CONTENT-AWARE LOCATION INFERENCE AND MISINFORMATION IN ONLINE SOCIAL NETWORKS

submitted by

Oluwaseun Ajao

for the degree of

Doctor of Philosophy

of the

Department of Computing
Sheffield Hallam University

October, 2019

ABSTRACT

Location inference is of potential use in the area of cybercrime prevention and misinformation detection. Inferring locations from user texts in Online Social Networks (OSN) is a non-trivial and challenging problem with regards to public safety. This work proposes LOCINFER - a novel non-uniform grid-based approach for location inference from Twitter messages using Quadtree spatial partitions. The proposed algorithm uses natural language processing (NLP) for semantic understanding and incorporates hybrid similarity measures for feature vector extraction and dimensionality reduction. LOCINFER addresses the sparsity problem which may be associated with training data following a biased clustering approach where densely populated regions within the data are partitioned into larger grids. The clustered grids are then classified using a logistic regression model. The proposed method performed better than the state-of-the art in grid-based content-only location inference by more than 150km in Average Error Distance (AED) and almost 300km in Median Error Distance (MED). It also performed better than by 24% in terms of accuracy at 161km. It was 400km better in prediction for MED and 250km better in terms of AED.

Also proposed is SENTDETECT - a technique that detects and classifies fake news messages from Twitter posts using extensive experiments with machine learning and deep learning models including those without prior knowledge of the domain. Following a text-only approach, SENTDETECT utilises an additional feature of the word sentiments alongside the original text of the messages. Incorporating these engineered features into the feature vector, provides an enrichment of the vector space prior to the deep learning classification task which utilised a Hierarchical Attention Networks (HAN) in pre-trained word embedding.

An emotional word ratio (EMORATIO) was deduced following the discovery of a positive relationship between negative emotional words and fake news posts. Finally, the work aimed to perform automatic detection of misinformation posts and rumors. A lot of work has been done in the area of detecting the truthfulness or veracity of posts from OSN messages. This work presents a novel feature-augmented approach using both text and sentiments in enriching features used during prediction. The end result performed better by up to 40% in Recall and F-Measure over the state of the art on benchmark misinformation PHEME dataset which relied on textual features only. The blend of location inference with misinformation detection provides an effective tool in the fight against vices on social media such as curtailing hate speech propagation, cyberbullying and fake news posts.

Dedicated to my darling wife Abolaji.

ACKNOWLEDGEMENTS

I would like to profoundly acknowledge the exceptional commitment of my two supervisors; Dr Deepayan Bhowmik and Dr Shahrzad Zargari. Their guidance and insight provided the much needed encouragement for achieving success on the PhD journey.

I deeply appreciate the support which I enjoyed from my dear wife Abolaji, my lovely daughters Ayomide, Esther and Daniella. I would forever be grateful for their understanding during the many long hours devoted to completing this work.

I recognise the support of my parents Mr and Mrs David Adekitan Ajao. The contributions and words of encouragement received from Engr and Dr (Mrs) Sangoyomi from the start of my postgraduate studies until the very end is priceless and most appreciated.

Publications

The following are the list of articles published during the PhD research project

Journal Publications
Two papers were published peer-reviewed journals in the course of the PhD research and they are given below.

- Ajao, O., Hong, J., and Liu, W. (2015). A survey of location inference techniques on Twitter. Journal of Information Science, 41(6), 855-864.
- Gough A., Hunter, R. F., Ajao, O., Jurek, A., McKeown, G., Hong, J., ... and Kee, F. (2017). Tweet for behavior change: using social media for the dissemination of public health messages. JMIR public health and surveillance, 3(1), e14..

Conference Publications
Three peer-reviewed conference articles were published during the PhD research and they are given below.

- Ajao, O., Bhowmik, D., and Zargari, S. (2018, July). Fake news identification on twitter with hybrid cnn and rnn models. In Proceedings of the 9th International Conference on Social Media and Society (pp. 226-230). ACM.
- Ajao, O., Bhowmik, D., and Zargari, S. (2019, April). Sentiment aware fake news detection on online social networks. In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 2507-2511). IEEE.
- Ajao, O., Bhowmik, D., and Zargari, S. (2018, August). Content-aware tweet location inference using quadtree spatial partitioning and jaccard-cosine word embedding. In 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM) (pp. 1116-1123). IEEE.

Contents

List of Figures

List of Tables

Acronyms

Acronym	Description
ACC@d	Distance-based Accuracy
AED	Average Error Distance
API	Appication Program Interface
BILOU	Beginning the Inside and the Last tokens of multi-token entities as well as Unit-length entities
CBOW	Continuous Bag of Words
CNN	Convolutional Neural Network
CRF	Conditional Random Field
ESN	Echo State Networks
F-M	F-Measure
GPS	Global Positioning System
HAN	Hierarchical Attention Neural Network
LDA	Latent Dirichlet Allocation
LIWC	Linguistic Inquiry and Word Count
LOGIT	Logistic Regression
LSA	Latent Semantic Analysis
LSI	Latent Semantic Indexing
LSTM	Long Short Term Memory
MED	Median Error Distance
MNB	Multinomial Naive Bayes
NER	Named Entity Recognition
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
OOV	Out-of-Vocabulary
OSN	Online Social Networks
POI	Point of Interest
POS	Part of Speech
PRE	Precision
REC	Recall
RELU	Rectifier Linear Unit
REST	Representational State Transfer
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines
TF-IDF	Term Frequency-Inverse Document Frequency
URL	Uniform resource Locator
XG-Boost	Extreme Gradient Boosting

Statement of Originality

The research conducted within the scope of this thesis produced the following novel and unique contributions towards a content-aware location inference and misinformation in Online Social Networks:

- improvement over the state-of-the-art in grid-based content-only location inference method (LOCINFER) using a Quadtree clustering in Jaccard-Cosine Similarity Measures for Natural language Processing
- an emotional ratio index (EMORATIO) of the negative to positive words used in messages posted to Online Social Networks
- a sentiment-aware misinformation classifier (SENTDETECT) that used an additional feature from a derived emotional index

Chapter 1

Introduction

1.1 Location Inference

The ability to accurately profile the location of social media users comes with immense benefits to service providers and consumers themselves (Kinsella et al., 2011). It is advantageous to have all the necessary meta-data from tweets, micro-blogs or generally in online social networks. However in the event they are unavailable it would be necessary to infer them. This continues to be a well-explored research domain (Mahmud et al., 2014). The dilemma of correctly identifying the authors location combined with the unique language of microblogs such as Twitter, Facebook and Foursquare has brought with it some challenges that were not associated with structured texts and online blogs, forums and conventional online media. Twitter now has more than 300 million monthly active users who on a daily basis generate over 500 million conversations popularly referred to as tweets which are text messages consisting of a maximum of 140 characters. The limited space requires brevity in writing, giving rise to an informal dictionary of words only used within the social media space. In addition, writing on Twitter tends to have lots of non-standard abbreviations, typographical errors, use of emoticons, irony, sarcasms and trending topics referred to as hashtags. Such unconventional, unstructured texts are regarded as noise as standard Natural Language Processing (NLP) tools do not handle such well (Ritter et al., 2011), leading to an interesting challenge in tweet content analysis. Location inference on Twitter can be used to identify offenders engaging in bullying of other online users via social media also known as cyberbullying. Also Twitter has been known to be a good platform for detecting the outbreaks of diseases and natural disasters. The ability to accurately infer the location of affected users can save lives and help in crisis management. It has been shown in McGee et al. (2013) that Twitter serves as a platform for building social relationships and is utilitarian for information purposes. In the former, users having a bidirectional following relationship (allowing the followers public posts to be continually displayed on the followers news feed) called friends. In the latter, a unidirectional following relationship exists where a user may only follow another influential user they are interested in. However, the followee may choose not to reciprocate the gesture thus being a unidirectional relationship. This is more common with corporations, celebrities, public figures and politicians who may have a significantly larger number of followers and just a handful of friends. Twitter users have the option to disclose their city level location which should normally be their primary residence. Text may be input within a location field as part of their Twitter user account registration. In reality, less than 14% of users accurately complete this field. In Hecht et al. (2011), it is discovered that 34% of Twitter users gave false or fictitious location names. Because this is an optional, free text field, Twitter does not regulate or enforce what their users can input. Also, to enhance the experience of its users, Twitter allows inclusion of location coordinates as metadata to tweets. This is called geotagging; the current location of the user can be included in tweets sent from mobile devices. Geotagged messages can give an accurate estimate of the current location of the user or the origin of a particular tweet up to the nearest kilometre. Similarly, even though virtually all recently manufactured smartphones now come with a GPS, less than 0.5% of Twitter users turn on the location function of their smartphones due to concerns over privacy (Li et al., 2012), cyber bullying and stalking. Other users switch off location services to conserve power and prevent their batteries from running out quickly (Lin et al., 2010). Various works have employed diverse kinds of spatial features to infer the location of online users including use of metadata information such as time of post (Li et al., 2011). Some have used only the content of the tweets (Chandra et al., 2011)(Chang et al., 2012)(Cheng et al., 2010). The others have looked at the social network relationship amongst users (Abrol & Khan, 2010). The user account information has also given useful insights for this purpose (Backstrom et al., 2010)(Bouillot et al., 2012), while some have followed a hybrid approach (Jurgens, 2013). There is also a growing trend for the use of location-based social networks (Ikawa et al., 2012; Li & Sun, 2014; Schulz et al., 2013). However most works observed still tend to include the message text as a key input for their study and techniques. Techniques have ranged from Natural Language Processing including Named Entity Recognition (NER), Parts of Speech (POS) tagging (Lingad et al., 2013), machine learning and probabilistic methods (Li et al., 2012) as well as gazetteers and location databases (Takhteyev et al., 2012). Results achieved by the various works are diverse and have been shown to be getting higher granularity levels with average error distances of less than 1 km (Li & Sun, 2014).

1.2 Misinformation Detection

This project also aimed to determine the veracity of a set of tweets based on the content of their messages without prior knowledge of the news domain. Subsequently, this work went further to establish the presence of *emotional signals* in Misinformation posts - deriving an *EMORATIO* index used for predicting rumor and non-rumor messages posted to Online Social Networks. The Rumor-Non Rumor dataset is an established benchmark dataset which was created by the PHEME research group[1]. This team of researchers are also working in the area of rumor and fake news detection and have made the tweets collected publicly available. [2]. This dataset consists of Twitter posts about five (5) global events namely:

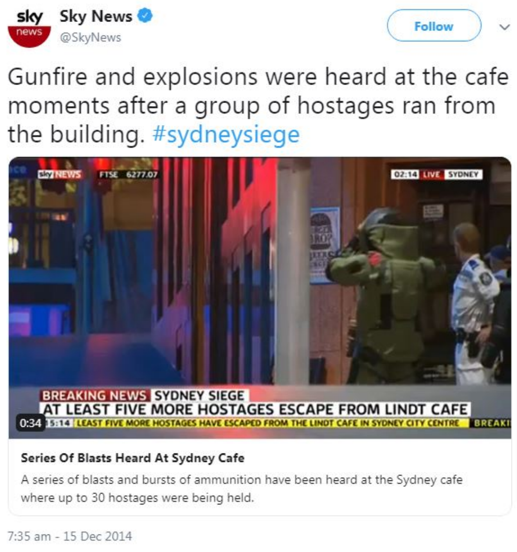
- Charlie Hebdo shooting: two brothers forced their way into the offices of the French satirical weekly newspaper Charlie Hebdo in Paris, killing 11 people and wounding 11 more, on January 7, 2015. [3]
- Ferguson unrest: citizens of Ferguson in Michigan, USA, protested after the fatal shooting of an 18-year-old African American, Michael Brown, by a white police officer Darren Wilson on August 9, 2014. [4]
- Germanwings plane crash: a passenger plane from Barcelona to Dusseldorf crashed in the French Alps on March 24, 2015, killing all passengers and crew. The plane was ultimately found to have been deliberately crashed by the co-pilot of the plane, Andreas Lubitz. [5]
- Ottawa shooting: shootings occurred on Ottawas Parliament Hill in Canada, resulting in the death of a Canadian soldier on October 22, 2014. [6]
- Sydney siege: a gunman held hostage ten customers and eight employees of a Lindt chocolate cafe located at Martin Place in Sydney, Australia, on December 15, 2014. [7]

The PHEME dataset is stored in JSON (Java Script Object Notation) file format which consisted of the 5800 tweets and their respective interactions to the source messages.

[1] `https://www.pheme.eu`
[2] `https://figshare.com/articles/PHEME_dataset_of_rumours_and_non-rumours/4010619`
[3] `https://www.bbc.co.uk/news/world-europe-30708237`
[4] `https://www.bbc.co.uk/news/world-us-canada-30193354`
[5] `https://www.nytimes.com/news-event/germanwings-flight-9525-crash`
[6] https://www.cbc.ca/news/politics/ottawa-shooting-a-day-of-chaos-leaves-soldier-gunman-dead-1.2808710
[7] https://www.theguardian.com/australia-news/2014/dec/20/sydney-siege-timeline-how-a-day-and-night-of-terror-unfolded-at-the-lindt-cafe

The breakdown of the distribution of the messages in terms of the rumor tweets and non-rumor tweets is given in Table 1.1. An example JSON record within the dataset is given in Appendix F. The variables and components of the dataset is presented in Appendix G. A typical tweet has almost 200 variables. For this work, only the 'text' variable or column was used being the actual words of the post as this is a content-only approach while the binary classes were 'rumor' and 'non-rumor' which was coded as one and zero respectively in the classification task.

Snapshots of actual non-rumor and rumor tweets from the PHEME dataset are presented in Figure 1.1 and Figure 1.2 respectively.


Sky News
@SkyNews

Gunfire and explosions were heard at the cafe moments after a group of hostages ran from the building. #sydneysiege

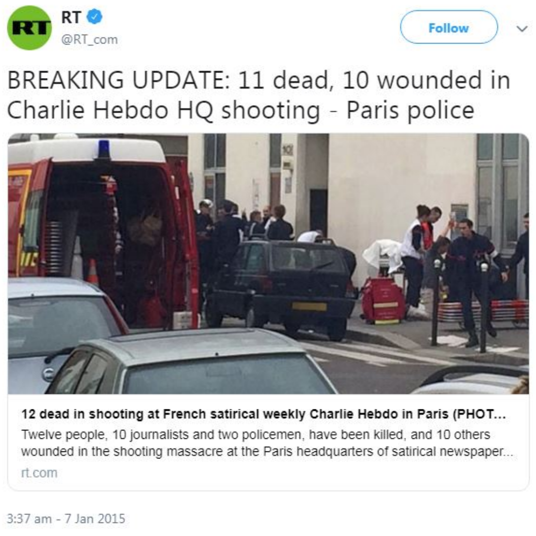
Series Of Blasts Heard At Sydney Cafe

A series of blasts and bursts of ammunition have been heard at the Sydney cafe where up to 30 hostages were being held.

7:35 am - 15 Dec 2014

Follow
BREAKING NEWS SYDNEY SIEGE
AT LEAST FIVE MORE HOSTAGES ESCAPE FROM LINDT CAFE
0:34

Figure 1.1: Snapshot of PHEME Non-Rumor Tweet Sample


RT
@RT_com

BREAKING UPDATE: 11 dead, 10 wounded in Charlie Hebdo HQ shooting - Paris police

12 dead in shooting at French satirical weekly Charlie Hebdo in Paris (PHOT...

Twelve people, 10 journalists and two policemen, have been killed, and 10 others wounded in the shooting massacre at the Paris headquarters of satirical newspaper...

rt.com

3:37 am - 7 Jan 2015

Figure 1.2: Snapshot of a PHEME Rumour Tweet Sample

Table 1.1: Distribution of PHEME Rumour-Non Rumor Dataset

RNR	CH	Distribution	FG	Distribution	GW	Distribution	OT	Distribution	SS	Distribution
Rumors	458	22%	284	25%	238	51%	470	53%	522	43%
Non-Rumors	1621	78%	859	75%	231	49%	420	47%	699	57%
TOTALS	2079	100%%	1143	100%	469	100%	890	100%	1221	100%

KEY

CH	Charlie Hebdo Shooting
FG	Ferguson Shooting
GW	Germanwings Crash
OT	Ottawa Shooting
SS	Sydney Siege
RNR	Rumor - Non Rumor Distribution

1.3 Aims and Objectives

This work aims to solve the problem of location inference on Twitter using a content-only approach. Also, it addresses the detection of fake news message from text as well as the consideration of sentiment-awareness that influences the dissemination of fake news messages.

The objectives set out in location inference was to improve on the state-of-the-art in grid-based content-only approach using a Quadtree clustering in Jaccard-Cosine Similarity Measures for Natural language Processing. The use of a discriminative clustering technique of the training dataset ensure that the density of some regions are well captured and considered in determining the size of the data partitions. In other words, the denser the geotagged area of tweets, the smaller the size of the associated grid labels assigned to the tweets from that region. Similarly, the smaller or more sparse the data from a region, the larger the size of the grid labels.

In the detection of fake news and rumors, their origin may largely be dependent on the location of the authors of these messages. Such unconfirmed messages could have far reaching political and socio-economic impact if their origins is not accurately checked and resolved.

In fulfilling the aim of misinformation detection, this work also introduces an emotional ratio index (EMORATIO) of the negative to positive words used in messages posted to Online Social Networks. The objective followed a sentiment-aware misinformation classifier that used an additional feature from a derived emotional index addressing the limitations associated with using a text-only approach in misinformation detection.

1.4 Methodology

1.4.1 Location Inference

The approach followed in this thesis is content-only approach. Using only the messages posted by the users onto social media without any meta-data from the tweets. This approach was considered as there are instances when the location of users may need to intuitively decided without additional information. This includes the automatic detection of user location from content in real-time posts made onto online social networks. This approach is much harder and non-trivial. As information and features used by the classifier comes solely from the text of the messages.

To achieve this an approach which was introduced in Section 1.3 is taken in two parts. The first part is the discriminate clustering of the data using a Quadtree partitioning technique. This method considers the geo-distribution of the training data before it is fed into the model. This addresses the sparse nature of the words such that more sparse locations would have bigger cluster and conversely more dense locations would have smaller clusters. The second part is the use of a classifier with hybrid word-embedding. A combination of the Jaccard similarity technique was first applied followed by the Cosine Similarity technique. This resulted in the reduction of the dimensionality of the created word vectors as well as the improvement in the classification task over the state-of-the-art in content-only grid-based location inference in online social networks.

1.4.2 Misinformation Detection

The this task was done in two parts; the first was intuitively detecting rumor posts without any feature engineering. The second was a feature augmented approach that involved the detection of emotion signals in the messages. Subsequently these derived emotional indexes served as features to enhance the performance of the misinformation classifier. The first part included the use of three algorithms namely: the Long-Short Term Memory (LSTM) Recurrent Neural Network (RNN), the second is the LSTM with dropout regularization and the third is the hybrid of the LSTM with Convolutional Neural Network (CNN) in one dimensional (1d) convolution. All three methods were applied onto the PHEME Rumor-Non Rumor dataset.

1.5 Datasets Used

In the conduct of the experiments for this research three different datasets were adopted for location inference and misinformation detection. These are briefly listed below:

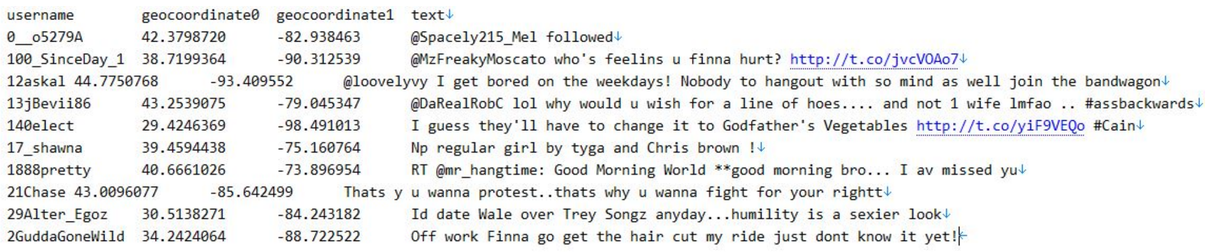
- UTGEO(Small)(Roller et al., 2012) - Location Inference. Due to the large size this dataset it is not published in this thesis. However, the link to access the dataset is given in Appendix E
- GEOTEXT(Eisenstein et al., 2010) - Location Inference. Similarly, due to the large size this dataset it is not published in this thesis. However, the link to access the dataset is given in Appendix E
- PHEME Rumor NON-Rumor Dataset(Zubiaga et al., 2016) - Fake News Detection. The repository to access the PHEME dataset is given in Appendix D

1.5.1 UTGEO-Small Dataset

This dataset was used in the works done by Roller et al. (2012) which included more than 670,000 tweets generated by a randomly selected group of 10,000 users. It is a subset dataset from a larger UTGEO-Large dataset which is made of 38 million tweets from. 449,000 users. The scaled down subset version of UTGeo-Small was chosen because it enables good comparison with the GEOTEXT dataset on the same models and algorithm. UTGEO-Small and GEOTEXT have fairly similar sizes. This avoids a bias and gives a fair reporting of the evaluation metrics. The structure of the UTGEO dataset which is labeled with geotags is given as username, geocoordinate0 (latitude), geocoordinate1 (longitude) and text which is the actual tweet message posted by the users. Snapshot of the UTEO-Small records is shown in Figure 1.3.

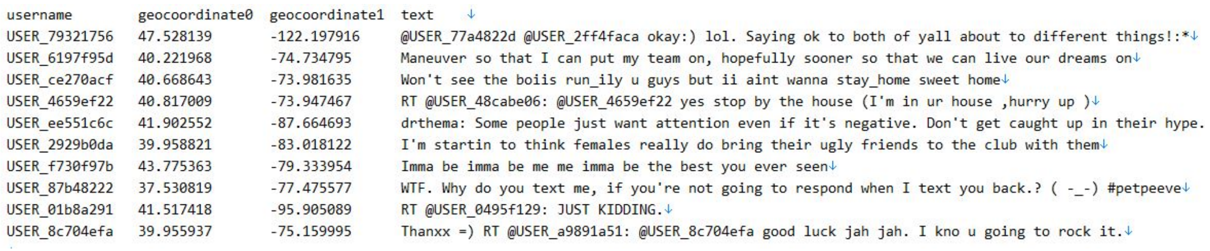
1.5.2 GEOTEXT Dataset

This corpus was created and used by Eisenstein et al. (2010). The dataset comprised of 377,616 messages gathered from more than 9,000 Twitter users based in the United



```
username        geocoordinate0  geocoordinate1  text
0__o5279A       42.3798720      -82.938463      @Spacely215_Mel followed
100_SinceDay_1  38.7199364      -90.312539      @MzFreakyMoscato who's feelins u finna hurt? http://t.co/jvcVOAo7
12askal 44.7750768      -93.409552      @loovelyvy I get bored on the weekdays! Nobody to hangout with so mind as well join the bandwagon
13jBevii86      43.2539075      -79.045347      @DaRealRobC lol why would u wish for a line of hoes.... and not 1 wife lmfao .. #assbackwards
140elect        29.4246369      -98.491013      I guess they'll have to change it to Godfather's Vegetables http://t.co/yiF9VEQo #Cain
17_shawna       39.4594438      -75.160764      Np regular girl by tyga and Chris brown !
1888pretty      40.6661026      -73.896954      RT @mr_hangtime: Good Morning World **good morning bro... I av missed yu
21Chase 43.0096077      -85.642499      Thats y u wanna protest..thats why u wanna fight for your rightt
29Alter_Egoz    30.5138271      -84.243182      Id date Wale over Trey Songz anyday...humility is a sexier look
2GuddaGoneWild  34.2424064      -88.722522      Off work Finna go get the hair cut my ride just dont know it yet!
```

Figure 1.3: Snapshot of UTGEO-Small Dataset



```
username        geocoordinate0  geocoordinate1  text
USER_79321756   47.528139       -122.197916     @USER_77a4822d @USER_2ff4faca okay:) lol. Saying ok to both of yall about to different things!:*
USER_6197f95d   40.221968       -74.734795      Maneuver so that I can put my team on, hopefully sooner so that we can live our dreams on
USER_ce270acf   40.668643       -73.981635      Won't see the boiis run_ily u guys but ii aint wanna stay_home sweet home
USER_4659ef22   40.817009       -73.947467      RT @USER_48cabe06: @USER_4659ef22 yes stop by the house (I'm in ur house ,hurry up )
USER_ee551c6c   41.902552       -87.664693      drthema: Some people just want attention even if it's negative. Don't get caught up in their hype.
USER_2929b0da   39.958821       -83.018122      I'm startin to think females really do bring their ugly friends to the club with them
USER_f730f97b   43.775363       -79.333954      Imma be imma be me me imma be the best you ever seen
USER_87b48222   37.530819       -77.475577      WTF. Why do you text me, if you're not going to respond when I text you back.? ( -_-) #petpeeve
USER_01b8a291   41.517418       -95.905089      RT @USER_0495f129: JUST KIDDING.
USER_8c704efa   39.955937       -75.159995      Thanxx =) RT @USER_a9891a51: @USER_8c704efa good luck jah jah. I kno u going to rock it.
```

Figure 1.4: Snapshot of GEOTEXT Dataset

States. It was collected in one week of March 2010. It contained the text, and geolocation of tweets. The users could not be directly identified and were anonymised. The structure of the GEOTEXT dataset is also similar to the UTGEO-small dataset which is also labeled with geotags but has anonymised usernames with latitudes, longitudes and message texts. Snapshot of the GEOTEXT records is shown in Figure 1.4.

1.6 Thesis Layout

The state-of-the-art in the field of location inference is introduced in the Literature Review Chapter 2, surveying the the most relevant and up-to date in location inference

Related literature in fake news and misinformation detection in online social networks is presented in Chapter 2.

Chapter 3 which is a contribution chapter on text-only location inference examines a grid-based content-only approach assuming the absence of other spatial clues or indicators.

Chapter 4 looks at the detection of fake messages and rumors also following a text-only approach, without prior knowledge of the topic domain, and the application of machine learning techniques for the classification task.

Chapter 5 builds on the work done in Chapter 4. In this chapter, a separate emotional ratio feature was introduced to the word vector and included in the classification task. This further helped improve the performance thus achieving a 5% improvement in accuracy and further significant improvements in terms of precision, recall and f-measures.

Summary findings, conclusion and future work are presented in Chapter 6

1.7 Contributions to New Knowledge

The expected contributions to new knowledge in this work includes:

- improvement over the state-of-the-art in grid-based content-only location inference method using a Quadtree clustering in Jaccard-Cosine Similarity Measures for Natural language Processing. Technique and method proposed is called LOCINFER
- an emotional ratio index (EMORATIO) of the negative to positive words used in messages posted to Online Social Networks
- a sentiment-aware misinformation classifier that includes an additional feature from a derived emotional index alongside the text within OSN messages. Technique and method proposed is called SENTDETECT

At the time of writing this dissertation, the results achieved in this work outperforms the state of the art in content-aware grid-only location inference and fake news identification from text in online social networks. Such results and the contributions achieved is hoped would be useful in the fight and curtailing the spread of fake news in microblogs such as Twitter and Facebook.

