of unique users we used to extract the tweets is 17 730. The tweets downloaded originated from 93 999 different users. After the data was gathered, we cleaned and modified it on the *databricks* platform.

To prevent overfitting on the dataset we created the accounts that extracted the tweets completely random. The downloaded data is saved as a JSON as this is an easy way of saving social media data. Next, we applied a process called serialization to transform all these individual JSON files into one big table, called *dataframe* on our platform *databricks*. Keeping in mind that we store each tweet in an individual JSON, our final *dataframe* will be a collection of JSONs. For more information about tweets in JSON format, you can look on the developer site from twitter. In figure below there is a visualization of a sample tweet in JSON structure. This whole process provides us with the data to analyze what would be the impact of different tweets from different users on the retweetability of tweets.

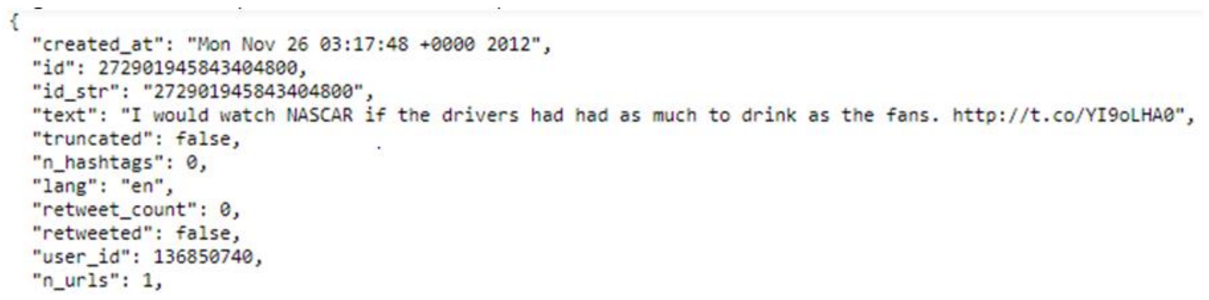
When we take a closer look at the different JSONs in attachment, we see a remarkable difference between the flattened and the normal shaped JSON. The big difference is that the normal JSON goes much deeper (multiple dimension). This gives us a lot of complication when we want to make a dataset of this deep JSON. Therefore we need to flatten this JSON first. The originally data contains concrete enumerations of concrete attributes where for our research we typically want only the number of attributes. In other places the data we want is deeply nested into the JSON structure and to fit in the *dataframe* model we don't want nested JSONs. You can find an example in figure below. You can find the complete JSONs in attachment 1.



```
{
  "created_at": "Mon Nov 19 11:33:36 +0000 2012",
  "id": 270490001706463232,
  "id_str": "270490001706463232",
  "text": "http://t.co/mberTsF5 I would watch NASCAR if the drivers had had as much to drink as the fans.",
  "truncated": false,
  "entities": {
    "hashtags": [

    ],
    "symbols": [

    ],
    "user_mentions": [

    ],
    "urls": [
      {
        "url": "http://t.co/mberTsF5",
        "expanded_url": "http://bitly.xaijo.com/GaBkOmFl",
        "display_url": "bitly.xaijo.com/GaBkOmFl",
        "indices": [
          0,
          20
        ]
      }
    ]
  },
  "source": "<a href=\"http://mobile.twitter.com\" rel=\"nofollow\">Mobile Web</a>",
```

…

After flattening becomes



```
{
  "created_at": "Mon Nov 26 03:17:48 +0000 2012",
  "id": 272901945843404800,
  "id_str": "272901945843404800",
  "text": "I would watch NASCAR if the drivers had had as much to drink as the fans. http://t.co/YI9oLHA0",
  "truncated": false,
  "n_hashtags": 0,
  "lang": "en",
  "retweet_count": 0,
  "retweeted": false,
  "user_id": 136850740,
  "n_urls": 1,
```

Figure 2: Example flattening JSON

4.3. Data Preparation

When we look at all different variables, we see that we have too many variables to start with. The goal is to make a model that includes the most predictive and most effective parameters. We include the variables as suggested in previous research (referred to as standard variables) and add the ones which we think could contribute (see above) .

For the classification problem, the dependent variable is retweeted_dummy. This is a boolean, which takes the value 1 if the tweet is retweeted, 0 otherwise. For the regression problem the dependent variable is retweet_count. This is the number of times that the original tweet/message is retweeted. The standard independent variables are the same for the classification and the linear problem. Table 2 summarizes the standard variables.

Independent variable	**Explanation**
followers_count	current number of followers of the user. A follower is a person who follows you on twitter. You don’t need to follow him/her back. It is just a follower.
friends_count	The number of users this account is following (AKA their “followings”).
favourites_count	The number of Tweets this user has liked in the account’s lifetime.
n_user_mentions	The number of users that are mentioned in the tweet. There is a list of which users are mentioned but we are only interested in how much users are included in the tweet
n_urls	The number of urls that are mentioned in the tweet. There is a list of which urls are mentioned but we are only interested in how much urls are included in the tweet
n_hashtags	Represents the number of hashtags which have been parsed out of the Tweet text.
n_listed	The number of public lists that this user is a member of. *
n_statuses	The number of Tweets (including retweets) issued by the user

**

Table 2: The list of independent variables

**A List is a curated group of Twitter accounts. You can create your own Lists or subscribe to Lists created by others. Viewing a List timeline will show you a stream of Tweets from only the accounts on that List (google)*

** Remark: n_favorited (= the number of times that the tweet is favorited) is not put in the list of independent variables. Because we won’t have that information if we have to predict the retweets of a tweet. Another research question could be: *“Predicting which tweets will be favorited most*”?

Next to these variables we have created additional variables that hopefully will add predictive value to this model. Table 3 summarizes these additional variables.

Extra Independent Variables	**Explanation**
count_positive	The number of positive words that are included in the text of the tweet. *
count_negative	The number of negative words that are included in the text of the tweet. *
n_symbols	The number of symbols that are mentioned in the tweet. There is a list of which symbols are mentioned but we are only interested in how much symbols are included in the tweet
n_media	Represents the number of media elements uploaded with the Tweet. A media element is a photo, an url, an expanded url,...

Table 3: The list of extra independent variables

*To get a sentiment score for each tweet, we use a lexicon-based approach. Before we are able to execute sentiment analysis, we need to clean the text and tokenize the tweets into English words. We make multiple strings from one large string. This means per tweet we filter out all irrelevant characters, remove RT, convert everything to lower case, tokenize the result and count how many positive and negative words appear in the resulting vector. The lists of words we used consist of more than 2 000 words (Ding, Liu, and Zhang 2009). For an example see Table 4.

Different steps	**Text**
Original text from retweet	RT @xfl2020: This is **great** football reimagined. This is the XFL. Watch the official announcement — LIVE NOW! #XFL2020 https://t.co/KFX5oLmkHw
Convert to lower case	rt @xfl2020: this is **great** football reimagined. this is the xfl. watch the official announcement — live now! #xfl2020 https://t.co/kfx5olmkhw
Remove rt, http and all special characters	this is **great** football reimagined this is the xfl watch the official announcement live now xfl2020
Remove stopwords	**great** football reimagined xfl watch official announcement live xfl2020
Tokenize	[“**great**”,"football","reimagined","xfl","watch","official", "announcement","live","xfl2020"]
Count positive words	**1**
Count negative words	0

Table 4:Example of steps to calculate count positive and negative words

It is important to mention that retweeted tweets and non-retweeted tweets have a different structure (JSON). If a tweet is retweeted we are interested in the data of the original tweeter, and not of the re-tweeter. So we create new variables that indicate whether the data is retweeted from another account or whether the data is purely user-generated. An example will make this clearer. For example, I as a student see a tweet that I want to share from prof. Dirk Van den Poel, and then I can retweet his original tweet. On that same evening there is a soccer game going about which I tweet. When we want to collect this information, we want to assemble for the first tweet the user and text data from the original tweeter prof. Dirk Van den Poel. For the second tweet, we want to collect user and text data from Jolan Bourdeaud'hui. Hence, we always want the data from the original tweet.

First we make a dataset for our classification problem. Is a tweet retweeted or not? For this more appropriate shaped JSON, we make sure that there are no variables present in the *dataframe* from retweeted tweets, because these variables are the consequence of a retweet and thus cannot be used to predict a retweet.

In Attachment 2 you see an example of a flattened JSON for a tweet for which the usage in our model is indicated.

First we look at the distribution number of tweets per user. We have 560 000 tweets in total and 17 730 unique users. Each user has an average of 31.5 tweets. When we look at the distribution of the number of tweets per user we see that this is not normally distributed. This is because there are a lot of users that tweet very rarely and only a small percentage of the twitter network is responsible for the most diffusion on twitter. Figure 3 shows that only 10 percent of the users are responsible for 90 percent of the total tweets. When we generate our user-id randomly, there is a bigger chance to take an account with fewer tweets. The maximum number of tweets per user we can fetch through the twitter api is 200. 50% of the unique users have 5 tweets or less that meet the requirements.

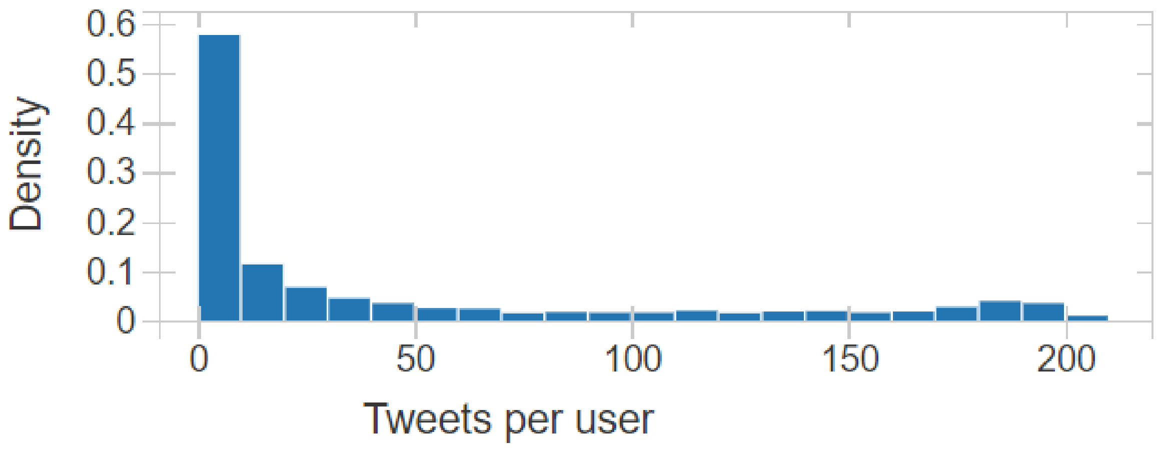


Figure 3: Overview tweets per user

Figure 4 shows that 50% is retweeted and 50% is not retweeted. Of the 560 000 there were 280 000 tweets retweeted and 280 000 not retweeted. We have retweeted tweets from original tweeters and retweeted tweets from the user timeline we scraped. 230 000 Of the retweets (or 83%) come from tweets from other tweeters. 47 678 Of the retweets (or 17%) are written by the users we scraped and are retweeted by others..

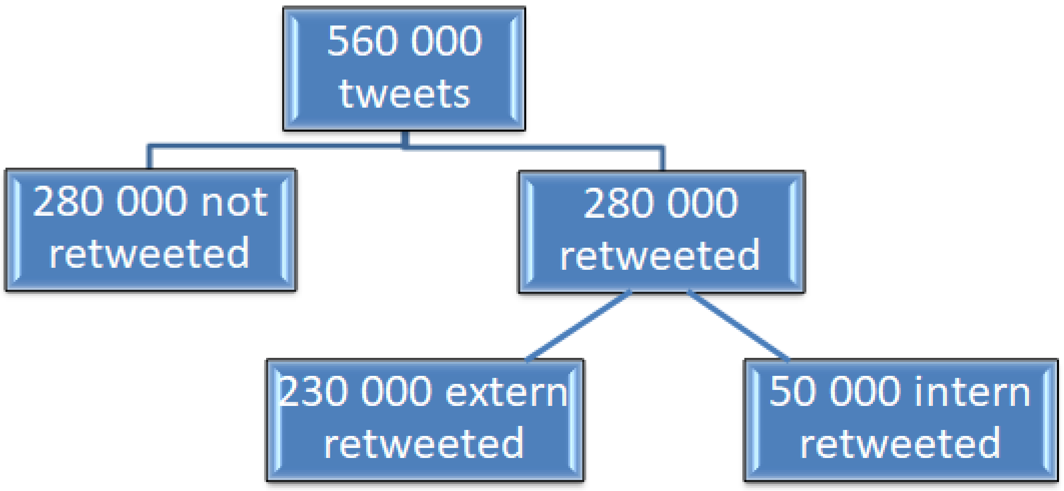


Figure 4: Overview distribution tweets (rounded numbers)

4.4. MODELING

4.4.1 Pipeline

To implement our predictive models we use *databricks python spark (pyspark)*. This is a pretty new programming language. *Python* allows us to run functions over collections on multiple processors in parallel. *Spark* allows us to run functions over multiple servers in parallel (Odersky, 2017). This should allow us to analyze massive amounts of tweets in a reasonable time. Working locally on *python* would be limited because we are tied to a single compute node (computer). To model everything in *PySpark*, we need to put our data in the right format. First of all we considers null values being the integer 0.

Most of the operations will be done on either the categorical values, the continuous values or the label. Therefore, a split-up in a label, the categorical features and the continuous features is the most efficient. The conversion is the easiest with *RDD's*, therefore we will transform the *DataFram*e to an *RDD* and afterwards use a map function to transform each of the rows.

After putting it in the right format we convert the *RDD* back to a *DataFrame* and we split the base table up in a train and test set. 70% of the data is used for training and 30% is used for testing. The splitting of the data is executed randomly to save computation time. Cross validation is not used because we have limited *Databricks* resources.
Now we have our *basetable* ready for a Machine Learning Pipeline. So now, we need to construct a pipeline.

The pipeline will consist out of five stages:

1. *StringIndexe*r: A label indexer that maps a string column of labels to an ML column of label indices. So this makes category indices of the strings in a column.
2. *VectorIndexer*: Class for indexing categorical feature columns in a dataset of Vector.
3. *StandardScaler*: Standardizes features by removing the mean and scaling to unit variance using column summary statistics on the samples in the training set.
4. *VectorAssemble*r: A feature transformer that merges multiple columns into a vector column.
5. Decide on the machine learning algorithm that we will be using. We will give an example of Logistic Regression, Decision Tree learning, Random Forest

The next step in our model is training the pipeline, the number of iterations we trained differs for the models we used.

- For logistic regression the number of iterations is 1000.
- For the decision tree classification our model came up with the solution of depth 5 with 63 nodes. Each node represent a test or decision.
- For random forest 500 number of trees were used. This is a good trade-off between computation time and error.

We made use of machine learning techniques classification and regression. There is an important difference between classification and regression problems. Fundamentally, classification is about predicting a label and regression is about predicting a quantity.

For the first research question we use classification. Classification gives answers to a binary problem: Is a tweet retweeted or not? We use logistic regression, random forest and decision tree learning in this scenario. We make a subdivision. Classification retweeter is using the new user specific data. Classification original tweeter is using the original user specific data

For the second research question we have to make use of regression. Regression gives answers to continuous problems: How many times is a tweet retweeted? We use linear regression, random forest and decision tree in this scenario.

4.4.2 Evaluation

4.4.2.1 Classification retweeter. Table 5 shows the results for the three previous discussed models. We first made our *basetable* with the data from the last person that tweeted, not the original. We had 12 independent variables and 1 dependent variable.

Model	**Performance**	**Total_accuracy**	**Accuracy_not_retweeted**	**Accuracy_retweeted**
Logistic regression	0.82	0.747	0.76	0.733
Decision tree	0.755	0.776	0.811	0.741
Random forest	0.853	0.782	0.836	0.727

Table 5: Results classification retweeter

The performance measure is AUC (i.e. Area Under Curve). For binary classification we will use one performance measure in this section called Area Under the Receiver Operating Characteristics curve (AUC). AUC is a measure between 0.5 (if the model is not doing better than random selection) and 1 (if the model makes perfect predictions). The area under the ROC curve (AUC) is a well-known measure of ranking performance, estimating the probability that a random positive is ranked higher than a random negative, without committing to a particular decision threshold. (book predictive and prescriptive analytics)

Figure 5 tells us that random forest performs best. We see that the AUC is quite high,, but there is definitely still room for improvement.

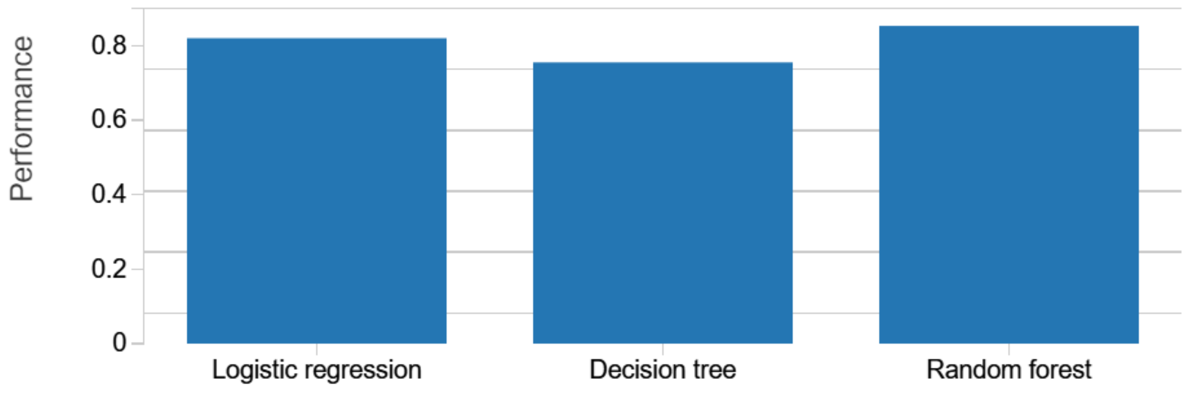


Figure 5: Performance (AUC) classification problem retweeter

4.4.2.2 Classification original tweeter

Model	Performance	Total_accuracy	Accuracy_not_retweeted	Accuracy_retweeted
Logistic regression	0.888	0.81	0.672	0.949
Decision tree	0.826	0.91	0.891	0.93
Random forest	0.969	0.916	0.88	0.953

Table 6: Results classification original tweeter

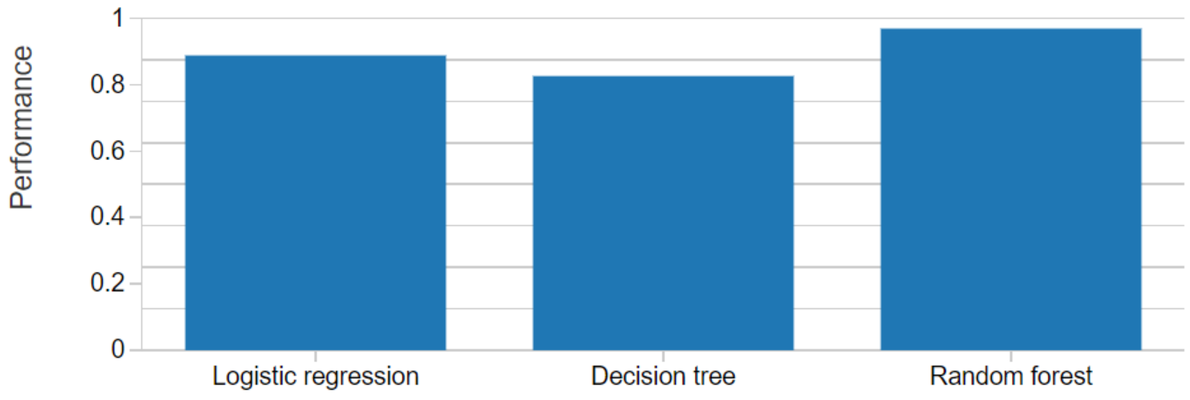


Figure 6: Performance (AUC) classification original tweeter

The performance from classification problem 2 is remarkably higher than the performance of classification problem 1. Replacing the user parameters of the new tweeter in the original tweeter has a positive influence on performance. We see that Random forest is our best model with an AUC of 0.969. When we want to predict which tweets will be retweeted most, we must change from a classification problem to a linear problem. The independent variable is no longer retweeted_dummy but retweeted_count. We don't change the dependent variables. They stay the same as the variables in classification problem original tweeter. We remove all data that is not retweeted from the dataset, because we are interested in how much a tweet get retweeted. So all observations must be retweeted at least 1 time.