Chapter 2

Literature Review

2.1 State-of-the-art in Location inference

The increasing popularity of the social networking service, Twitter, has made it more involved in day-to-day communications, strengthening social relationships and information dissemination. Conversations on Twitter are now being explored as indicators within early warning systems to alert of imminent natural disasters such earthquakes and aid prompt emergency responses to crime. Producers are privileged to have limitless access to market perception from consumer comments on social media and Microblogs. Targeted advertising can be made more effective based on user profile information such as demography, interests and location. While these applications have proven beneficial, the ability to effectively infer the location of Twitter users has even more immense value. However, accurately identifying where a message originated from or authors location remains a challenge thus essentially driving research in that regard. In this chapter,a range of techniques were examined which infer the location of Twitter users from inception to state-of-the-art. We find significant improvements over time in the granularity levels and better accuracy with results driven by refinements to algorithms and inclusion of more spatial features.

2.2 Types of location on Twitter

Initial works in the field of location inference made no differentiation between the home residence of a Twitter user and their current location. It is observed that some

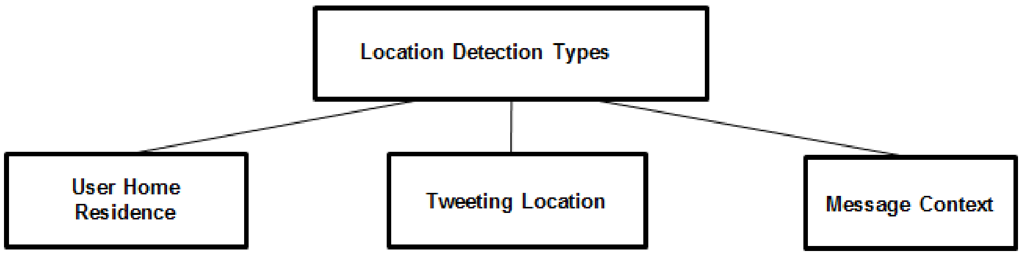


Figure 2.1: Types of Locations inferred on Twitter

authors had earlier referred to it as User Location (Abrol & Khan, 2010; Chandra et al., 2011; Gonzalez et al., 2011) assuming the geotagged location of the tweet to be the user location. Hecht et al. (2011) infers the home residence of the user to be already contained within the location field provided as part of the user account information. (Ikawa et al., 2013) cites the fact that it was possible to tweet about a particular location and not be in that location at the time. It illustrates the concept of space and time. Li & Sun (2014) examines the concept of determining Points Of Interests (POI) with a temporal awareness of the past, present and future as mentioned in the message text. Ikawa et al. (2013) also defines 4 distinct location types on Twitter, namely, locations directly mentioned in the message text, focused locations i.e. described by the message context, users current location (from where a tweet was sent) and their location profile which can be a combination of their current, previous home locations and other places they frequently visit . A diagrammatic illustration of locations inferred on Twitter is given in Figure 2.1.

2.3 Spatial features and indicators

As illustrated in Figure 2.2, diverse indicator types that help to infer the location of Twitter users have been employed over the years and we shall look at them in more detail.

2.3.1 Message Context

Twitter message text forms the backbone of most research within the field of location inference as this helps understand the context of the messages themselves. The challenges associated with tweet text processing can be very much linked to the unstructured format of those messages as opposed to online articles and blogs that have

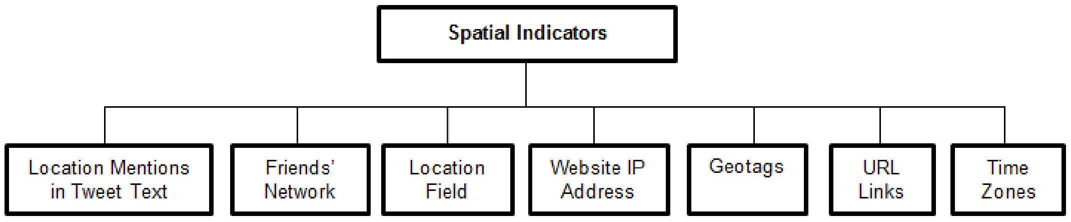


Figure 2.2: Indicators of user location

more content and follow conventional grammatical and semantic usage. These include abbreviations and more so non-standard ones as there is no precise rule of writing on the social media platform. Because most of tweets are sent via mobile devices their users have a large leeway for typos and brevity. An instance would be the abbreviation for the United Kingdom which could be UK, GB, GBR or GR8 Britain. Li & Sun (2014) uses the Brown clustering to handle Out-Of-Vocabulary (OOV) words. While Cheng et al. (2010) uses the Jaccard coefficient to resolve and accommodate similar words. Ikawa et al. (2012) uses cosine similarity to match actual location with a list of keywords. A good content analysis approach would take into consideration all possible instances of the location entities being expressed within the message. It is important to note that even when locations are identified within the messages, it cannot be automatically inferred as the user location or even the tweet location (Ikawa et al., 2012). A good example would be where a tweet contained the city name Belfast; however it may not necessarily imply the author was based in Belfast or that the tweet was even sent from Belfast. Some works have used the URL links within the body of the text as spatial indicators for inferring the location of the users. Schulz et al. (2013) uses these links to infer the country level location by inputting the corresponding domain server IP addresses into the InfoDB database - a free online query service that matches geographical location with IP addresses and domain names. The most successful techniques have employed use of the message content alongside one or two other features to have a robust output.

In this work, location inference follows a hybrid approach of both NLP and machine learning, however only the text of the messages are considered as spatial cues for determining the location of the users on Twitter. There is an assumption that Points of Interest and context in which the words were used would be intuitively recognised by the machine learning model. Also, the natural language processing approach using a hybrid of similarity measures would remove redundancy within the word vectors used in the text classification task, this also handles Out-of-Vocabulary occurrences across messages posted by the users for detecting their location.

2.3.2 Social Networks

The followers of a user have been shown to be a good indicator of their home residence. While reciprocal following relationship can provide evidence of strong user connections, other indicators can include regular exchange of messages or frequent mention of each others names within messages. Jurgens (2013); McGee et al. (2013, 2011) have shown that two users are likely to communicate frequently if they reside within the same city and vice versa. Li et al. (2012) mentions the possibility of having multiple location profiles based on the users offline social relationships with other users. According to Li et al. (2012) the more influential a user is, the higher the diversity of their followers and friends would be from around the world. Abrol & Khan (2010) shows that the network of a user would be optimal for inferring location up to the third depth.

2.3.3 User Profiles

The account information given at the point of registering a Twitter user account can give very useful insight into their location allowing advertisers to accurately target their customers. It can also help emergency services and first responders to immediately locate the scene of a crisis or disaster or to help track down potential offenders in cyber bullying crimes. Usually the location field follows a free text format enabling the users to manually type in their city name. It would normally be in the City or State granularity level such as Glasgow, Scotland. However, instances of less conventional phrases such as the The Big Apple or even meaningless expressions such as Bieber Town make it difficult for conventional Natural Language Processing (NLP) and machine learning algorithms to effectively extract the location entities and in some instances are likely to give misleading results. The users website or personal web page could also be listed on the account information and would normally hold useful information. This would be so in particular if the website listed by the user was hosted by a provider resident within their home country and with possibly city-level information, if they resided in the same city. However, there is the possibility of hosting their website in one geographical location and living elsewhere. For instance a user based in the US might had initially signed up for web hosting with a provider based in the US but if they relocated to say, Australia but had not switched service providers. This would mean that their web domain and server IP address would still be indexed to their former country of residence which is the United States whereas they currently reside in Australia.

2.3.4 Geotags

Most smartphones are now equipped with the Global Positioning System (GPS) function as a standard feature and working with this, geo-satellites are able to accurately pinpoint the users geographical location i.e. latitudes and longitudes coordinates. This would usually be an optional feature for users to enable due to their privacy concern and it has been found that less than 0.5% turn on their location services (Li et al., 2012) making this a challenging feat. This indicator is very useful where the user is mobile and frequently updates their location profile. Jurgens (2013) uses Vincenty's geometric median - an estimate well applied to the field of Geography and land surveying (Vincenty, 1975). The formula was used to estimate the location of a Twitter user using their last 5 geotagged tweets that occurred within a 15km radius, as shown in Equation 2.1. m is the geometric median while L is their GPS location with latitude x and longitude y.

$$m = argmin_{x \epsilon L} \sum_{y \epsilon L} Distance(x, y) \tag{2.1}$$

2.3.5 Third Party Sources

The popularity of location-based social media sites has enabled means of interaction also referred to as Geo-social networking. Foursquare and Yelp are good examples of these sites offering companies, small businesses and restaurants the opportunity of registering on their directory which gets such businesses enlisted as part of a geographic database. Online users are able to find the location of a place of interest, say a restaurant in Belfast, simply by searching their online directory. Previous visitors to these locations are able to leave reviews and comments about these places called check-ins. Foursquare allows its users to connect their Twitter accounts to Foursquare posts which are usually geotagged thus allowing to infer their location from a Foursquare message post even though they have not disclosed their location on Twitter (Li & Sun, 2014).

2.3.6 Time Zones

Tweets metadata usually contain a timestamp of the message and the time zone as captured by the Twitter API. This is a useful feature that can allow the inference of the location to at least country-level granularity (Mahmud et al., 2014). This would be quite useful where there is limited and sparse location information within the body of the message text.

2.3.7 Web Snippets

Li et al. (2011) addresses the sparsity problem of tweets in locating points of interests by employing webpage snippets. Rae et al. (2012) searches Wikipedia to get structured information about places to complement tweets about Points of Interests (PoIs).

2.4 Methods of inferring locations on Twitter

Diverse approaches and techniques have been used in the past and are currently being employed to better improve the accuracy of location inference methodologies and algorithms. This burgeoning field lends techniques ranging from several fields of study involving machine learning, statistics, probability, natural language processing to geographical information systems and surveying. Diverse methods have achieved varying levels of success; in any case the effectiveness and granularity levels achieved by these methods continue to improve rapidly. However, the informal nature of the social media platform as well as unique language of expression brings with it some challenges in trying to properly deduce the meaning and context of these conversations. They contain frequent use of emoticons, sarcasms, hashtags, abbreviations and typographical errors. This leads to the need for robust methods and algorithms that will factor that into its input. In the analysis of text messages, names of places mentioned could be ambiguous. For example the word Washington could refer to the state or a place bearing the same name within the District of Columbia both in the United States. Washington DC and Washington State are 3,000 miles apart. The process of trying to disambiguate place names is called toponym resolution. It becomes more complicated when noun types could have similar names; for example, a person could also be called Washington. Techniques used in location inference can be broadly grouped into three categories namely natural language processing, machine learning and use of location databases or gazetteers as shown in Figure 2.3.

2.4.1 Natural Language Processing (NLP) Techniques

Natural processing methods applied include the named entity recognition which could be either segment-based or word-based representation (Li & Sun, 2014) with the former showing more effectiveness in recognising entities within tweets and the widely used tool for this technique is the StanfordNER. Gelernter & Mushegian (2011) found that use of the StanfordNER on social media texts did not accurately detect entities including location names, especially if they were unusually abbreviated thus having a high probability of type I error (false negatives). However, Lingad et al. (2013) retrained four NER tools namely StanfordNER, OpenNLP, TwitterNLP and Yahoo! Placemaker on 2,878 disaster-related tweets applying a 10-fold cross validation and found the retrained StanfordNER to have the highest F-measure of 0.9. In Hinduja & Patchin (2010) a hybrid approach was adopted where location entities were extracted and parsed into a gazetteer to accurately geocode the place names mentioned in the tweets. The conditional random field (CRF) technique is recommended for handling complex dependencies within phrases and sentences. The University of Illinois at Urbana Champaign NETagger (Ratinov & Roth, 2009) has also been well used to date. NLP techniques often tend to be applied with probabilistic tools such as multinomial Bayesian and generative probability models. It requires training data and may be complex to apply. However, it allows the development of sophisticated algorithms that suit the users needs. It has also been shown to have a quicker processing time. Another benefit of using NLP is its flexibility in identifying unconventional words (which is quite common on Twitter) as similarity checks between words can be done in order to identify entities listed within a keyword list (Ikawa et al., 2012).

2.4.2 Gazetteers

Gazetteers and Geographical Databases are also well applied to the study and some tools used include the United States board on geographic names popularly called GeoNames[1], GeoNet[2] and the US census TIGER Gazetteers[3]. Some works have also used a hybrid of the earlier mentioned techniques. For example Paradesi (2011) proposed a system for inferring the current location of a Twitter user using the PipePOS tagger and the USGS location database to resolve ambiguous location names. Gazetteers are easy to implement (Schulz et al., 2013). Also they do not require training data but there is a challenge of slow processing speed. Abrol & Khan (2010) show that varying

[1]`http://geonames.usgs.gov`
[2]`http://www.geonet.org.nz`
[3]`http://www.census.gov/geo/maps-data/data/gazetteer.html`

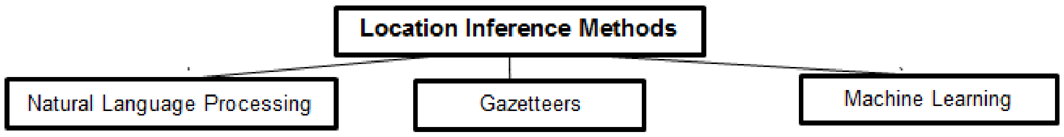


Figure 2.3: Main categories of location inference techniques

the size of their Twitter dataset by increasing the depth of friends/followers relationship has no impact on the time taken to compute and detect the location of a tweeter using the gazetteer method. This can be especially frustrating in databases with very large dataset. Thus there also exists the challenge of toponym resolution and matching of words with the location database to cater for the abbreviations and unconventional writing style on Twitter and in most cases location names which are found in messages but do not exactly match the database thus are discarded and could lead to a type I error (false negative). This would be synonymous to an error asserting that the user location being estimated was absent or not found in the database of possible geographical locations.

2.4.3 Probabilistic and Machine Learning Techniques

Techniques for the detection of location of Twitter users have also been adopted from data mining and machine learning techniques. These methods have been shown to be good methods of clustering Twitter users locations (Pennacchiotti & Popescu, 2011). Techniques used have included k-nearest neighbour, fuzzy matching (Abrol & Khan, 2010), Naives Bayes, probabilistic clusters, Markov chain models. Ryoo & Moon (2014) used a probabilistic model that incorporated the local words used by users while users who had not mentioned sufficient local words had their location inferred from the local words of their friends network. Also, in Chandra et al. (2011); Chang et al. (2012); Cheng et al. (2010) location was inferred from probabilistic distribution of users local words. (Backstrom et al., 2010; Li et al., 2012) use a probabilistic algorithm based on friends relationship. Jurgens (2013) uses a graph-based approach applying label propagation to predict location from that of other users in their network. Eisenstein et al. (2010) develop geographic and topic models adopting Mean Field Variational inference and Kullback-Leiber divergence. Hecht et al. (2011) proposes a Naives Bayes model classifier. Ikawa et al. (2012) learns the patterns of location based services from past messages to predict current location. Kinsella et al. (2011) developed language models using Bayesian inversion. Li & Sun (2014) used a CRF classifier to identify points of interest (PoIs) incorporating four classes i.e. lexical, grammatical, geographical and BILOU schema features. In Li et al. (2012) a model that considered both the tweeting and following relationships was used. Ratinov & Roth (2009) looked at dynamically weighted ensemble method to create a combination of Naives Bayes, Naives Bayes Multinomial and Heuristic classifiers that can predict user location at all levels of granularity.

2.4.4 Multinomial Naive Bayes

This is quite popular with discrete probability distributions such as word counts in text classification (Han et al., 2011) . They are straight forward to implement. Using the default blackbox settings in the Scikit Learn [4] having alpha value at 1 and learn class prior probabilities; they were also adjusted according to the classes within the dataset - all being the default settings on the sklearn MNB classifier. With a multinomial event model, samples (feature vectors) represent the frequencies with which certain events have been generated by a multinomial $(p_1, ..., p_n)$ where p_i is the probability that event i occurs (or K such multinomials in the multiclass case). A feature vector $X = (x_1, ..., x_n)$ is then a histogram, with x_i counting the number of times event i was observed in a particular instance. This is the event model typically used for document classification, with events representing the occurrence of a word in a single document (Witten et al., 2016). The likelihood of observing a histogram x is given by:

$$p(x|C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i pk_i^{x_i} \tag{2.2}$$

2.4.5 Logistic Regression

Also known as the Maximum Entropy or Logit classifier is a regression model where the dependent variable(s) are categorical (Han et al., 2011); The default blackbox settings in the Scikit Learn classifier [5]. Theoretically, Logistic Regression is defined as the log-likelihood of the Linear Regression model (Witten et al., 2016). Thus in the generalized additive form consider a set of independent variables $X_1, X_2, ..., X_p$ predicting a likely outcome (Y) with $f_j(f_1, f_2, ..., f_p)$ unspecified smooth functions where α is the intercept and $\mu(X)$ represents the probability of the response variable $P(Y = 1)$. Then the

[4]`https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html`

[5]`https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

logistic regression model is given in Equation 2.3

$$log\left(\frac{\mu(X)}{1-\mu(X)}\right) = \alpha + f_1(X_1) + f_2(X_2) + ... + f_p(X_p) \tag{2.3}$$

In the experiments the Logit classifier outperformed all the classifiers in terms of precision, accuracy and recall. To avoid over-fitting, the L2-penalty also called Lasso regularization (James et al., 2013) was used as it is more robust for handling large number of features [6] and handles sparse data well; A phenomena that is quite common with geo-located Twitter data sets. There was no implementation of dual or primal formulation as the number of samples far exceeded the number of the features in the task. A balanced class scaling was applied to handle the L2 penalty. The maximum number of iterations was set at 100, being the default settings for the Scikit Learn machine learning classifier that was used.

2.4.6 Neural Networks

The artificial neural network (ANN) model was also considered with the multilayer perceptron (MLP) architecture aiming for higher predictability (Cheng & Titterington, 1994). The model adopted for the evaluation follows equation 2.4

$$v_k = g_k(\varphi_k(x, v_k)) \tag{2.4}$$

v_k denotes the output from the kth hidden layer of the ANN where $k = 1, 2, ..., M$, for a single output $y = f(\phi(v, w))$. In expressing y as a function of x with parameters $M + 1$ sets of weights $v_1, v_2, ...v_M, w$ this becomes a non-linear regression problem

The default settings for ANN implementation in scikit learn [7] were used. The number of neurons was 100 for two hidden layers using the rectified linear unit (ReLu) activation function which returns for any function f(x) = max(0, x) (Glorot et al., 2011). Due to the large size of the data the stochastic gradient-based approach as a solver for the weight optimization and used the L2 penalty as the regularization term parameter. The maximum number of iterations was 200.

[6]`https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c`

[7]`https://scikit-learn.org/stable/modules/neural_networks_supervised.html`

2.4.7 Decision Trees

In addition, decision trees (Han et al., 2011) was used, with a mean square error (MSE) criterion and chose the best split at each node. A maximum depth was not specified but continued to expand the nodes until all leaves of the tree are pure and only one sample remained on an internal node. All features were considered when looking for the best split of the tree nodes. while decision trees tend to be handling outliers and missing values in the input space, they have poor predictive power and unable to extract linear combinations of features. The default settings for Decision Trees implementation in scikit learn [8] were used.

Mathematically, let the data at node m be represented by Q. For each candidate split $\theta = (j, t_m)$ consisting of a feature j and threshold t_m the data is likewise separated into two subsets at each split of the node and the parameters that minimizes the impurity at m

$$\theta = argmin_{\theta} G(Q, \theta) \tag{2.5}$$

2.4.8 Random Forests

Also the performance was examined using the ensemble method of Random Forest. This involved constructing an ensemble of 10 random decision trees in the 'forest' as estimators (Breiman, 2001) (Han et al., 2011). The *Gini* impurity was instead of the Information Gain Criterion. The limit of the maximum feature size equal to the square root value of the number of features. The default settings for Random Forest implementation in scikit learn [9] were used.

Given t trees created in random subspaces, a discriminant function is needed to combine their classification of a test point. For a point x, let $v_j(x)$ be the terminal node that x is assigned to when it descends down tree T_j $(j = 1, 2, , t)$ Given this, let the posterior probability that x belongs to class $c(c = 1, 2, , n)$ be denoted by $P(c|v_j(x))$

[8] `https://scikit-learn.org/stable/modules/tree.html`

[9] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

2.5 Data Partitioning with Quadtrees

A Quadtree is a hierarchical data structure and partitioning technique for efficiently organizes data in a pre-defined discriminative manner (Samet, 1984). A Quadtree illustration is given in Figure 2.4. For example given an object is first decomposed into four quadrants or nodes; numbered 1,2,3 and 4. Nodes 2 and 3 are further split based on spatial interest (population density in this work) into 4 leaves each as they had not been fully decomposed until pure nodes were obtained. Quadtrees effectively handle spatial querying of geographic data (Samet et al., 1984) and proven applications in the areas of collision detection and image processing (Mehta & Sahni, 2004). LOCINFER employed the method because in the area of location inference application of the algorithm would be to query the location of users around a predefined radius or some location of interest such as a geo-political region. This method fulfills this requirement and as such within a conventional database ultimately support searches, insertions and deletions within the parent and leaf nodes of the dataset. Another advantage that the Quadtree method has over uniform grids is also the time saved in implementation as the nodes with little or no data points can be easily dropped from the query.

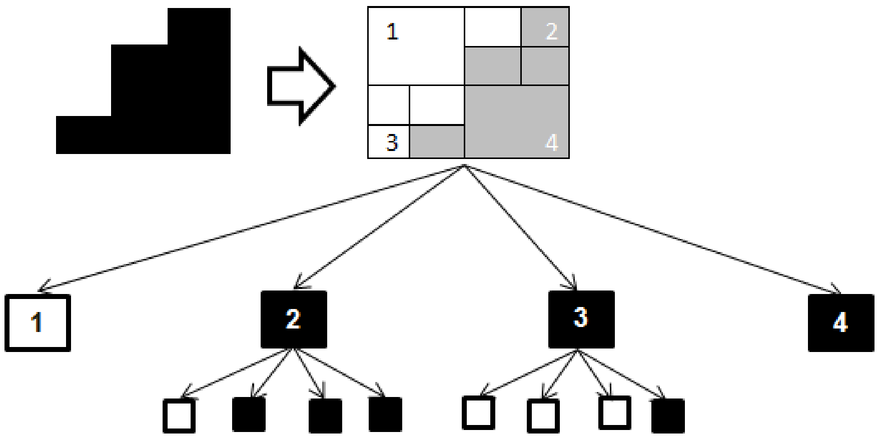


Figure 2.4: Spatial partitioning method illustrating decomposition and equivalent Quadtree representation of an object

2.6 Location Inference Applications in Public Health

The early detection of an epidemic outbreak is hinged upon a surveillance system that effectively captures the prevalence of syndromic conditions expressed by a population of interest. Paul & Dredze (2011) shows there exists a positive relationship between tweet mentions of disease symptoms and public health data. Syndromic data gathered from tweets would be of immense benefit if spotted on time as an interesting pattern or anomaly and better still, if the precision of the location or the part(s) within the entire population is known or accurately inferred. Thus, Twitter user locations inferred and known on time could help forestall the spread of a deadly disease outbreak thereby saving lives. It will also ultimately save money as it would cost less to administer and treat infected patients if the disease is contained in its early stages of manifestation.

2.7 Location Inference Applications in Cyberbullying

There are increasing reports of stalking and cyberbullying where people are being verbally assaulted and at times sexually harassed by people they may or may not know. In most cases the users would veil themselves with anonymous user accounts with the belief that they cannot be identified. This continues to remain a challenge for police and law enforcement, proving to be even more difficult to produce sufficient evidence to prosecute such offenders in the court of law thus even more sophisticated technological methods such as cryptography are being applied (Burmester et al., 2005). There are cases that have led to the eventual suicide of their victims as well as the demise of offenders themselves (Hinduja & Patchin, 2010). This has prompted a lot of privacy concerns and raises questions as to how safe online social communication is. Tri & Jung (2015) extended the Hyperlink-Induced Topic Search (HITS) algorithm to identify and rank the relationships existing between a set of keywords (tags) and a set of location-aware content such as videos and photos on Flickr. This further illustrates the need to accurately map topics and conversations to related location resources within the broader social media space.

2.8 Location Inference Applications in Crisis and Disaster Management

Also, potential applications of this would be better public enlightenment as to what level of information they should disclose online if they want to remain anonymous because their location could be implicitly inferred from other means such as content of their tweet messages, relationship with other users and their account information just to mention a few. While some Twitter users would like to switch on the location services of their smart phones, there is the limitation of mobile device battery life thus some only enable the GPS function once in a while. However in event of a natural disaster such as an earthquake or a Tsunami, Twitter users may switch on this service (MacEachren et al., 2011) to support emergency rescue efforts. TEDAS is a system developed in Li et al. (2012) for the identification of crime and disaster related event (CDE) tweets while extracting the location from such messages from the users past tweets as well as their friend networks using a rule-based classifier. It is expected that future work would look at ways of further improving the granularity levels of locations inferred on Twitter. Better algorithms would imply fewer friend network and information are then required to infer locations accurately. Sakaki et al. (2010) applied semantic information gathered from tweets to develop a system that detects and provides early warning alerting its users of an earthquake occurring in a location. The location accuracy of such a system is crucial for first responders and for emergency medical services to formulate effective evacuation strategies. Streamlining the detection in these locations would mean a more efficient and effective earthquake detection system.

2.9 Tweet Gathering and Analysis

Tweets made public are usually accessible in the online domain method and can be retrieved using the Twitter REST API[10] while live updates on individual or multiple users can be extracted as required in real-time using its streaming API. This accessibility makes Twitter a powerful tool in the gathering and analysis of public views allowing its users to become social sensors within the population.

2.9.1 Tweet Corpus

Corpus sizes of tweets gathered have varied from relatively small datasets of under 62,000 tweets (Hecht et al., 2011) to as large as 615 million tweets (Ryoo & Moon, 2014). Table 2.1 Time span of the data collected was usually in the range of few weeks to a couple of months. On the one hand, the REST API is also useful for the collection of specific user tweets allowing for the backtracking of their timeline to gather their most recent 3,200 tweets. At the time of writing this thesis, the Twitter search API allows the collection of tweets by defined keywords or around a specified location name or coordinates (geotagged messages) for tweets posted up to the previous 6-9 days. On the other hand, the streaming API that collects the messages as they are being broadcast would only be able to receive 1% of the Firehose. Twitter data partners such

[10]`http://dev.twitter.com/overview/documentation`

Table 2.1: Datasets and collection periods of some works

Reference	Corpus Size	Period Covered	Duration (Months)
Bouillot et al. (2012)	2,495,000	Jan 11 May 11	5
Eisenstein et al. (2010)	380,000	Mar 10	1
Hecht et al. (2011)	62,000	Apr 10 May 10	2
Jurgens (2013)	47,700,000	Apr 12 Nov 12	8
Li & Sun (2014)	4,330,000	Jun 10	1
Mahmud et al. (2012)	1,524,000	Jul 11 Aug 11	2
McGee et al. (2013)	100,000,000	Jun 10	1
Ikawa et al. (2013)	20,000,000	Apr 11	1
Ryoo & Moon (2014)	615,000,000	Jun 10 Apr 11	13
Schulz et al. (2013)	80,000,000	Sep 11 Feb 12	6

as GNIP[11] or Datasift[12] provide a premium service that supplies messages covering a longer duration as well as 100% access to the Firehose.

Another means of gathering Twitter data for training and testing location inference algorithms would be from other researchers within the field. An example is the Social Network Analytics Platform[13] (SNAP) provided via open access by Stanford University. It includes large tweet corpuses and social networking data which can be used for graph analysis.