4.4.2.3 Regression original tweeter. Note that the dependent variable is now changed to retweet count. The independent variables remain the same.

Model	**Performance**	**MeanAbsoluteError**
Logistic regression	0.046	0.81
Decision tree	0.529	0.91
Random forest	0.408	0.916

Table 7 : Mean Absolute Error regression original tweeter

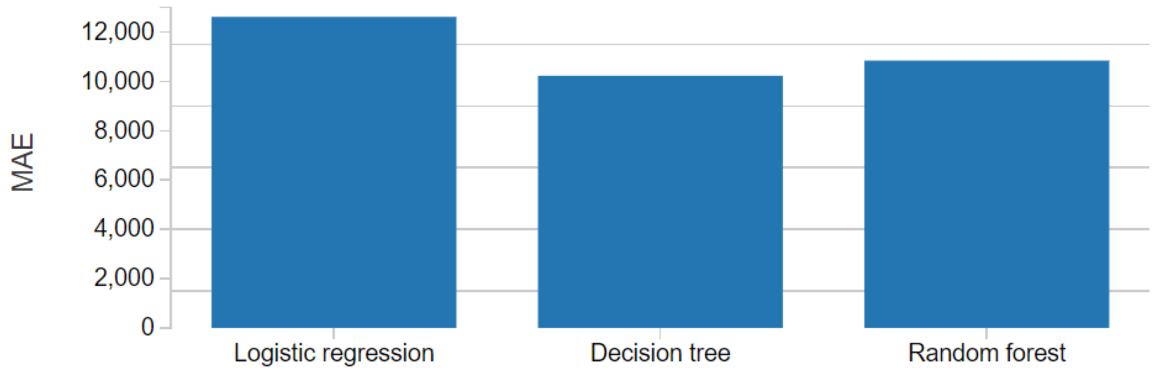


Figure 7: Mean Absolute Error regression original tweeter

Figure 8 shows that decision tree is outperforming random forest, which is at least remarkable. When we look at the figure 7 of the Mean Absolute Error, we see that the MAE has risen significantly. It is a lot harder to predict how many times a tweet is going to be retweeted than the classification if a tweet is going to be retweeted.

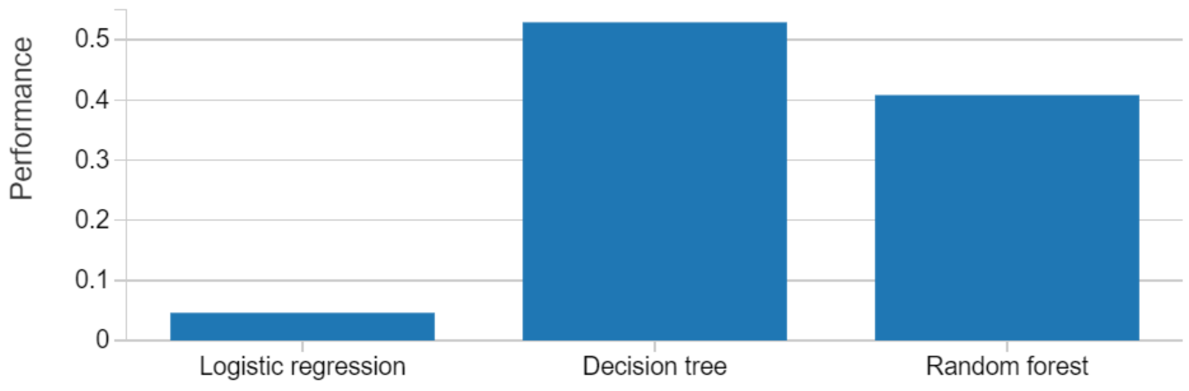


Figure 8: Accuracy (R^2) regression original tweeter

The performance of the models is being expressed in R^2. This is the proportion of the variance in the dependent variable that is predictable from the independent variables. We see that it dropped relative to the classification problems. There is a lot of uncertainty how much a tweet is going to be retweeted.

5. Data Analysis

5.1. Classification retweeter

The total accuracy is the percentage of tweets that is actually correctly predicted. If all retweeted tweets are predicted as retweeted tweets and all not retweeted tweets are retweeted as not retweeted tweets. This would result in an accuracy of 100%. The total accuracy in classification problem 1 is quite high, but as said before there is still room for improvement. We can see in Figure 9 that Random forest is our best model with a performance of 0.853.

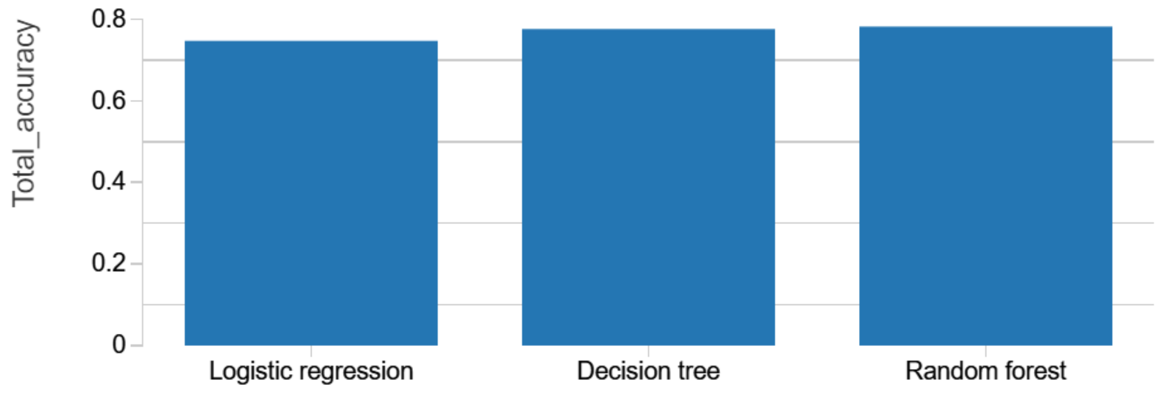


Figure 9: Total accuracy classification retweeter

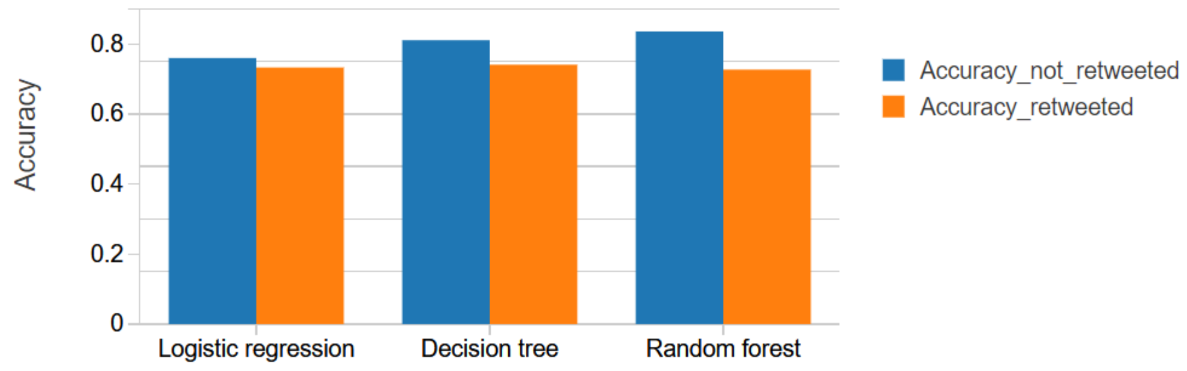


Figure 10: Specific accuracy classification retweeter

Remark that the accuracy of the not retweeted tweets is higher than the accuracy of the retweeted tweets (Figure 10). This is very surprising. Our model predicts more that a tweet will not be retweeted. A possible explanation for this phenomenon is that we have a lot of not famous people who don’t get retweeted so much. Because we don’t use - in this classification problem - the data from the original tweeter, we get a lot of user data from ordinary people.

5.2 Classification original tweeter

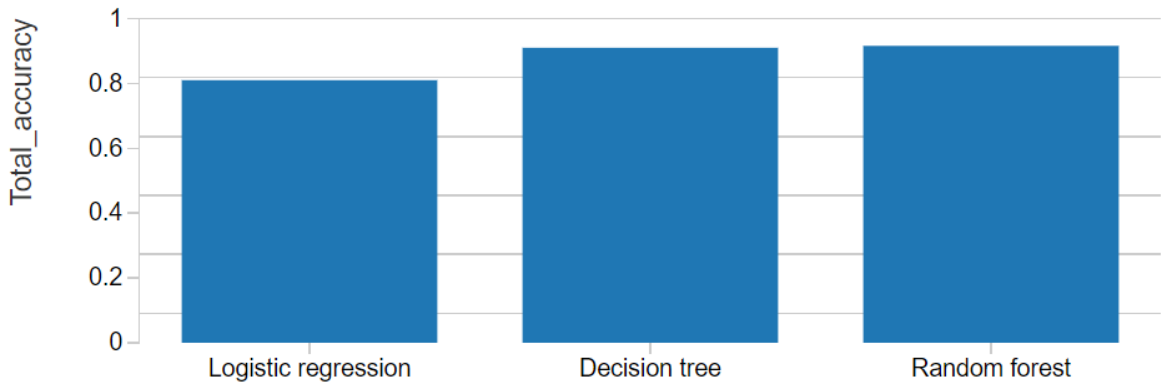


Figure 11: Total accuracy classification problem original retweeter

Replacing the user parameters of the new tweeter in the original tweeter also has a positive influence on the accuracy.

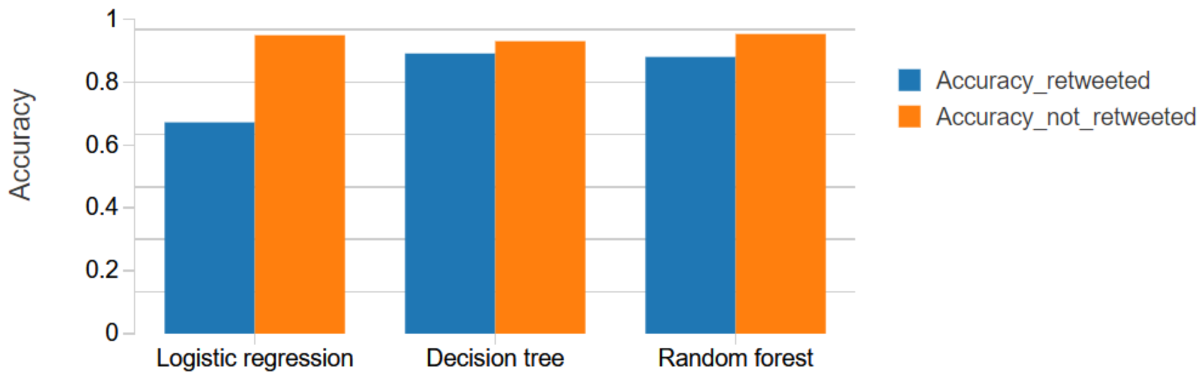


Figure 12: Specific accuracy classification problem original retweeter

Just as in classification problem 1, the accuracy of the not retweeted tweets is higher than the accuracy of the retweeted tweets.

5.3 Regression original tweeter

For the regression problem we don’t have meaningful results concerning accuracy and residuals. We refer to attachment 3 for the residuals.

6. Results

6.1. Variable importance vector

Table 9 shows the variable importance vector for the best model (2 times Random Forest , 1 time Decision Tree). The total of the sum of the feature importance vector is 1. The (absolute) magnitude of each non-zero weights can give an idea about the importance of the corresponding attribute. Irrelevant variables are set to zero. Many variable selection algorithms include variable ranking as a principal or auxiliary selection mechanism because of its simplicity, scalability, and good empirical success (Guyon & Elisseeff, 2003).

Table 8 is ordered from most important features above to least important features below. When we look at Table 8, we see variables changing position when we change classification into regression problem. We also see a remarkably difference in the variable importance between the two classification problems. In classification problem 1 (with the new user retweeted data) the number of users mentioned is the most important variable. In contrary with classification problem 2 (with the original user retweeted data) and the linear problem (with the number of retweets as dependent variable), the number of statuses of the user makes the reserve movement. At classification problem number 1 it is less important, than in problem 2 and 3. However, the most special perception of the table is that the number of followers is not that important in problem 1 and 3. In problem 2 - this is also the problem with the best results in terms of performance - the number of followers is the most important feature. We see a global trend that the user specific data (followers, friends, statuses,..) are more important than tweet specific data like number of hashtags, urls or positive/negative words. This confirms our hypotheses that user related features are more important than content related features.

In the past, a lot of research has been done to find out which parameters of the tweet correlate with the number of times a tweet is retweeted. Luckily there is always room for improvement and twitter provides us with extra parameters. For instance, there are the number of the memberships of lists and the different types of media attached to the tweet. The results of these parameters are particularly interesting.

Place (variable importance)	Classification retweeter	Classification original tweeter	Regression
1.	n_user_mentions	followers_count	n_statuses
2.	n_media	n_statuses	friends_count
3.	followers_count	n_listed	favourites_count
4.	friends_count,	friends_count	n_listed
5.	favourites_count	favourites_count	followers_count
6.	n_urls	n_media	n_media
7.	n_listed	n_urls	n_hashtags
8.	n_statuses	n_user_mentions	count_positive
9.	n_hashtags	n_hashtags	n_urls
10.	count_negative	count_negative	n_user_mentions
11.	count_positive	count_positive	count_negative
12.	n_symbols	n_symbols	n_symbols

Table 8: Overview features variable importance vector (3 models: 2 times Random Forest, 1 time Decision Tree)

Classification retweeter	Classification original tweeter	Classification retweeter	Regression
favourites_count	0.154	0.059	0.142
followers_count	0.042	0.399	0.114
friends_count,	0.038	0.090	0.200
n_hashtags	0.010	0.002	0.044
n_urls	0.012	0.008	0.011
n_user_mentions	0.629	0.007	0.006
n_listed	0.011	0.203	0.117
n_symbols	0.0	0.0	0.0
n_media	0.085	0.014	0.107
count_positive	0.001	0.0	0.022
count_negative	0.001	0.0	0.004
n_statuses	0.010	0.217	0.231

Table 9: Overview values of the features variable importance vector (3 models: 2 times Random Forest, 1 time Decision Tree

6.2. Correlation matrix

6.2.1 Classification original tweeter

An interesting result is the correlation matrix. Correlation is a scaled measure of how two variables change with respect to each other. We knew already which variables were more important than others (see tables 8 and 9), but the correlation matrix learns us how the variables influence the dependent variable. In table 10 we see that there are only two negative variables namely the number of urls in the tweet and the number of users mentioned in the tweet. This means that if you add one (or more) url or users in your tweet, the number of times that you are going to be retweeted drops. All other parameters are positive. This means that, for example, the more followers you have, the more chance you have to be retweeted. The variables with the highest correlation values are numbers of media elements that are uploaded with the tweet, the number of favorites of the user and the number of followers of the user. Adding media to your tweet helps to get your message retweeted.

Features	**Correlation with feature retweeted**
Retweeted	1
favourites_count	0.188
followers_count	0.155
friends_count	0.114
n_hashtags	0.017
n_urls	-0.059
n_user_mentions	-0.129
n_listed	0.112
n_symbols	0.008
n_media	0.282
count_positive	0.055
count_negative	0.051
n_statuses	0.289

Table 10: Correlation matrix classification original retweeter for random forest (the corresponding line of the dependent variable retweeted)

6.2.2 Regression problem

Table 11 shows the corresponding line of the dependent variable (retweet_count) for the regression problem. When we look at the correlation between the retweeted count and all the variables in Table 11, we see similar results as before. There are two extra variables that have a negative impact on the retweet count, namely the number of hashtags and the number of symbols. But the correlation values are relatively low. The number of public lists that a user is member of, the number of followers and the number of statuses has the biggest correlation values with the retweet count.

Features	**Correlation with feature retweet_count**
favourites_count	0.044
followers_count	0.187
friends_count	0.055
n_hashtags	-0.011
n_urls	-0.001
n_user_mentions	-0.027
n_listed	0.118
n_symbols	-0.001
n_media	0.071
count_positive	0.004
count_negative	0.007
n_statuses	0.127
retweet_count	1

Table 11 : Correlation matrix regression problem for decision tree (the corresponding line of the dependent variable retweet_count)

7. Discussion

The results are in line with the expectations. The global trend is that most of the time user specific information is more decisive than text content specific information to be retweeted. This confirms our first hypothesis that user related features are more important than content related features. The most remarkable and maybe surprising result is the relatively high positive correlation between the variables “memberships lists” and “number of retweets”. Further, the negative correlation between “number of urls” and the variable “retweeted” is at least equally interesting.

The answer on the second hypothesis “the most important feature of the original tweeter is the number of followers” can be confirmed from the variable importance vector from the classification model. This is in contradiction to the regression model. Note that the classification has proven to be much more reliable than the regression model so we can confirm the hypothesis.

The answer on the third hypothesis “messages that score high on sentiment, positive or negative, will be retweeted more” can be confirmed from the solution from the correlation matrix. Nevertheless, our results showed that there was only very limited correlation. The most remarkable finding was that as well tweets with positive as tweets with negative sentiment both get more retweets.

The last hypothesis declares that the importance of the individual features for the binary classification model is about the same as the importance of the features for the regression model. The top six features in the importance matrix (table 8) are the same, although they don’t appear in the same order. This is logical because we have another but related dependent variable (i.e. retweet count).

This thesis can have several implications for future research and practice. As our dataset and our model are made available, these products can be reused in future research. We are aware that our dataset is still in an early stage. However, as our dataset and model is freely available and self-explanatory we empower the reader to contribute in this educational Machine Learning project and unleash the power of Big Data in the area of social media. One can contribute to our research by enhancing either the dataset or the model (e.g. in adding more independent variables). Further, we believe that marketeers could benefit from adopting our model to his/her specific setting.

We also have to keep in mind that this research is conducted with completely random generated userIDs and this research is not applicable to all tweets of the world.

We can conclude that we have reached several meaningful models with great performance measures, but there is definitely room for improvement and there are some limitations.

Bibliography

Boyd, D., S. Golder, & G. Lotan. (2010). "Tweet, Tweet, Retweet: Conversational Aspects of Retweeting on Twitter." In 2010 43rd Hawaii International Conference on System Sciences. https://doi.org/10.1109/hicss.2010.412.

Breiman, L. (1999). Using adaptive bagging to debias regressions. Technical Report, Department of Statistics, University of California, Berkeley.

Breiman, L. & Oct (2001). Random Forests. *Machine Learning 45* (1), 5–32.

Chao-Ying Joanne Peng , Kuk Lida Lee & Gary M. Ingersoll, (2002). An Introduction to Logistic Regression Analysis and Reporting, *The Journal of Educational Research, 96*(1), 3-14. doi: 10.1080/00220670209598786

Chapman, Pete (2000). *CRISP-DM 1.0: Step-by-Step Data Mining Guide*.

Dang-Xuan, L., & Stieglitz, S. (2012). Political Communication and Influence through Microblogging. An Empirical Analysis of Sentiment in Twitter Messages and Retweet Behavior. 47th Hawaii International Conference on System Sciences. doi:10.1109/HICSS.2012.476.

Ding, Xiaowen, Bing Liu, & Lei Zhang, 2009. "Entity Discovery and Assignment for Opinion Mining Applications." In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '09. https://doi.org/10.1145/1557019.1557141.

Flach, P., Hernandez-Orallo, J., Ferri, C., 2011. A Coherent Interpretation of AUC as a Measure of Aggregated Classification Performance. In: Proceedings of the 28th International Conference on Machine Learning. Bellevue,WA, USA, p. 8.