2.9.2 Results and Metrics

The results achieved by various works have significantly improved over time with regards to increased accuracy and granularity levels. This has been largely driven by refinements to algorithms and inclusion of more spatial features. In the same vein, the effectiveness of spatial features and/or accuracy of the algorithms required to achieve finer granularity levels increase progressively for time zones, country, region, city and post codes respectively. For example a more accurate prediction method would be required to estimate a Twitter users home postal code as opposed to one that infers their country of residence. Several other metrics have been presented to compare the performance and results of the methods with one another. They include accuracy within a specified range say 10 km, error distance - Average Error Distance (AED) and Median Error Distance (MED). To validate the effectiveness of the location inference methods against other baselines, the k-fold cross validation of accuracy has been well utilized while other metrics used in evaluating the geolocation classifier performance include

[11]`http://www.gnip.com`
[12]`http://www.datasift.com`
[13]`http://snap.stanford.edu/snap`

Precision as given in Equation 2.6, Recall shown in Equation 2.7 and F-measure, which is the harmonic mean of Precision and Recall as given in Equation 2.8.

2.10 Precision, Recall and F-Measure

There are certain metrics used in evaluating the performance of location inference techniques. They include: Precision, Recall and F-Measure (Witten et al., 2016). Formulae for their calculations is given in Equation 2.6, Equation 2.7 and Equation 2.8 respectively.

From the equations, the following terms are explained. True Positives (TP): These are correct classifications made by the location inference technique. The classifier correctly identifies a tweet comes from a particular location grid or class.

True Negatives (TN): These are also correct classifications made by the location inference technique. Similarly, the classifier correctly identifies a tweet does not comes from a particular location grid or class.

False Positive (FP): This occurs when the location is incorrectly predicted as yes (or positive) within the class or location grid when it is actually negative or not from that class.

False Negative (FN) occurs when the location is incorrectly predicted as negative when it is actually positive or within the class or location grid.

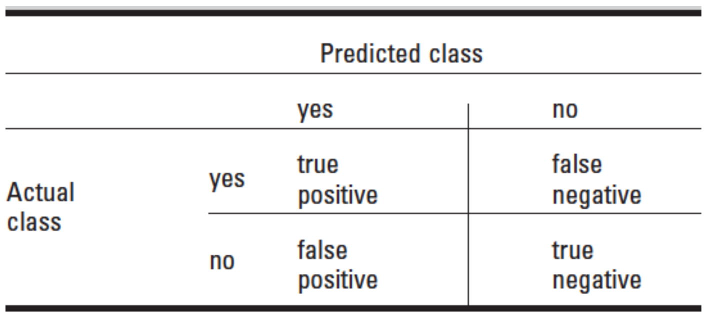
The relationships between TP, TN, FP and FN. Also, all possible prediction outcomes is illustrated in Figure 2.5

These class metrics used in location inference can be defined as follows:

Accuracy: This is the simple ratio of the correctly predicted locations to the total number of tweets within either of the (GEOTEXT or UTGEO) datasets.

Precision: This is the ratio of the correctly predicted positive observations to the total predicted positive observations.

This is the ratio of the correctly inferred locations (positive observations) to the total



		Predicted class	
		yes	no
Actual class	yes	true positive	false negative
	no	false positive	true negative

Figure 2.5: Two Class Prediction Outcomes

inferred positive observations

Recall (Sensitivity): This is the ratio of correctly predicted positive observations to all the observations in the actual class

FMeasure: This is the weighted average of the Precision and the Recall evaluation metrics

AL: Actual Location IL: Inferred Location

- True Positive(TP): IL=Yes, AL=Yes
- True Positive(TP): IL=Yes, AL=Yes
- False Negative(FN): IL=No, AL=Yes
- True Negative(TN): IL=No, AL=No

$$Precision = \frac{TruePositives}{(TruePositives + FalsePositives)} \tag{2.6}$$

$$Recall = \frac{TruePositives}{(TruePositives + FalseNegatives)} \tag{2.7}$$

$$F - Measure = \frac{2 * Precision * Recall}{(Precision + Recall)} \tag{2.8}$$

The confusion matrix of the actual location grid versus the inferred location grid is given in Table 2.2. It can be seen from Table 2.3 the growing trend of finer grained location inference on Twitter. Over time, accuracy levels and granularity of results have continued to improve starting from 2010 when inference was only precise to the city-level. This resulted from the fact that location was inferred solely on the basis of the tweet content without giving consideration to other information such as web links, friend the user profile and other metadata associated with the message, however with the subsequent adoption of spatial features such as user check-ins gathered from location-based services including Foursquare, accuracy has improved significantly subsequently (Ryoo & Moon, 2014) achieving a 60% accuracy within a 10km. This is a remarkable improvement as opposed to a performance of 51% accuracy over a 160km radius recorded by Cheng et al. (2010).

2.11 Calculating Error Distance

The concept of location inference on Twitter is such that given a set of geo-tagged tweets $T_i = \{t_1, t_2, ..., t_N\}$ with ground truth location of $ActualLoc_i$ and a predicted location $ExpectedLoc_i$. The classifiers performance are also evaluated in terms of the Average Error Distance (AED), Median Error Distance (MED) and Distance-Based Accuracy which is usually within a radius of 100 miles which is equivalent to 161 kilometers. The lower the error distance, the better the performance of the classifier.

In calculating the error distance (in km) between the predicted and actual location the Haversine formula (Shumaker & Sinnott, 1984) (Laylavi et al., 2016) was applied. Also known as the *Great Circle Distance* between any two geo-coordinates on the earth's surface assuming an spherical shape of the Earth. This method was chosen as was found suitable and stable in determining the distance estimation of several diverse and closely located geo-coordinate pairs.

Table 2.2: Confusion Matrix

		Inferred Location Grid	
		TRUE	FALSE
Actual Location Grid	TRUE	True Positive	False Negative
	FALSE	False Positive	True Negative

Table 2.3: Improvement in granularity levels over the past 5 years

Ref	Year	Technique	ACC(%)	Coverage	Location Type
Cheng et al. (2010)	2010	Probabilistic (ML)	51.00	160km	User location
Tri & Jung (2015)	2010	Geographic topic model (NLP)	24.00	State level	Home location
Kinsella et al. (2011)	2011	Language models	13.90	Zip code level	Tweet location
Kinsella et al. (2011)	2011	Language models	29.80	Town level	Tweet location
Chang et al. (2012)	2012	Gaussian Mixture models & MLE (ML)	49.90	160km	Home location
Li et al. (2012)	2012	Probabilistic (ML)	62.30	160km	Home location
Ikawa et al. (2012)	2012	Machine learning	20.00	10km	Tweet location
Sadilek et al. (2012)	2012	Dynamic Bayesian Networks	57.00	0.1km	Home location
Schulz et al. (2013)	2013	Gazetteer	37.00	10km	Tweet location
Mahmud et al. (2012)	2014	Probabilistic (ML)	60.00	10km	Users location

2.11.1 Average Error Distance

The average error distance (AED) measures the arithmetic average error of predictions for the messages within the dataset, however it should be noted that this metric can be easily skewed by large ranges of values within the dataset and would particularly be unreliable if there where a presence of an anomaly in the training of the classifier. The AED for the classifier is given in Equation 2.9.

$$AED = \frac{1}{N}\sum_{i=1}^{N} \left(\{|ActualLoc_i - ExpectedLoc_i|\}\right). \tag{2.9}$$

2.11.2 Median Error Distance

The median error distance (MED) overcomes the limitation of the AED by considering only the error values close to the median. The errors are sorted in ascending order prior to the estimation of the Median value. This was found to be most reliable and gives a truer indication of the performance. This is represented in Equation 2.10.

$$MED = MedianDistance\{|ActualLoc_i, ExpectedLoc_i|\}. \tag{2.10}$$

2.11.3 Distance-Based Accuracy

Accuracy levels at a set distance and more specifically around a distance d is a renowned benchmark and this was applied. It is estimated as the ratio of correctly predicted location with an error margin less than $d = 161$km compared to the entire tweets count of N. Equation for distance-based accuracy is given

$$ACC@161 = \sum_{i=1}^{N} \frac{\{|ActualLoc_i - ExpectedLoc_i|\} \leq 161km}{N} \tag{2.11}$$

2.12 State of the Art in Misinformation Detection

The Merriam Webster Online Dictionary (multi-element approach to location inference of twitter: A case for emergency response, mer) states Fake News as *'News reports that are intentionally false or misleading'.* In this work, Fake News in online social media is defined as *'any story circulated, shared or propagated which cannot be authenticated.'*

Thus, going by these definitions, it is posited that Fake News can also include rumors, clickbait, propaganda, satire and parody as the truthfulness of the stories could often times be unverifiable. Several methods have been aimed in the recent past to identify and tackle the problem of fake news. These could be broadly categorised into:

- Content-based: Text (linguistics(Hardalov et al., 2016)); Media (images(Gupta et al., 2013), GIFs and video) and URLs
- User-based: activity tracking (bots and spam (Ferrara et al., 2016)); bio information (registration age(Castillo et al., 2011)); opposing views of other online users(Jin et al., 2016)
- Metadata-based: GPS Geotags, device source, Followers and Friends Network(Tacchini et al., 2017)

Tambuscio et al. (2015) proposed a model for tracking the spread of hoaxes using the four parameters; spreading rate, gullibility, probability to verify a hoax and forgetting one's current belief. Many organisations now employ social media accounts on Twitter, Facebook and Instagram for announcement of corporate information such as earnings report and new product releases. Thus consumers, investors and other stake holders take these news messages as seriously as they would for any other mass media (Kaplan & Haenlein, 2010). Other reasons that fake news has been widely proliferated include for humour, or just to get their readers to click on sponsored content on their websites also referred to as 'clickbait'. This is aimed at unethically driving up their advertising revenues.

2.13 Definition of Fake News

Fake News is defined by Shu et al. (2017) as a news article that is intentionally and verifiably false. While (Stroud, Stroud) refers to it as intentionally false information or propaganda published under the guise of being authentic". Thus, for the purpose of the project; manually or hand-labeled examples will serve as ground information to cross check the veracity of the stance class of the tweets being classified by MISDETECT algorithm. Fake News is synonymous with rumors as they are found to be some form of false information also. However, according to Zubiaga et al. (2018), rumors are circulating items of information whose veracity status cannot be verified at the time they were posted. Further stating that, unlike Fake News, which is always false, rumors cannot be verified at the time they were posted. It can be implied that every fake news then starts off as a rumor. However, not all rumors are fake news as some may be confirmed subsequently as being true. For the purpose of analysis and dataset used for the illustration of methods and techniques adopted for this study, it will be assumed that rumor stories eventually are fake news stories.

It is noteworthy that fake and false information spreads much quicker and deeper than true information. Vosoughi et al. (2018) found 126,000 messages spread by almost 3 million people and found that fake news diffused up to 100,000 people while the truth only reached 1,000 people. This is in multiples of more than a hundred. Hence, its not a surprise that people tend to promote false information online and in some cases the use of social bots. Kumar & Shah (2018) identified that lone wolves spread their message faster by creating fake accounts which express the same opinion in multiple ways to help propagate their message faster. A more effective way of achieving this by using social botnets that retweet and share the same messages indiscriminately to gain popularity and achieve greater spread and coverage.

2.14 Related Works in Misinformation Detection

The work on fake news detection have been initially reviewed by several authors often referring to it the past as 'rumors' not until recently in 2016, during the US presidential elections the phrase became popular with the elected president Donald Trump, Twitter only contains or allows their users to communicate with 140 characters on its platform hence there is only so much that they can say to other people. However those that propagate fake news, rumors and questionable posts have been found to incorporate other mediums to make their messages go 'viral' as was seen in the aftermath of Hurricane

Sandy, Gupta et al. (2013) used a Decision Tree classifier to distinguish between fake and real images posted about fake news events Neural networks are a form of machine learning method that found to exhibit high accuracy and Precision in the clustering and classification of text (Ma et al., 2016). Also it showed effectiveness in the prompt detection of spatio-temporal trends in the content propagation on social media. In this approach, it was combined with the efficiency of the recurrent neural networks (RNN) in the detection and semantic interpretation of images. Although this hybrid approach in semantic interpretation of text and images is not new (Karpathy & Fei-Fei, 2015) (Wang et al., 2016), at the time of writing this thesis, this is the first attempt involving the use of a hybrid approach in the detection of the origin and propagation of fake news posts.

Kwon et al. (2013) identified and utilised three hand crafted feature types associated with rumor classification including (1) Temporal features - how tweet propagates from one time window to another. (2) Structural Features - how the influence or followership of posters affect other posts. (3) Linguistic Features - sentiment categories of words.

Previous work done by Gupta et al. (2013) achieved 97% accuracy in detecting fake images from tweets posted during the Hurricane Sandy disaster in the United States They performed a characterization analysis, to understand the temporal, social reputation and influence patterns for the spread of fake images by examining more than 10,000 images posted on Twitter during the incident. They used two broad types of features in the detection of fake images posted during the event. These include 7 user-based features such as age of the user account, followers size and the follower-followee ratio. Also they deduced 18 tweet-based features such as tweet length, retweet count, presence of emoticons and exclamation marks.

Aggarwal et al. (2012) had identified 4 certain features based on URLs, WHOIs, content and followers networks of tweets associated with the phishing tweets which usually are a problem similar to fake and non-credible tweets but in their case also has the potential to cause significant financial harm illegally to someone who clicked on the links associated with these ‘phishing‘ messages

Yardi et al. (2009) developed three feature types for spam detection on Twitter; which includes searches for URLs, matching of username patterns and detection of keywords from supposedly spam messages. O’Donovan et al. (2012) identified the most useful indicators of credible and non-credible tweets as URLs, mentions, retweets and tweet lengths. Other works on the credibility and veracity identification on Twitter include Gupta et al. (2014) that developed a framework and real-time assessment system for validating authors content on Twitter as they are being posted. Their approach assigns a graduated credibility score or rank to each tweet as they are posted live on the social network platform.

2.14.1 Text-Based Fake News Detection

It would be shown subsequently in Chapter 4 of this thesis, that fake news can be detected using the text-only approach without prior knowledge of the topic domain. It is worth noting that fake and false information spreads much quicker and deeper than true information. Vosoughi et al. (2018) has so far created the largest rumour dataset of 126,000 messages spread by almost 3 million people and found that fake news diffused up to 100,000 people while the truth only reached 1,000 people. Kumar et al. (2018) identified that 'lone wolves spread their message faster by creating fake accounts which express the same opinion in multiple ways to help propagate their message faster. A more effective way of achieving this by using social botnets - that re-tweet and share the same messages indiscriminately to gain popularity and achieve greater spread and coverage. In this chapter, the aim was to explore other semantic and multi-modal signal for misinformation in online social networks.

A conditional random field (CRF) was used by Zubiaga et al. (2016) for text based rumor detection on the PHEME dataset. Chapter 4 of this thesis looks at a hybrid of recurrent neural networks and convolutional neural networks to show that fake news and rumors could be predicted achieving high accuracy without prior knowledge of the topic domain and no feature engineering. Ruchansky et al. (2017) also used a text-based approach for fake news detection but considered the test, response and clustering of user features determined by support vector decomposition and integrated into a hybrid model.

2.14.2 Text Sentiment Analysis

Sentiment analysis also known as opinion mining seeks to understand the effective meaning of sentences and phrases. It assigns levels of classification to declarations made by the authors; also referred to as "polarity". It could be as simple as binary levels such as positive and negative or sometimes neutral level of classification. Similar tools and methods were employed by Baccianella et al. (2010) that used a weak supervised, semi-supervised and random-walk step to create lexicons and bag-of-words sentiments. Similarly, in O'Connor et al. (2010) using moving average of text sentiment scores over a period, established that negative and positive sentiments extracted from users on Twitter are true reflections of voters' confidence and approval ratings of the President. While sentiment analysis from text goes beyond polarity it could also include the determination of the emotional state of the authors such as *angry, anxious, depressed* and *excited*. Some sentiment dictionaries exist to help in the achievement of this task such as Miller (1995) and Hu & Liu (2004). Sentiment analysis from text such as Twitter and blogs are well researched topic areas. However, at the time of writing this thesis, this is the first time emotions and sentiment analysis would be examined in the context of fake news detection in OSN. For the scope of the current work, the sentiment analysis of the text was limited to the negative (false/rumour) and positive (true/non-rumor) polarities of keywords from the text messages.

2.14.3 Machine Learning Algorithms

A range of machine learning algorithms were utilised in the classification and clustering of the data used for the prediction of fake news. The detection of the occurrence of a fake news can draw strengths from a probabilistic learning approach (Conroy et al., 2015) where the models using examples or a *training dataset* are able to *learn* about patterns and build a model which can predict an occurrence from a *testing* dataset based on these previous examples shown to the model.

2.14.4 Classification Models

Machine Learning-based classification has more to do with prediction unlike clustering which has to do with finding out groups and associations within datasets. Examples include Support Vector Machine (SVM) (Tong & Koller, 2001), Logistic Regression (LOGIT) (Pregibonet al. , 1981), Multinomial Naive Bayes (MNB)(McCallum et al., 1998), Decision Trees(Quinlan, 1987), Random Forests (Breiman, 2001) and Artificial Neural Networks (ANN) (Braspenning et al., 1995).

2.14.5 Deep Learning Models

Deep learning models stem from the Artificial Neural networks (LeCun et al., 2015), however they include the use of multiple layers for the training of the model and often times generate better prediction with larger datasets (Goodfellow et al., 2016). Different architectures have been proposed for Deep Learning and they include - Recurrent Neural Networks (RNN) (Mikolov et al., 2010) and Convolutional Neural Networks (CNN) which was initially developed and applied to image classification (Krizhevsky et al., 2012). CNN have been successfully applied to text classification (Kim, 2014) and Hierarchical Attention Neural Network (Yang et al., 2016). Deep learning models used in this work include the RNN, CNN and HAN.

2.15 Discussion

Location inference can be applied to many areas and its applications include cyber-bullying prevention, disaster management and in public health event prediction. The importance and popularity of location-based social networking services continues to grow as billions of videos are being uploaded daily and shared worldwide on Twitter and other social networking platforms. It has been reviewed in this chapter that there has been improvement in granularity level of inference of user locations often achieved achieving better results where hybrid techniques are adopted. This work improves on the performance of previously done work, proposing a grid-based content-only location inference technique would be adopted. This approach addresses the sparsity problem associated with various other machine learning techniques. The inclusion of the similarity measure processing of the tweets removes redundancy and would help in dimensionality reduction of the feature vector. In addition the task of misinformation detection would greatly benefit from the findings of the work done in Chapter 5 and in Chapter 3 on location inference - As this could be applied in the detecting the origin and geolocation of users who spread misinformation posts.

The various metrics adopted in location inference include the use of Precision, Recall, F-Measure, Average Error Distance (AED) and Median Error Distance (MED). Types of spatial clues in OSN messages include URLs, text, Points of Interests, location field, IP addresses, friends network and time zones. while third party sources such as Foursquare allow the tracking of users with links to other OSN services such as Twitter.

Chapter 3

Content Aware Tweet Location Inference using Quadtree Spatial Partitioning

Inferring locations from user texts on social media platforms is a non-trivial but challenging problem relating public safety. This work proposes LOCINFER - a novel non-uniform grid-based approach for location inference from Twitter messages using Quadtree spatial partitions. The proposed algorithm uses Natural Language Processing (NLP) for semantic understanding and incorporates Cosine similarity and Jaccard similarity measures for feature vector extraction and dimensionality reduction. Twitter was chosen as the experimental social media platform due to its popularity and effectiveness for the dissemination of news and stories about recent events happening around the world. This approach is the first of its kind to make location inference from tweets using Quadtree spatial partitions and NLP, in hybrid word-vector representations. The proposed algorithm achieved significant classification accuracy and outperformed state-of-the-art grid-based content-only location inference methods by up to 24% in correctly predicting tweet locations within a 161km radius and by 300km in median error distance on benchmark dataset.

3.1 Introduction

The task of inferring users' locations on Twitter as well as most social media platforms is non-trivial spurring the interests of many researchers in the field of artificial intelligence, computer science and behavioural sciences alike for almost a decade. Studies show that only less than 2% of Twitter users disclose or geotag the location of tweets (Leetaru et al., 2013) (Li et al., 2012) due to fears of being tracked by online predators thus preserving their personal safety or by advertisers that use cookies to continually send them often times unsolicited product advertisements that have been personalised to their tracked location. Some social media sites even offer tailored location-based services such as Snapchat offering a new addition called SnapMap[1] where one can track the location of friends using the App and even know the status of their current activity including if they are sleeping or in ridding a car or shopping. These information are quite private and the users may not even be aware they have provided such information which could lead to stalking and posing threats especially for children (Field, 2017). However location tracking of the online users also has benefits relating public safety and security.

The growing threat of online crimes ranging from messages focused at propagating hatred, to cyberbullying and spread of fake news and false information for the purpose of promoting malicious selfish intentions; personal or political gains have continued to cause governments, corporate organisations and individuals cause for concern. Social media is a good tool for the promotion of information but the fact that it is uncensored - stemming from the notion of freedom of speech which obtains in most democracies tend to be abused. It is crucial that law enforcement bodies are able to track down the location of these offenders and the origin of these messages to curtail their spread before they begin to 'infect' the behaviours and actions of other online users.

The large footprint of Twitter makes it an important marketplace for advertisers to reach their consumers, and serve as projection platforms for the government to its citizens. Knowledge of users who interact on Twitter may be quite useful for organisations that render these services. There exist third party domains and other sources such as knowledge bases. These sources amongst others are useful for estimating user locations (Ajao et al., 2015). However, they may be unreliable and insufficient for effectively estimating the location of users. This brings the need to infer locations from transmitted messages solely based on the content alongside other relevant metadata information captured with the tweets such as user description and time zone information

[1] `www.snapchat.com`

etc.

In this chapter, there's a proposal for a novel non-uniform grid-based approach for location inference from Twitter messages combining quadtree spatial partitions and semantic understanding using Natural Language Processing (NLP). The contributions made in this chapter is given as follows.

- A discriminative grid-based approach for the determination of tweet locations based on the content,
- A Quadtree spatial indexing technique for inferring locations based on variable nodes,
- A NLP based hybrid word embedding model consisting of Cosine and Jaccard similarity measures (Huang, 2008) for dimensionality reduction in the feature vector, and representation (Christopher et al., 2008).
- Improvement in city-level grid-based location inference based solely on the content of Twitter messages.

Location inference also referred to in literature as *Geolocation Prediction* has enjoyed a fair amount of research interests by several authors working within the space. A few works have been written on the inference of location of Twitter users. The one most related to this work is (Cheng et al., 2010) where the authors estimated user locations solely based on the content of their messages using supervised classification. The work extracted *local words* from the messages of users with the assumption that users from specific geographic locations would normally use words that are *local* to that geographic location. For example the word *howdy* which is *hello* in English would be considered to be more frequently used in the US state of Texas. However, the authors did not actively seek out to recognise entities such the names of people, places and organisation within twitter messages as part of their location inference technique unlike the proposal in this chapter. It should be noted that some of such local words they identified could also be geo-entities, for instance their probabilistic method identified the word *ucsb* to show a peak distribution around the state of California as this was the abbreviation for the University of California located in the city of Santa Barbara.

Location inference and privacy of geo-spatial data have always been an area of concern. Krumm (Krumm, 2007) examined the identification of users from web search data and was able to successfully identify their locations to the granularity level of home addresses from GPS data. This is possible due to the very high degree of correctness that GPS data typically offers. However, the availability of location information is not always guaranteed which introduce additional challenges. The proposed approach aims to address this issue by inferring the user's locations to a city-level accuracy by analysing users texts available from social media. Privacy continues to be an emerging area of research discussion with people choosing to hide their online identities to keep an anonymous profile from other users and in some cases for the safety and the fear of being *trolled* online by cyberbullies especially in the social media and Twitter in particular. (Han et al., 2016)

Han *et al.* (Han et al., 2014) used words referred as *Location Indicative Words* (LIWs) and provided a spatial clue to indicate the whereabouts of the users. It was proposed that users were more likely to be successful in preserving their privacy if they refrain from mentioning these LIWs in their online conversations and also to actively delete location meta data from their online footprint. This seems far from being realistic as users are most likely to be tracked by the social media platforms who passively collect and retain time-stamped information such as time-zones and IP addresses of their users, Most of these meta data is then made available to the public via the Twitter API and can be linked it to the users who created them. Other work done in the field can be found in (Ikawa et al., 2012) that proposed a method which learns association from locations and keywords from previous user messages to predict subsequent messages. The challenge with this method is that to effectively train a location classifier the past tweets of a user would have to be collected and analysed and may be prove to be technically unfeasible because at the moment the Twitter REST API only allows the retrieval of the last 3200 messages of any user. Secondly there is the possibility that users can relocate over time from one city to another or even from one country and time zone to another. Thus online themes and conversations that they tweet about today may be different tomorrow. Our approach is not user-specific and relies on word-usage and geo-entities associated with locations.

Jurgens (Jurgens, 2013) applied label propagation of location assignments to the knowledge of locations. The work relied on the friends connections also known as their *ego* network locations including self-reported ones found in the free-text fields of the user profiles. Compton *et al.* (Compton et al., 2014) inferred location from the friends network with known locations. Their work presented the largest dataset utilised till date for the training and testing of their location inference classifier accounting for tweets captured from over 100 million Twitter users. Chang *et al.* (Chang et al., 2012) used Gaussian mixture models and Maximum Likelihood Estimation (MLE) which is purely content-based in addition to the use of local words distribution within messages. Mahmud *et al.* (Mahmud et al., 2014) used an ensemble of statistical and heuristic classifiers. Their approach also followed a hybrid of both tweet content and social network profile information including the friends networks. Chapter 2 gave an insight into a range of clues for estimating user locations in addition to the message content. They outlined three various locations that had been inferred in on Twitter including tweeting location, user home residence and message context that have mentions or references to certain geographical locations or points of interests. Various partitioning algorithms are proposed in the literature to infer Tweet locations including k-dimensional trees (Roller et al., 2012; Wing & Baldridge, 2014) or uniform grids (Hulden et al., 2015; Wing & Baldridge, 2011). A further breakdown of reported results from related works is presented in Table 3.1.

Table 3.1: Methods and Outcomes from Related Works in Twitter Location Inference

Author	**Input**	**Method**	**Technique**	**ACC(%)**	**Radius**
Cheng *et al.* (Cheng et al., 2010)	content	words	Probabilistic(ML)	51	160km
Eisenstein *et al.* (Eisenstein et al., 2010)	content	geo-topic	Geo-Topic Model	24	State
Wing *et al.* (Wing & Baldridge, 2011)	content	locations	Grid-based(Uniform)	-	-
Kinsella *et al.* (Kinsella et al., 2011)	content	locations	Language Models	13.9	Zip Code
Kinsella *et al.* (Kinsella et al., 2011)	content	locations	Language Models	29.8	Town
Ikawa *et al.* (Ikawa et al., 2012)	content	words	ML classification	20-60	10-30km
Chang *et al.* (Chang et al., 2012)	content	words	GMM & MLE	49.9	160km
Roller *et al.* (Roller et al., 2012)	content	locations	Grid-based(kd-tree)	34.6	160km
Li et al(Li et al., 2012)	content, network	hybrid	Probabilistic(ML)	66	160km
Schulz *et al.* (Schulz et al., 2013)	content, context	hybrid	Gazetteer	-	-
Compton *et al.* (Compton et al., 2014)	Network	closeness	Network	80	
Mahmud *et al.* (Mahmud et al., 2014)	content, context	locations	Ensemble classifiers	58	city-level
Wing *et al.* (Wing & Baldridge, 2014)	content	locations	Grid-based(kd-tree)	90.2	160km
Ryoo & Moon(Ryoo & Moon, 2014)	content	words	Location services	56.7	10km
Hulden *et al.* (Hulden et al., 2015)	content	words	Grid-based(Uniform)	-	-
Han *et al.* (Han et al., 2016)	content	words	Neural Net	40.9	-

It is posited that the task of location inference from tweets and other sources which involves the use of text, relies on natural language processing models and machine learning techniques to understand the semantics. There are over 500 million messages sent by Twitter users each day[2]. Thus, it is humanly impossible to manually sift through the contents of these messages and make meaning of them. NLP models such as word embedding and pattern recognition capabilities of machine learning models are useful in the identification of patterns (Zhong et al., 2012) within the text. This helps in machine understanding of the human language and drawing insights suitable for the process. NLP methods applied in this chapter includes the use of the continuous bag of word (CBOW) model (Mikolov et al., 2013) for embedding the words into vectors. Additionally, Jaccard similarity and Cosine distance of word vectors (Cha, 2007; Huang,

[2]`www.twitter.com`

2008) was computed for feature extraction and word dimensionality reduction to get prediction-relevant text. At the time of writing this thesis LOCINFER is the first to use Quadtree spatial indexes in combination with NLP for content-aware location prediction on Twitter.