Hong, Liangjie, Ovidiu Dan, & Brian D. Davison (2011). "Predicting Popular Messages in Twitter." In Proceedings of the 20th International Conference Companion on World Wide Web - WWW '11. https://doi.org/10.1145/1963192.1963222.

Isabelle, C. (2003). An Introduction to Variable and Feature Selection, *JMLR, 3*(Mar), 1157-1182.

Wiley, J. & Sons, I. (2011). *WIREs Data Mining Knowl Discov (1),* 14–23. doi: 10.1002/widm.8

Chapman, P. (2000). CRISP-DM 1.0: Step-by-step Data Mining Guide.

Kupavskii A, Umnov A, Gusev G, & Serdyukov P. (2013). Predicting the audience size of a tweet. In ICWSM. Cambridge, Massachusetts, USA, The AAAI Press. Google Scholar

Lekha. R. N., & Sujala, D. S. (2015). Streaming Twitter data analysis using Spark. *Journal of Theoretical and Applied Information Technology, 80*(2).

Shoro, A. G., & Soomro, T. R. (2015). Big Data Analysis: Ap Spark Perspective. *Global Journal of Computer Science and Technology: C Software & Data Engineering, 15*(1). USA: Global Journals Inc.

Odersky, M.(2017). Online course: Big Data Analysis with Scala and Spark. Coursera. École Polytechnique Fédérale de Lausanne.

Suh, Bongwon, Lichan Hong, Peter Pirolli, & Ed H. Chi. (2010). “Want to Be Retweeted? Large Scale Analytics on Factors Impacting Retweet in Twitter Network.” In 2010 IEEE Second International Conference on Social Computing. https://doi.org/10.1109/socialcom.2010.33.

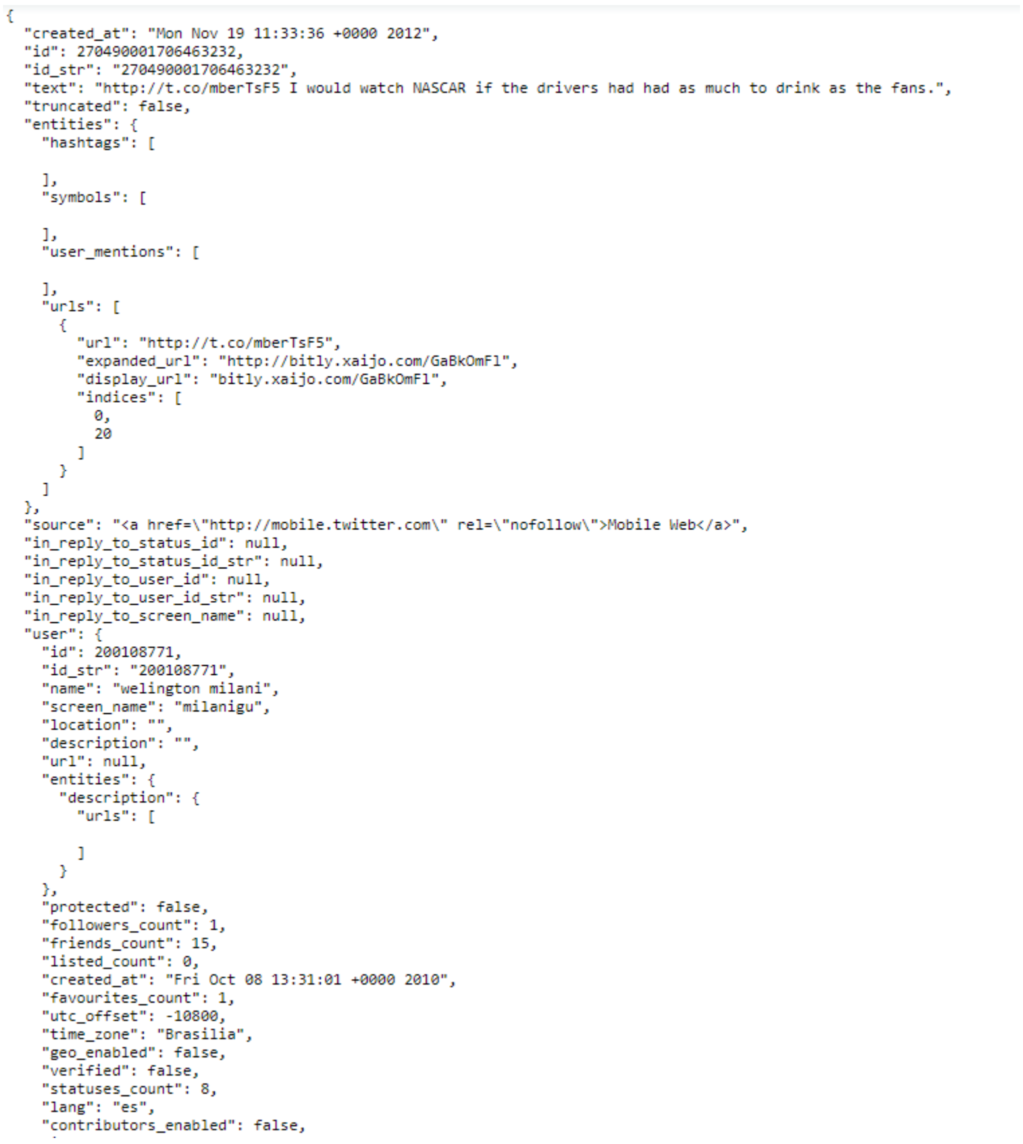
Tang X., Miao Q., Quan Y., Tang, J. & Deng, K. (2015). Predicting Individual Retweet Behavior by User Similarity: A Multi-Task Learning Approach. *Knowledge-Based Systems, 89,* 681–88.

Yang, Q., & Xu Z. (2012). Advances in Social Networks Analysis and Mining (ASONAM). Analyzing User Retweet Behavior on Twitter. International Conference. Turkey. Ddoi: 10.1109/ASONAM.2012.18.

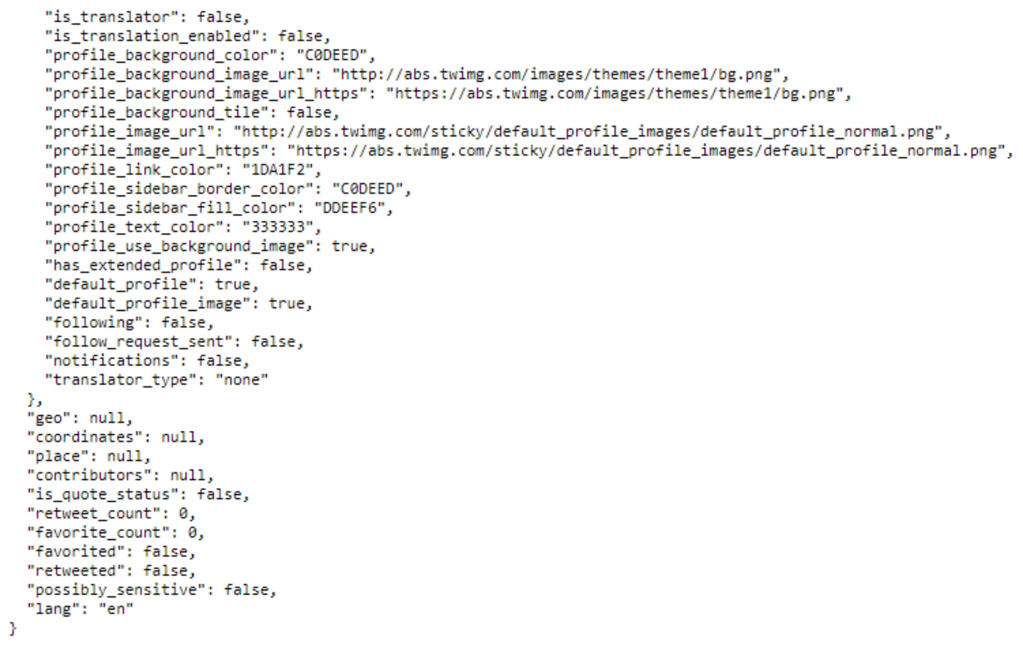
Zaharia, Matei, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, Ion Stoica, Reynold S. Xin, et al. 2016. “Apache Spark.” *Communications of the ACM 59 (11),* 56–65.

Attachments

Attachment 1: Examples JSON structure downloaded and flattened

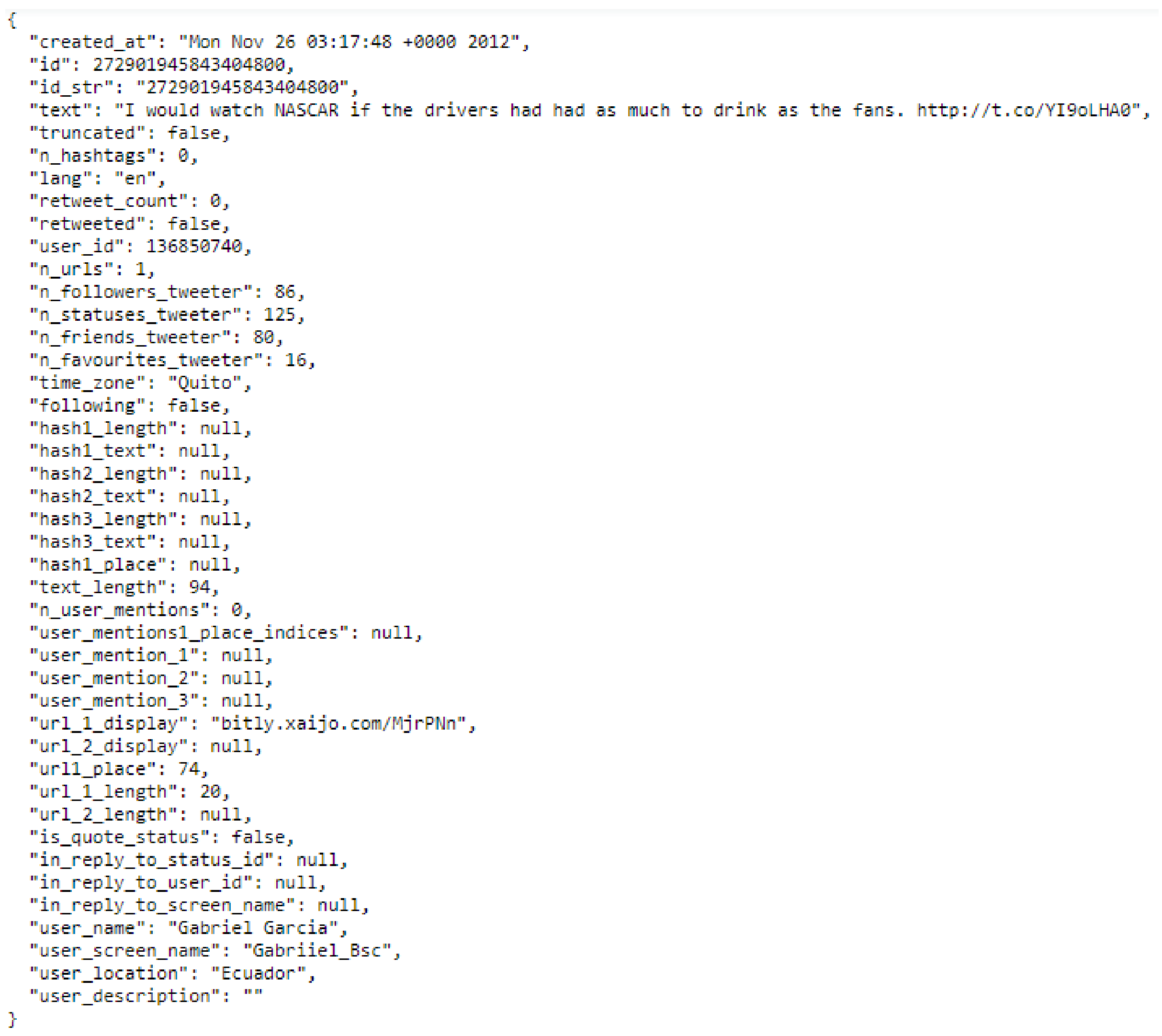


```
{
  "created_at": "Mon Nov 19 11:33:36 +0000 2012",
  "id": 270490001706463232,
  "id_str": "270490001706463232",
  "text": "http://t.co/mberTsF5 I would watch NASCAR if the drivers had had as much to drink as the fans.",
  "truncated": false,
  "entities": {
    "hashtags": [

    ],
    "symbols": [

    ],
    "user_mentions": [

    ],
    "urls": [
      {
        "url": "http://t.co/mberTsF5",
        "expanded_url": "http://bitly.xaijo.com/GaBkOmFl",
        "display_url": "bitly.xaijo.com/GaBkOmFl",
        "indices": [
          0,
          20
        ]
      }
    ]
  },
  "source": "<a href=\"http://mobile.twitter.com\" rel=\"nofollow\">Mobile Web</a>",
  "in_reply_to_status_id": null,
  "in_reply_to_status_id_str": null,
  "in_reply_to_user_id": null,
  "in_reply_to_user_id_str": null,
  "in_reply_to_screen_name": null,
  "user": {
    "id": 200108771,
    "id_str": "200108771",
    "name": "welington milani",
    "screen_name": "milanigu",
    "location": "",
    "description": "",
    "url": null,
    "entities": {
      "description": {
        "urls": [

        ]
      }
    },
    "protected": false,
    "followers_count": 1,
    "friends_count": 15,
    "listed_count": 0,
    "created_at": "Fri Oct 08 13:31:01 +0000 2010",
    "favourites_count": 1,
    "utc_offset": -10800,
    "time_zone": "Brasilia",
    "geo_enabled": false,
    "verified": false,
    "statuses_count": 8,
    "lang": "es",
    "contributors_enabled": false,
```



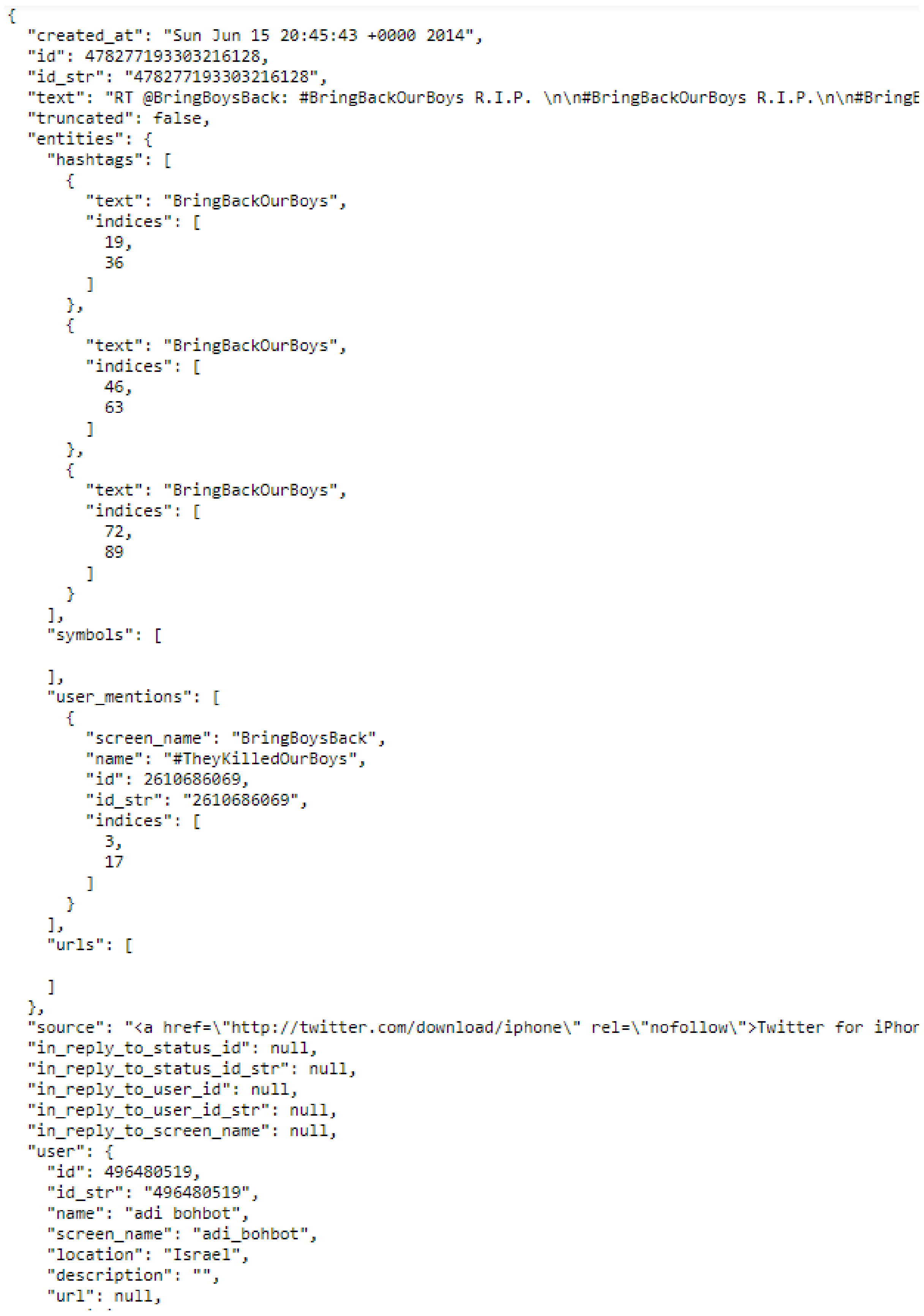
```
    "is_translator": false,
    "is_translation_enabled": false,
    "profile_background_color": "C0DEED",
    "profile_background_image_url": "http://abs.twimg.com/images/themes/theme1/bg.png",
    "profile_background_image_url_https": "https://abs.twimg.com/images/themes/theme1/bg.png",
    "profile_background_tile": false,
    "profile_image_url": "http://abs.twimg.com/sticky/default_profile_images/default_profile_normal.png",
    "profile_image_url_https": "https://abs.twimg.com/sticky/default_profile_images/default_profile_normal.png",
    "profile_link_color": "1DA1F2",
    "profile_sidebar_border_color": "C0DEED",
    "profile_sidebar_fill_color": "DDEEF6",
    "profile_text_color": "333333",
    "profile_use_background_image": true,
    "has_extended_profile": false,
    "default_profile": true,
    "default_profile_image": true,
    "following": false,
    "follow_request_sent": false,
    "notifications": false,
    "translator_type": "none"
  },
  "geo": null,
  "coordinates": null,
  "place": null,
  "contributors": null,
  "is_quote_status": false,
  "retweet_count": 0,
  "favorite_count": 0,
  "favorited": false,
  "retweeted": false,
  "possibly_sensitive": false,
  "lang": "en"
}
```