(Cha et al., 2015) used sparse coding and dictionary learning (PCA whitening, feature augmentation and voting-based grid selection). While in terms of predicting Twitter locations in real-time (Yamaguchi et al., 2014) proposed a solution that constantly infers location of users from the social stream. (Zheng et al., 2017) categorised location inference into the prediction of user home locations, tweet locations and the mentioned locations.

The task of location inference from tweets and more specifically the method that relies on the use of text bears a lot of similarity from Natural Language Processing (NLP) techniques. Considering the vast amounts of messages being posted onto Twitter each day, it is humanly impossible to sift through the contents of these messages and make meaning of them. However, machine learning models are useful in the identification of patterns thus helping in the understanding of the human language and drawing insights suitable for the process. Hence the adoption of NLP techniques proves invaluable for this procedure. However, when applying NLP methods caution has to be exercised this is due to the fact that tweets do not necessarily imply the exact same resemblance to text found in blogs and corporate or news websites. There is an air of informality on social media openly embracing the use of abbreviations, sarcasm, irony and non-conventional text such as emojis and emoticons. These tend to mislead machine learning classifiers and constitutes as noise in the task of training or testing the algorithms. Also, the brevity of standard Twitter messages only constitute of 140 characters each thus limiting the user expressions. Some users especially government and corporate accounts have found a way around this brevity by breaking down a single message into multiple tweets e.g. a lengthy message with 700 characters can be transmitted in 3 parts and sent as 1/3, 2/3 and 3/3 etc.

3.2 Methodology

LOCINFER proposes a new grid-based approach for location inference from Twitter messages using quadtree spatial partitions. The proposed algorithm incorporates Cosine similarity and Jaccard similarity measures for NLP-based feature vector extraction and dimensionality reduction. The summary of the illustration of the approach towards content-based location inference by LOCINFER is given in Figure 3.1 and described in detail in this section.

A functional block diagram of the proposed algorithm is depicted in Figure 3.1.

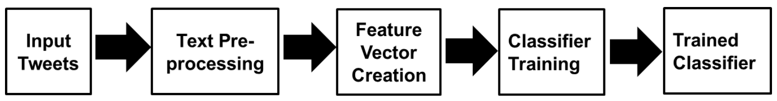


Figure 3.1: Functional diagram illustrating the tweets location inference task.

3.2.1 Uniform Grid Clustering versus Discriminate Partitioning Technique

In the determining of the location of the tweets, following a supervised machine learning approach, *classes* or *labels* need to be created for the classifier. These classes would be the location targets in the prediction task. Although all the tweets would be geotagged as part of the training and testing data input into the classifier. The geotag of each tweet may be different and also the varaince of the location of each tweet may not be uniform. For example, tweets collected from users in the united States may be across the entire continental which would be quite a large spread.

Another important need for having labels of some sort is that it helps improve classification accuracy. For example, if 670,000 geotagged tweets are collected in one corpora, such as in the case of the GEOTEXT dataset. Trying to predict the location of all the 10,000 users which sent these messages would be practically impossible as the number of labels would have been too much for the classifier. This would be extremely computationally expensive to run in terms of the time and computing power required to execute this task.

However, by simply dividing the plotted geotags into 'regions' or 'grids' following a row-major ordering, it would help create clusters of location labels. An illustration of the uniform grid approach is shown in Table 3.2. This is the most naive form of tweet clustering aimed at being used as prediction targets or labels for location inference classifiers. They include clusters of uniform squares of 4x4, 8x8, 16x16, 32x32. The lattices named A, B, C and D respectively each have grids that contained individual tweets having their geotags of latitudes and longitudes fall into the derived polygon. The more the number of grids created, the higher the degree of granularity, accuracy and detail required of the classifier. For example, very large grid sizes as in Lattice A with only 16 grids could be prediction to the level of a timezone such as East Coast or

Table 3.2: Grids of Message Geotags

4 × 4 Lattice (A)				**8 × 8 Lattice (B)**				**16 × 16 Lattice (C)**				**32 × 32 Lattice (D)**			
G1	G2	...	G4	G1	G2	...	G8	G1	G2	...	G16	G1	G2	...	G32
G5	G6	...	G8	GG9	G10	...	G16	G17	G18	...	G32	G33	G34	...	G64
.	.		.	.	.		.	.	.		.	.	.		.
.	.		.	.	.		.	.	.		.	.	.		.
.	.		.	.	.		.	.	.		.	.	.		.
G12	G13	...	G16	GG57	G58	...	G64	G241	G242	...	G256	G993	G994	...	G1024

the West Coast. In this case, not much precision is required of the classification task. Similarly, very small sized grids applied on the same US dataset as seen in Lattice D (1,024 grids) could give a higher level of precision and granularity to as much as the post code level.

While a finer grained cluster of training data could aid the classifier to give more precise prediction, there is a major restriction that could hinder the achievement of this. The main challenge is the tweet geo-sparsity problem. As it is seen that the tweets collected have close similarity to the demographic population of the united states cities of under 5000 people. The tweet distribution aligns with the demographic spread of the United States. For example, there are more tweets collected around New York as opposed to Seattle. This is because the former is more densely populated than the latter. Thus it is expected that more users and messages would be sent from around New York than from Seattle.

In this instance, the application of a uniform grid approach to the map would mean tweets sent from the east coast are going to have a lot of tweets while those from the west may even have empty or very few thereby creating an unfair class imbalance. To address this challenge, a discriminate partitioning technique is required which would introduce a bias in determining non-uniform sizes of the grids based on their density. The proposed discriminate technique is the Quadtree approach.

3.2.2 Text Preprocessing

As the first step in the processing pipeline, perform text pre-processing was done. In the various datasets the words serves as the features while the grids served as the labels for each of the tweets. Normally a tweet would contain 140 characters (from November 2017 this is now extended to 280 characters). However there was a need to perform feature reduction to obtain only words relevant in determining the geo-spatial properties of the words. This includes text cleaning and character processing

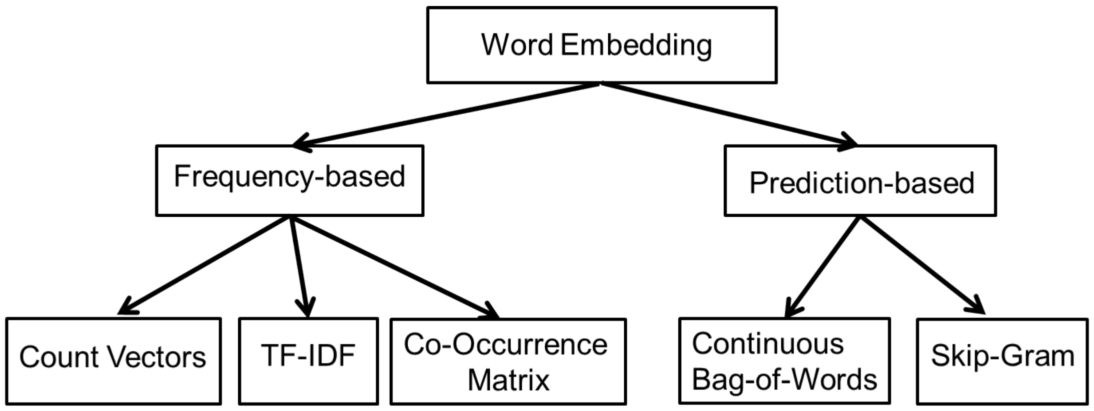


Figure 3.2: Methods of Word Embedding in Natural Language Processing

such as, *a)* removal of duplicate tweets, blank tweets, URLs/hyper-links, user-names, stop-words and numbers; *b)* normalisation of words; *c)* word stemming; *d)* handling punctuations or *e)* tokenisation. Description of these preprocessing techniques are collated in Table 3.3.

3.2.3 Converting Clean Tweets to Word Vectors

A neural word embedding model called word2vec was adopted for converting the word tokens extracted from the clean pre-processed tweets into numerical form (vectors) serving as input for the machine learning classifiers. This was chosen as it has achieved recent success in word embedding for text mining and natural language processing tasks. The ability of Word2Vec to perform algebraic operations and vector additions makes it suitable in representing words in dimensions as well as the context in which they have been used. Using such approach, allows the discrete state in which words normally occur be better understood by examining the transitional probabilities between these states. This implies that not only can similar-looking words be identified, also the context in which they have been used can be discovered. This provides a form of similarity discovery based on word usage in the vocabulary. This intuitive function that makes it quite useful also in deep learning models such as Recurrent Neural Networks (RNNs). For the purpose of this work, two similarity functions which exist in word2vec were harnessed in building the word vectors; namely the Jaccard Similarity measure and Cosine Similarity. The former looks at how identical any two sets of word tokens are while the latter measures the angular distance between the word vectors. Similarity scores for both functions range between 0 and 1.

Word2Vec has the potential to use either of two types of approach to predict or compare

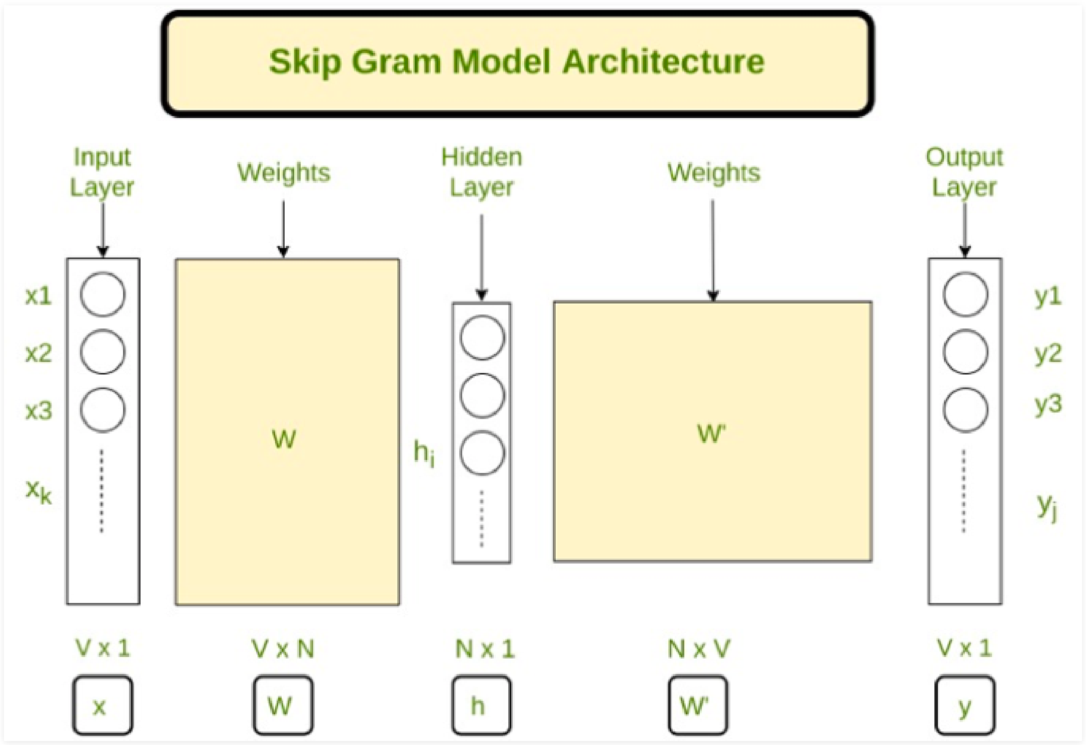


Figure 3.3: Skip Gram Model Architecture for Word2Vec

target words namely, the continuous bag of words (CBOW) which uses context to predict the current word or the continuous skip gram predicts surrounding words given the current word. The latter has been found to be more effective in learning word vector representations in unstructured text (Mikolov et al., 2013) (Mikolov et al., 2013). Thus the continuous skip-gram approach is the one used in this work. The skip gram model architecture is given in Figure 3.3. This illustrates how a typical word tokens from the tweets would be converted into vectors by the architecture.

The basic illustration of the skip-gram input-output operation in Word2Vec is given in Figure 3.4. The input is the *center word* while the context words are the prediction targets. Given W is an array of words, selecting a sliding window size of 2 words, if W(i) is the input (*center word*), then W(i-2), W(i-1), W(i+1), and W(i+2) will be the context words.

An example of word prediction from a sentence is given in Figure 3.5

The variables in the diagram are explained formally and thus; Given V unique word tokens in the tweet corpus, x Input layer N Number of neurons in the hidden layer of neural network W Weights between input layer and hidden layer h Hidden layer W'

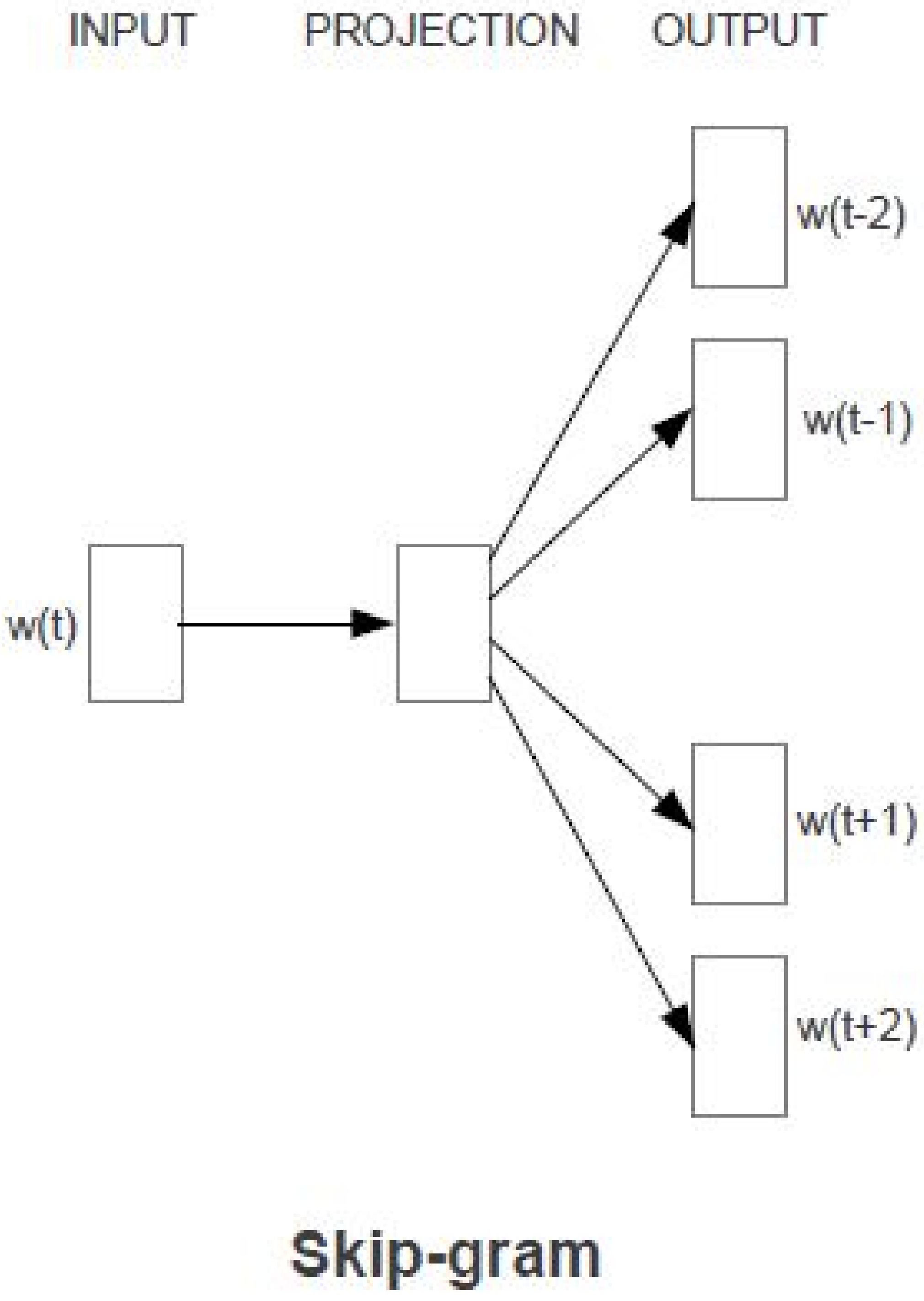


Figure 3.4: Basic Skip Gram Input-Output Illustration

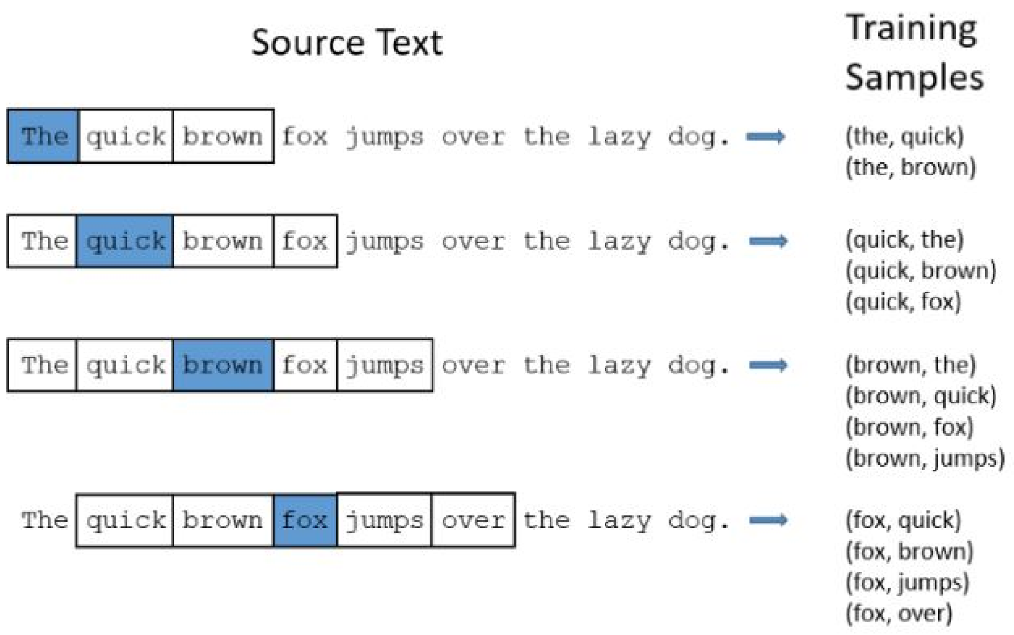


Figure 3.5: SkipGram Sentence Example

Weights between hidden layer and output layer y A softmax output layer

3.2.4 Feature Vector Creation using NLP

The proposed work incorporates natural language processing methods in creating the feature vector. This includes *Word Embedding*, where the words are converted to numbers in order to process them effectively (Mikolov et al., 2013) and forming the vector representation of those words. There are two broad categories of word embeddings namely, *a)* frequency-based and *b)* prediction-based word embeddings as seen in Figure 3.2. Available models for word embedding include the continuous bag-of-words CBOW model (Zhang et al., 2010), Word2vec model (Mikolov et al., 2013) and Glove model (Pennington et al., 2014). Prediction based word embedding techniques such as *word2vec* are proved to be the state-of-the-art technique and have the advantages over deterministic methods such as conventional bag-of-words or count vectors. These also have the ability to incorporate neural networks improves their performance compared to their predecessors. For LOCINFER the *word2vec* model proposed by (Mikolov et al., 2013) was adopted.

The NLP-based text processing for feature vector creation includes following three central steps:

Calculation of linear vector: Linear vector calculations are implemented on feature vectors using *word2vec*. An example of this is *King - Man + Woman = Queen*. This is inherent in the fact that once words are converted into vectors they lend themselves to algebraic and mathematical operations thus revealing the association and relationships that exist between them. In the above example the gender is the relationship between them.

Identification of synonyms used in the messages: Words that have the same semantic meaning are given the same representation. In essence it looks out for word synonyms avoids redundancy and significantly reduces the size of the feature vector and computing time. For example the two sentences $S_1 = \{$*That is a small thing*$\}$ and $S_2 = \{$*That is a little thing*$\}$ will be considered the same, thus improving the effectiveness of their respective word-vector representation.

Determination of similarity threshold: Similarity thresholds can be specified where the distance between the feature vectors is measured with the cosine similarity and Jaccard similarity functions (described in Eq. (3.1) and Eq. (3.2) respectively). In this regard, words that have similar syntactic appearance but were however mis-spelt, exaggerated or abbreviated would be recognised and given the same representation within the vector space. This can be achieved by the cosine function to compare their similarity with the English language dictionary. For example, *Yeeeees* is equivalent to *Yes* or *Gooooood* is recognised as *Good*.

In order to measure the closeness between the word feature vectors two types of similarity measures (Huang, 2008) were used namely, *a) Cosine similarity* and *b) Jaccard similarity* as described below. Considering two non-zero vectors, p and q, each having component values $1, 2, ...n$, their cosine similarity $(Sim_c(p, q))$ is calculated in Equation 3.2 and Equation 3.1 respectively.

$$Sim_c(p, q) = \frac{p.q}{||p||\,||q||} = \frac{\sum\limits_{i=1}^{n} p_i.q_i}{\sqrt{\sum\limits_{i=1}^{n} p_i^2}\sqrt{\sum\limits_{i=1}^{n} q_i^2}} \tag{3.1}$$

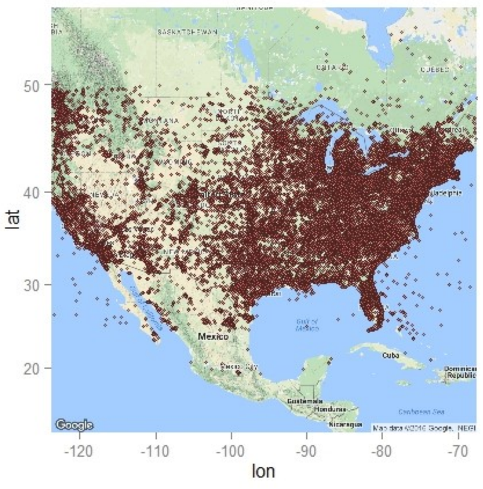


Figure 3.6: Geo-located US Tweets captured in the simulation dataset

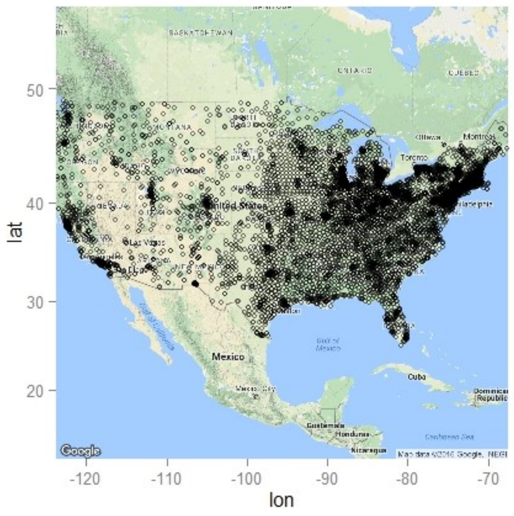


Figure 3.7: US cities with population over 5,000.

For the same vectors the Jaccard similarity ($Sim_j(p, q)$) is calculated by:

$$Sim_j(p, q) = \frac{|p \cap q|}{|p \cup q|}. \tag{3.2}$$

3.2.5 Sparsity of Tweets and Quadtree

To perform exploratory data analysis and investigate how the simulation dataset of 730,000 geotagged tweets was spatially distributed in terms of their geographical representation, there was a need to plot the actual latitudes and longitudes of these tweets onto the map of the United States. Since the Twitter API had the geographical boundary box setting restricted to the continent of the United States at the time of the data collection. Using the GGPLOT2 library in R programming language [3], the graphical plot of the tweets onto the map of the United States is illustrated in Figure 3.6.

Similarly, Figure 3.7 which was also created using GGPLOT2 in R, show the plot of US cities with a population density of 5000 or more. The figures used were for the United States Census Bureau at the 2010 US census (Agency, 2013). Each point on the plot are equally represented as long as the city was recorded to have a population that exceeded 5000 people. This approach follows a similar method which was adopted by Cheng et al. (2010).

Comparing Figure 3.6 and Figure 3.7, it can be seen that the tweeting geo-locations bear close resemblance to actual population demographies, showing similar density patterns. Considering that the continental United States has a total geographic area of 6,110,264 square miles (Agency, 2013), it should be noted that this challenge with the dataset is due to the geographical outlay of the country as some areas were more inhabited than others. Thus tweets were considered to have a sparse distribution in some areas. As such when the map was uniformly split into grid sizes (G_i), as depicted in Table 3.2, some grids contained too little dataset to be used for training the classifier while some other grids contained too much tweets. This presented a major limitation in the estimation of location of the users using a uniform grid approach.

Evaluation results in terms of uniform grid classification precision on the simulation dataset presented in Figure 3.8 shows the strong correlation between the number of tweets within each of the grids namely 4x4, 8x8, 16x16 and 32x32 all done using the uniform spatial partitioning method illustrated in Table 3.2. The precision value was plot against each of the 4 split grids. There was a direct relationship between the log-value of the counts of observations each uniformly partitioned grid and their precision. This implies the presence of a bias favouring highly populated grids while the less populated ones got lower precision.