Figure 20: JSON (not retweeted tweet) normal structure

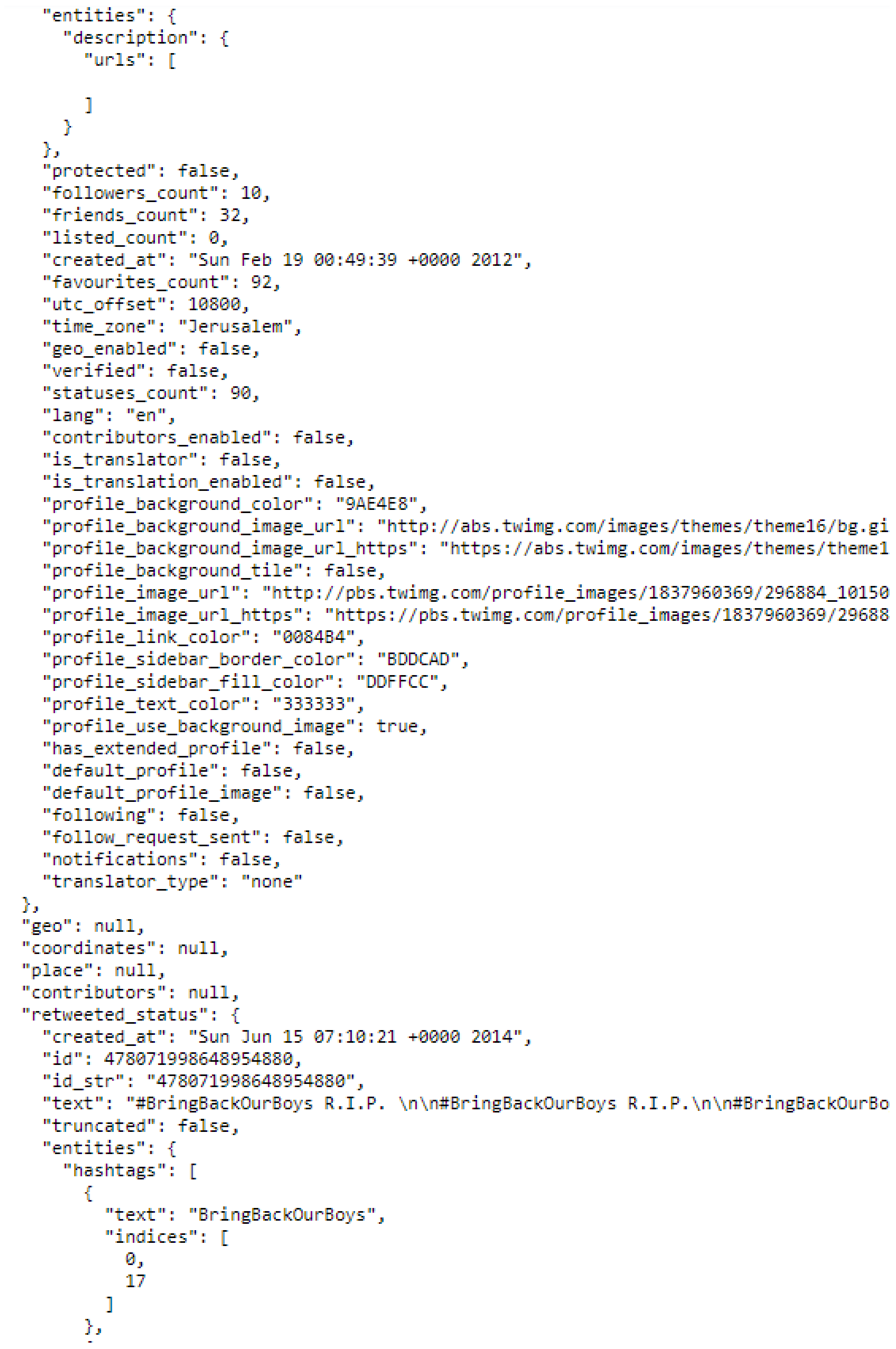


```
{
  "created_at": "Mon Nov 26 03:17:48 +0000 2012",
  "id": 272901945843404800,
  "id_str": "272901945843404800",
  "text": "I would watch NASCAR if the drivers had had as much to drink as the fans. http://t.co/YI9oLHA0",
  "truncated": false,
  "n_hashtags": 0,
  "lang": "en",
  "retweet_count": 0,
  "retweeted": false,
  "user_id": 136850740,
  "n_urls": 1,
  "n_followers_tweeter": 86,
  "n_statuses_tweeter": 125,
  "n_friends_tweeter": 80,
  "n_favourites_tweeter": 16,
  "time_zone": "Quito",
  "following": false,
  "hash1_length": null,
  "hash1_text": null,
  "hash2_length": null,
  "hash2_text": null,
  "hash3_length": null,
  "hash3_text": null,
  "hash1_place": null,
  "text_length": 94,
  "n_user_mentions": 0,
  "user_mentions1_place_indices": null,
  "user_mention_1": null,
  "user_mention_2": null,
  "user_mention_3": null,
  "url_1_display": "bitly.xaijo.com/MjrPNn",
  "url_2_display": null,
  "url1_place": 74,
  "url_1_length": 20,
  "url_2_length": null,
  "is_quote_status": false,
  "in_reply_to_status_id": null,
  "in_reply_to_user_id": null,
  "in_reply_to_screen_name": null,
  "user_name": "Gabriel Garcia",
  "user_screen_name": "Gabriiel_Bsc",
  "user_location": "Ecuador",
  "user_description": ""
}
```

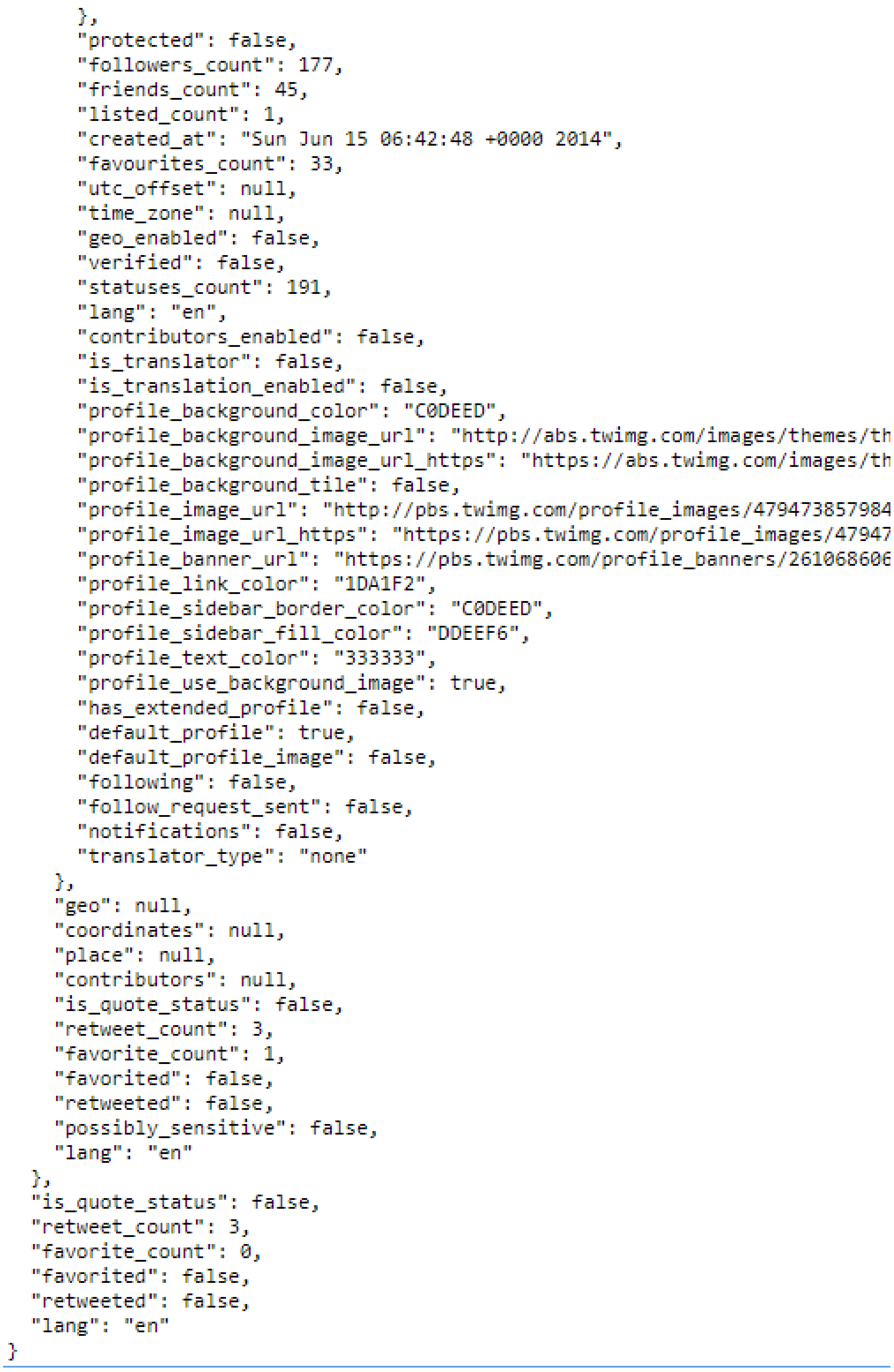
Figure 21: JSON (not retweeted tweet) reshaped structure



```
{
  "created_at": "Sun Jun 15 20:45:43 +0000 2014",
  "id": 478277193303216128,
  "id_str": "478277193303216128",
  "text": "RT @BringBoysBack: #BringBackOurBoys R.I.P. \n\n#BringBackOurBoys R.I.P.\n\n#BringE
  "truncated": false,
  "entities": {
    "hashtags": [
      {
        "text": "BringBackOurBoys",
        "indices": [
          19,
          36
        ]
      },
      {
        "text": "BringBackOurBoys",
        "indices": [
          46,
          63
        ]
      },
      {
        "text": "BringBackOurBoys",
        "indices": [
          72,
          89
        ]
      }
    ],
    "symbols": [

    ],
    "user_mentions": [
      {
        "screen_name": "BringBoysBack",
        "name": "#TheyKilledOurBoys",
        "id": 2610686069,
        "id_str": "2610686069",
        "indices": [
          3,
          17
        ]
      }
    ],
    "urls": [

    ]
  },
  "source": "<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for iPhor
  "in_reply_to_status_id": null,
  "in_reply_to_status_id_str": null,
  "in_reply_to_user_id": null,
  "in_reply_to_user_id_str": null,
  "in_reply_to_screen_name": null,
  "user": {
    "id": 496480519,
    "id_str": "496480519",
    "name": "adi bohbot",
    "screen_name": "adi_bohbot",
    "location": "Israel",
    "description": "",
    "url": null,
```

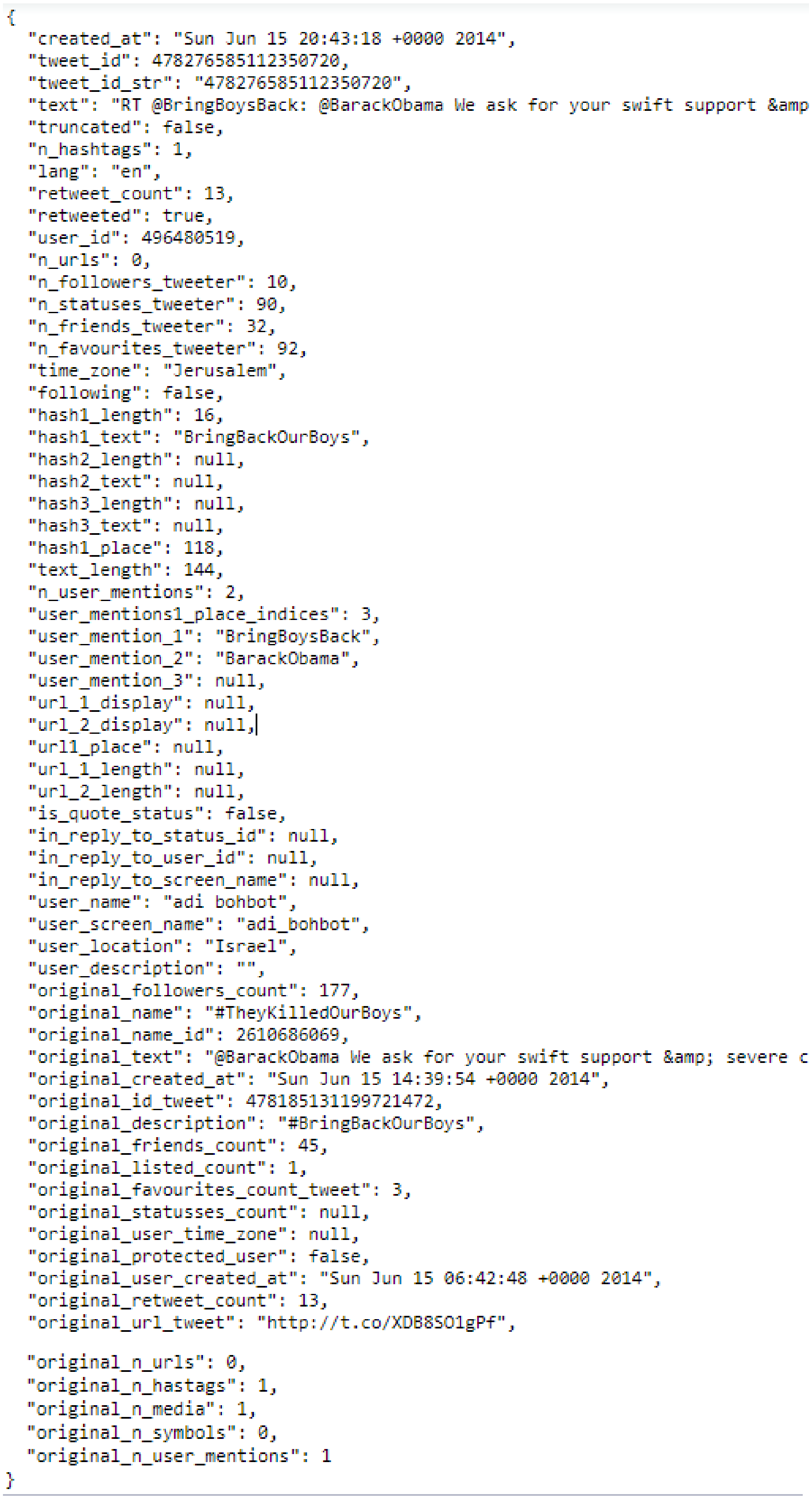


```
    "entities": {
      "description": {
        "urls": [

        ]
      }
    },
    "protected": false,
    "followers_count": 10,
    "friends_count": 32,
    "listed_count": 0,
    "created_at": "Sun Feb 19 00:49:39 +0000 2012",
    "favourites_count": 92,
    "utc_offset": 10800,
    "time_zone": "Jerusalem",
    "geo_enabled": false,
    "verified": false,
    "statuses_count": 90,
    "lang": "en",
    "contributors_enabled": false,
    "is_translator": false,
    "is_translation_enabled": false,
    "profile_background_color": "9AE4E8",
    "profile_background_image_url": "http://abs.twimg.com/images/themes/theme16/bg.gi
    "profile_background_image_url_https": "https://abs.twimg.com/images/themes/theme1
    "profile_background_tile": false,
    "profile_image_url": "http://pbs.twimg.com/profile_images/1837960369/296884_10150
    "profile_image_url_https": "https://pbs.twimg.com/profile_images/1837960369/29688
    "profile_link_color": "0084B4",
    "profile_sidebar_border_color": "BDDCAD",
    "profile_sidebar_fill_color": "DDFFCC",
    "profile_text_color": "333333",
    "profile_use_background_image": true,
    "has_extended_profile": false,
    "default_profile": false,
    "default_profile_image": false,
    "following": false,
    "follow_request_sent": false,
    "notifications": false,
    "translator_type": "none"
  },
  "geo": null,
  "coordinates": null,
  "place": null,
  "contributors": null,
  "retweeted_status": {
    "created_at": "Sun Jun 15 07:10:21 +0000 2014",
    "id": 478071998648954880,
    "id_str": "478071998648954880",
    "text": "#BringBackOurBoys R.I.P. \n\n#BringBackOurBoys R.I.P.\n\n#BringBackOurBo
    "truncated": false,
    "entities": {
      "hashtags": [
        {
          "text": "BringBackOurBoys",
          "indices": [
            0,
            17
          ]
        },
    {
      "text": "BringBackOurBoys",
      "indices": [
        27,
        44
      ]
    },
    {
      "text": "BringBackOurBoys",
      "indices": [
        53,
        70
      ]
    }
  ],
  "symbols": [

  ],
  "user_mentions": [

  ],
  "urls": [

  ],
  "media": [
    {
      "id": 478071995481870336,
      "id_str": "478071995481870336",
      "indices": [
        108,
        130
      ],
      "media_url": "http://pbs.twimg.com/media/BqJz8mvCUAAV2vl.jpg",
      "media_url_https": "https://pbs.twimg.com/media/BqJz8mvCUAAV2vl.jpg",
      "url": "http://t.co/0mnM8ZL3JB",
      "display_url": "pic.twitter.com/0mnM8ZL3JB",
      "expanded_url": "https://twitter.com/BringBoysBack/status/478071998648
      "type": "photo",
      "sizes": {
        "thumb": {
          "w": 150,
          "h": 150,
          "resize": "crop"
        },
        "large": {
          "w": 500,
          "h": 539,
          "resize": "fit"
        },
        "medium": {
          "w": 500,
          "h": 539,
          "resize": "fit"
        },
        "small": {
          "w": 500,
          "h": 539,
          "resize": "fit"
        }
      }
    }
  ]
},
"extended_entities": {
  "media": [
    {
      "id": 478071995481870336,
      "id_str": "478071995481870336",
      "indices": [
        108,
        130
      ],
      "media_url": "http://pbs.twimg.com/media/BqJz8mvCUAAV2v1.jpg",
      "media_url_https": "https://pbs.twimg.com/media/BqJz8mvCUAAV2v1.jpg",
      "url": "http://t.co/0mnM8ZL3JB",
      "display_url": "pic.twitter.com/0mnM8ZL3JB",
      "expanded_url": "https://twitter.com/BringBoysBack/status/478071998648954
      "type": "photo",
      "sizes": {
        "thumb": {
          "w": 150,
          "h": 150,
          "resize": "crop"
        },
        "large": {
          "w": 500,
          "h": 539,
          "resize": "fit"
        },
        "medium": {
          "w": 500,
          "h": 539,
          "resize": "fit"
        },
        "small": {
          "w": 500,
          "h": 539,
          "resize": "fit"
        }
      }
    }
  ]
},
"source": "<a href=\"http://twitter.com/download/android\" rel=\"nofollow\">Twi
"in_reply_to_status_id": null,
"in_reply_to_status_id_str": null,
"in_reply_to_user_id": null,
"in_reply_to_user_id_str": null,
"in_reply_to_screen_name": null,
"user": {
  "id": 2610686069,
  "id_str": "2610686069",
  "name": "#TheyKilledOurBoys",
  "screen_name": "BringBoysBack",
  "location": "",
  "description": "#BringBackOurBoys",
  "url": null,
  "entities": {
    "description": {
      "urls": [

      ]
    }
  },
```



```
      },
      "protected": false,
      "followers_count": 177,
      "friends_count": 45,
      "listed_count": 1,
      "created_at": "Sun Jun 15 06:42:48 +0000 2014",
      "favourites_count": 33,
      "utc_offset": null,
      "time_zone": null,
      "geo_enabled": false,
      "verified": false,
      "statuses_count": 191,
      "lang": "en",
      "contributors_enabled": false,
      "is_translator": false,
      "is_translation_enabled": false,
      "profile_background_color": "C0DEED",
      "profile_background_image_url": "http://abs.twimg.com/images/themes/th
      "profile_background_image_url_https": "https://abs.twimg.com/images/th
      "profile_background_tile": false,
      "profile_image_url": "http://pbs.twimg.com/profile_images/479473857984
      "profile_image_url_https": "https://pbs.twimg.com/profile_images/47947
      "profile_banner_url": "https://pbs.twimg.com/profile_banners/261068606
      "profile_link_color": "1DA1F2",
      "profile_sidebar_border_color": "C0DEED",
      "profile_sidebar_fill_color": "DDEEF6",
      "profile_text_color": "333333",
      "profile_use_background_image": true,
      "has_extended_profile": false,
      "default_profile": true,
      "default_profile_image": false,
      "following": false,
      "follow_request_sent": false,
      "notifications": false,
      "translator_type": "none"
    },
    "geo": null,
    "coordinates": null,
    "place": null,
    "contributors": null,
    "is_quote_status": false,
    "retweet_count": 3,
    "favorite_count": 1,
    "favorited": false,
    "retweeted": false,
    "possibly_sensitive": false,
    "lang": "en"
  },
  "is_quote_status": false,
  "retweet_count": 3,
  "favorite_count": 0,
  "favorited": false,
  "retweeted": false,
  "lang": "en"
}
```

Figure 22: JSON (retweeted tweet) not reshaped structure



```
{
  "created_at": "Sun Jun 15 20:43:18 +0000 2014",
  "tweet_id": 478276585112350720,
  "tweet_id_str": "478276585112350720",
  "text": "RT @BringBoysBack: @BarackObama We ask for your swift support &amp
  "truncated": false,
  "n_hashtags": 1,
  "lang": "en",
  "retweet_count": 13,
  "retweeted": true,
  "user_id": 496480519,
  "n_urls": 0,
  "n_followers_tweeter": 10,
  "n_statuses_tweeter": 90,
  "n_friends_tweeter": 32,
  "n_favourites_tweeter": 92,
  "time_zone": "Jerusalem",
  "following": false,
  "hash1_length": 16,
  "hash1_text": "BringBackOurBoys",
  "hash2_length": null,
  "hash2_text": null,
  "hash3_length": null,
  "hash3_text": null,
  "hash1_place": 118,
  "text_length": 144,
  "n_user_mentions": 2,
  "user_mentions1_place_indices": 3,
  "user_mention_1": "BringBoysBack",
  "user_mention_2": "BarackObama",
  "user_mention_3": null,
  "url_1_display": null,
  "url_2_display": null,
  "url1_place": null,
  "url_1_length": null,
  "url_2_length": null,
  "is_quote_status": false,
  "in_reply_to_status_id": null,
  "in_reply_to_user_id": null,
  "in_reply_to_screen_name": null,
  "user_name": "adi bohbot",
  "user_screen_name": "adi_bohbot",
  "user_location": "Israel",
  "user_description": "",
  "original_followers_count": 177,
  "original_name": "#TheyKilledOurBoys",
  "original_name_id": 2610686069,
  "original_text": "@BarackObama We ask for your swift support &amp; severe c
  "original_created_at": "Sun Jun 15 14:39:54 +0000 2014",
  "original_id_tweet": 478185131199721472,
  "original_description": "#BringBackOurBoys",
  "original_friends_count": 45,
  "original_listed_count": 1,
  "original_favourites_count_tweet": 3,
  "original_statusses_count": null,
  "original_user_time_zone": null,
  "original_protected_user": false,
  "original_user_created_at": "Sun Jun 15 06:42:48 +0000 2014",
  "original_retweet_count": 13,
  "original_url_tweet": "http://t.co/XDB8SO1gPf",

  "original_n_urls": 0,
  "original_n_hastags": 1,
  "original_n_media": 1,
  "original_n_symbols": 0,
  "original_n_user_mentions": 1
}
```

Figure 23: JSON (retweeted tweet) reshaped structure

Attachment 2: example of a flattened JSON for a tweet for which the usage in our model is indicated

Usage	Feature in flattened JSON
Used as feature in final model	"text": "RT @BringBoysBack: @BarackObama We ask…", "n_hashtags": 1, "user_listed_count": 0, "n_urls": 0, "n_followers_tweeter": 10, "n_media": null, "n_statuses_tweeter": 90, "n_friends_tweeter": 32, "n_favourites_tweeter": 92, "n_user_mentions": 2, "n_symbols": 0, "original_followers_count": 177, "original_text": "@BarackObama We ask …", "original_friends_count": 45, "original_listed_count": 1, "original_favourites_count_user": 33, "original_n_urls": 0, "original_n_hastags": 1, "original_n_media": 1, "original_n_symbols": 0, "original_n_user_mentions": 1, "original_statuses_count": 191
Tested as feature but not included in final model	"n_favorite_count": 0, "hash1_length": 16, "hash2_length": null, "hash3_length": null, "hash1_place": 118, "text_length": 144, "user_mentions1_place_indices": 3, "user_description": ""

	"original_description": "#BringBackOurBoys", "original_favourites_count_tweet": 3
Dependent variable	"retweet_count": 13, "retweeted": true, "original_retweet_count": 13
Used for technical reasons (sorting, identifier, grouping)	"tweet_id": 478276585112350720, "lang": "en", "user_id": 496480519, "original_name_id": 2610686069, "original_id_tweet": 478185131199721472, "original_user_id": 2610686069
Not used	"created_at": "Sun Jun 15 20:43:18 +0000 2014", "tweet_id_str": "478276585112350720", "truncated": false, "time_zone": "Jerusalem", "following": false, "hash1_text": "BringBackOurBoys", "hash2_text": null, "hash3_text": null, "user_mention_1": "BringBoysBack", "user_mention_2": "BarackObama", "user_mention_3": null, "url_1_display": null, "url_2_display": null, "url1_place": null, "url_1_length": null, "url_2_length": null, "is_quote_status": false, "in_reply_to_status_id": null, "in_reply_to_user_id": null, "in_reply_to_screen_name": null, "user_name": "adi bohbot", "user_screen_name": "adi_bohbot",

	"user_location": "Israel", "original_name": "#TheyKilledOurBoys", "original_created_at": "Sun Jun 15 14:39:54 +0000 2014", "original_user_time_zone": null, "original_user_created_at": "Sun Jun 15 06:42:48 +0000 2014", "original_protected_user": false, "original_url_tweet": "http://t.co/XDB8SO1gPf"

Attachment 3: Program to download and flatten tweets in JSON format

```
import tweepy #https://github.com/tweepy/tweepy

import csv

import json
import random
import time
import csv
from tweepy import OAuthHandler


#Twitter API credentials

consumer_key = '***'
consumer_secret = '***'
access_key = '***'
access_secret = '***'


auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)

list1=['jolskiii']

api = tweepy.API(auth)
countmax = 200
totallines =10
retweeted =5
notretweeted =5


#api = tweepy.API(auth)
#countmax = 200
#totallines =60000
#retweeted =30000
#notretweeted =30000

def loop_over_user_names():
    #file1 = open("C://temp//dataset//5000_balanced.json", "a")
    file1 = open("C://temp//ww.json", "a")
    #file1.write("[")


    n = 0
    r=0
    while (n < notretweeted or r <retweeted):
        rand = random.randint(100000000, 999999999)
        try:
            # orriginally count was 200
            count = 0
            print(n,r)
            time.sleep(1)
            new_tweets = api.user_timeline(user_id=rand, count=countmax)
            print(new_tweets)
            while count < countmax:
                try:
                    if (new_tweets[count]._json["lang"] == "en"):
                        toFile = json.dumps(new_tweets[count]._json)

                        if (new_tweets[count]._json["retweet_count"] == 0):
                            if (n < notretweeted):
                                file1.write(toFile + "\n")
                                n = n + 1
                        else:
                            if (r < retweeted):
                                file1.write(toFile + "\n")
                                r = r + 1
                    count += 1

                except Exception as e:
                    print(str(e))
                    count +=1
                    break

        except:
            print("skipping")


    #file1.write("{}]")

    file1.close()



loop_over_user_names()









import tweepy
import json
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener


#consumer key, consumer secret, access token, access secret.
consumer_key = '***'
consumer_secret = '***'
access_key = '***'
access_secret = '***'



auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)



def hashtaglength(hashtags,i):
    try:
        return len(hashtags[i]["text"])

    except:
        return None

def hashtagname(hashtags,i):
    try:
        return (hashtags[i]["text"])

    except:
        return None

def hashtagplace(hashtags):
    try:
        return (hashtags[0]["indices"][0] )

    except:
        return None

def usermentionplace(hashtags):
    try:
        return (hashtags[0]["indices"][0])

    except:
        return None

def usermentionname(hashtags,i):
    try:
        return (hashtags[i]["screen_name"])

    except:
        return None

def url(hashtags,i):
    try:
        return (hashtags[i]["display_url"])

    except:
        return None

def urlplace(hashtags):
    try:
        return (hashtags[0]["indices"][0] )

    except:
        return None

def urllength(hashtags,i):
    try:
        return len(hashtags[i]["url"])

    except:
        return None

def symbols(hashtags):
    try:
        return (hashtags[0]["text"])

    except:
        return None
```

```
def retweeted(data):
    if (data["retweet_count"]>0):
        return True
    else:
        return False

def original_followers_count(data):
    try:
        return data["retweeted_status"]["user"]["followers_count"]
    except :
        return None



def original_name(data):
    try:
        return data["retweeted_status"]["user"]["name"]
    except:
        return None

def original_name_id(data):
    try:
        return data["retweeted_status"]["user"]["id"]
    except:
        return None

def original_text(data):
    try:
        return data["retweeted_status"]["text"]

    except:
        return None


def original_created_at(data):
    try:
        return data["retweeted_status"]["created_at"]

    except:
        return None

def original_id_tweet(data):
    try:
        return data["retweeted_status"]["id"]

    except:
        return None


def original_description(data):
    try:
        return data["retweeted_status"]["user"]["description"]

    except:
        return None

def original_id_tweet(data):
    try:
        return data["retweeted_status"]["id"]
    except:
        return None


def original_friends_count(data):
    try:
        return data["retweeted_status"]["user"]["friends_count"]

    except:
        return None

def original_listed_count(data):
    try:
        return data["retweeted_status"]["user"]["listed_count"]

    except:
        return None

def original_favourites_count_tweet(data):
    try:
        return data["retweeted_status"]["favorite_count"]

    except:
        return None

def original_favourites_count_user(data):
    try:
        return data["retweeted_status"]["user"]["favourites_count"]

    except:
        return None

def original_statuses_count(data):
    try:
        return data["retweeted_status"]["user"]["statuses_count"]

    except:
        return None

def original_protected_user(data):
    try:
        return data["retweeted_status"]["user"]["protected"]

    except:
        return None

def original_user_created_at(data):
    try:
        return data["retweeted_status"]["user"]["created_at"]

    except:
        return None

def original_user_time_zone(data):
    try:
        return data["retweeted_status"]["user"]["time_zone"]

    except:
        return None
```

```
def orignal_retweet_count(data):
    try:
        return data["retweeted_status"]["retweet_count"]

    except:
        return None

def original_url_tweet(data):
    try:
        return
data["retweeted_status"]["extended_entities"]["media"][0]["url"]

    except:
        return None

def original_type_extended_entities_tweet(data):
    try:
        return data["retweeted_status"]["entities"]["media"][0]["type"]

    except:
        return None

def original_n_urls(data):
    try:
        return len(data["retweeted_status"]["entities"]["urls"])

    except:
        return None
def original_n_hashtags(data):
    try:
        return len(data["retweeted_status"]["entities"]["hashtags"])

    except:
        return None
def original_n_user_mentions(data):
    try:
        return len(data["retweeted_status"]["entities"]["user_mentions"])

    except:
        return None
def original_n_media(data):
    try:
        return len(data["retweeted_status"]["entities"]["media"])

    except:
        return None
def original_n_symbols(data):
    try:
        return len(data["retweeted_status"]["entities"]["symbols"])

    except:
        return None

def n_media(data):
    try:
        return len(data["entities"]["media"])

    except:
        return None

def original_user_id(data):
    try:
        return data["retweeted_status"]["user"]["id"]

    except:
        return None



def make_json1(data):
    n = 0
    q = 0
    r = 0
    jsontest = {"created_at": data["created_at"], "tweet_id": data["id"],
"tweet_id_str": data["id_str"] , "text": data["text"], "truncated":
data["truncated"],
                 "n_hashtags": len(data["entities"]["hashtags"]),
'n_hashtags': len(data["entities"]["hashtags"]),
                "lang": data["lang"], 'retweet_count':
data["retweet_count"], 'retweeted': retweeted(data),
                'user_id': data["user"]["id"], "user_listed_count":
data["user"]["listed_count"],
                'n_urls': len(data["entities"]["urls"]),
                'n_followers_tweeter':
data["user"]["followers_count"],'n_media': n_media(data),
                'n_statuses_tweeter': data["user"]["statuses_count"],
                'n_friends_tweeter': data["user"]["friends_count"],
'n_favorite_count': data["favorite_count"],
                'n_favourites_tweeter': data["user"]["favourites_count"],
'time_zone': data["user"]["time_zone"],
                'time_zone': data["user"]["time_zone"], 'following':
data["user"]["following"],

'hash1_length':hashtaglength(data["entities"]["hashtags"],0),'hash1_text':h
ashtagname(data["entities"]["hashtags"],0),
                'hash2_length': hashtaglength(data["entities"]["hashtags"],
1), 'hash2_text': hashtagname(data["entities"]["hashtags"], 1),
                'hash3_length': hashtaglength(data["entities"]["hashtags"],
2), 'hash3_text': hashtagname(data["entities"]["hashtags"], 2),
                'hash1_place': hashtagplace(data["entities"]["hashtags"]),
"text_length": len(data["text"]),
                'n_user_mentions': len(data["entities"]["user_mentions"]),
'user_mentions1_place_indices':
usermentionplace(data["entities"]["user_mentions"]),
                'user_mention_1':
usermentionname(data["entities"]["user_mentions"],0), 'user_mention_2':
usermentionname(data["entities"]["user_mentions"],1),
                'user_mention_3':
usermentionname(data["entities"]["user_mentions"], 2), 'n_urls':
len(data["entities"]["urls"]),
                'url_1_display': url(data["entities"]["urls"],0),
'url_2_display': url(data["entities"]["urls"],1), 'url1_place':
urlplace(data["entities"]["urls"]),
                'url_1_length': urllength(data["entities"]["urls"], 0),
'url_2_length': urllength(data["entities"]["urls"], 1), 'n_urls':
len(data["entities"]["urls"]),
                'is_quote_status': data["is_quote_status"],
'in_reply_to_status_id' : data["in_reply_to_status_id"],
                'in_reply_to_user_id': data["in_reply_to_user_id"],
'in_reply_to_screen_name': data["in_reply_to_screen_name"],
'user_name':data["user"]["name"],
'user_screen_name':data["user"]["screen_name"],'user_location':data["user"]
["location"], 'user_description':data["user"]["description"], 'n_symbols':
len(data["entities"]["symbols"]),

                #'rr': data.get('retweeted_status',
'')['user']['friends_count']

                'original_followers_count':
original_followers_count(data),'original_name': original_name(data),
'original_name_id': original_name_id(data),
                'original_text': original_text(data),
'original_created_at': original_created_at(data), 'original_id_tweet':
original_id_tweet(data),
                'original_description': original_description(data),
'original_friends_count': original_friends_count(data),
'original_listed_count': original_listed_count(data),
                'original_favourites_count_tweet':
original_favourites_count_tweet(data),'original_favourites_count_user':
original_favourites_count_user(data) , 'original_user_time_zone':
original_user_time_zone(data),
                'original_user_created_at': original_user_created_at(data),
'original_retweet_count':
orignal_retweet_count(data),'original_protected_user':
original_protected_user(data),

                'original_url_tweet': original_url_tweet(data),
'original_n_urls': original_n_urls(data),'original_n_hastags':
original_n_hashtags(data),'original_n_media': original_n_media(data),
                'original_n_symbols': original_n_symbols(data),
'original_n_user_mentions': original_n_user_mentions(data),
'original_statuses_count': original_statuses_count(data),
                'original_user_id': original_user_id(data)


        #,'original_friends_count': original_friends_count(data),
'original_listed_count': original_listed_count(data)
        #'original_protected_user': original_protected_user(data),

                }

    return jsontest


#'text_symbols': symbols(data["entities"]["symbols"])
    # file1.write("{}]")
#Bij runnen file1 hashtag wegnemen
file1 = open("C://temp//jsonplat// 560000_balanced_finall.json", "w")
#file1 = open("C://temp//jsonplat//test.json", "w")

with open("C://temp//ww.json","r") as f:

    for line in f:
        data = json.loads(line)
        jsont= make_json1(data)


        file1.write(json.dumps(jsont) + "\n")

file1.close()
```

Attachment 4: Data analytics in Databricks PySpark

databricks Twitter_test

Predicting which tweets will be retweeted most

Table of content

4.3 Results

1. Data preparation

Read in flattened JSON-file:

- Json-file is produced in pythonscript and stored in the filestore of Databricks
- Each row represent one tweet and all the corresponding features
- We have a total of 560 000 tweets and each tweet has 70 features before cleaning

```
testJsonData =
sqlContext.read.json("/FileStore/tables/560000_balanced_finall.json")

display(testJsonData)
```

created_at	following	hash1_length	hash1_place	hash1_text	hash2_
Fri Apr 29 04:09:42 +0000 2011	false	null	null	null	null
Fri Apr 29 01:13:52 +0000 2011	false	5	42	Quote	null
Thu Apr 28 22:15:30 +0000 2011	false	null	null	null	null
Sat Mar 17	false	null	null	null	null

Showing the first 1000 rows.

```
result= testJsonData
#display(result)
```

Describing basetable:

- The describe function shows the range of the features.
- It also shows the count on the not null values per feature.

```
display(result.describe())
```

summary	created_at	hash1_length	hash1_place	hash1_text
count	560000	118916	118916	118916
mean	null	9.570739009048404	55.528675703858184	2.0
stddev	null	4.711061935222937	35.22245249532093	1.511857892036909
min	Fri Apr 01 00:05:05 +0000 2016	1	0	0000ff
max	Wed Sep 30 23:59:49 +0000 2015	64	152	T M1D N

Data exploration

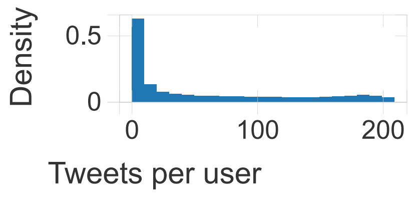
```
## Quick df facts
df_size = result.count()
num_unique_tweets = result.select('tweet_id').distinct().count()
num_unique_users = result.select('user_id').distinct().count()
num_unique_users_reteweeted =
result.select('original_name_id').distinct().count()

print("Number of records: {}".format(df_size))
print("Number of unique tweets: {}".format(num_unique_tweets))
print("Number of unique users: {}".format(num_unique_users))
print("Number of unique users retweeted:
{}".format(num_unique_users_reteweeted))
```

```
Number of records: 560000
Number of unique tweets: 560000
Number of unique users: 17730
Number of unique users retweeted: 93999
```

```
# User tweet frequency distribution
user_tweet_count_df = result.groupBy("user_id").count()

# Retrieve max user tweet frequency
max_count = user_tweet_count_df.agg({"count": "max"}).collect()[0][0]
```

```
user_tweet_count_df = user_tweet_count_df.withColumnRenamed("count", "Tweets
per user")
```

```
# 80% of the tweeters have a frequency of tweets of 50 or less
display(user_tweet_count_df)
```



Cleaning basetable

- Transform features in correct form
- Create new features
- Delete not used features

```
#transform boolean to binary (retweeted)

result = result.withColumn("retweeted1", (result.retweeted ==
'true').astype('int'))
result = result.withColumn("retweeted_boolean",  result["retweeted"])
result = result.withColumn("retweeted",  result["retweeted1"])
result = result.drop('retweeted1')
```

```
result = result.withColumn("retweeted_dummy", (result.retweeted_boolean ==
'true').astype('int'))
#result = result.drop('following_dummy')
#display(result)
```

```
data=result
```

2. Create basetable

New features

- Check if the feature has an original tweeter
- YES -> put the value in the new feature
- NO -> put the value of the new tweeter in the new feature
- No difference between non retweeted tweets and retweeted tweets in the form of number of variables

```
from pyspark.sql.functions import *

data=
data.withColumn('n_followers_tweeter_new',when(data.retweeted_dummy<=0,data.
n_followers_tweeter).otherwise(data.original_followers_count))
#data= data.drop('n_followers_tweeter')
data=
data.withColumn('n_favourites_tweeter_new',when(data.retweeted_dummy<=0,data
.n_favourites_tweeter).otherwise(data.original_favourites_count_user))
#data= data.drop('n_followers_tweeter')
data=
data.withColumn('n_statuses_tweeter_new',when(data.retweeted_dummy<=0,data.n
_statuses_tweeter).otherwise(data.original_statuses_count))
#data= data.drop('n_followers_tweeter')
data=
data.withColumn('retwteet_count_new',when(data.retweeted_dummy<=0,data.retwe
et_count).otherwise(data.original_retweet_count))
#data= data.drop('n_followers_tweeter')
data=
data.withColumn('n_user_mentions_new',when(data.retweeted_dummy<=0,data.n_us
er_mentions).otherwise(data.original_n_user_mentions))
#data= data.drop('n_followers_tweeter')
data=
data.withColumn('n_urls_new',when(data.retweeted_dummy<=0,data.n_urls).other
wise(data.original_n_urls))
#data= data.drop('n_followers_tweeter')
data=
data.withColumn('n_symbols_new',when(data.retweeted_dummy<=0,data.n_symbols)
.otherwise(data.original_n_symbols))
#data= data.drop('n_followers_tweeter')
data=
data.withColumn('n_media_new',when(data.retweeted_dummy<=0,data.n_media).oth
erwise(data.original_n_media))
#data= data.drop('n_followers_tweeter')


data=
data.withColumn('n_hashtags_new',when(data.retweeted_dummy<=0,data.n_hashtag
s).otherwise(data.original_n_hastags))
#data= data.drop('n_followers_tweeter')
data=
data.withColumn('n_listed_new',when(data.retweeted_dummy<=0,data.user_listed
_count).otherwise(data.original_listed_count))
#data= data.drop('n_followers_tweeter')
data=
data.withColumn('n_friends_count_new',when(data.retweeted_dummy<=0,data.n_fr
iends_tweeter).otherwise(data.original_friends_count))
#data= data.drop('n_followers_tweeter')
data=data.withColumn('n_favourites_tweet_new',when(data.retweeted_dummy<=0,d
ata.n_favorite_count).otherwise(data.original_favourites_count_tweet))
```

```
#data= data.drop('n_followers_tweeter')
data=
data.withColumn('text_new',when(data.retweeted_dummy<=0,data.text).otherwise
(data.original_text))
#data= data.drop('n_followers_tweeter')
#data=
data.withColumn('n_followers_tweeter_new',when(data.retweeted_dummy<=0,data.
n_followers_tweeter).otherwise(data.original_n_symbols))
#data= data.drop('n_followers_tweeter')
#data=
data.withColumn('n_favourites_tweeter_new',when(data.retweeted_dummy<=0,data
.n_favourites_tweeter).otherwise(data.original_n_media))

display(data)
```

created_at	following	hash1_length	hash1_place	hash1_text	hash2_
Fri Apr 29 04:09:42 +0000 2011	false	null	null	null	null
Fri Apr 29 01:13:52 +0000 2011	false	5	42	Quote	null
Thu Apr 28 22:15:30 +0000 2011	false	null	null	null	null
Sat Mar 17	false	null	null	null	null

Showing the first 1000 rows.