Following this observation, LOCINFER was used to cluster the dataset in a biased manner now dependent on the counts of observations within each grid. This created more effective labeled training dataset for the classifiers. In contrast to using a uniform splitting approach, Quadtrees being hierarchical spatial data structures (Mehta & Sahni, 2004) offer a solution that dynamically addresses the sparsity problem by discriminatively splitting denser parts of the map into smaller grids while the more

[3] `https://www.rdocumentation.org/packages/ggplot2/`

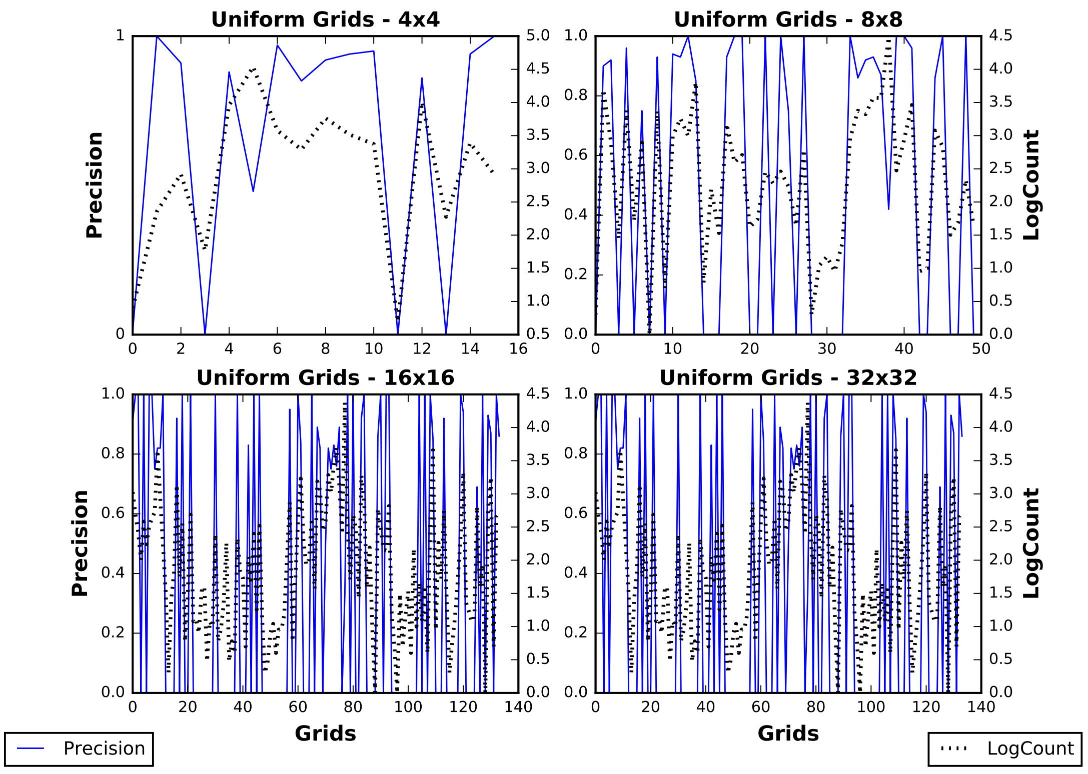


Figure 3.8: Interaction between precision and log of uniform grid counts

sparsely distributed parts would then be captured in larger grids.

3.2.6 LOCINFER and Quadtree Data Partitioning

LOCINFER implementation considered a variable resolution constraint which can be adjusted. As a result, it addresses the bias mentioned earlier in Section 3.2.7. For empirical purposes and during the decomposition process, the maximum number of points in each grid was set in multiples of 5000, *i.e.,* 10000, 15000, 20000 etc. It can be seen in Figure 3.9 As the dataset is now more fairly split across more grids it was observed significant correlation between the log values of the grid counts as well as improvement in the level of accuracy, Average Error Distance (AED) and Median Error Distance (MED) which are further described in Section 3.3. The rationale for choosing these sizes was based on previous experimental trials done with random sample sizes.

3.2.7 Tweet based bias removal

In addition to the Quadtree partitioning, a population based bias removal technique was adopted. The geo-spatial visualisation of the US tweets in Figure 3.6 indicates more visible activities towards the North East of the country; this bears a true resemblance of Figure 3.7 which illustrates the population of the United States (US Census Bureau, 2016) as discussed earlier. This implies that there is a bias favoring a larger count size as opposed to less dense grids. This is clearly a problem due to the sparse distribution of the tweets and as seen from the geographical map, tweets on the East coast (around New York etc tend to have normally a larger population density and thus more user tweets are included in the training data for this purpose). In order to further remove this bias a weighted measurements of the outcome and incorporated this within the measurement metric to be further discussed in Section 3.3 was used in Figure 3.9 was generated using LOCINFER Quadtree structure.

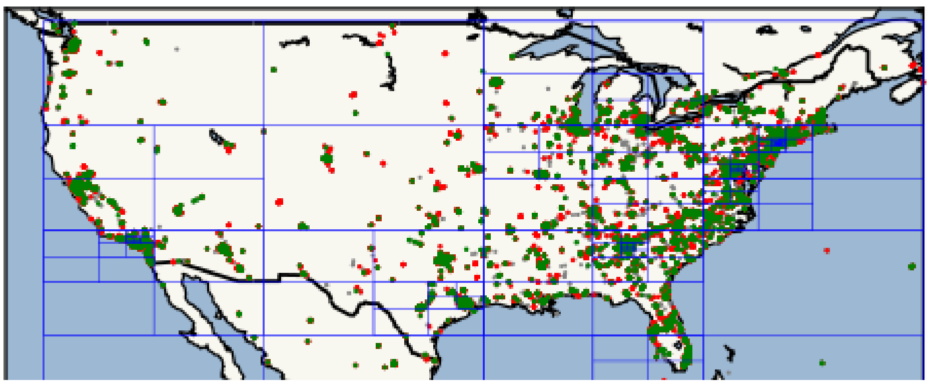


Figure 3.9: Geo-located US Tweets partitioned with LOCINFER Quadtree algorithm

3.2.8 Training of Location Classifier

The task of content-based location inference can be interpreted a classification problem. A number of machine learning classifiers (Han et al., 2011) were examined including the Logistic Regression (Maximum Entropy), Random Forests, Decision trees, Artificial Neural Networks and the Multinomial Naive Bayes (MNB) classifier for supervised classification of more than 730,000 messages geotagged to the continental United States. These served as preliminary investigation before the training on baseline datasets namely GEOTEXT(Eisenstein et al., 2010) and UTGEO-Small(Roller et al., 2012) indicates better performances by Multinomial Naive Bayes and Logistic Regression which are also commonly used in similar dataset by other researchers. This subsection briefly revisited these two classifier before reporting the results. In the classification, the words served as the features while the grids served as the labels or predicted results of the task. In training and testing the classifier, 75% of the data was randomly split into training while the rest 25% used for testing.

3.2.9 Data

The benchmark datasets used include the UTGEO-Small Dataset(Roller et al., 2012); This consists of 670,000 geolocated twitter messages from the continental united states. This was collected and used as part of the training the model. Also the GEOTEXT dataset of (Eisenstein et al., 2010) sufficient for the evaluation of the two MNB and Logistic Regression models. This one was also comprised of geolocated twitter messages of approximately 380,000 messages collected from twitter. Both datasets form the baseline datasets for comparison with the LOCINFER technique.

3.3 Results and Discussions

This section describes the measurement metrics that were used for evaluation of the LOCINFER technique, the results and related discussions.

3.3.1 Experimental Results and Discussion

The summary performance of LOCINFER, measured against various metrics such as average error distance (AED) as calculated in Eq. (2.9), median error distance (MED) and predicted accuracy to the nearest 161km from Eq. (2.10) and Eq. (2.11) respectively are shown in Table 3.6. A detailed breakdown of each of the classifiers (MNB) and (Logit) for both GEOTEXT datasets is given in Table 3.4 while that of the UTGEO-Small dataset is given in Table 3.5.

On the GEOTEXT dataset from Table 3.4 LOCINFER achieved significant improvements as the grid counts reduces from 20,000 all the way to 5,000 tweets in all three metrics specifically MED of 39.15km, AED of 598.44km and 59.47km performing better than methods that had been applied on the same dataset (Eisenstein et al., 2010), (Wing & Baldridge, 2011), performing better than (Hulden et al., 2015) by more than 150km in AED and almost 300km in MED Similarly, from Table 3.5 and 3.6 LOCINFER performed better than (Roller et al., 2012) by 24% in terms of ACC@161, more

400km better prediction for median error distance and over 250km more accurate average error distance.This implies that it was unable to go beyond the maximum grid size and a granularity level finer than 5000 tweets as this could be lead to overfitting of the training data.

In terms of the computing time to execute both methods using LOCINFER algorithm, it was seen from Table 3.4 and Table 3.5 that on both datasets the MNB was quicker to implement in terms of the processing time than the Logit which took a little bit longer as it was more computationally expensive. The implementation of both machine learning classification was done on Windows i7 desktop processor. The difference in processing time can be seen in Table 3.5 and Table 3.4. On average about 20 minutes longer to execute. While LOCINFER performs better on all metrics namely AED, MED and ACC@ 161, it should be noted that the AED is less influenced by anomalous values in the training dataset as it relies on median values. However, MED should be given more consideration over the AED as the latter can be affected by a range of very low and very high error distances thus not giving a fair assessment of the classifier performance.

This work also shows the performances of the method with and without considering demographic biases as discussed in 3.2.7. It is evident the performance has improved significantly while bias was removed using a weighting parameter that is proportional to the demographic distribution. Finally LOCINFER was compared with other grid-based methods in Table 3.6. The result indicates that LOCINFER outperformed the existing grid-based location inference techniques on Twitter. Showing significantly better results in terms of AED, MED and Accuracy at an error distance of 161km radius.

3.3.2 Comparison of Classifier Performance

Evaluating the performances of the two classifiers and how well they adapt to grid sizes, From the table of the optimal grid values, it can see that the LOGIT classifier does a better job than the MNB classifier in terms of achieving higher precision from less populated grids even as low as 300 tweets in a grid for the 5k partition. This performance is consistent across both datasets of UTGEO and GEOTEXT.

3.3.3 Behaviours of the LOGIT and MNB Classifiers

For the LOGIT classifier, a general trend that can be observed from the 4 sets of quadtree graphs plotted in quadruples; at that the onset, there is an inverse relationship between the precision and recall. It is observed that irrespective of the classifier used, provided the level of true positives (TP) stays constant, as the population of the clustering grids increases, then the following happens based on the expressions in equation (1): - False Positives also increase leading to lower Precision (bad) - False Negatives decrease leading to higher Recall (good)

The MNB classifier tend to follow more the trend of Precision, Recall and FMeasure right from small to large grid sizes. This is a contrast to the LOGIT classifier. As the grid counts further increases, an equilibrium point is reached. At this point, precision, recall and F-measure seem to reach a close value for all of the three metrics. These points is denoted with vertical red lines in graphs of Figure 3.10, 3.11, 3.12 and 3.13.

3.4 Performance of the Classifiers on Various Datasets and Splitting Criterion

In evaluating the performance of the classifiers, there was a comparison of how the various grid clusters/sizes of 20000 tweets (20k), 15000 tweets (15k), 10000 tweets (10k) and 5000 tweets (5k) influenced the various metrics of precision, recall and F-Measure. These performances were also evaluated against the two benchmark datasets - namely the UTGEO-Small and GEOTEXT datasets respectively. The results of these evaluations are plotted in 4 graphs (Figure 3.10, Figure 3.11, Figure 3.12 and Figure 3.13).

The black dotted line on each graph is the tweet count within each grid. For purposes of clarity, the clustered dataset has been sorted by grid size in ascending order such that the metrics of smaller grids are plotted first on the graph which leads to a positive slope as the grid counts are plotted as black dots on each of the graphs. Precision, Recall and F-measure metrics are represented as dashed blue, dashed orange and solid green lines respectively. Each graph has two Y-axes; one on the left and the other on the right; one of the vertical axes is for the grid counts while the other axis is a measure of the Precision, Recall or F-Measure; this metric value would range from 0 to 1. The dashed red vertical lines represent the region of convergence of the three metric values. While the x-axis denotes the labels of each grid.

Also, it can be observed that the corresponding regions of convergence vary from one classifier/dataset to another. Further discussions about each of these graphs and their findings are presented in subsequent sections. A maximum splitting value for the algorithm was set. This implies that by the generic nature of Quadtrees, no minimum grid count would be set. Thus, some grids could have very few tweet counts relative to other grids under the same split criterion

Generally, it can be seen from all the four charts that the larger the specified tweet counts per grid, the fewer the number of grids that the data is clustered into. On the UTGEO-Small dataset as shown in Figure 3.10 and Figure 3.12, setting a criterion of 5k splits the dataset into approximately 420 grids, 10k resulted into 200 grids, 15k produced 150 grids and 20k gave 115 grids. While the splits of the GEOTEXT dataset illustrated in Figure 3.11 and Figure 3.13 shows the approximate number of grids to be 250, 120, 80 and 60 under 5k, 10k, 15k and 20k split criterion respectively.

3.4.1 Performance of LOGIT Classifier on UTGEO-Small Dataset

Figure 3.10 presents the performance of the LOGIT classifier on the UTGEO dataset under each of the four splitting criterion of 5k, 10k, 15k and 20k respectively. Smaller grids tend to give unreliable results this is evident by the large disparity between the precision, recall and measure. For these grids, precision appeared to be quite high; in some instances as high as 1 while the values of recall and f-measure for the predicting the same grids were quite close to zero. However, as the grid sizes increased from left to right on the graph, it can be seen that the large disparity between the metrics tended to get smaller, leading to points of convergence where grid sizes ranged from 2500 to 3500 tweets. These converging points were around Grid labels G380, G170, G135 and G95 for 5k, 10k, 15k and 20k splits respectively. As expected, the smaller the splitting criterion, the smaller the grid size at the point of convergence for example around 4000 tweets for the 4k criterion in grid G380 and around 14000 tweets for the 20k splitting criterion in grid G95.

3.4.2 Performance of LOGIT Classifier on GEOTEXT Dataset

Figure 3.11 presents the performance of the LOGIT classifier on the GEOTEXT dataset under each of the four splitting criterion of 5k, 10k, 15k and 20k respectively. Also, initially on the left hand side of the graph, the smaller grids were plotted first with their metrics and grid sizes, again very high fluctuations at the start which gradually

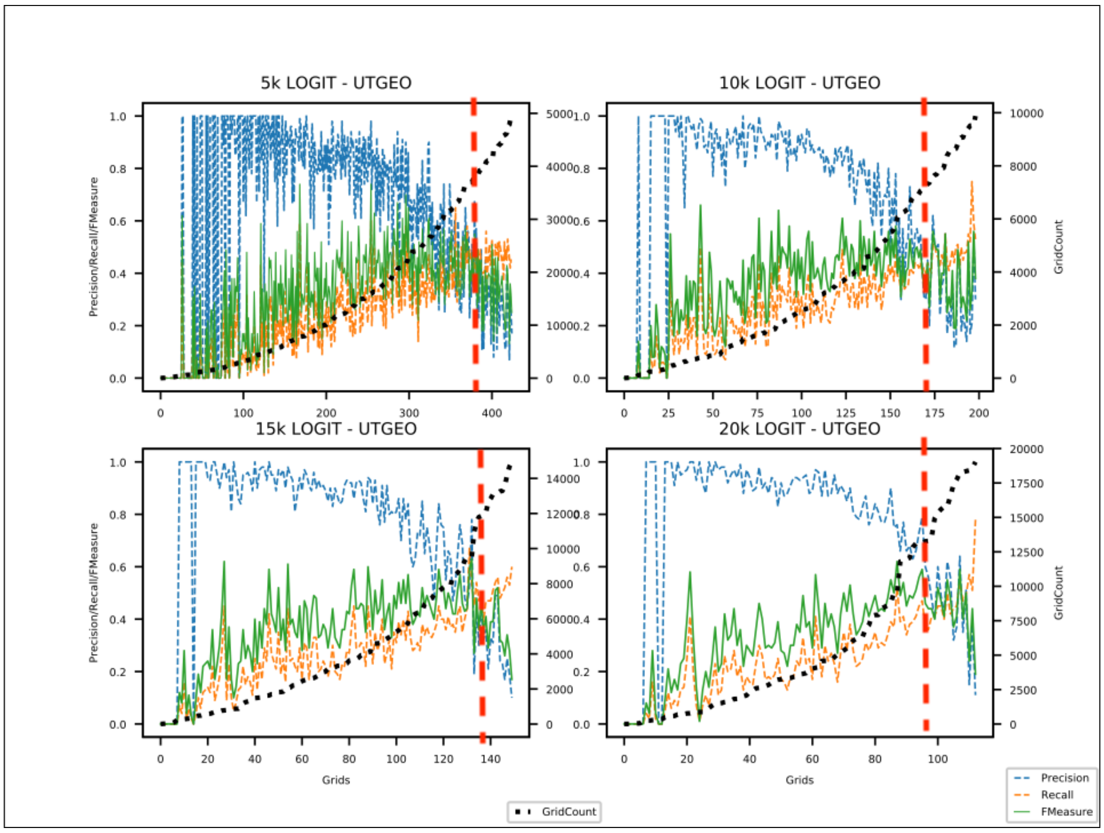


Figure 3.10: Combined Performance of the LOGIT Classifier on UTGEO dataset variants

settles around grid counts of 3500(G230), 8000(G110), 12000(G70) and 13000(G50) for split criterion of 5k, 10k, 15k and 20k respectively.

3.4.3 Performance of MNB Classifier on UTGEO-Small Dataset

Similarly, Figure 3.12 presents the performance of the MNB classifier on the UTGEO dataset under each of the four splitting criterion of 5k, 10k, 15k and 20k respectively. The disparity and divergence between all three metrics was quite high around the small grids but as they increased in size, it would be seen that the convergence and more reliable estimates was achieved around G380, G180, G140 and G100 for grid sizes of 4000, 8000, 12000 and 15000 tweets respectively.

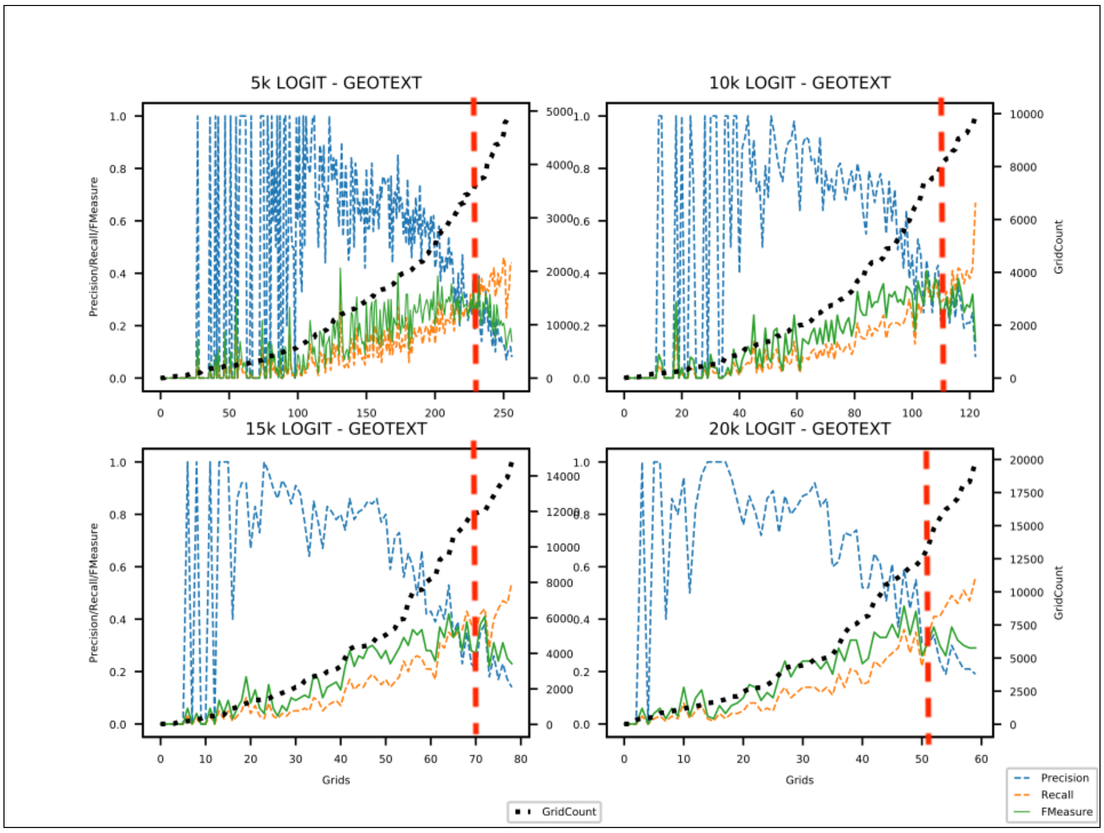


Figure 3.11: Combined Performance of the LOGIT Classifier on GEOTEXT dataset variants

3.4.4 Performance of MNB Classifier on GEOTEXT Dataset

Figure 3.13 presents the performance of the MNB classifier on the GEOTEXT dataset under each of the four splitting criterion of 5k, 10k, 15k and 20k respectively. Points of convergence in this instance was around G240, G118, G70 and G55 for grid counts of 4000, 9000, 12000 and 15000 tweets respectively.

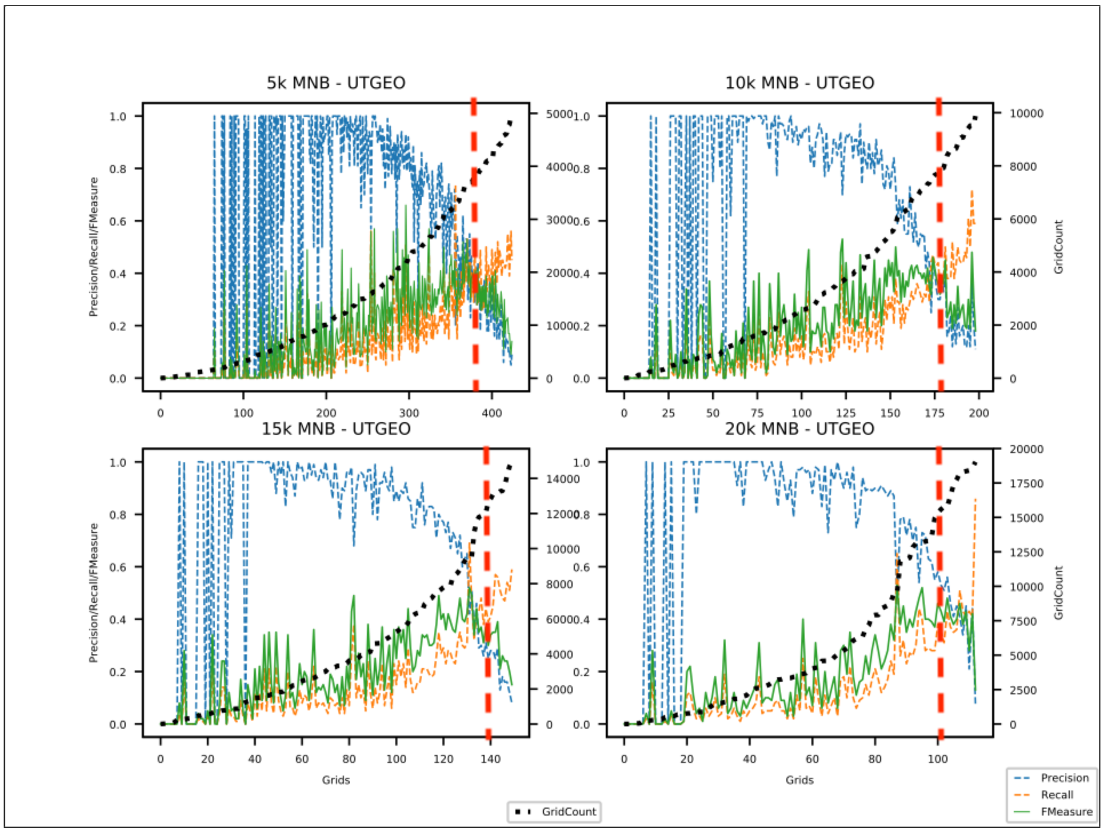


Figure 3.12: Combined Performance of the MNB Classifier on UTGEO dataset variants

Table 3.3: Text Cleaning and Pre-Processing Steps

TASK	DESCRIPTION
Duplicate Tweets	it was discovered that some tweets were unnecessary being posted, leading to multiple instances of the same features but with predictive significance. They tended to be more like spam messages hence removed from the data set
Blank Tweets	There is also no benefit to the classifier where tweets have no content or characters in them and no sort of text processing can be on the messages even if the tweets contain location metadata and are geotagged
URLs and Hyperlinks	With regular expressions this ensures that only words remain in the analysis of the messages. Although URLs may be unique to each message and can sometimes be used to decipher web sources of message text and embedded images, these were removed from the training and testing corpus as its raw form distort the performance of the classifier models
Usernames	These are generally mentions of other users with the @ prefix aimed at quoting, retweeting or replying their messages. As these only bring into repetition their names and have no location correlation or significance hence their exclusion from the refined text corpora
Tokenisation	strip white spaces as well as the splitting of words into 'tokens' to handle each tokenized word as a feature in the classification task
Word Normalisation	As conversations on social media tend to appear in all forms of capitals and lower case characters it is essential to avoid redundancy; This ensures words are not unnecessarily repeated within the vector space. For example the word 'Miami' will be transformed into 'miami'
Stopwords	These would be words that are commonly used in the English language vocabulary and have no significant impact on the geospatial identification of the messages or their authors. Such words would have such a frequent occurrence such that the sensitivity and accuracy of the classifier is hindered and not effective. Example words include This, his, the etc.
Word Stemming	It was found necessary to shorten words all still aimed at feature reduction and vector space optimisation. For example words such as 'sudden' and 'suddenly' are stemmed to 'sudden'
Punctuations	As part of cleaning up the text, punctuation, special and non-ASCII characters such as emojis and emoticons are cleaned out of the corpus
Numbers	As the scope of the task is strictly a word-based approach, numbers were not found useful in the training as they were removed from the corpora

Table 3.4: Quadtree-based classification showing Error Distance and Compute Time for two different classifiers on GeoText Dataset

Grid Count	Med-ED (km)	Avg-ED (km)	ACC@161	Time (mins)
Logit - GeoText dataset				
20,000	143.98	571.39	51.72	68
15,000	125.73	700.76	53.84	70
10,000	129.18	620.81	52.04	73
5,000	39.15	598.44	59.47	79
MNB - GeoText dataset				
20,000	411.22	721.11	38.57	58
15,000	1009.82	579.44	41.61	58
10,000	279.78	876.67	30.76	58
5,000	400.62	853.33	42.57	58

Table 3.5: Quadtree-based classification showing Error Distance and Compute Time for two different classifiers on UTGeo-small Dataset

Grid Count	Med-ED (km)	Avg-ED (km)	ACC@161	Time (mins)
Logit - UTGeo-small dataset				
20,000	249.45	833.10	43.70	78
15,000	124.44	651.81	54.75	78
10,000	92.86	618.07	57.30	79
5,000	45.00	600.79	60.24	81
MNB - UTGeo-small dataset				
20,000	665.31	1093.45	30.20	71
15,000	449.78	907.65	40.07	71
10,000	418.08	828.13	42.56	71
5,000	380.76	855.64	43.76	71

Table 3.6: Our Method and other Grid-Based Results

Author	Method	AED	MED	ACC@161
GeoText dataset				
Eisenstein *et al.* (Eisenstein et al., 2010)	Topic Models	900	494	24
Wing *et al.* (Wing & Baldridge, 2011)	Uniform	967	479	N/A
Hulden *et al.* (Hulden et al., 2015)	Uniform	764.8	357.2	N/A
Our	Quadtree	**598.44**	**39.15**	**59.47**
UTGeo-small dataset				
Roller *et al.* (Roller et al., 2012)	kd-tree	860.0	463.0	34.6
Our	Quadtree	**600.79**	**45.00**	**60.24**

Table 3.7: Minimum Suggested Grid Sizes to achieve good Precision

GRID	Logit Geotext	Logit UTGeo	MNB Geotext	MNB - UTGeo
20K	895	522	1771	719
15K	534	484	1771	1337
10K	562	484	2011	1492
5K	650	299	1436	1105