An example of the basetable (without extra self added features (positive and negative count))

```
example_basetable = data[['retweeted_dummy',
'n_favourites_tweeter_new','n_followers_tweeter_new','n_friends_count_new',
'n_hashtags_new',   'n_urls_new','n_user_mentions_new',
'n_listed_new','n_symbols_new','n_media_new',
'n_statuses_tweeter_new','retweet_count']]
display(example_basetable)
```

retweeted_dummy	n_favourites_tweeter_new	n_followers_tweeter_new	n_fri
0	0	0	0

0	0	0	0
0	0	0	0
0	0	5	18
0	0	5	18
0	0	5	18
1	22	55888	2

Showing the first 1000 rows.

```
data = data.na.fill(0)
#display(data)
```

```
result=data
```

Making new variables

```
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.ml import *
from pyspark.ml.clustering import *
from pyspark.ml.classification import *
from pyspark.ml.regression import *
from pyspark.ml.feature import *
from pyspark.ml.evaluation import *
from scipy.stats import wilcoxon
from array import array
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors
from urllib2 import urlopen
from json import loads
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pyspark.sql.functions as f
```

```
test = result
```

```
#Define splitcol function: splits column in two columns
def splitcol(df, col, sep, name1, name2):
  split_col = split(df[col], sep)
  df = df.withColumn(name2, split_col.getItem(1))
  df = df.withColumn(name1, split_col.getItem(0))
  return df
```

```
#Check how much tweets are retweeted (not original creator) (start with RT)
#Remove @Rt and username from original text
test = splitcol(test, 'text', 'RT @', 'not_retweeted_text',
'original_textt')
#display(test)
```

```
test = splitcol(test, 'original_textt', ': ', 'username_original',
'text_retweeted')
#display(test)
```

```
#remove text retweeted
test = test.na.fill('')
#display(test)
```

```
#Put together to merge
test = test.withColumn("text_final", concat(test["not_retweeted_text"],
test["text_retweeted"]))
#display(test)
```

```
test = test.withColumn("dummy_retweet_retweeted", (test.text_retweeted !=
"").astype('int'))
#display(test)
```

```
newdf =
test.join(test.groupBy('dummy_retweet_retweeted').count(),on='dummy_retweet_
retweeted')
#display(newdf)
#232322 van de retweets zijn afkomstig van tweets van andere tweeters. 83%
van de retweets dus .
#280000-232322=47678 zijn zelf geschreven en toch geretweeted
```

```
test = test.withColumn("number_of_words_tweet",
f.size(f.split(f.col("text_final")," ")))
#display(test)
#F.size(F.split(F.col("_1"), " "))).show()
```

Sentiment analysis text

Clean up text

```
result_text=result[['text_final','retweeted']]

#display(result_text)
```

Data preparation text

We apply the following transformation to the input text data:

- Clean strings
- Tokenize (`String -> Array<String>`)
- Remove stop words
- Stem words
- Create bigrams

1. Clean text string

```
from pyspark.sql.functions import col, lower, regexp_replace, split

def clean_text(c):
  c = lower(c)
  c = regexp_replace(c, "^rt ", "")
  c = regexp_replace(c, "(https?\://)\S+", "")
  c = regexp_replace(c, "[^a-zA-Z0-9\\s]", "")
  #c = split(c, "\\s+") tokenization...
  return c

clean_text_df =
result.select(clean_text(col("text_final")).alias("text_final"), "tweet_id")

clean_text_df.printSchema()
display(clean_text_df)
#clean_text_df.show(10)
```

text_final
just chillin watching teen mom cleaning paying bills and decorating
the place i walked to following the time quote
looks bad out there i am truly terrified of falling today
gettin ready 4 rugby lads dungannon comm well get 2 the final p
hart is feelin his ass
just up and i found a cat lyin beside me then it talked then i woke up again and it was sleepin never so
this is football reimagined this is the xfl watch the official announcement live now xfl2020
come on come on come on lets get it on

Showing the first 1000 rows.

1. Tokenizer

```
from pyspark.ml.feature import Tokenizer

tokenizer = Tokenizer(inputCol="text_final", outputCol="vector")
vector_df = tokenizer.transform(clean_text_df).select("vector")

vector_df.printSchema()
vector_df.show(10)
```

```
root
 |-- vector: array (nullable = true)
 |    |-- element: string (containsNull = true)

+--------------------+
|              vector|
+--------------------+
|[just, chillin, w...|
|[the, place, i, w...|
|[looks, bad, out,...|
|[gettin, ready, 4...|
|[hart, is, feelin...|
|[just, up, and, i...|
|[this, is, footba...|
|[come, on, come, ...|
|[retweet, , follo...|
|[me, gust, un, vi...|
+--------------------+
only showing top 10 rows
```

1. Remove stop words

```
from pyspark.ml.feature import StopWordsRemover

# Define a list of stop words or use default list
remover = StopWordsRemover()
stopwords = remover.getStopWords()

# Display default list
stopwords[:100]
```

```
Out[44]:
[u'i',
 u'me',
 u'my',
 u'myself',
 u'we',
 u'our',
 u'ours',
 u'ourselves',
 u'you',
 u'your',
 u'yours',
 u'yourself',
 u'yourselves',
 u'he',
 u'him',
 u'his',
 u'himself',
 u'she',
 u'her',
 u'hers',
```

```
# Specify input/output columns
remover.setInputCol("vector")
remover.setOutputCol("vector_no_stopw")

# Transform existing dataframe with the StopWordsRemover
vector_no_stopw_df = remover.transform(vector_df).select("vector_no_stopw")

# Display
vector_no_stopw_df.printSchema()
vector_no_stopw_df.show()
```

```
root
 |-- vector_no_stopw: array (nullable = true)
 |    |-- element: string (containsNull = true)

+--------------------+
|     vector_no_stopw|
+--------------------+
|[chillin, watchin...|
|[place, walked, f...|
|[looks, bad, trul...|
|[gettin, ready, 4...|
| [hart, feelin, ass]|
|[found, cat, lyin...|
|[football, reimag...|
|[come, come, come...|
|[retweet, , follo...|
|[gust, un, video,...|
|[gust, un, video,...|
|[city, mania, hac...|
|[videos, , , , , ...|
|[society, look, i...|
|[everyone, proble...|
|[hellolalit, hey,...|
|[warehousing, tak...|
|[inspired, americ...|
|[list, americanni...|
|[dcopperfield, al...|
+--------------------+
only showing top 20 rows
```

```
text_dataset_production=vector_no_stopw_df
display(text_dataset_production)
```

vector_no_stopw
▸ ["chillin","watching","teen","mom","cleaning","paying","bills","decorating"]
▸ ["place","walked","following","time","quote"]
▸ ["looks","bad","truly","terrified","falling","today"]
▸ ["gettin","ready","4","rugby","lads","dungannon","comm","well","get","2","final","","p"]
▸ ["hart","feelin","ass"]
▸ ["found","cat","lyin","beside","talked","woke","sleepin","never","sooooooo","scared","life"]
▸ ["football","reimagined","xfl","watch","official","announcement","","live","xfl2020"]
▸ ["come","come","come","lets","get","","todays","raw","superstars","ready","show","ruthless","aggressic
▸ ["retweet","","follow","mikeyacestudios","","4","best","daily","wwe","cartoons","twitter","ff","fridayfeelinc

Showing the first 1000 rows.

Save production data as Spark table for analysis

```
#production_df.write.saveAsTable("text_dataset_production")
#text_dataset_production = spark.sql("SELECT * FROM
text_dataset_production")
#display(text_dataset_production.select("*"))
```

```
positive = sqlContext.read.csv("/FileStore/tables/positive.txt")
negative = sqlContext.read.csv("/FileStore/tables/neg.csv")

display(negative)
#display(txt)
```

_c0
abnormal
abolish
abominable
abominably
abominate
abomination
abort
aborted

Showing the first 1000 rows.

```
negative = negative.withColumnRenamed("_c0", "words")
positive = positive.withColumnRenamed("_c0", "words")
```

```
#positive = txtx.select('words').collect()
negative = negative.select("words").rdd.flatMap(lambda x: x).collect()
positive = positive.select("words").rdd.flatMap(lambda x: x).collect()
```

```
def positive_count(row):
  cnt=0
  print row
  for item in row:
    if item in positive:
      cnt+=1
  return cnt

def negative_count(row):
  cnt=0
  print row
  for item in row:
    if item in negative:
      cnt+=1
  return cnt


pos_udf = udf(positive_count, IntegerType())
neg_udf = udf(negative_count, IntegerType())
```

```
sample3 = text_dataset_production.withColumn('count_positive',
pos_udf(text_dataset_production['vector_no_stopw']))
sample3 = sample3.withColumn('count_negative',
neg_udf(sample3['vector_no_stopw']))
#display(sample3)
```

```
# combine the table with the positve and negative word count with the
original data
# We used the function monotonically_increasing_id() that randomly assign a
number to the respectively row
df11 =  sample3.withColumn("rowId", monotonically_increasing_id())
df22 =  result.withColumn("rowId", monotonically_increasing_id())
newDF = df11.join(df22, df11.rowId == df22.rowId, 'inner').drop(df22.rowId)
#display(newDF)
```

vector_no_stopw
▸ ["great","times","great","times","great","times"]
▸ ["love"]

▸ ["cravemythoughts","emilyemmons3","us","lol"]

Showing the first 997 rows.

```
newDF= newDF.withColumn('count_pos_NEG', newDF.count_positive -
newDF.count_negative)
#display(newDF)
```

vector_no_stopw
▸ ["great","times","great","times","great","times"]
▸ ["love"]
▸ ["cravemythoughts","emilyemmons3","us","lol"]
▸ ["beatty","street","grocery","building"]

Showing the first 995 rows.

```
basetablee = spark.sql("SELECT * FROM basetable_first")
display(basetablee)
```

vector_no_stopw
▸ ["everyone","problem","area","treated","hasansurgery"]
▸ ["earn","money","using","facebook","twitter","etc"]

Showing the first 1000 rows.

3. Classification retweeter and classification original tweeter:

```
from pyspark.ml.linalg import Vectors
from array import array
from pyspark.sql import Row
```

```
display(basetablee.describe())
```

summary	count_positive	count_negative	rowId	created_
count	560000	560000	560000	560000
mean	0.5406946428571429	0.31935178571428574	3.353321848742147E10	null
stddev	0.8057552568702495	0.6487205891167308	2.1861263450107533E10	null
min	0	0	0	Fri Apr 0 00:05:05 +0000 20
max	21	19	68719528052	Wed Sep 23:59:49 +0000 20

```
#final = final.filter(result.n_followers_tweeter_new. isNotNull())
#display(final)
```

Show result

-Classification retweeter

```
aa = basetablee[['retweeted_dummy',
'n_favourites_tweeter','n_followers_tweeter','n_friends_tweeter',
'n_hashtags',   'n_urls','n_user_mentions',
'user_listed_count','n_symbols','n_media','count_positive',
'count_negative', 'n_statuses_tweeter','retweet_count']]
display(aa)
```

retweeted_dummy	n_favourites_tweeter	n_followers_tweeter	n_friends_tweet
1	24	3	624
0	3	459	90
0	10	24	368
1	1761	8649	9503
0	0	340	1739
0	0	340	1739
0	0	340	1739
1	0	32	28

Showing the first 1000 rows.

```
basetable = aa.rdd.map(lambda r : Row( retweeted = r.retweeted_dummy,
                        continuousFeatures = Vectors.dense([
r.n_favourites_tweeter , r.n_followers_tweeter , r.n_friends_tweeter,
r.n_hashtags, r.n_urls,r.n_user_mentions,r.user_listed_count,r.n_symbols,
r.n_media, r.count_positive,r.count_negative, r.n_statuses_tweeter]),
                        categoricalFeatures = Vectors.dense([])
                        )).toDF()
#extra:
r.hash1_length,r.hash2_length,r.hash3_length,r.number_of_words_tweet,r.count
_pos_neg(afgetrokken) (not included in final model)
#extra: r.following_dummy (not included in final model)
basetable.show(20)
```

```
+-------------------+--------------------+---------+
|categoricalFeatures|  continuousFeatures|retweeted|
+-------------------+--------------------+---------+
|                 []|[24.0,3.0,624.0,1...|        1|
|                 []|[3.0,459.0,90.0,0...|        0|
|                 []|[10.0,24.0,368.0,...|        0|
|                 []|[1761.0,8649.0,95...|        1|
|                 []|[0.0,340.0,1739.0...|        0|
|                 []|[0.0,340.0,1739.0...|        0|
|                 []|[0.0,340.0,1739.0...|        0|
|                 []|[0.0,32.0,28.0,0....|        1|
|                 []|[4.0,114.0,379.0,...|        0|
|                 []|[486.0,440.0,360....|        0|
|                 []|[0.0,37.0,166.0,0...|        0|
|                 []|[36.0,165.0,166.0...|        0|
|                 []|[6.0,3.0,77.0,0.0...|        0|
|                 []|[145.0,534.0,196....|        1|
|                 []|[145.0,534.0,196....|        1|
|                 []|[7.0,231.0,300.0,...|        1|
|                 []|[7.0,231.0,300.0,...|        0|
|                 []|[123.0,277.0,20.0...|        0|
|                 []|[2.0,34.0,86.0,2....|        1|
|                 []|[4009.0,489.0,345...|        0|
+-------------------+--------------------+---------+
only showing top 20 rows
```

- Classification original tweeter

```
aa = basetablee[['retweeted_dummy',
'n_favourites_tweeter_new','n_followers_tweeter_new','n_friends_count_new',
'n_hashtags_new',   'n_urls_new','n_user_mentions_new',
'n_listed_new','n_symbols_new','n_media_new','count_positive',
'count_negative', 'n_statuses_tweeter_new','retweet_count']]
display(aa)
```

retweeted_dummy	n_favourites_tweeter_new	n_followers_tweeter_new	n_friends_
1	24	790	254
0	3	459	90
0	10	24	368
1	67	32867	13
0	0	340	1739
0	0	340	1739
0	0	340	1739
1	7936	45472	11513
0	4	114	379

Showing the first 1000 rows.

```
basetable = aa.rdd.map(lambda r : Row( retweeted = r.retweeted_dummy,
                        continuousFeatures = Vectors.dense([
r.n_favourites_tweeter_new , r.n_followers_tweeter_new ,
r.n_friends_count_new,
r.n_hashtags_new,r.n_urls_new,r.n_user_mentions_new,r.n_listed_new,r.n_symbo
ls_new, r.n_media_new, r.count_positive,r.count_negative,
r.n_statuses_tweeter_new]),
                        categoricalFeatures = Vectors.dense([])
                        )).toDF()
#extra: r.count_negative,r.count_positive]
basetable.show(20)
```

```
+------------------+--------------------+---------+
|categoricalFeatures|  continuousFeatures|retweeted|
+------------------+--------------------+---------+
|                []|[24.0,790.0,254.0...|        1|
|                []|[3.0,459.0,90.0,0...|        0|
|                []|[10.0,24.0,368.0,...|        0|
|                []|[67.0,32867.0,13....|        1|
|                []|[0.0,340.0,1739.0...|        0|
|                []|[0.0,340.0,1739.0...|        0|
|                []|[0.0,340.0,1739.0...|        0|
|                []|[7936.0,45472.0,1...|        1|
|                []|[4.0,114.0,379.0,...|        0|
|                []|[486.0,440.0,360....|        0|
|                []|[0.0,37.0,166.0,0...|        0|
|                []|[36.0,165.0,166.0...|        0|
|                []|[6.0,3.0,77.0,0.0...|        0|
|                []|[0.0,0.0,0.0,0.0,...|        1|
|                []|[0.0,0.0,0.0,0.0,...|        1|
|                []|[0.0,0.0,0.0,0.0,...|        1|
|                []|[7.0,231.0,300.0,...|        0|
|                []|[123.0,277.0,20.0...|        0|
|                []|[17795.0,139516.0...|        1|
|                []|[4009.0,489.0,345...|        0|
+------------------+--------------------+---------+
only showing top 20 rows
```

```
''' Split data into trainig and test data'''
(trainingData, testData) = basetable.randomSplit([0.70, 0.30], 1)

#print trainingData.count()
#print testData.count()
```

Machine Learning Pipeline Construction

Now we have our basetable that is ready to be put in our Machine Learning Pipeline. First we need to construct a pipeline.

The pipeline will consist out of 5 stages:

- StringIndexer: A label indexer that maps a string column of labels to an ML column of label indices. So this makes category indices of the strings in a column.
- VectorIndexer: Class for indexing categorical feature columns in a dataset of Vector.