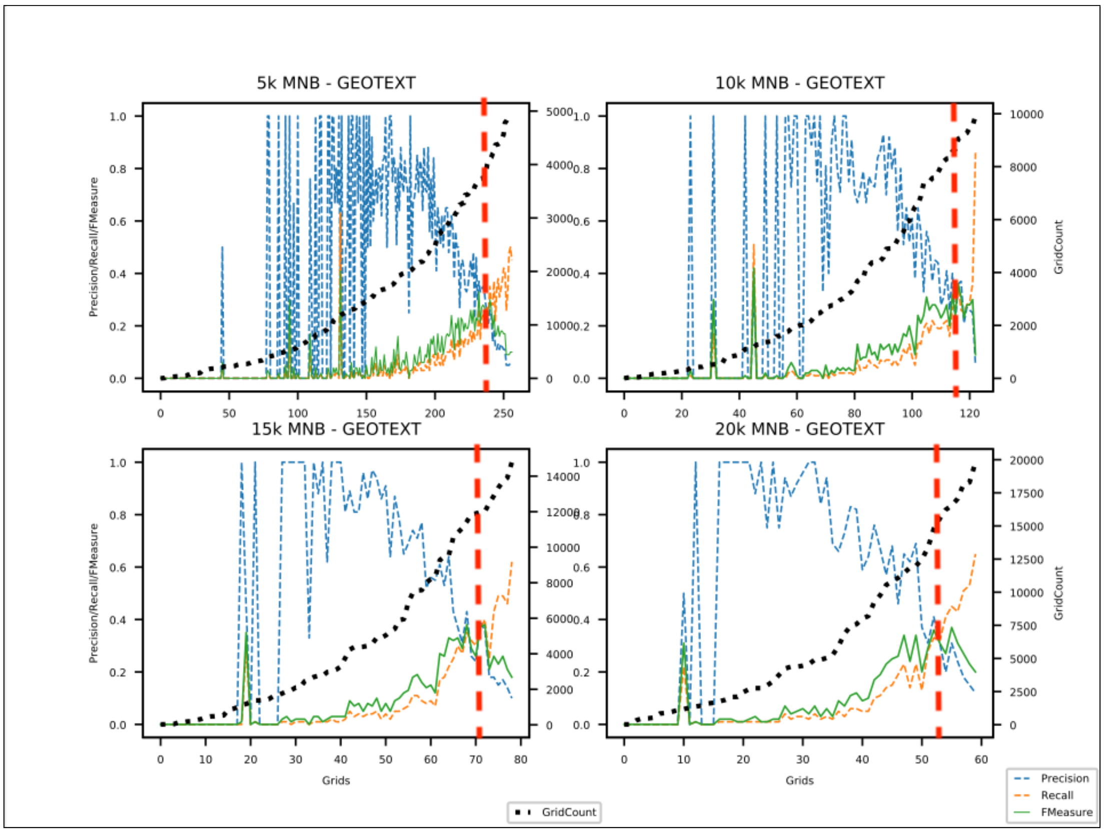


Figure 3.13: Combined Performance of the MNB Classifier on GEOTEXT dataset variants

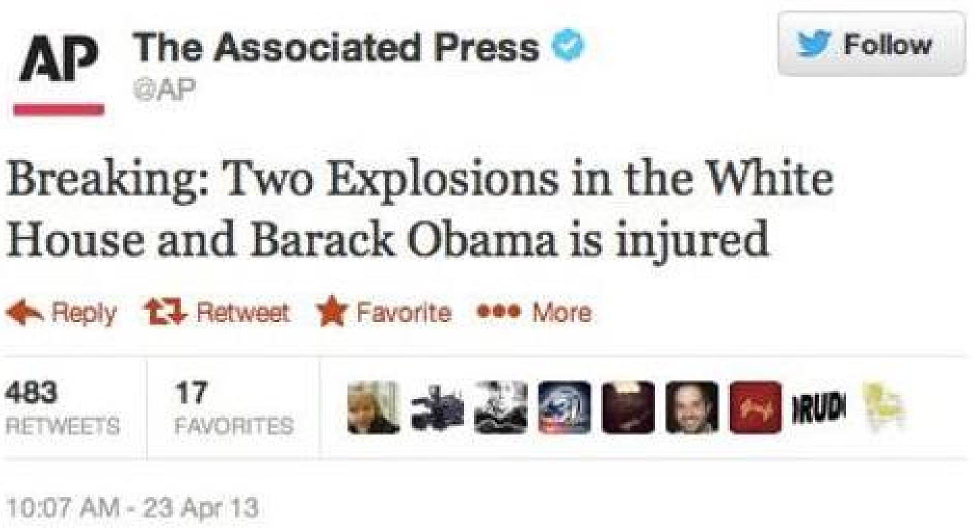
Chapter 4

Fake News Identification on Twitter with Text - content only

The problem associated with the propagation of fake news continues to grow at an alarming scale. This trend has generated much interest from politics to academia and industry alike. The misinformation detection technique (MISDETECT) that detects and classifies fake news messages from Twitter posts using a hybrid of convolutional neural networks and long-short term recurrent neural network models. It is reported in this work that using this deep learning approach achieves an 82% accuracy. Intuitively identifying relevant features associated with fake news stories without previous knowledge of topic domain.

4.1 Introduction

The growing influence experience by the propaganda of fake news author is now cause for concern for all walks of life. Election results are argued on some occasions to have been manipulated through the circulation of unfounded and some time doctored stories on social media including microblogs such as Twitter. All over the world, the growing influence of fake news is felt on daily basis from politics to education and financial markets. This has continually become a cause of concern for politicians and citizens alike. The impact could also be severe. On April 23rd 2013 the Twitter account of the news agency, Associated Press which had almost 2 million followers at the time was hacked. The following message was sent "Breaking Two Explosions in the White


AP The Associated Press @AP

Follow

Breaking: Two Explosions in the White House and Barack Obama is injured

Reply Retweet Favorite More

483 RETWEETS 17 FAVORITES

10:07 AM - 23 Apr 13

Figure 4.1: Tweet allegedly sent by the Syrian Electronic Army from hacked Twitter account of Associated Press

House and Barack Obama is injured." shown in Figure 4.1. This message led to a flash crash on the New York Stock Exchange where more than 140 points was shaved off the Dow Jones Industrial Average translating to investors losing 136 billion dollars on the Standard & Poors Index in two minutes (Keller, 2013). It would be interesting and indeed beneficial if the origin of messages could be verified and filtered where the fake messages were separated from authentic ones. The information that people listen to and share in social media is largely influenced by the social circles and relationships they form online (Leskovec & Mcauley, 2012). Accurately tracking the spread of fake messages and especially news content would be of interest to researchers, politicians, citizens as well as individuals all around the world. This can be achieved by using effective and relevant 'social sensor tools' (Schifferes et al., 2014). This need is more so important in countries that have trusted and embraced technology as part of their electoral process and thus adopted e-voting. Ceron et al. (2014) found in France and Italy even though internet users may not accurately represent the demographics of the entire population, opinions on social media and mass surveys of citizens are correlated as they are both found to be largely influenced by external factors such as news stories from newspapers, TV and ultimately on social media.

In addition, there's a growing and alarming use of social media for anti-social behaviours such as cyberbullying, hate propaganda, crime and for the radicalisation and recruitment of individuals into terrorism organisations such as ISIS (Ferrara, 2015). A study by Burgess et al. (2012) into the top 50 most retweeted stories with pictures of the Hurricane Sandy disaster found that less than 25% were real while the rest were either fake or from unsubstantiated sources. Facebook announced the use of 'filters' for

Table 4.1: Most circulated and engaging fake news stories on Facebook in 2016

S/N	Fake News Headlines	Category
1	Obama Signs Executive Order Banning The Pledge Of Allegiance In Schools Nationwide	Politics
2	Woman arrested for defecating on boss' desk after winning the lottery	Crime
3	Pope Francis Shocks World, Endorses Donald Trump for President, Releases Statement	Politics
4	Trump Offering Free One-Way Tickets to Africa & Mexico for Those Who Wanna Leave America	Politics
5	Cinnamon Roll Can Explodes Inside Man's Butt During Shoplifting Incident	Crime
6	Florida man dies in meth-lab explosion after lighting farts on fire	Crime
7	FBI Agent Suspected in Hillary Email Leaks Found Dead in Apparent Murder-Suicide	Politics
8	RAGE AGAINST THE MACHINE To Reunite And Release Anti Donald Trump Album	Politics
9	Police Find 19 White Female Bodies In Freezers With "Black Lives Matter" Carved Into Skin	Crime
10	ISIS Leader Calls for American Muslim Voters to Support Hillary Clinton	Politics

removing hoaxes and fake news from the news feed on the world's largest social media platform especially in Germany (BBC, 2017) This was prior to the presidential elections in the country. The development followed concerns that the spread of fake news on the platform might have helped Donald Trump win the US presidential elections held in 2016 (Solon, 2016) According to the social media site, (Silverman, 2016) 46% of the top fake news stories circulated on Facebook was about US politics and election. Table 4.1 gives detail of the top ranking news stories that was circulated on Facebook in year 2016.

4.1.1 Background of the Problem

Misinformation and fake news is growing at an alarming rate on the Media - social and conventional media, print and electronic. The work of detecting the veracity a message in Online Social Networks (OSN) remains a problem that poses a lot of interest to academic research, industry and global citizens. To appropriately address the domain, there's a definition of the subject, the aims, objectives and contributions of the study were set. Afterwards, some tools and dataset to be used are introduced.

4.1.2 Spatio-temporal awareness of fake news stories

The concept of fake news detection can also be considered in a spatio-temporal awareness. Its common for there to be different types of fake news stories. They could have been partially reported having half-truths of the actual events that have occurred or be total misinformation where the actual sequence of events that have been altered to suit a particular agenda or motive of the promoter. While it may be generally assumed that fake news is the spread of false information. There's a need to be mindful that there could be varying dynamics with respect to the location and the time that the news is considered fake. However this currently out of the scope of this work.

4.1.3 Research Questions

In this work the following are the research questions aimed to be answered:

- Given tweets about a news item or story, is it possible to determine their truth or authenticity based on the content of the messages
- Can semantic features or linguistic characteristics associated with a fake news story on Twitter be automatically identified without prior knowledge of the domain or news topic?

4.1.4 Problem Definition

Given a set of tweets collected in a corpus. Its assumed that the veracity of some examples is used infer some which are unknown. A model is trained based on the veracity of the known examples while the veracity of the unknown is determined from the weights assigned to the trained model.

4.2 Methodology

The approach of this work involves the automatic identification of features within Twitter post without prior knowledge of the subject domain or topic of discussion using the application of a hybrid deep learning model of LSTM and CNN models. This work posits that since the use of deep learning models enables automatic feature extraction; the dependencies amongst the words in fake messages can be identified automatically (Ma et al., 2016) without expressly defining them in the network. The knowledge of the news topic or domain being discussed would not be necessary to achieve the feat of fake news detection

4.2.1 The Deep learning Architectures

Deep learning models such as Convolutional Neural Networks (CNN) (Yang et al., 2018) and Recurrent Neural Networks (RNN) (Ma et al., 2016) are good for text and image classification with recent architectures aimed at being simple, fast and accurate. This attribute helps in the determination of fake news veracity as the models could implement faster and with little or no feature engineering required. As the fully connected hidden layers of the AI algorithms intuitively search for related features in the words or inputs of the model.

This work implemented three deep neural network variants. The models applied to train the PHEME dataset include:

- Long-Short Term (LSTM) recurrent neural network (RNN) was adopted for the sequence classification of the data. The LSTM (Greff et al., 2017) remains a popular method for the deep learning classification involving text since when they first appeared 20 years ago (Hochreiter & Schmidhuber, 1997).The architecture for the plain LSTM model is shown in Figure 4.2. From the diagrammatic illustration, it consisted of the input or embedding layer which accepts the word tokens as vectors and these are passed on to the LSTM layer where the sequential classification is done. A sequence length of 100 was used. There is an additional hidden layer called the *Dense* layer provides an extra level abstraction before this is passed on to the output layer. the binary outputs of this layer is either Fake or not Fake.

- LSTM - RNN with dropout regularization (Srivastava et al., 2014) layers between the word embedding layer and the LSTM layer to avoid over-fitting to the training dataset. Following this approach, randomly selected and dropped weights amounting to 20% gate-specific dropouts of neurons in the LSTM layer.

- LSTM with convolutional neural networks (CNN) (Karpathy & Fei-Fei, 2015) immediately after the word embedding layer of the model it was further included a 1d CNN and a max pooling layer to reduce dimensionality of the input layer while preserving the depth and avoid over-fitting of the training data. This also helps in reducing computational time and resources in the training of the model. The overall aim is to ultimately improve model prediction accuracy. The architecture for the LSTM-CNN model is shown in Figure4.3. From the diagrammatic illustration, it consisted of the input layer which accepts the word features and these are passed on to the one dimensional CNN layer then LSTM

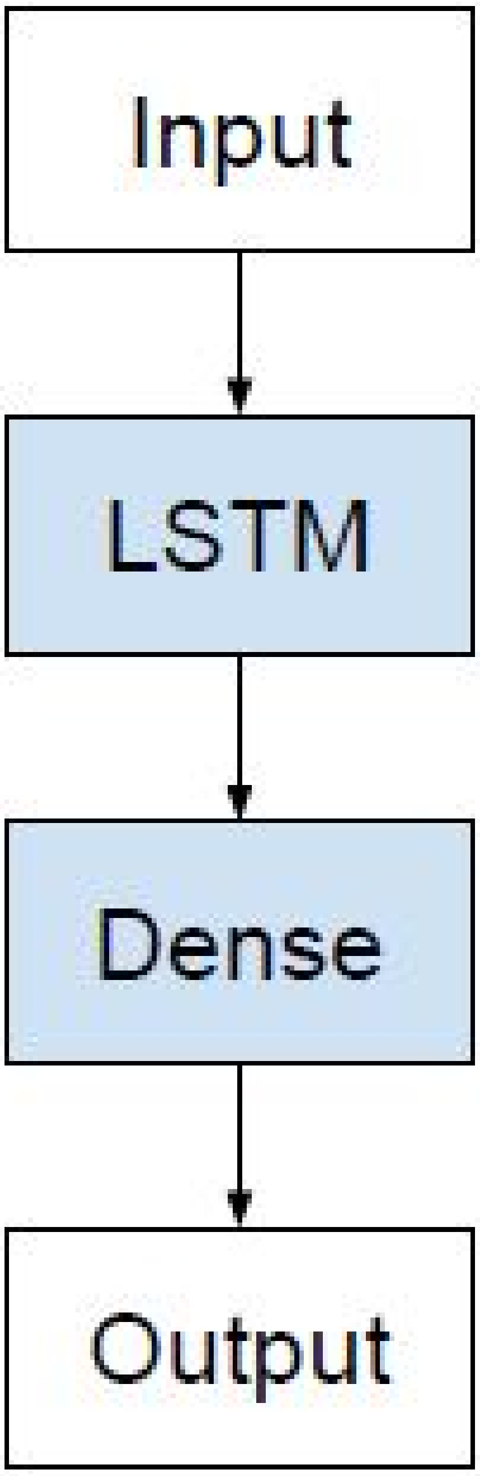


Figure 4.2: Architecture of the LSTM Model

layer. Outputs of the LSTM layer is then passed on to the *Dense* layer which is then passed on to the output layer.

CNNs have been widely usually used for the classification of image data and in computer vision. However, they have also shown success in text classification and NLP[1]. In place of image pixels from computer vision then the use of the word tokens would serves as input where each row of the vector matrix represents a word token. The work utilizes this convolution power of the CNN for use in text classification of content-aware misinformation detection.

4.2.2 Frameworks and equipment Hardware

The experiments were conducted over using an Intel i7 desktop processor. The time taken for the processing runs were recorded and compared for each of the deep learning algorithms. The program was written in Python programming language and the Keras [2] library of Python was adopted this was due to it's ease of processing and available functions for various deep learning tasks including CNN and RNN models.

4.2.3 About the Dataset

The dataset consisted of approximately 5800 tweets centered on five rumor stories. The tweets were collected and used in the works by Zubiaga et al. (2016). These stories were being consisted of original tweets and they were labeled as rumor and non-rumors. The events were widely reported in online, print and conventional electronic media such radio and Television at the time of occurrence:

- CharlieHebdo
- SydneySiege
- Ottawa Shooting
- Germanwings-Crash
- Ferguson

[1]`http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/?source=post_page-----98c86a0dd361----------------------`

[2]`https://keras.io/`

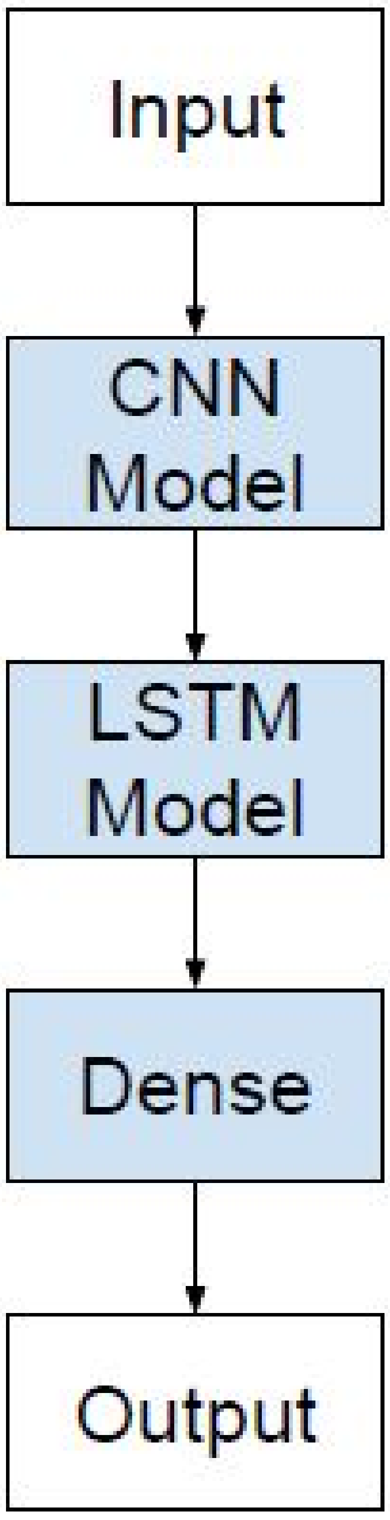


Figure 4.3: Architecture of the combined LSTM-CNN Model

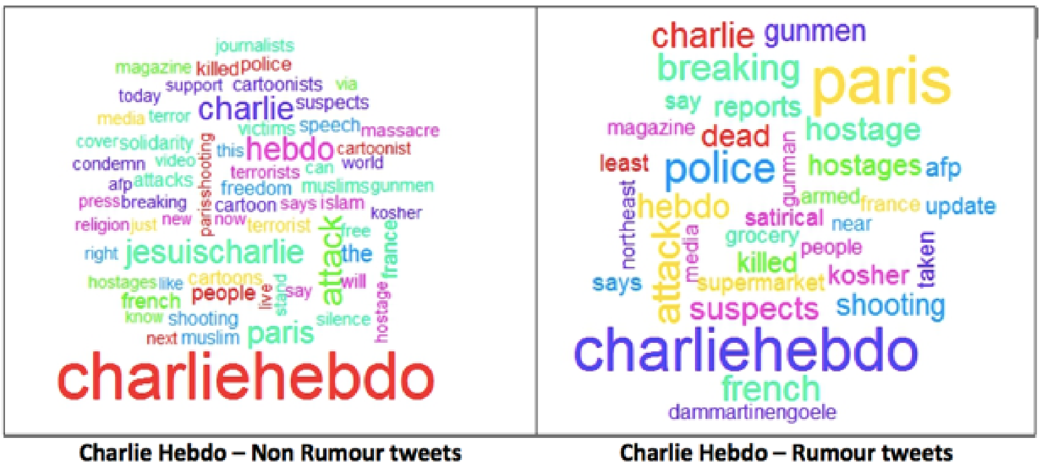

Charlie Hebdo – Non Rumour tweets
Charlie Hebdo – Rumour tweets

Figure 4.4: Wordcloud visualisation of the Charlie Hebdo Incident

This work applied ten-fold cross validation on the entire dataset of 5800 tweets and performed padding of the tweets i.e. adding zeros to the tweets for uniform inclusion in the feature vector for analysis and processing.

4.2.4 Description of the PHEME Rumor-Non Rumor Dataset

In the determination of the veracity of fake news stories, the Rumor-Non Rumor dataset was examined. To perform initial exploratory data analysis of the dataset, below is the exploratory analysis of the dataset. Highlighting wordcloud visualisations of the dataset for Charlie Hebdo. At this stage, the word clouds are not used in the classification but only to view the word frequency usage in the data. Figure 4.4, Ferguson Figure 4.5, Germanwings Figure 4.6, Ottawa Figure 4.7 and Sydney Siege Figure 4.8.

4.2.5 Recurrent Neural Network $RNNs$

This type of Neural network has been shown to be effective in time and sequence based predictions (Ma et al., 2015). Twitter posts can be likened to events that occur in time where the intervals between the retweet of one user to another is contained within a time window and treated in sequential modes.(Kwon et al., 2013) Rumours have been examined in the context of varying time windows (Kwon et al., 2017)

Recurrent Neural Networks were initially limited by the problem associated with the adjustment of weights over time. Several methods have been adopted in solving the

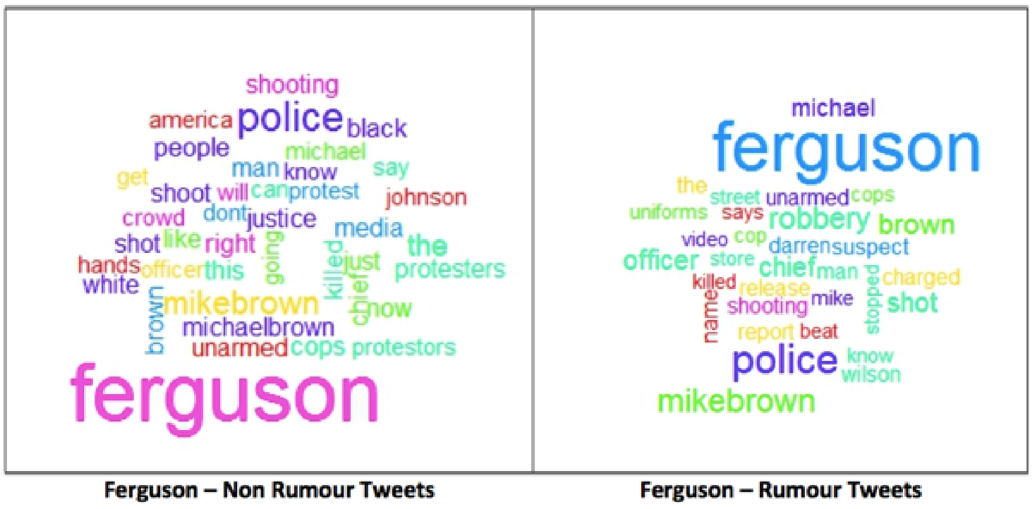


Figure 4.5: Wordcloud visualisation of the Ferguson

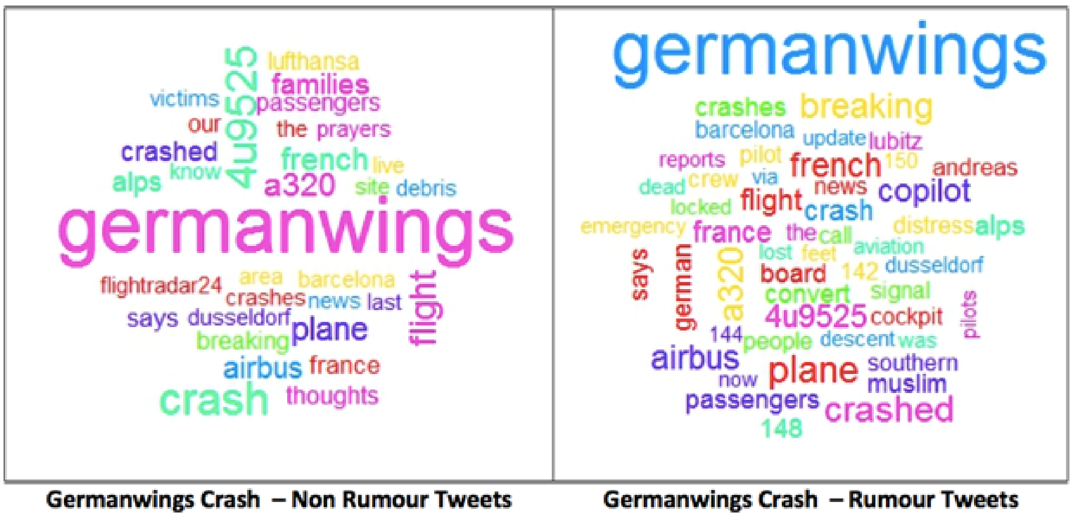


Figure 4.6: Wordcloud visualisation of the GermanWings Crash

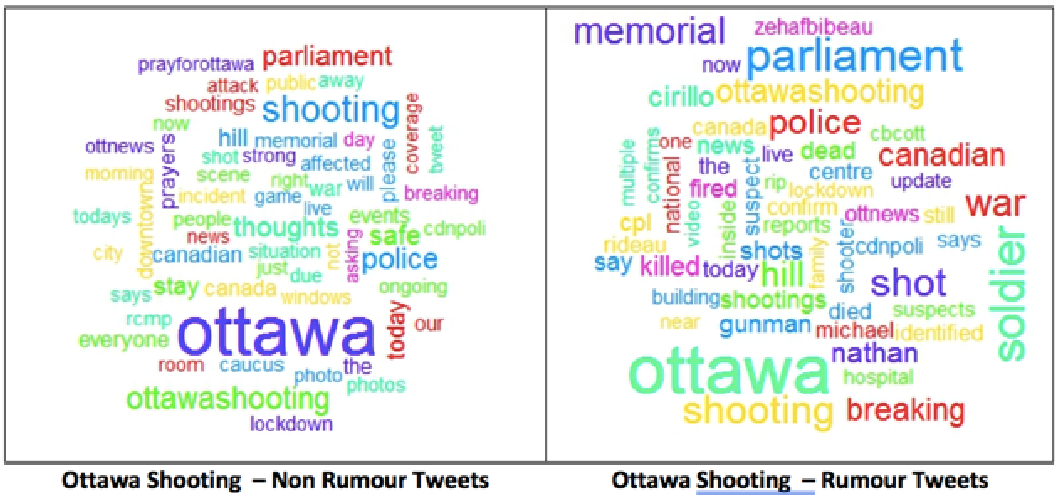


Figure 4.7: Wordcloud visualisation of the Ottawa Shooting

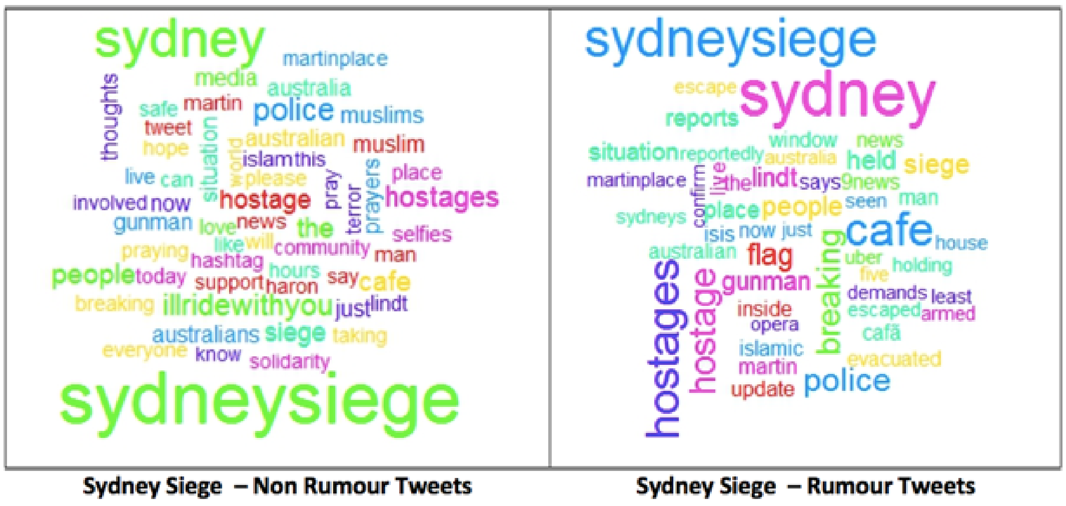


Figure 4.8: Wordcloud visualisation of the Sydney Siege

vanishing gradient problem but can largely be categorized into two types namely the exploding gradient and the vanishing gradient. Solutions adopted for the earlier include truncated back propagation, penalties and gradient clipping (these solve the exploding gradient problem) while this problem has been resolved using dynamic weight initializations, the echo state networks (ESN) and Long-Short Term Memory (LSTMs). LSTMs will be the main focus of this work as they preserve the memory from the last phase and incorporate this in the prediction task of the neural network model. Weights are the long term memories of the neural network.

4.2.6 Incorporating Convolutional Neural Network

Another popular model is the convolutional neural network (CNN) which has been well known for their application in image processing as well as their use in text mining(Hsu et al., 2017). It is posited that addition of the hybrid method would improve performance of the model and give much better results for the content based fake news detection. However, the hybrid implementation for this work so far involves a text-only approach.

4.2.7 Selection of Training Parameters

The following Hyper-parameters were optimized using a grid search approach and optimal values derived for the following batch size, epochs, learning rates, activation function and dropout regularization rate which is set at 20%.

4.2.8 Batch Size

This is the number of training examples included in one forward or back-propagation. It is recommended by (Goodfellow et al., 2016) that the batch size of power of 2 between 32-256 would produce optimal performance in the training and development of a conventional neural network. Thus 64 tweets (data rows) were used as the batch size in the model.

4.2.9 Number of Epochs

This is equivalent to a forward pass or backward pass of the training examples. It is recommended that 50 epochs consistent with the size of the dataset would be sufficient. This value implied that all the examples in the training dataset were examined 50 times to sufficiently train the model. Epoch sizes for deep learning models are usually set in multiples of 50 (Goodfellow et al., 2016)

4.2.10 Optimization Parameters

There are optimization parameters such as the Stochastic Gradient Descent (SGD) (Ruder, 2016) alongside other models Mini Batch Gradient Descent (RMSProp), Momentum, Adagrad, Adadelta, Adaptive Momentum (Adam), Adamax and Nesterov-accelerated Adaptive Moment Estimation (Nadam). Adam which is a combination of the RMSProp and Momentum (Kingma & Ba, 2014), being an adaptive learning rate optimization algorithm is suggested to be a reasonable optimization parameter for the task (Goodfellow et al., 2016)

4.2.11 Learning Rate

The learning rate was recommended to be optimal at 0.001 as this is also the suggested learning rate for the ADAM optimization parameter (Kingma & Ba, 2014). This value allowed the convergence of the training parameters in good time and efficiently. If the learning rate was too low the time taken for the gradient descent to converge would be too long and other hand a very large value would result in overshooting the minimum loss function and convergence impossible. Thus an ideal optimal learning rate enabled us to strike a balance between training time and accuracy of the model.

4.2.12 Network weight initialization

A network initialization (also referred to as the Xavier initialization (Glorot & Bengio, 2010)) of zero was recommended . This value ensured that the gradients were not too little or too steep for the training learning process of the network. They ensure uniform levels of distributions of the neuron activations within the model having a zero mean and an optimal variance.

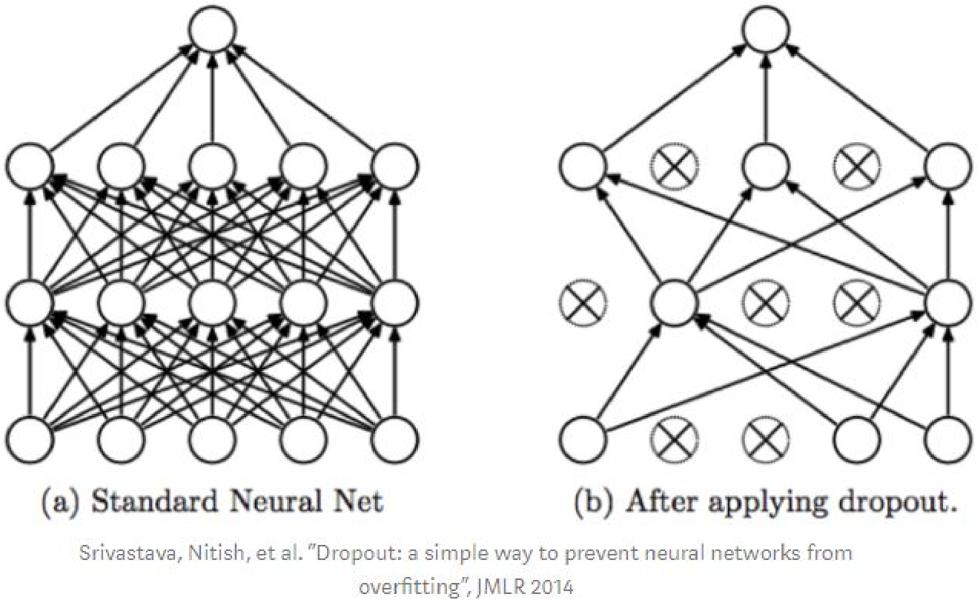

(a) Standard Neural Net
(b) After applying dropout.
Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Figure 4.9: Avoiding over-fitting using Dropout technique Illustration

4.2.13 Neuron activation function

To determine the activation thresholds of each nodes, the Rectified Linear Unit (ReLU) (Glorot et al., 2011) activation function was adopted within the hidden layers of the neural network. This is a well used, simple and effective function [3]. To map the input to the output via the activation node, the Sigmoid function was used due to the binary classification of fake and *not fake* outcomes as the output.

4.2.14 Dropout regularization

For dropping out neurons from LSTM layer it was found from literature [4] that 20% dropout rate was suitable. This was achieved by randomly changing values of the predefined proportion of nodes within the deep neural network to zero. Thereby enhancing a fair use of other features within the network and preventing overfitting (Srivastava et al., 2014) in the MISDETECT algorithm. The illustration is given in Figure 4.9

[3]`https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f`
[4]`https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/`

4.2.15 Number of neurons in the hidden layer

As applicable to all neural networks, the number of neurons in the input layer is equivalent to the number of feature types or variables within the data. The approach followed in this work is a text-only approach only the *text* column of the PHEME dataset was used in training and testing the deep learning model. According to (Heaton, 2008) the optimal size of the hidden layer is usually between the size of the input and size of the output layers.

4.3 Evaluation, Results and Discussions

The deep learning LSTM model intuitively achieves an 82% accuracy on the classification task in the detection fake news posts without prior domain knowledge of the topics being discussed.

So far in the experiments completed it is revealed that the plain vanilla LSTM model achieved the best performance in terms of Precision, Recall, F-measure and having an accuracy of 82% as shown in Table 4.2. On the other hand, the LSTM method with a dropout regularization performed the least in terms of the metrics adopted. This is likely to be as a result of under fitting of the model and the lack of a sufficient training data and examples within the network which would have negatively impacted the performance of the model and thus becoming counter-productive instead of helping improve model gain and efficiency. Another reason for the low performance of the dropout regularisation would be the depth of the network since the network is relatively shallow, the drop-out layer is quite close to the input and output layers of the model; this could severely degrade the performance of the method. An alternative to improve model performance could be through *Batch Normalisation* (Ioffe & Szegedy, 2015) where the input values in the layers have a mean activation of zero and standard deviation of one as in a standard normal distribution, this is beyond the scope of this current work.

The LSTM-CNN hybrid did not perform as badly as the dropout regularisation model having 74% accuracy and an FMeasure of 39.70%. However due to insufficient training examples for the neural network model led to negative appreciation against the plain-vanilla LSTM model.

The precision of 68% achieved by the state of the art on the PHEME dataset by Zubiaga

Table 4.2: Table of values for 3 different proposed deep learning methods in fake news detection

Technique	ACC	PRE	REC	F-M
LSTM	82.29	44.35	40.55	40.59
LSTMDrop	73.78	39.67	29.71	30.93
LSTM-CNN	80.38	43.94	39.53	39.70

et al. (2016) was still higher than the results obtained so far. However, it was expected that the inclusion of more training data from the reactions to the original twitter posts there will be more significant improvement in model performance.

4.3.1 Improvements through Feature Engineering from Sentiments

Improving the performance of the model could be achieved through the inclusion of more features. The low precision recorded from the words-only approach could be further boosted by considering other signals or features that were embedded within the context of the words usage. Using this approach the appearance of the words would be considered alongside the ways in which they were used. This would be a more detailed and involved approach where the meaning of the words are now being considered. The use of machines to understand text and their context have been previously used in field such as sentiment analysis - where polarity could be assigned to text ranging from 'negative' for bad and 'positive' for good. As tweets are posted in text, a use of emotions or sentiments extracted from the messages could prove helpful in determining the veracity of the posts, providing additional features that would enable and enrich the classifier performance.

Chapter 5

Sentiment Aware Fake News Detection in Online Social Networks

5.1 Introduction

Messages posted to Online Social Networks (OSN) cause a recent stir due to the intended spread of fake news or rumor. In this work, the aim was to understand and analyse the characteristics of fake news especially in relation to sentiments, to determine the automatic detection of fake news and rumors. Based on empirical observation, this work proposes a hypothesis that there exists a relation between a fake message/rumour and the sentiment of the texts posted online. The hypothesis of this work is verified by comparing with the state-of-the-art baseline text-only fake news detection methods that do not consider sentiments. This work performed experiments on standard Twitter fake news dataset and show good improvements in detecting fake news/rumor. This techniques proposed at the time of this writing this thesis is the first that considers sentiment awareness, in the task of fake news detection.

In the task of detecting fake news in social media it is beneficial if all features associated with each message type are properly identified and utilised. Twitter posts with images offer more impression and influence over text only tweets. A Twitter message has been shown to have a lifespan of as little as less than one day and up to a 70 day span depending on the type of content and URL being shared (Wu et al., 2011). This implies that except a message goes *viral* where it *infects* other users - leading to more engagements such as retweets, it normally tends to be short lived thus over-ridden by other posts before the end of the day. To create more engagements, often images are used which may not even be related to the post nor be true images of the event.

Previous work has shown that deception and false statements can be detected from the writing style of the authors or linguistics and sometimes be used to infer their personalities (Pennebaker & King, 1999). Some authors have shown from face-to-face interview transcriptions that liars can even be detected as they tell complex stories, make fewer self-references -to disassociate themselves from the story, and tend to have more frequent use of negative emotion words as a sign of guilt (Newman et al., 2003). Therefore, it is logical to consider emotions within the posted texts as a cue in relation to spreading fake news and rumour. The approach of this work is different as it looks the emotional context of the words used in online social networks. It proposes a hypothesis that there exists a relation between a fake message or rumour and the emotion or sentiment of the texts posted online. The proposed hypothesis is proven on a standard benchmark PHEME Rumor Non-Rumor dataset presented in Appendix D by comparing with the state-of-the-art (Zubiaga et al., 2016) baseline text-only fake news detection methods that does not consider these emotional words but rather relying on the text only. The overall flow of SENTDETECT algorithm is shown in Figure 5.1. The contribution in this chapter is an emotional ratio feature infused as part of the word vectors prior to input into the deep learning classifier. Thus a sentiment-aware classifier called SENTDETECT is proposed.

- proposing a relationship that exists between fake news messages and emotional words used in the message text, and
- improvement in fake news detection and prediction following a sentiment-aware classification .

5.2 Methodology

5.2.1 Sentiment-Aware Misinformation

This work presents an hypothesis that there exists a relationship between a fake message or rumour and the sentiment of the texts posted online. Authors of misinformation

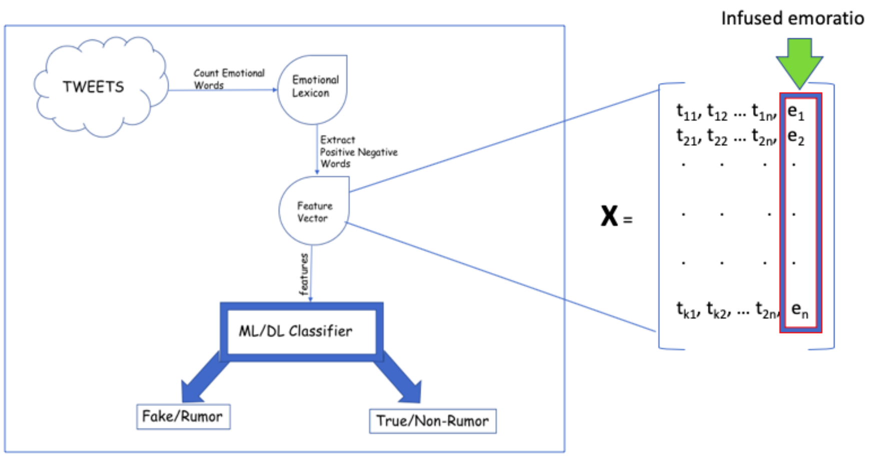


Figure 5.1: Schematic Diagram of Text Rumor Classifier

posts have been found to conceal their emotions by use of negative emotional words as a sign of guilt in their communication (Newman et al., 2003). Also could be that negative emotions tend to spread fast and thus become mechanisms with which these author convey their messages.

This work posited that sentiment may place a role in determination of the class of a tweet as a rumor or non-rumor. It is observed that such characteristics by analyzing the benchmark data (Zubiaga et al., 2016) using world cloud visualization after text cleaning. Example of wordclouds from the Charlie Hebdo event is shown in Figure 5.2. Therefore a sentiment analysis is proposed to be performed on each of the event corpus with a focus on the sentiment scoring function using Linguistic Word Count application's (LIWC) (Tausczik & Pennebaker, 2010) psychological and linguistic analytic capabilities. Our sentiment analysis rely on an emotional ratio score as calculated below:

$$emoratio = \frac{\text{count of negative emotional words}}{\text{count of positive emotional words}} \tag{5.1}$$

In order to check if there was any level of significance between the two types of tweets (rumor and non-rumor), there was a calculation of the t-statistic:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\left(\frac{(N_1-1)s_1^2+(N_2-1)s_2^2}{N_1+N_2-2}\right)\left(\frac{1}{N_1}+\frac{1}{N_2}\right)}}, \tag{5.2}$$

and the Null Hypothesis:

$$H_0 : u_1 - u_2 = 0, \tag{5.3}$$

where u_1 is the mean of rumor corpus and u_2 is the mean of non-rumor corpus of

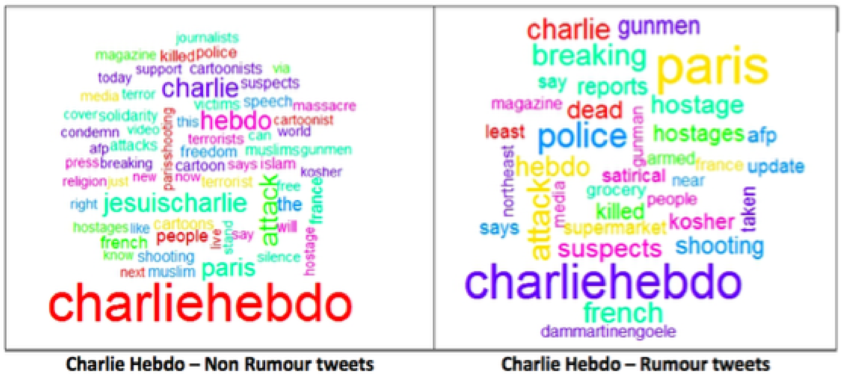


Figure 5.2: Word Cloud of Charlie Hebdo Tweets

the data. The initial assumption (Ho) is there's no difference between the average sentiment scores of the two populations i.e. rumors N_1 and non-rumors N_2 each having means $\bar{X}_1$ and $\bar{X}_2$ respectively.

In the analysis, the Treatment 1 was considered as the emoratio of rumor tweets of the 5 classes of events, $N_1 = 5$, average across the groups given as $\bar{X}_1 = 3.74$, and variance of $s_1^2 = 3.15$. Similarly Treatment 2 is the emoratio values of Non-Rumor events with $N_2 = 5$, $\bar{X}_2 = 1.65$ and $s_2^2 = 0.48$. Thus the T-value calculation computed from Equation 5.2 is given as $t = 2.45058$ is greater than the p-value is 0.01995 (at 0.05 level of significance). It implies that the the null hypothesis H_0, would be rejected *i.e.,* there's significant difference in the mean of the sentiment scores of the two types of tweets.

Table 5.1 show the initial findings derived from LIWC application. The last column of the table with the title Emotion Ratio could be considered as being equivalent to Equation 5.1.

as part of the input features used in the classification. Overview of the proposed algorithm and description of the algorithm are shown in Figure 5.1 and Algorithm 1, respectively. Results for various Machine learning and deep learning classifiers are also presented in Table 5.3. Given the proof that there is a strong significance and association between tweets spread as false rumors and Sentiment Analysis. The task is to develop a machine learning classifier that factors the sentiment score of each tweet corpus in determining the weights used in the prediction model.This is achieved using the emotional ratio as describer earlier.

Table 5.1: Emotion ratio in rumor and non-rumor Tweets

Corpus	Word Count	LIWC Positive Emotion	LIWC Negative Emotion	Emotion Ratio
Rumors				
Charlie	7054	0.82	4.34	5.29
Ferguson	5512	0.71	2.38	3.35
Germanwings	3895	0.41	2.31	5.63
Ottawashoot	7721	1.17	3.67	3.14
Sydneysiege	8250	0.81	1.03	1.27
Non Rumors				
Charlie	26004	2.52	5.78	2.29
Ferguson	14208	1.63	2.94	1.8
Germanwings	3689	0.73	1.68	2.3
Ottawashoot	6719	3.17	2.68	0.85
Sydneysiege	11874	2.7	2.73	1.01

5.2.2 Machine Learning and Deep Learning Classification

An initial classification of the labeled dataset was done using a series of machine learning algorithms: logistic regression (LOGIT), support vector machines (SVM), decision trees, random forest and extreme gradient boosting (XG-Boost). This work includes the implementation of the long short term memory (LSTM) recurrent neural network implementation with hierarchical attention networks (HAN). This work examined the benefits of using varied word embeddings as pre-trained language models for the input layer of the HAN model. The pre-trained word vectors by (Pennington et al., 2014) was used this included the *Wikipedia 2014 Gigaword5 collection* which was pre-trained on six billion word tokens and the *Twitter collection* which was pre-trained on 2 billion tweets with 27 billion tokens; both in sizes of 100 dimensions. Both LSTM-HAN models were trained with an epoch size of 50, while a batch size of 64 was found to be optimal and learning rate was set at 10%.

```
Input: TweetCorpus, PosemoLexicon, NegemoLexicon;
Extract top k words; Extract relevant words for each k;
Extract negative emotion words;
Extract positive emotion words;
repeat
    Input: Receive next relevant tweets;
    Calculate emoratio;
    Extract word features from tweets into vector;
    Append the emoratio to the word feature vector;
    repeat
    until all tweets have been appended;
    Parse feature vector into classifier;
until end of sequence;
Output: $y_1$ Predicted label of tweet - Rumor or Non-Rumor;
```

Algorithm 1: Rumor Classifier Algorithm

Table 5.2: Summary Statistics of Dataset

Name of Event	Event Date	Size	With Images
Charlie Hebdo	7th Mar 2015	2,058	1,087
Ferguson	9th Aug 2014	1,142	4390
Germanwings	24th Mar 2015	468	213
OttawaShoot	22nd Oct 2014	886	301
SydneySiege	15th Dec 2014	1,211	509
TOTAL		5,765	2,600

5.3 Results and Discussions

5.3.1 Dataset

This work used the PHEME (Zubiaga et al., 2016) labeled Twitter dataset, The corpora consists of 5800 tweets about 5 notable world events widely reported in the electronic, print and conventional news media. They occurred at various times between August 2014 and March 2015. The statistic about these news stories is presented in Table 5.2. All items were hand labeled by journalists. About 45% of the dataset had images and only these were further selected for further enriching the feature set in terms of the embedded texts.

Table 5.3: Range of Classifier Results after Emotional Analysis

Classifier	Accuracy	Precision	Recall	F-M
LOGIT	0.84	0.84	0.84	0.84
SVM-Linear	**0.86**	0.86	0.86	0.86
Decision Trees	0.77	0.77	0.77	0.77
Random Forest	0.85	0.85	0.85	0.85
XG-Boost	0.84	0.83	0.84	0.83
LSTM_HAN(Wiki)	0.85	0.86	0.81	0.84
LSTM_HAN(Twitt)	**0.86**	0.86	0.82	0.84
Baseline (Ajao et al., 2018)	0.82	0.82	0.44	0.44
Baseline (Zubiaga et al., 2016)	N/A	N/A	0.68	0.55

5.3.2 Discussion

The emotional ratio of negative to positive words is computed in Table 5.1. Our statistical test shows that the rumor dataset were significantly different in terms of being more negative sentiments and adverse emotional words from the emotional lexicon (Hu & Liu, 2004). This is further proven in the fake news classifier models where the focus on using emotional words in the classification feature set gave better results over the state of the art which used the same dataset shown in Chapter 4 and (Zubiaga et al., 2016). Specifically as shown from Table 5.3. SVM and HAN model with Twitter pretrained word embedding performed best with 86% for sentiment-aware text only rumor detection. Also, SENTDETECT results comprises of four variants of the classification feature set; the features from words within the text (TX), the emotional ratio (ER) and use of additional features (AD) including counts of uppercase words, exclamation marks, positive and negative emoticons, user mentions, hashtags and quotations.

Table 5.4: Combined features (subset with image-only Tweets)

Classifier	ER+TX	AD+TX	ER+AD+TX
LOGIT	0.84	0.82	0.83
SVM	**0.89**	0.81	0.80
Decision Tree	0.77	0.81	0.81
Random Forest	0.85	0.86	0.85
Grad Boosting	0.85	0.85	0.85
XG-Boost	0.83	0.82	0.83

Table 5.4 gives summary results in terms of accuracy for these feature combination types. However, considering only the 2600 tweets that had images in Table 5.4 i.e. column (ER+TX) shows that there's a further 3% improvement to 89% when there's a combination of the text with the emotional ratio if they contained an embedded image within the message. This further strengthens the impact of images in conveying rumors in online social networks. However, these additional features (AD) did not improve the performance of the models.

Chapter 6

Conclusions and Future Work

6.1 Location Inference

This thesis proposed a new non-uniform Quadtree content-only approach called LOCINFER for location inference from Twitter messages. The proposed algorithm uses natural language processing for semantic understanding and incorporates Cosine similarity and Jaccard similarity measures for feature vector extraction and dimensionality reduction. The result of the grid classification shows good improvement over the existing state-of-the-art grid based approaches in city-level location inference on existing benchmark dataset of the GEOTEXT and UTGEO-Small Twitter corpuses. 60% of tweets are accurately predicted within an error distance of 161km (100 miles radius). The results show the effectiveness of the LOCINFER Quadtree technique in combination with a Logistic regression classification model which outperforms other grid-based methods in location inference. Future work could look the location prediction in real-time from live Twitter data streams and possibly linking with other location-based networks and for other geographical regions of the world. There is also a potential application in helping to address online social media issues such as fake news detection and tracking the origin of online malicious content-based messages.

A more efficient grid-based content-only classifier *LOCINFER* was was implemented with a Logistic Regression and Multinomial Naive Bayes classifiers. The Logistic Regression Model version of this technique performed best. It showed significant progress in addressing the sparsity problem associated with disparately distributed tweets. The clustering approach along with a hybrid word embedding model allowed a outperformed state-of-the art grid-based content-only location inference methods by up to

24% in correctly predicting tweet locations within a 161km radius and by 300km in median error distance on benchmark datasets - UTGEO (small) and the GEOTEXT datasets (Appendix E).

6.2 Improvements over existing state-of-the-art location inference methods

The location inference approach followed in the work presented in this thesis is a grid-based content-only approach. At the time of writing this thesis, the state-of-the-art which applied this technique on the GEOTEXT dataset is given by (Hulden et al., 2015) with AED and MED of 765km and 357km respectively. While on the UTGEO-Small dataset is (Roller et al., 2012) with AED of 860km and MED of 463km. This work improves on the previous work done by (Hulden et al., 2015) and (Roller et al., 2012) by incorporating the hybrid word embedding in the determination of the word vectors - combining the Jaccard Similarity and cosine similarity approach for the reduction of the word vectors dimensionality. Thus, the difference between this work and these existing works is a content-only discriminate grid-based location inference with hybrid word embedding classifier. The variant in natural language processing the text incorporated in this work resulted in an improvement in the performance achieving 598km/600km AED and 39km/45km MED for the GEOTEXT and UTGEO-Small datasets respectively. This performance represents a lower MED than the most effective state-of-the-art methods by up to 300km and 400km on the GEOTEXT and UTGEO-Small benchmark datasets respectively.

6.3 Critical Analysis of LOCINFER technique

While better results than the state-of-the-art were presented by the approach followed in implementing LOCINFER. It should be noted that one of the reasons the location inference classifier did well was due to sufficient benchmark data which consisted of hundreds of thousands of training examples. Similarly, posts of the users included had tweeted often such that the classifier could detect a pattern in terms of their linguistic word used in relation with their geolocation labels. These grid patterns gave an ideal cluster for the algorithm to afterwards estimate their locations up to some level of accuracy. However, in the instance where a user had only tweeted once, it's very unlike that their location could be efficiently determined by LOCINFER only without the use of other spatial indicators such as timezones, IP addresses, free-text profile information and their friends spatial network. The use of a multi-spatial indicator approach for fine-grained location determination is not captured within the scope of this current work. Also, the tweets used where only English language tweets thus, it would be interesting to see how this could be translated to other non-English language tweets and geo-locations.

6.4 Misinformation Detection

Two other contributions of this thesis is to present content-based approach for the detection of the origin of fake news messages considering only the words of the authors. Using a sentiment-aware approach classifier called SENTDETECT, it was found that the sentiments of the authors words helped in determining the veracity of the tweets; as more negative sentiments were more associated with rumor messages. An emotional ratio *EMORATIO* was derived which takes into account the ratio of negative words to that of positive words was also created to achieve better results on a sentiment-aware classifier. Results obtained running the proposed methods on the PHEME dataset (Appendix D) showed a significant improvement by up to 5% in the task of fake news detection. The study aimed to detect the veracity of posts on Twitter. A good application of this would help law enforcement agencies in curtailing the spread and propaganda of such messages having negative implications and consequences for the believers of these messages. The earlier these messages are checked and stopped the better the chances of preventing them go 'viral'.

For the future, using images posted to online social networks for misinformation detection is bright considering the advancements in the facial recognition, object detection and convolutional neural networks coupled with adversarial neural networks. The extraction of embedded text in images, coupled with visual emotional analysis could be prospective domains that enhance the misinformation detection AI models. Also, the consideration of spatio-temporal awareness where fake images from past events have been added to recent messages - could bring an interesting dimension into the research problem. Finally, the recent upsurge in the proliferation of fake videos using easily accessible AI tools specifically - Generative Adversarial Networks (GANs) has led to people with limited skills in video editing producing *compelling fakes.* The power of videos and images frames in motivating individuals online is quite significant. Thus it would be expected that these would present interesting research challenges in the years to come.