- StandardScaler: Standardizes features by removing the mean and scaling to unit variance using column summary statistics on the samples in the training set.
- VectorAssembler: A feature transformer that merges multiple columns into a vector column.
- The machine learning algorithm that we will be using. We will give an example of Logistic Regression, Decision Trees, Random Forest

```
from pyspark.ml.feature import StringIndexer, VectorIndexer,
VectorAssembler, StandardScaler
from pyspark.ml import Pipeline

# Import for Logistic Regression
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import LogisticRegressionModel

# Import for Decision Tree
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import DecisionTreeClassificationModel

# Import for Random Forest
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import RandomForestClassificationModel
```

- String Indexer

For the String and Vector Indexer it is important that you fit them to the entire data. Not just the training data, because it might be possible that not all different labels are present in the training set. In this case you will get errors when you try to make predictions on the test set because he will encounter labels that he doesnt recognize because they were not in the training set.

```
# What is a StringIndexer? (just to display help info)
#help(StringIndexer)

# String Indexer: the input column is our default_next_month column and our
output will be called label
labelIndexer = StringIndexer(inputCol = "retweeted", outputCol =
"label").fit(basetable)
```

```
## What is a VectorIndexer? (just to display help info)
#help(VectorIndexer)

# Vector Indexer: the input column are the categorical features and they
will be transformed to a column of indexed categorical features
vtrIndexer = VectorIndexer(inputCol = "categoricalFeatures", outputCol=
"indexedCatFeatures").fit(basetable)
```

```
# What is a StandardScaler? (just to display help info)
#help(StandardScaler)
```

```
# Standard Scaler: scales the continuous features to scaled continuous
features
standardScaler = StandardScaler(inputCol="continuousFeatures",
outputCol="scaledContFeatures")
```

```
# What is a VectorAssembler? (just to display help info)
#help(VectorAssembler)

# Vector Assembler: assembles the different vectors of the categorical and
continuous features and outputs them as a feature column
vtrAssembler = VectorAssembler(inputCols= ["indexedCatFeatures",
"scaledContFeatures"], outputCol= "features")
```

The machine learning model:

```
# How to do a Linear Regression? What are the parameters? (just to display
help info)

from pyspark.ml.classification import LogisticRegression
# Logistic Regression
logRegressor = LogisticRegression(labelCol="label", featuresCol="features",
maxIter=100)
```

```
# How to do a Decision Tree Regression? What are the parameters? (just to
display help info)
#help(DecisionTreeRegressor)

# Decision Tree
dtRegressor = DecisionTreeClassifier(labelCol = "label", featuresCol =
"features")
```

```
# How to do a Random Forest Regression? What are the parameters? (just to
display help info)
#help(RandomForestRegressor)

# numTrees 5000 out of memory after 33 min , 2500 trees; training and
fitting last 20 minutes, 500 trees 5 min
# Random Forest Regressor: we define the label and the features columns and
the number of trees we want to build. 500 is a good trade-off between
computation time and error.
# The Random Forest will contain 500 different decision trees.
rfRegressor = RandomForestClassifier(labelCol="label",
featuresCol="features", numTrees=500)
```

```
#Logistic regression: Defining all the stages of the pipeline
lrPipeline = Pipeline(stages = [labelIndexer,vtrIndexer, standardScaler,
vtrAssembler, logRegressor])

# Fitting the pipeline to the training data: This is the execution of all
the stages and the training of the logistic regression Model.
lrModel = lrPipeline.fit(trainingData)
```

```
lrModel.stages
```

```
Out[16]:
[StringIndexer_4affb6492dd6097b1004,
 VectorIndexer_41bf9a3e9e1053f6d719,
 StandardScaler_455bb927110fb7685ddf,
 VectorAssembler_4316943654fe9d38ad4e,
 LogisticRegression_456a81a80ab509bbb33c]
```

```
#Decision tree: Defining all the stages of the pipeline
dtPipeline = Pipeline(stages = [labelIndexer, vtrIndexer, standardScaler,
vtrAssembler, dtRegressor])

# Fitting the pipeline to the training data: This is the execution of all
the stages and the training of the Random Forest Model.
dtModel = dtPipeline.fit(trainingData)
```

```
#Random Forest: Defining all the stages of the pipeline
rfPipeline = Pipeline(stages = [labelIndexer, vtrIndexer, standardScaler,
vtrAssembler, rfRegressor])

# Fitting the pipeline to the training data: This is the execution of all
the stages and the training of the Random Forest Model.
rfModel = rfPipeline.fit(trainingData)
```

```
# Deployment of Logistic Regression: Make predictions on test data.
lrPredictions = lrModel.transform(testData)

# The label denotes what should have been the prediction. The prediction
column is the prediction that was made with the trained machine learning
model
lrPredictions.show(4)
#lrPredictions.printSchema()
```

```
+------------------+--------------------+---------+-----+------------------
+--------------------+----------+--------------------+--------------------+-
---------+
|categoricalFeatures|  continuousFeatures|retweeted|label|indexedCatFeatures
|  scaledContFeatures|  features|       rawPrediction|         probability|p
rediction|
+------------------+--------------------+---------+-----+------------------
+--------------------+----------+--------------------+--------------------+-
---------+
|                 []|[0.0,0.0,0.0,0.0,...|        1|  1.0|                []
|[0.0,0.0,0.0,0.0,...|(12,[],[])|[0.88766247090405...|[0.70840755318084...|
      0.0|
|                 []|[0.0,0.0,0.0,0.0,...|        1|  1.0|                []
|[0.0,0.0,0.0,0.0,...|(12,[],[])|[0.88766247090405...|[0.70840755318084...|
      0.0|
|                 []|[0.0,0.0,0.0,0.0,...|        1|  1.0|                []
|[0.0,0.0,0.0,0.0,...|(12,[],[])|[0.88766247090405...|[0.70840755318084...|
      0.0|
|                 []|[0.0,0.0,0.0,0.0,...|        1|  1.0|                []
|[0.0,0.0,0.0,0.0,...|(12,[],[])|[0.88766247090405...|[0.70840755318084...|
      0.0|
+------------------+--------------------+---------+-----+------------------
+--------------------+----------+--------------------+--------------------+-
---------+
only showing top 4 rows
```

```
# Deployment of Decision Tree: Make predictions on test data.
dtPredictions = dtModel.transform(testData)

# The label denotes what should have been the prediction. The prediction
column is the prediction that was made with the trained machine learning
model
dtPredictions.show(4)
#dtPredictions.printSchema()
```

```
+------------------+--------------------+---------+-----+------------------
+--------------------+----------+-------------+--------------------+--------
--+
|categoricalFeatures|  continuousFeatures|retweeted|label|indexedCatFeatures
|  scaledContFeatures|  features|rawPrediction|         probability|predicti
```

```
on|
+-------------------+--------------------+---------+-----+------------------
+--------------------+----------+-------------+--------------------+--------
--+
|                 []|[0.0,0.0,0.0,0.0,...|        1|  1.0|                []
|[0.0,0.0,0.0,0.0,...|(12,[],[])|[2.0,14078.0]|[1.42045454545454...|
1.0|
|                 []|[0.0,0.0,0.0,0.0,...|        1|  1.0|                []
|[0.0,0.0,0.0,0.0,...|(12,[],[])|[2.0,14078.0]|[1.42045454545454...|
1.0|
|                 []|[0.0,0.0,0.0,0.0,...|        1|  1.0|                []
|[0.0,0.0,0.0,0.0,...|(12,[],[])|[2.0,14078.0]|[1.42045454545454...|
1.0|
|                 []|[0.0,0.0,0.0,0.0,...|        1|  1.0|                []
|[0.0,0.0,0.0,0.0,...|(12,[],[])|[2.0,14078.0]|[1.42045454545454...|
1.0|
+-------------------+--------------------+---------+-----+------------------
+--------------------+----------+-------------+--------------------+--------
--+
only showing top 4 rows
```

```
# Deployment of Random Forest: Make predictions on test data.
rfPredictions = rfModel.transform(testData)

# The label denotes what should have been the prediction. The prediction
column is the prediction that was made with the trained machine learning
model
rfPredictions.show(1000)
#rfPredictions.printSchema()
```

Show result

3. Model Evaluation

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

1. Logistic Regression

```
# Our PipelineModel consists of 5 stages starting of index 0 to index 4.
Each of these index places contains the model of the respective stage. The
more stages you add the larger the pipeline will become, but in this example
it contains 5 stages.
# The machine learning model is on index 4
# Print the coefficients and intercept for logistic regression
print("Coefficients: " + str(lrModel.stages[4].coefficients))
print("Intercept: " + str(lrModel.stages[4].intercept))
```

```
Coefficients: [1.2490986877346895,874.9525560342598,-4.199810970818677,-0.05
646538865809754,-0.42702785624500833,-0.326421570809041,705.8257709006641,0.
022928759948021585,0.31947377573726016,0.11065618779743354,0.019602001505486
147,0.6280888835783143]
Intercept: -0.887662470904
```

```
# With the binary evaluator
lrEvaluator = BinaryClassificationEvaluator()

lrAUC = lrEvaluator.evaluate(lrPredictions, {lrEvaluator.metricName:
"areaUnderROC"})

print "The AUC of the Logistic Regression Model is %f" %(lrAUC)
```

```
The AUC of the Logistic Regression Model is 0.887920
```

```
#Results with 8+1 variables continuousFeatures = Vectors.dense([
r.n_favourites_tweeter,
r.n_followers_tweeter,r.n_friends_tweeter,r.n_hashtags,
r.n_statuses_tweeter]),,r.n_urls,r.n_user_mentions,r.count_pos_neg(afgetrokk
en)]
                        # categoricalFeatures =
Vectors.dense([r.following_dummy]
  #Accuracy of Logistic Regression: 0.743158
#Results with 8+1 variables continuousFeatures = Vectors.dense([
r.n_favourites_tweeter,
r.n_followers_tweeter,r.n_friends_tweeter,r.n_hashtags,
r.n_statuses_tweeter]),,r.n_urls,r.n_user_mentions,r.count_pos_neg(opgeteld)
]
                        # categoricalFeatures =
Vectors.dense([r.following_dummy]
  #Accuracy of Logistic Regression: 0.743158
#Results with 8+1 variables continuousFeatures = Vectors.dense([
r.n_favourites_tweeter,
r.n_followers_tweeter,r.n_friends_tweeter,r.n_hashtags,
r.n_statuses_tweeter]),,r.n_urls,r.n_user_mentions,r.number_of_words_tweet]
                        # categoricalFeatures =
Vectors.dense([r.following_dummy]
  #Accuracy of Logistic Regression: 0.743891
#Results with 7+1 variables continuousFeatures = Vectors.dense([
r.n_favourites_tweete
#r, r.n_followers_tweeter,r.n_friends_tweeter,r.n_hashtags,
r.n_statuses_tweeter]),,r.n_urls,r.n_user_mentions
                        # categoricalFeatures =
Vectors.dense([r.following_dummy])
#Accuracy of Logistic Regression: 0.748672
# Results with 5+1 variables continuousFeatures = Vectors.dense([
r.n_favourites_tweeter,
r.n_followers_tweeter,r.n_friends_tweeter,r.n_hashtags,
r.n_statuses_tweeter]),
                        # categoricalFeatures =
Vectors.dense([r.following_dummy])
#Accuracy of Logistic Regression: 0.601272
#Results with 10+1 variables continuousFeatures = Vectors.dense([
r.n_favourites_tweeter,
r.n_followers_tweeter,r.n_friends_tweeter,r.n_hashtags,
r.n_statuses_tweeter]),,r.n_urls,r.n_user_mentions,r.hash1_length,r.hash2_le
ngth,r.hash3_length]
                        # categoricalFeatures =
Vectors.dense([r.following_dummy])
#Accuracy of Logistic Regression: 0.747654
lrMultiEvaluator =
MulticlassClassificationEvaluator(predictionCol="prediction")
lrAcc = lrMultiEvaluator.evaluate(lrPredictions,
{lrMultiEvaluator.metricName: "accuracy"})
```

```
print "Accuracy of Logistic Regression: %f" %(lrAcc)
print "Test Error of Logistic Regression: %g" % (1.0 - lrAcc)
```

```
Accuracy of Logistic Regression: 0.810059
Test Error of Logistic Regression: 0.189941
```

```
retweetedTweets = lrPredictions.filter(lrPredictions.label == 1)
notretweetedTweets = lrPredictions.filter(lrPredictions.label == 0)

#print lrPredictions.count()
#print notretweetedTweets.count()
#print retweetedTweets.count()


#Results with 7+1 variables
# Accuracy of retweeted tweets: 0.811439
#Accuracy of not retweeted tweets: 0.685824
lrAccDefault = lrMultiEvaluator.evaluate(retweetedTweets,
{lrMultiEvaluator.metricName: "accuracy"})
lrAccNonDefault = lrMultiEvaluator.evaluate(notretweetedTweets,
{lrMultiEvaluator.metricName: "accuracy"})

print "Accuracy of retweeted tweets: %f" %(lrAccDefault)
print "Accuracy of not retweeted tweets: %f" %(lrAccNonDefault)
```

```
Accuracy of retweeted tweets: 0.671972
Accuracy of not retweeted tweets: 0.949389
```

2. Decision Tree

```
print dtModel.stages[4].toDebugString
```

Show result

```
dtModel.stages[4].featureImportances
```

```
Out[29]: SparseVector(12, {0: 0.0271, 1: 0.6547, 2: 0.0008, 5: 0.0002, 6: 0.
0072, 8: 0.0128, 9: 0.0, 10: 0.0, 11: 0.297})
```

```
# With the binary evaluator
dtEvaluator = BinaryClassificationEvaluator()

dtAUC = dtEvaluator.evaluate(dtPredictions, {dtEvaluator.metricName:
"areaUnderROC"})

print "The AUC of the Decision Tree Model is %f" %(dtAUC)
```

```
The AUC of the Decision Tree Model is 0.825964
```

```
#Results of 8+1 variables
#Accuracy of Decision Tree: 0.776932
#Test Error of Decision Tree: 0.223068


#Results of 7+1 variables
#Accuracy of Decision Tree: 0.777301
#Test Error of Decision Tree: 0.222699
dtMultiEvaluator =
MulticlassClassificationEvaluator(predictionCol="prediction")
dtAcc = dtMultiEvaluator.evaluate(dtPredictions,
{dtMultiEvaluator.metricName: "accuracy"})

print "Accuracy of Decision Tree: %f" %(dtAcc)
print "Test Error of Decision Tree: %g" % (1.0 - dtAcc)
```

```
Accuracy of Decision Tree: 0.910215
Test Error of Decision Tree: 0.0897845
```

```
retweetedTweets = dtPredictions.filter(dtPredictions.label == 1)
notretweetedTweets = dtPredictions.filter(dtPredictions.label == 0)

#print lrPredictions.count()
#print notretweetedTweets.count()
#print retweetedTweets.count()



#Results of 7+1 variables
#Accuracy of retweeted tweets: 0.821603
#Accuracy of not retweeted tweets: 0.732942
dtAccDefault = dtMultiEvaluator.evaluate(retweetedTweets,
{dtMultiEvaluator.metricName: "accuracy"})
dtAccNonDefault = dtMultiEvaluator.evaluate(notretweetedTweets,
{dtMultiEvaluator.metricName: "accuracy"})

print "Accuracy of retweeted tweets: %f" %(dtAccDefault)
print "Accuracy of not retweeted tweets: %f" %(dtAccNonDefault)
```

```
Accuracy of retweeted tweets: 0.890977
Accuracy of not retweeted tweets: 0.929627
```

3. Random Forest

```
print rfModel.stages[4].toDebugString
```

Show result

To get an idea of how important the different features were in predicting the outcome, you can print the feature importances. The higher the number the more important the feature was.

```
rfModel.stages[4].featureImportances
```

```
Out[35]: SparseVector(12, {0: 0.0597, 1: 0.3996, 2: 0.0901, 3: 0.0021, 4: 0.
008, 5: 0.0069, 6: 0.2037, 7: 0.0, 8: 0.0142, 9: 0.0, 10: 0.0, 11: 0.2157})
```

```
# With the binary evaluator
rfEvaluator = BinaryClassificationEvaluator()

rfAUC = rfEvaluator.evaluate(rfPredictions, {rfEvaluator.metricName:
"areaUnderROC"})

print "The AUC of the Random Forest Model is %f" %(rfAUC)
```

```
The AUC of the Random Forest Model is 0.969339
```

```
#number of trees =500 -> ACC of 0.778033
#number of trees =2000 -> ACC of 0.778170?
rfMultiEvaluator =
MulticlassClassificationEvaluator(predictionCol="prediction")
rfAcc = rfMultiEvaluator.evaluate(rfPredictions,
{rfMultiEvaluator.metricName: "accuracy"})

print "Accuracy of Random Forest: %f" %(rfAcc)
print "Test Error of Random Forest: %g" % (1.0 - rfAcc)
```

```
Accuracy of Random Forest: 0.916401
Test Error of Random Forest: 0.0835995
```

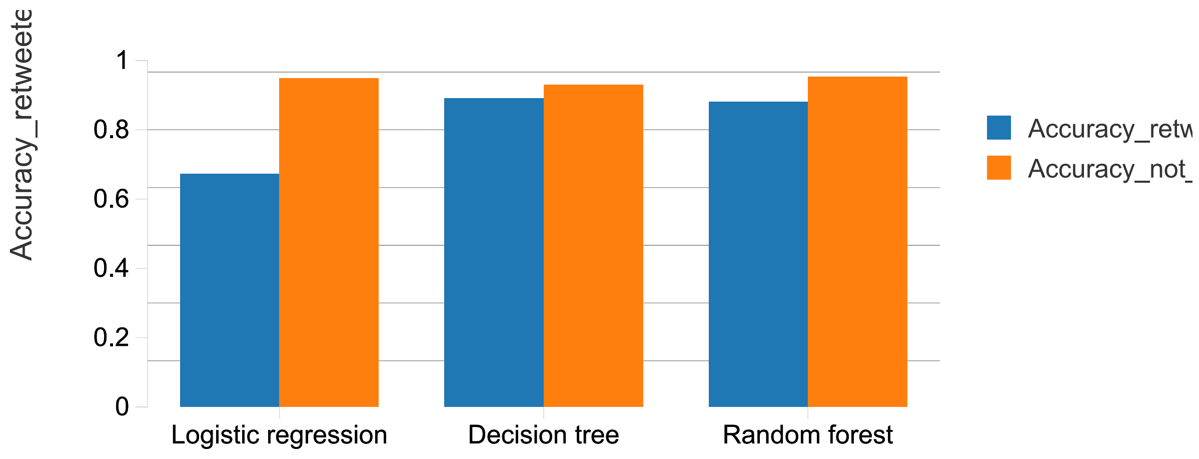
```
retweetedTweets = rfPredictions.filter(rfPredictions.label == 1)
notretweetedTweets = rfPredictions.filter(rfPredictions.label == 0)

#print rfPredictions.count()
#print notretweetedTweets.count()
#print retweetedTweets.count()
```

```
# 500 number of trees
# retweeted tweets: 0.827649
# not retweeted tweets: 0.728354

# 2500 number of trees
# retweeted tweets: 0.826280
# not retweeted tweets: 0.729998

rfAccDefault = rfMultiEvaluator.evaluate(retweetedTweets,
{rfMultiEvaluator.metricName: "accuracy"})
rfAccNonDefault = rfMultiEvaluator.evaluate(notretweetedTweets,
{rfMultiEvaluator.metricName: "accuracy"})

print "Accuracy of retweeted tweets: %f" %(rfAccDefault)
print "Accuracy of not retweeted tweets: %f" %(rfAccNonDefault)
```

```
Accuracy of retweeted tweets: 0.880225
Accuracy of not retweeted tweets: 0.952902
```

```
#create dataframe overview results
# import pyspark class Row from module sql
from pyspark.sql import *
import pyspark.sql.functions as func

# Create Example Data - Model, Performance, accuracy

# Create the Departments
ModelLOGR = 'Logistic regression'
ModelDT = 'Decision tree'
ModelRF = 'Random forest'


performanceLOGR = lrAUC
performanceDT = dtAUC
performanceRF = rfAUC

TOTaccuracyLOGR = lrAcc
TOTaccuracyDT = dtAcc
TOTaccuracyRF = rfAcc

accuracyNotRetLOGR = lrAccNonDefault
accuracyNotRetDT = dtAccNonDefault
accuracyNotRetRF = rfAccNonDefault

accuracyRetLOGR = lrAccDefault
accuracyRetDT = dtAccDefault
accuracyRetRF = rfAccDefault


ModelwithresultsLOGR = Row(Model=ModelLOGR, Performance=performanceLOGR,
Total_accuracy=TOTaccuracyLOGR,
                               Accuracy_not_retweeted=accuracyNotRetLOGR,
Accuracy_retweeted=accuracyRetLOGR)
ModelwithresultsDT = Row(Model=ModelDT, Performance=performanceDT,
Total_accuracy=TOTaccuracyDT,
                               Accuracy_not_retweeted=accuracyNotRetDT,
Accuracy_retweeted=accuracyRetDT)
ModelwithresultsRF = Row(Model=ModelRF, Performance=performanceRF,
Total_accuracy=TOTaccuracyRF,
                               Accuracy_not_retweeted=accuracyNotRetRF,
Accuracy_retweeted=accuracyRetRF)




ModelwithresultsSeq1 = [ModelwithresultsLOGR, ModelwithresultsDT,
ModelwithresultsRF]

df = sqlContext.createDataFrame(ModelwithresultsSeq1)
```

```
#df2 = df.withColumn("Model", df1.Model)

df2 = df.withColumn("Accuracy_retweeted",
func.round(df["Accuracy_retweeted"], 3))
df2 = df2.withColumn("Accuracy_not_retweeted",
func.round(df["Accuracy_not_retweeted"], 3))
df2 = df2.withColumn("Performance", func.round(df["Performance"], 3))
df2 = df2.withColumn("Total_accuracy", func.round(df["Total_accuracy"], 3))

df3=df2[['Model','Performance','Total_accuracy','Accuracy_not_retweeted','Ac
curacy_retweeted']]

display(df3)
```



Correlation matrix

```
aa = basetablee[['retweeted_dummy',
'n_favourites_tweeter_new','n_followers_tweeter_new','n_friends_count_new',
'n_hashtags_new',   'n_urls_new','n_user_mentions_new',
'n_listed_new','n_symbols_new','n_media_new','count_positive',
'count_negative', 'n_statuses_tweeter_new','retweet_count']]
```

```
features = aa.rdd.map(lambda row: row[0:])

from pyspark.mllib.stat import Statistics

corr_mat=Statistics.corr(features, method="pearson")
print corr_mat
```

```
[[ 1.00000000e+00  1.88070964e-01  1.55125596e-01  1.14687640e-01
   1.72495133e-02 -5.88822443e-02 -1.28777171e-01  1.11794910e-01
   8.13467870e-03  2.82361102e-01  5.48576318e-02  5.15052711e-02
   2.88728695e-01  7.92549094e-02]
 [ 1.88070964e-01  1.00000000e+00  3.15324589e-05  1.20886932e-01
  -7.83156450e-03  1.60515124e-02 -2.46204666e-02  1.94714495e-03
  -2.73672769e-04  1.06218087e-01  3.50167679e-03  3.49298275e-02
   2.28068000e-01  4.42246956e-02]
 [ 1.55125596e-01  3.15324589e-05  1.00000000e+00  2.84132758e-01
   3.80716923e-03 -1.77159977e-03 -2.40452583e-02  8.36349546e-01
  -1.88031458e-03  4.99063185e-02  3.41546012e-02 -1.81587123e-04
   9.10626709e-02  1.87128641e-01]
 [ 1.14687640e-01  1.20886932e-01  2.84132758e-01  1.00000000e+00
   9.03528286e-03 -9.15530276e-03 -2.75191725e-02  2.84029198e-01
   4.06646543e-03  5.76125209e-02  6.42964011e-03  1.23042500e-02
   1.41648936e-01  5.55007383e-02]
 [ 1.72495133e-02 -7.83156450e-03  3.80716923e-03  9.03528286e-03
   1.00000000e+00  1.26003743e-01  2.26674391e-02  4.58856497e-03
   9.76867065e-03  9.99069814e-02  2.16339823e-02 -4.91456345e-02
   1.66408384e-02 -1.07796311e-02]
 [-5.88822443e-02  1.60515124e-02 -1.77159977e-03 -9.15530276e-03
```

4.1 Regression original tweeter

```
from pyspark.sql.types import *
from pyspark.sql.functions import *
```

```
basetablee = spark.sql("SELECT * FROM basetable_first")
#display(basetablee)
```

```
#filter; only retweeted tweets (280 000)
```

```
result= basetablee.where(col('retweeted_dummy') == 1)
```

```
from pyspark.ml.linalg import Vectors
from array import array
from pyspark.sql import Row


basetable = result.rdd.map(lambda r : Row( retweeted = r.retwteet_count_new,
                        continuousFeatures = Vectors.dense([
r.n_favourites_tweeter_new , r.n_followers_tweeter_new ,
r.n_friends_count_new,
r.n_hashtags_new,r.n_urls_new,r.n_user_mentions_new,r.n_listed_new,r.n_symbo
ls_new, r.n_media_new, r.count_positive,r.count_negative,
r.n_statuses_tweeter_new]),
                        categoricalFeatures = Vectors.dense([])
                        )).toDF()
#extra: r.count_negative,r.count_positive]
basetable.show(20)
```

```
+-------------------+--------------------+---------+
|categoricalFeatures|  continuousFeatures|retweeted|
+-------------------+--------------------+---------+
|                 []|[24.0,790.0,254.0...|      161|
|                 []|[67.0,32867.0,13....|        5|
|                 []|[7936.0,45472.0,1...|        7|
|                 []|[0.0,0.0,0.0,0.0,...|        0|
|                 []|[0.0,0.0,0.0,0.0,...|        0|
|                 []|[0.0,0.0,0.0,0.0,...|        0|
|                 []|[17795.0,139516.0...|       48|
|                 []|[14680.0,1965.0,1...|        1|
|                 []|[4230.0,2733.0,43...|        1|
|                 []|[54486.0,1995.0,9...|        1|
|                 []|[0.0,0.0,0.0,0.0,...|        0|
|                 []|[1178.0,1.7527795...|    20317|
|                 []|[14603.0,531864.0...|      388|
|                 []|[10296.0,24839.0,...|      315|
|                 []|[182.0,3146152.0,...|    41128|
|                 []|[2198.0,510.0,401...|       60|
|                 []|[51554.0,2812.0,1...|        8|
|                 []|[11497.0,1656450....|     5519|
|                 []|[29608.0,4273.0,4...|     2152|
|                 []|[80.0,3.1819537E7...|   110477|
+-------------------+--------------------+---------+
only showing top 20 rows
```

```
basetable.show()
(trainingData, testData) = basetable.randomSplit([0.7, 0.3])
```

```
+-------------------+--------------------+---------+
|categoricalFeatures|  continuousFeatures|retweeted|
+--------------------+--------------------+---------+
|                  []|[24.0,790.0,254.0...|      161|
|                  []|[67.0,32867.0,13....|        5|
|                  []|[7936.0,45472.0,1...|        7|
|                  []|[0.0,0.0,0.0,0.0,...|        0|
|                  []|[0.0,0.0,0.0,0.0,...|        0|
|                  []|[0.0,0.0,0.0,0.0,...|        0|
|                  []|[17795.0,139516.0...|       48|
|                  []|[14680.0,1965.0,1...|        1|
|                  []|[4230.0,2733.0,43...|        1|
|                  []|[54486.0,1995.0,9...|        1|
|                  []|[0.0,0.0,0.0,0.0,...|        0|
|                  []|[1178.0,1.7527795...|    20317|
|                  []|[14603.0,531864.0...|      388|
|                  []|[10296.0,24839.0,...|      315|
|                  []|[182.0,3146152.0,...|    41128|
|                  []|[2198.0,510.0,401...|       60|
|                  []|[51554.0,2812.0,1...|        8|
|                  []|[11497.0,1656450....|     5519|
|                  []|[29608.0,4273.0,4...|     2152|
|                  []|[80.0,3.1819537E7...|   110477|
+--------------------+--------------------+---------+
only showing top 20 rows
```

```
from pyspark.ml.feature import VectorIndexer, VectorAssembler,
StandardScaler
from pyspark.ml import Pipeline

# Import for Logistic Regression
from pyspark.ml.regression import LinearRegression

# Import for Decision Tree
from pyspark.ml.regression import DecisionTreeRegressor

# Import for Random Forest
from pyspark.ml.regression import RandomForestRegressor
```

```
from pyspark.ml.feature import Binarizer

binarizer = Binarizer(threshold = 1.0, inputCol = "continuousFeatures",
outputCol = "features")
binarizer.transform(trainingData).show()
```

```
+--------------------+--------------------+---------+----------+
|categoricalFeatures|  continuousFeatures|retweeted|  features|
+--------------------+--------------------+---------+----------+
|                  []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                  []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
|                 []|[0.0,0.0,0.0,0.0,...|        0|(12,[],[])|
+-------------------+--------------------+---------+----------+
only showing top 20 rows
```

```
## What is a VectorIndexer? (just to display help info)
#help(VectorIndexer)

# Vector Indexer: the input column are the categorical features and they
will be transformed to a column of indexed categorical features
vtrIndexer = VectorIndexer(inputCol = "categoricalFeatures", outputCol=
"indexedCatFeatures").fit(basetable)

# What is a StandardScaler? (just to display help info)
#help(StandardScaler)

# Standard Scaler: scales the continuous features to scaled continuous
features
standardScaler = StandardScaler(inputCol="continuousFeatures",
outputCol="scaledContFeatures")

# What is a VectorAssembler? (just to display help info)
#help(VectorAssembler)

# Vector Assembler: assembles the different vectors of the categorical and
continuous features and outputs them as a feature column
vtrAssembler = VectorAssembler(inputCols= ["indexedCatFeatures",
"scaledContFeatures"], outputCol= "features")
```

```
# How to do a Linear Regression? What are the parameters? (just to display
help info)
#help(LinearRegression)

# Logistic Regression
lrRegressor = LinearRegression(labelCol="retweeted", featuresCol="features",
maxIter=100)

# How to do a Decision Tree Regression? What are the parameters? (just to
display help info)
#help(DecisionTreeRegressor)

# Decision Tree
dtRegressor = DecisionTreeRegressor(labelCol = "retweeted", featuresCol =
"features")

# How to do a Random Forest Regression? What are the parameters? (just to
display help info)
#help(RandomForestRegressor)

# Random Forest Regressor: we define the label and the features columns and
the number of trees we want to build. 500 is a good trade-off between
computation time and error.
# The Random Forest will contain 500 different decision trees.
rfRegressor = RandomForestRegressor(labelCol="retweeted",
featuresCol="features", numTrees=500)
```

4. Training the pipeline

1. Linear Regression

```
# Defining all the stages of the pipeline
lrPipeline = Pipeline(stages = [vtrIndexer, standardScaler, vtrAssembler,
lrRegressor])

# Fitting the pipeline to the training data: This is the execution of all
the stages and the training of the Random Forest Model.
lrModel = lrPipeline.fit(trainingData)
```

```
# Make predictions on test data.
lrPredictions = lrModel.transform(testData)

# The label denotes what should have been the prediction. The prediction
column is the prediction that was made with the trained machine learning
model
lrPredictions.show(4)
#lrPredictions.printSchema()
```

```
+-------------------+--------------------+---------+-----------------+-----
---------------+----------+------------------+
|categoricalFeatures|  continuousFeatures|retweeted|indexedCatFeatures|  sca
ledContFeatures|  features|        prediction|
+-------------------+--------------------+---------+-----------------+-----
---------------+----------+------------------+
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|3010.1933302449497|
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|3010.1933302449497|
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|3010.1933302449497|
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|3010.1933302449497|
+-------------------+--------------------+---------+-----------------+-----
---------------+----------+------------------+
only showing top 4 rows
```

```
from pyspark.ml.evaluation import RegressionEvaluator

# Our PipelineModel consists of 5 stages starting of index 0 to index 4.
Each of these index places contains the model of the respective stage. The
more stages you add the larger the pipeline will become, but in this example
it contains 5 stages.
# The machine learning model is on index 4
# Print the coefficients and intercept for linear regression
print("Coefficients: " + str(lrModel.stages[3].coefficients))
print("Intercept: " + str(lrModel.stages[3].intercept))
```

```
Coefficients: [2799.371154504995,21151.278960461983,283.04008446110674,-164
7.191189434722,1575.8189545042499,-1907.0616617386952,-9273.515041288427,3.7
385389630400963,4633.379640080822,-281.0185201965542,696.3083497575342,-246
3.4014849957143]
Intercept: 3010.19333024
```

```
summ = lrModel.stages[3].summary
summ
```

```
Out[102]: <pyspark.ml.regression.LinearRegressionTrainingSummary at 0x7f0bb7
d38610>
```

For Linear Regression we have an extra class called LinearRegressionTrainingSummary which gives us statistics about our model performance on the **training** dataset. Calculating performance metrics on the training dataset is not good practice since it will show better results than realistic.

To get the performance on the **test** set we have to use RegressionEvaluator. Therefore you always need to refer to the test statistics if you want to compare different models.

```
#final dataset with not retweeted still in
#The p-values are: [0.0, 0.0, 0.21635917423454454, 6.468159341466162e-13,
0.124091058302914, 0.0, 0.0, 0.9270422296973089, 0.0]
#the R^2 is:0.0453167329296
#The MSE is: 2636715307.51
# Two-sided p-value of estimated coefficients and intercept.
print "The p-values are: " + str(summ.pValues)
# Returns R^2^, the coefficient of determination.
print "the R^2 is:" + str(summ.r2)
# Returns the mean squared error
print "The MSE is: " + str(summ.meanSquaredError)
```

```
The p-values are: [0.0, 0.0, 0.10033585604243545, 0.0, 0.0, 0.0, 0.0, 0.9817
660815841176, 0.0, 0.08715402529025074, 2.746541335252317e-05, 0.0, 0.0]
the R^2 is:0.0441290146063
The MSE is: 5244070163.22
```

If you want to compare this Linear Regression to the models of Random Forest and Decision Trees, use the performance metrics of the RegressionEvaluator. These metrics are calculated on the test data instead of the training data and give, therefore, a good view of the performance of the model on new data.

```
from pyspark.ml.evaluation import RegressionEvaluator

##final dataset with not retweeted still in
#R^2 on test data = 0.0362238
#Mean Absolute Error (MAE)) on test data = 7187.73

lrEvaluator = RegressionEvaluator(labelCol="retweeted",
predictionCol="prediction")

lrr2 = lrEvaluator.evaluate(lrPredictions, {lrEvaluator.metricName: "r2"})
lrmae = lrEvaluator.evaluate(lrPredictions, {lrEvaluator.metricName: "mae"})

print("R^2 on test data = %g" % lrr2)
print("Mean Absolute Error (MAE)) on test data = %g" % lrmae)
```

```
R^2 on test data = 0.0385547
Mean Absolute Error (MAE)) on test data = 12611.4
```

```
from pyspark.ml.evaluation import RegressionEvaluator

lrEvaluator = RegressionEvaluator(labelCol="retweeted",
predictionCol="prediction")

lrr2 = lrEvaluator.evaluate(lrPredictions, {lrEvaluator.metricName: "r2"})
lrmae = lrEvaluator.evaluate(lrPredictions, {lrEvaluator.metricName: "mae"})

print("R^2 on test data = %g" % lrr2)
print("Mean Absolute Error (MAE)) on test data = %g" % lrmae)
```

```
R^2 on test data = 0.0385547
Mean Absolute Error (MAE)) on test data = 12611.4
```

2. Decision Tree

```
# Defining all the stages of the pipeline
dtPipeline = Pipeline(stages = [vtrIndexer, standardScaler, vtrAssembler,
dtRegressor])

# Fitting the pipeline to the training data: This is the execution of all
the stages and the training of the Random Forest Model.
dtModel = dtPipeline.fit(trainingData)
```

```
# Make predictions on test data.
dtPredictions = dtModel.transform(testData)

# The label denotes what should have been the prediction. The prediction
column is the prediction that was made with the trained machine learning
model
dtPredictions.show(4)
#dtPredictions.printSchema()
```

```
+-------------------+--------------------+---------+-----------------+-----
---------------+----------+------------------+
|categoricalFeatures|  continuousFeatures|retweeted|indexedCatFeatures|  sca
ledContFeatures|  features|        prediction|
+-------------------+--------------------+---------+-----------------+-----
---------------+----------+------------------+
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|146.87198505265542|
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|146.87198505265542|
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|146.87198505265542|
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|146.87198505265542|
+-------------------+--------------------+---------+-----------------+-----
---------------+----------+------------------+
only showing top 4 rows
```

```
print dtModel.stages[3].toDebugString
```

Show result

```
dtModel.stages[3].featureImportances
```

```
Out[111]: SparseVector(12, {0: 0.2591, 1: 0.3449, 2: 0.0396, 4: 0.0, 5: 0.00
05, 6: 0.1585, 8: 0.0947, 10: 0.0031, 11: 0.0996})
```

```
# Select (prediction, true label) and compute test error
dtEvaluator = RegressionEvaluator(labelCol="retweeted",
predictionCol="prediction",)

dtr2 = dtEvaluator.evaluate(dtPredictions, {dtEvaluator.metricName: "r2"})
dtmae = dtEvaluator.evaluate(dtPredictions, {dtEvaluator.metricName: "mae"})

print("R^2 on test data = %g" % dtr2)
print("Mean Absolute Error (MAE) on test data = %g" % dtmae)
```

```
R^2 on test data = 0.527149
Mean Absolute Error (MAE) on test data = 10224.4
```

3. Random Forest

```
# Defining all the stages of the pipeline
rfPipeline = Pipeline(stages = [vtrIndexer, standardScaler, vtrAssembler,
rfRegressor])

# Fitting the pipeline to the training data: This is the execution of all
the stages and the training of the Random Forest Model.
rfModel = rfPipeline.fit(trainingData)
```

```
# Make predictions on test data.
rfPredictions = rfModel.transform(testData)

# The label denotes what should have been the prediction. The prediction
column is the prediction that was made with the trained machine learning
model
rfPredictions.show(4)
#rfPredictions.printSchema()
```

```
+-------------------+--------------------+---------+-----------------+-----
---------------+----------+-----------------+
|categoricalFeatures|  continuousFeatures|retweeted|indexedCatFeatures|  sca
ledContFeatures|  features|       prediction|
+-------------------+--------------------+---------+-----------------+-----
---------------+----------+-----------------+
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|69.32680555202086|
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|69.32680555202086|
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|69.32680555202086|
|                 []|[0.0,0.0,0.0,0.0,...|        0|                []|[0.0,
0.0,0.0,0.0,...|(12,[],[])|69.32680555202086|
+-------------------+--------------------+---------+-----------------+-----
---------------+----------+-----------------+
only showing top 4 rows
```

```
print rfModel.stages[3].toDebugString
```

Show result

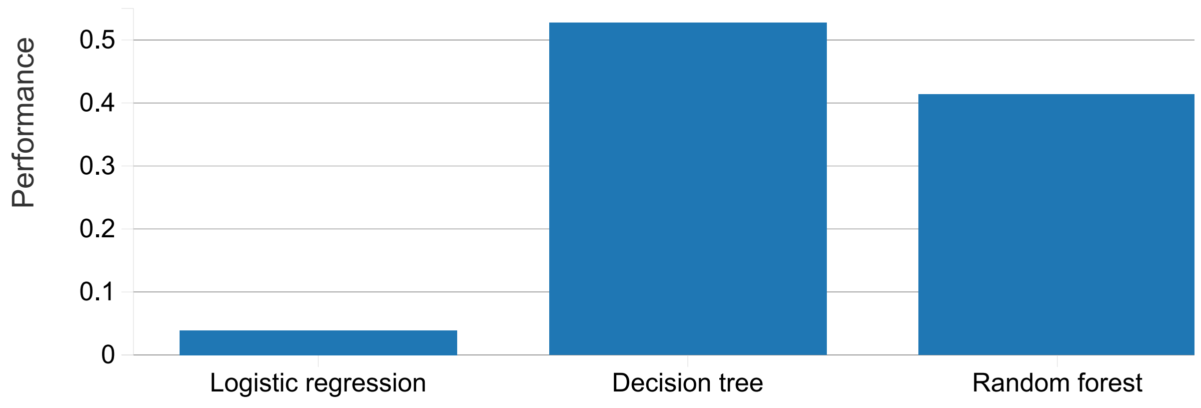
To get an idea of how important the different features were in predicting the outcome, you can print the feature importances. The higher the number the more important the feature was.

```
rfModel.stages[3].featureImportances
```

```
Out[116]: SparseVector(12, {0: 0.1561, 1: 0.1141, 2: 0.2009, 3: 0.0315, 4:
 0.0101, 5: 0.0068, 6: 0.1045, 7: 0.0, 8: 0.1413, 9: 0.0213, 10: 0.0059, 11:
 0.2075})
```

```
# Select (prediction, true label) and compute test error
rfEvaluator = RegressionEvaluator(labelCol="retweeted",
predictionCol="prediction",)

rfr2 = rfEvaluator.evaluate(rfPredictions, {rfEvaluator.metricName: "r2"})
rfmae = rfEvaluator.evaluate(rfPredictions, {rfEvaluator.metricName: "mae"})

print("R^2 on test data = %g" % rfr2)
print("Mean Absolute Error (MAE) on test data = %g" % rfmae)
```

```
R^2 on test data = 0.413803
Mean Absolute Error (MAE) on test data = 10840.3
```


Performance
Logistic regression
Decision tree
Random forest