The presented approach MISDETECT using the LSTM-only achieved an 82% accuracy performance beating the state of the art on the PHEME Dataset. It could be interesting to see the incorporation of fake image disambiguation which is found in these tweets; usually aimed at making the author's posts go viral. The approach gives a boost in the achievement of a higher performance while not requiring a large amount of training data typically associated with deep learning models. Future work could progressively examine the inference of the tweet geo-locations and origin of the authors of these fake news items who propagate them. It would be interesting if also the training data required in this task was relatively smaller such that fake news items can be quickly detected and located in a small amount of time saving computational resources and time. This feat would better aid the task of tracking the origin and location fake Twitter posts especially in real-time detection.

Deep learning models such as CNN and RNN often require much larger datasets as well as in some cases multiple layers of neural networks for the effective training of their models. In this case there was a fairly smaller dataset of 5800 tweets. In ongoing and future work the reaction of other users (retweets and replies) to these messages via the Twitter API would be in in the magnitude of hundreds of thousands with the aim of enriching the size of the training dataset thus improving the robustness of the model performance. Also to help draw more actionable insights for the propagation of these messages from one user to another and how they react; specifically if they embraced or refrained from becoming evangelists and promoters of these messages to other users on the platform.

This work proposed a new hypothesis that the use of emotional words is beneficial in sentiment-aware misinformation detection. This was support by proposing a novel sentiment-aware fake new detection algorithm and show improvement on a benchmark dataset over state-of-the-art algorithm that does not consider sentiment. The terrain of fake news and it's detection remains a actively researched topic because it continues to evolve rapidly and yet to be fully understood. This gap presents opportunities for progressive work to be done in the area. Additional sources of sentiment extracted from, *e.g.,* images, embedded text in the image and other visual media such as animations (GIFs) and videos may enhance model performance and is considered as future work.

6.5 Critical Analysis of SENTDETECT technique

It is worth mentioning that in an ideal experimental setting - similar to the one used in this study, there would be sufficient labeled training examples of fake and non-fake misinformation posts or rumors. This is not usually the case in a real life scenario. A more specifically challenging task could be where the the veracity or authenticity of these tweets needed to determined in real-time and automatically. In this case, there would be tweets which could have very few or no meaningful characters included. Another limitation with the proposed approach was that other non-text characters such as emojis, emoticons and GIFs could be sent by the users. All these were not captured or considered in the proposed approach presented in SENTDETECT. As it is current out of the scope of this work, it is hoped that this would be implemented in future work.

6.6 Combining Fake news detection with location inference

The blend of the fields of misinformation detection with location inference of user posts, presents a great opportunity in fighting cybercrimes, cyberbullying, curtailing the spread of malicious information as well as hate propaganda messages in online social networks. Law enforcement agencies and authorities can make the best of this approach to check the rise of such unwanted behaviours within the social media space. While location inference presents great opportunities in the detection of disasters emergencies, there is a possibility for the detection of misinformation posts such as fake news and rumors. Location inference presents an opportunity to track their origin hereby complementing the detection of the instance of such posts when they have been created and circulated.

Chapter 7

References

Fake news - political scandal words.

Abrol, S. & Khan, L. (2010). Tweethood: Agglomerative clustering on fuzzy k-closest friends with variable depth for location mining. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, (pp. 153–160). IEEE.

Agency, C. I. (2013). The world factbook: United states. `https://www.cia.gov/library/publications/the-world-factbook/geos/us.html`. Accessed: 2016/12/05.

Aggarwal, A., Rajadesingan, A., & Kumaraguru, P. (2012). Phishari: automatic real-time phishing detection on twitter. In *eCrime Researchers Summit (eCrime), 2012*, (pp. 1–12). IEEE.

Ajao, O., Bhowmik, D., & Zargari, S. (2018). Fake news identification on twitter with hybrid cnn and rnn models. In *9th Int'l Conference on Social Media & Society. Copenhagen (July 18)*, number Jul 2018.

Ajao, O., Hong, J., & Liu, W. (2015). A survey of location inference techniques on twitter. *Journal of Information Science*, *41*(6), 855–864.

Baccianella, S., Esuli, A., & Sebastiani, F. (2010). Sentiwordnet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining. In *Lrec*, volume 10, (pp. 2200–2204).

Backstrom, L., Sun, E., & Marlow, C. (2010). Find me if you can: improving geographical prediction with social and spatial proximity. In *Proceedings of the 19th international conference on World wide web*, (pp. 61–70). ACM.

BBC (2017). Facebook to tackle fake news in germany 2017.

Bouillot, F., Poncelet, P., & Roche, M. (2012). How and why exploit tweet's location information? In *AGILE'2012: 15th International Conference on Geographic Information Science*, (pp. N–A).

Braspenning, P. J., Thuijsman, F., & Weijters, A. J. M. M. (1995). *Artificial neural networks: an introduction to ANN theory and practice*, volume 931. Springer Science & Business Media.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Burgess, J., Vis, F., & Bruns, A. (2012). Hurricane sandy: The most tweeted pictures. *The Guardian Data Blog, November*, *6*.

Burmester, M., Henry, P., & Kermes, L. S. (2005). Tracking cyberstalkers: a cryptographic approach. *ACM SIGCAS Computers and Society*, *35*(3), 2.

Castillo, C., Mendoza, M., & Poblete, B. (2011). Information credibility on twitter. In *Proceedings of the 20th international conference on World wide web*, (pp. 675–684). ACM.

Ceron, A., Curini, L., Iacus, S. M., & Porro, G. (2014). Every tweet counts? how sentiment analysis of social media can improve our knowledge of citizens' political preferences with an application to italy and france. *New Media & Society*, *16*(2), 340–358.

Cha, M., Gwon, Y., & Kung, H. (2015). Twitter geolocation and regional classification via sparse coding. In *Proceedings of the 9th International Conference on Weblogs and Social Media (ICWSM 2015)*, (pp. 582–585).

Cha, S.-H. (2007). Comprehensive survey on distance/similarity measures between probability density functions. *City*, *1*(2), 1.

Chandra, S., Khan, L., & Muhaya, F. B. (2011). Estimating twitter user location using social interactions–a content based approach. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, (pp. 838–843). IEEE.

Chang, H.-w., Lee, D., Eltaher, M., & Lee, J. (2012). @ phillies tweeting from philly? predicting twitter user locations with spatial word usage. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, (pp. 111–118). IEEE Computer Society.

Cheng, B. & Titterington, D. M. (1994). Neural networks: A review from a statistical perspective. *Statistical science*, 2–30.

Cheng, Z., Caverlee, J., & Lee, K. (2010). You are where you tweet: a content-based approach to geo-locating twitter users. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, (pp. 759–768). ACM.

Christopher, D. M., Prabhakar, R., & Hinrich, S. (2008). *Introduction to information retrieval*. Cambridge University Press.

Compton, R., Jurgens, D., & Allen, D. (2014). Geotagging one hundred million twitter accounts with total variation minimization. In *Big Data (Big Data), 2014 IEEE International Conference on*, (pp. 393–401). IEEE.

Conroy, N. J., Rubin, V. L., & Chen, Y. (2015). Automatic deception detection: Methods for finding fake news. *Proceedings of the Association for Information Science and Technology*, *52*(1), 1–4.

Eisenstein, J., O'Connor, B., Smith, N. A., & Xing, E. P. (2010). A latent variable model for geographic lexical variation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, (pp. 1277–1287). Association for Computational Linguistics.

Ferrara, E. (2015). Manipulation and abuse on social media by emilio ferrara with ching-man au yeung as coordinator. *ACM SIGWEB Newsletter*, (Spring), 4.

Ferrara, E., Varol, O., Davis, C., Menczer, F., & Flammini, A. (2016). The rise of social bots. *Communications of the ACM*, *59*(7), 96–104.

Field, M. (2017). Police issue child safety warning over snapchat maps update that reveals users' locations.

Gelernter, J. & Mushegian, N. (2011). Geo-parsing messages from microtext. *Transactions in GIS*, *15*(6), 753–773.

Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, (pp. 249–256).

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, (pp. 315–323).

Gonzalez, R., Cuevas, R., Cuevas, A., & Guerrero, C. (2011). Where are my followers? understanding the locality effect in twitter. *arXiv preprint arXiv:1105.3682*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222–2232.

Gupta, A., Kumaraguru, P., Castillo, C., & Meier, P. (2014). Tweetcred: Real-time credibility assessment of content on twitter. In *International Conference on Social Informatics*, (pp. 228–243). Springer.

Gupta, A., Lamba, H., Kumaraguru, P., & Joshi, A. (2013). Faking sandy: characterizing and identifying fake images on twitter during hurricane sandy. In *Proceedings of the 22nd international conference on World Wide Web*, (pp. 729–736). ACM.

Han, B., Cook, P., & Baldwin, T. (2014). Text-based twitter user geolocation prediction. *Journal of Artificial Intelligence Research*, *49*, 451–500.

Han, B., Rahimi, A., Derczynski, L., & Baldwin, T. (2016). Twitter geolocation prediction shared task of the 2016 workshop on noisy user-generated text. In *Proceedings of the W-NUT Workshop*.

Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

Hardalov, M., Koychev, I., & Nakov, P. (2016). In search of credible news. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, (pp. 172–180). Springer.

Heaton, J. (2008). *Introduction to neural networks with Java*. Heaton Research, Inc.

Hecht, B., Hong, L., Suh, B., & Chi, E. H. (2011). Tweets from justin bieber's heart: the dynamics of the location field in user profiles. In *Proceedings of the SIGCHI conference on human factors in computing systems*, (pp. 237–246). ACM.

Hinduja, S. & Patchin, J. W. (2010). Bullying, cyberbullying, and suicide. *Archives of suicide research*, *14*(3), 206–221.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hsu, S. T., Moon, C., Jones, P., & Samatova, N. (2017). A hybrid cnn-rnn alignment model for phrase-aware sentence classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, (pp. 443–449).

Hu, M. & Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 168–177). ACM.

Huang, A. (2008). Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, (pp. 49–56).

Hulden, M., Silfverberg, M., & Francom, J. (2015). Kernel density estimation for text-based geolocation.

Ikawa, Y., Enoki, M., & Tatsubori, M. (2012). Location inference using microblog messages. In *Proceedings of the 21st International Conference on World Wide Web*, (pp. 687–690). ACM.

Ikawa, Y., Vukovic, M., Rogstadius, J., & Murakami, A. (2013). Location-based insights from the social web. In *Proceedings of the 22nd international conference on World Wide Web*, (pp. 1013–1016). ACM.

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, (pp. 448–456).

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Jin, Z., Cao, J., Zhang, Y., & Luo, J. (2016). News verification by exploiting conflicting social viewpoints in microblogs. In *AAAI*, (pp. 2972–2978).

Jurgens, D. (2013). That's what friends are for: Inferring location in online social media platforms based on social relationships. *ICWSM*, *13*(13), 273–282.

Kaplan, A. M. & Haenlein, M. (2010). Users of the world, unite! the challenges and opportunities of social media. *Business horizons*, *53*(1), 59–68.

Karpathy, A. & Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (pp. 3128–3137).

Keller, J. (2013). A fake ap tweet sinks the dow for an instant. *Bloomberg Businessweek*.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kinsella, S., Murdock, V., & O'Hare, N. (2011). I'm eating a sandwich in glasgow: modeling locations with tweets. In *Proceedings of the 3rd international workshop on Search and mining user-generated contents*, (pp. 61–68). ACM.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, (pp. 1097–1105).

Krumm, J. (2007). Inference attacks on location tracks. In *International Conference on Pervasive Computing*, (pp. 127–143). Springer.

Kumar, S., Jiang, M., Jung, T., Luo, R. J., & Leskovec, J. (2018). Mis2: Misinformation and misbehavior mining on the web. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, (pp. 799–800). ACM.

Kumar, S. & Shah, N. (2018). False information on web and social media: A survey. *arXiv preprint arXiv:1804.08559*.

Kwon, S., Cha, M., & Jung, K. (2017). Rumor detection over varying time windows. *PloS one*, *12*(1), e0168344.

Kwon, S., Cha, M., Jung, K., Chen, W., & Wang, Y. (2013). Prominent features of rumor propagation in online social media. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, (pp. 1103–1108). IEEE.

Laylavi, F., Rajabifard, A., & Kalantari, M. (2016). A multi-element approach to location inference of twitter: A case for emergency response. *ISPRS International Journal of Geo-Information*, *5*(5), 56.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436.

Leetaru, K., Wang, S., Cao, G., Padmanabhan, A., & Shook, E. (2013). Mapping the global twitter heartbeat: The geography of twitter. *First Monday*, *18*(5).

Leskovec, J. & Mcauley, J. J. (2012). Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, (pp. 539–547).

Li, C. & Sun, A. (2014). Fine-grained location extraction from tweets with temporal awareness. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, (pp. 43–52). ACM.

Li, R., Lei, K. H., Khadiwala, R., & Chang, K. C.-C. (2012). Tedas: A twitter-based event detection and analysis system. In *Data engineering (icde), 2012 ieee 28th international conference on*, (pp. 1273–1276). IEEE.

Li, R., Wang, S., & Chang, K. C.-C. (2012). Multiple location profiling for users and relationships from social network and content. *Proceedings of the VLDB Endowment*, *5*(11), 1603–1614.

Li, R., Wang, S., Deng, H., Wang, R., & Chang, K. C.-C. (2012). Towards social user profiling: unified and discriminative influence model for inferring home locations. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 1023–1031). ACM.

Li, W., Serdyukov, P., de Vries, A. P., Eickhoff, C., & Larson, M. (2011). The where in the tweet. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, (pp. 2473–2476). ACM.

Lin, K., Kansal, A., Lymberopoulos, D., & Zhao, F. (2010). Energy-accuracy trade-off for continuous mobile device location. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, (pp. 285–298). ACM.

Lingad, J., Karimi, S., & Yin, J. (2013). Location extraction from disaster-related microblogs. In *Proceedings of the 22nd international conference on world wide web*, (pp. 1017–1020). ACM.

Ma, J., Gao, W., Mitra, P., Kwon, S., Jansen, B. J., Wong, K.-F., & Cha, M. (2016). Detecting rumors from microblogs with recurrent neural networks. In *IJCAI*, (pp. 3818–3824).

Ma, J., Gao, W., Wei, Z., Lu, Y., & Wong, K.-F. (2015). Detect rumors using time series of social context information on microblogging websites. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, (pp. 1751–1754). ACM.

MacEachren, A. M., Jaiswal, A., Robinson, A. C., Pezanowski, S., Savelyev, A., Mitra, P., Zhang, X., & Blanford, J. (2011). Senseplace2: Geotwitter analytics support for situational awareness. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, (pp. 181–190). IEEE.

Mahmud, J., Nichols, J., & Drews, C. (2012). Where is this tweet from? inferring home locations of twitter users. *ICWSM*, *12*, 511–514.

Mahmud, J., Nichols, J., & Drews, C. (2014). Home location identification of twitter users. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *5*(3), 47.

McCallum, A., Nigam, K., et al. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, (pp. 41–48). Citeseer.

McGee, J., Caverlee, J., & Cheng, Z. (2013). Location prediction in social media based on tie strength. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, (pp. 459–468). ACM.

McGee, J., Caverlee, J. A., & Cheng, Z. (2011). A geographic study of tie strength in social media. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, (pp. 2333–2336). ACM.

Mehta, D. P. & Sahni, S. (2004). *Handbook of data structures and applications.* CRC Press.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association.*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, (pp. 3111–3119).

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, *38*(11), 39–41.

Newman, M. L., Pennebaker, J. W., Berry, D. S., & Richards, J. M. (2003). Lying words: Predicting deception from linguistic styles. *Personality and social psychology bulletin*, *29*(5), 665–675.

O'Connor, B., Balasubramanyan, R., Routledge, B. R., Smith, N. A., et al. (2010). From tweets to polls: Linking text sentiment to public opinion time series. *Icwsm*, *11*(122-129), 1–2.

O'Donovan, J., Kang, B., Meyer, G., Hollerer, T., & Adalii, S. (2012). Credibility in context: An analysis of feature distributions in twitter. In *Privacy, Security, Risk and Trust (PASSAT), 2012 international conference on and 2012 international confernece on social computing (SocialCom)*, (pp. 293–301). IEEE.

Paradesi, S. M. (2011). Geotagging tweets using their content. In *FLAIRS conference.*

Paul, M. J. & Dredze, M. (2011). You are what you tweet: Analyzing twitter for public health. *Icwsm*, *20*, 265–272.

Pennacchiotti, M. & Popescu, A.-M. (2011). A machine learning approach to twitter user classification. *Icwsm*, *11*(1), 281–288.

Pennebaker, J. W. & King, L. A. (1999). Linguistic styles: Language use as an individual difference. *77*(6), 1296.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*, volume 14, (pp. 1532–1543).

Pregibon, D. et al. (1981). Logistic regression diagnostics. *The Annals of Statistics*, *9*(4), 705–724.

Quinlan, J. R. (1987). Simplifying decision trees. *International journal of man-machine studies*, *27*(3), 221–234.

Rae, A., Murdock, V., Popescu, A., & Bouchard, H. (2012). Mining the web for points of interest. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, (pp. 711–720). ACM.

Ratinov, L. & Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, (pp. 147–155). Association for Computational Linguistics.

Ritter, A., Clark, S., Etzioni, O., et al. (2011). Named entity recognition in tweets: an experimental study. In *Proceedings of the conference on empirical methods in natural language processing*, (pp. 1524–1534). Association for Computational Linguistics.

Roller, S., Speriosu, M., Rallapalli, S., Wing, B., & Baldridge, J. (2012). Supervised text-based geolocation using language models on an adaptive grid. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, (pp. 1500–1510). Association for Computational Linguistics.

Ruchansky, N., Seo, S., & Liu, Y. (2017). Csi: A hybrid deep model for fake news detection.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Ryoo, K. & Moon, S. (2014). Inferring twitter user locations with 10 km accuracy. In *Proceedings of the 23rd International Conference on World Wide Web*, (pp. 643–648). ACM.

Sadilek, A., Kautz, H., & Bigham, J. P. (2012). Finding your friends and following them to where you are. In *Proceedings of the fifth ACM international conference on Web search and data mining*, (pp. 723–732). ACM.

Sakaki, T., Okazaki, M., & Matsuo, Y. (2010). Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, (pp. 851–860). ACM.

Samet, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, *16*(2), 187–260.

Samet, H., Rosenfeld, A., Shaffer, C. A., & Webber, R. E. (1984). A geographic information system using quadtrees. *Pattern Recognition*, *17*(6), 647–656.

Schifferes, S., Newman, N., Thurman, N., Corney, D., Göker, A., & Martin, C. (2014). Identifying and verifying news through social media: Developing a user-centred tool for professional journalists. *Digital Journalism*, *2*(3), 406–418.

Schulz, A., Hadjakos, A., Paulheim, H., Nachtwey, J., & Mühlhäuser, M. (2013). A multi-indicator approach for geolocalization of tweets. In *ICWSM*, (pp. 573–582).

Shu, K., Sliva, A., Wang, S., Tang, J., & Liu, H. (2017). Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, *19*(1), 22–36.

Shumaker, B. & Sinnott, R. (1984). Astronomical computing: 1. computing under the open sky. 2. virtues of the haversine. *Sky and telescope*, *68*, 158–159.

Silverman, C. (2016). Here are 50 of the biggest fake news hits on facebook from 2016. *BuzzFeed, https://www. buzzfeed. com/craigsilverman/top-fake-news-of-2016.*

Solon, O. (2016). Facebook's failure: Did fake news and polarized politics get trump elected. *The Guardian*, *10.*

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Stroud, F. fake news.

Tacchini, E., Ballarin, G., Della Vedova, M. L., Moret, S., & de Alfaro, L. (2017). Some like it hoax: Automated fake news detection in social networks. *arXiv preprint arXiv:1704.07506.*

Takhteyev, Y., Gruzd, A., & Wellman, B. (2012). Geography of twitter networks. *Social networks*, *34*(1), 73–81.

Tambuscio, M., Ruffo, G., Flammini, A., & Menczer, F. (2015). Fact-checking effect on viral hoaxes: A model of misinformation spread in social networks. In *Proceedings of the 24th International Conference on World Wide Web*, (pp. 977–982). ACM.

Tausczik, Y. R. & Pennebaker, J. W. (2010). The psychological meaning of words: Liwc and computerized text analysis methods. *Journal of language and social psychology*, *29*(1), 24–54.

Tong, S. & Koller, D. (2001). Support vector machine active learning with applications to text classification. *Journal of machine learning research*, *2*(Nov), 45–66.

Tri, N. T. & Jung, J. J. (2015). Exploiting geotagged resources to spatial ranking by extending hits algorithm. *Computer Science and Information Systems*, *12*(1), 185–201.

US Census Bureau, P. D. (2016). Annual estimates of resident population change for incorporated places of 50,000 or more in 2014, ranked by percent change: July 1, 2014 to july 1, 2015. `http://factfinder.census.gov/faces/tableservices/jsf/pages/productview.xhtml?src=bkmk`. Accessed: 2016/12/05.

Vincenty, T. (1975). Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey review*, *23*(176), 88–93.

Vosoughi, S., Roy, D., & Aral, S. (2018). The spread of true and false news online. *Science*, *359*(6380), 1146–1151.

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., & Xu, W. (2016). Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (pp. 2285–2294).

Wing, B. & Baldridge, J. (2014). Hierarchical discriminative classification for text-based geolocation. In *EMNLP*, (pp. 336–348).

Wing, B. P. & Baldridge, J. (2011). Simple supervised document geolocation with geodesic grids. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, (pp. 955–964). Association for Computational Linguistics.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann.

Wu, S., Hofman, J. M., Mason, W. A., & Watts, D. J. (2011). Who says what to whom on twitter. In *Proceedings of the 20th international conference on World wide web*, (pp. 705–714). ACM.

Yamaguchi, Y., Amagasa, T., Kitagawa, H., & Ikawa, Y. (2014). Online user location inference exploiting spatiotemporal correlations in social streams. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, (pp. 1139–1148). ACM.

Yang, Y., Zheng, L., Zhang, J., Cui, Q., Li, Z., & Yu, P. S. (2018). Ti-cnn: Convolutional neural networks for fake news detection. *arXiv preprint arXiv:1806.00749.*

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, (pp. 1480–1489).

Yardi, S., Romero, D., Schoenebeck, G., et al. (2009). Detecting spam in a twitter network. *First Monday*, *15*(1).

Zhang, Y., Jin, R., & Zhou, Z.-H. (2010). Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, *1*(1-4), 43–52.

Zheng, X., Han, J., & Sun, A. (2017). A survey of location prediction on twitter. *arXiv preprint arXiv:1705.03172*.

Zhong, N., Li, Y., & Wu, S.-T. (2012). Effective pattern discovery for text mining. *IEEE transactions on knowledge and data engineering*, *24*(1), 30–44.

Zubiaga, A., Aker, A., Bontcheva, K., Liakata, M., & Procter, R. (2018). Detection and resolution of rumours in social media: A survey. *ACM Computing Surveys (CSUR)*, *51*(2), 32.

Zubiaga, A., Liakata, M., & Procter, R. (2016). Learning reporting dynamics during breaking news for rumour detection in social media. *arXiv preprint arXiv:1610.07363*.

Appendices

Appendix A

Quadtree Location Inference Codes

This section contains the source Python codes used in the implementation of the methods and techniques for the grid-based quadtree content-only location inference with hybrid word embeddings. The python libraries used include: Scikit Learn (`www.scikit-learn.org`) Word2Vec by (Mikolov et al 2013) GeoPy (`https://geopy.readthedocs.io/en/stable/`) Matplotlib (`https://matplotlib.org/`) Pandas (`https://pandas.pydata.org/`) Numpy (`https://numpy.org/`) NLTK Tool Kit (`http://www.nltk.org/`)

```
"""
This is the main module of the project. It contains the entry point
CLI function of the application.
"""


import sys
import numpy as np
import preprocess
import classify
import cluster
#import draw
import settings
import pandas as pd
import joblib
```

```
from geopy.distance import great_circle
from matplotlib import pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import cohen_kappa_score
from scipy import interp, sparse


def _cluster(input_filename):
    return cluster.by_grid(input_filename, settings.CLUSTER.
        OUTPUT_FILENAME,
                            map_bounds=settings.MAP.BOUNDS, by_depth=settings
                                .CLUSTER.BY_DEPTH)


def _learn(input_filename, centroids, distance):
    # prepare the data
    corpus, labels = preprocess.read_corpus(input_filename)
    embedder = classify.TextEmbedder(distance)
    corpus, labels = embedder(corpus, labels)

    # split data and learn the model
    X, y = corpus, labels
    X_train, X_test, y_train, y_test = train_test_split (X, y,
                                                         test_size =settings.
                                                             CLASSIFY.TEST_SIZE
                                                             ,
                                                        random_state=settings.
                                                             CLASSIFY.
                                                             RANDOM_SEED)
    tfidf = preprocess.Vectorizer()
    logreg = classify.learn( tfidf . fit (X_train), y_train)

    # predict labels
    X_transformed = tfidf.transform(X)
    y_pred = logreg.predict(X_transformed)
```

```
sparse.save_npz(settings.ROC.XINPUT, X_transformed)
del X_transformed

X_test_transformed = tfidf.transform(X_test)
del tfidf
y_test_pred = logreg.predict(X_test_transformed)
accuracy, _ = classify.test(logreg, X_test_transformed, y_test)
kappa = cohen_kappa_score(y_test, y_test_pred)

# save data for the roc curve
y.to_csv(settings.ROC.YINPUT, sep='\t', header=True)

# save predicted labels
original_df = preprocess.read_dataframe(input_filename)
text = original_df[settings.CSV.INPUT.TEXT]
actual_latitude = original_df[settings.CSV.OUTPUT.ACTUAL_LATITUDE]
actual_longitude = original_df[settings.CSV.OUTPUT.
    ACTUAL_LONGITUDE]
y_pred = pd.Series(y_pred, name=settings.CSV.OUTPUT.PREDICTED_GRID)
predicted_latitude = pd.Series(
    [centroids[grid][settings.CSV.INPUT.LATITUDE] for grid in y_pred],
    name=settings.CSV.OUTPUT.PREDICTED_LATITUDE
)
predicted_longitude = pd.Series(
    [centroids[grid][settings.CSV.INPUT.LONGITUDE] for grid in y_pred],
    name=settings.CSV.OUTPUT.PREDICTED_LONGITUDE
)

error = pd.Series(
    list(map(lambda x, y: round(great_circle(x, y).kilometers, 2),
             zip(actual_latitude, actual_longitude), zip(
                 predicted_latitude, predicted_longitude)
             )),
    name=settings.CSV.OUTPUT.ERROR
)

dataframe = pd.concat([text,
                       actual_latitude,
                       actual_longitude,
                              predicted_latitude ,
                              predicted_longitude,
                              error ,
                              labels ,
                              y_pred],  axis=1)

    output_filename = settings.CLASSIFY.OUTPUT_FILENAME
    dataframe.to_csv(output_filename, sep='\t', header=True)
    return accuracy, kappa, (y_test, y_test_pred)


def _visualize (coords, y_values, tree, map_bounds, output_filename):
    draw.draw_result(coords, y_values, tree, max_depth=tree.height,
                     map_bounds=map_bounds)
    plt . savefig (output_filename, format='svg', dpi=2000)
    print("Map file:", output_filename)


def _roc(X, y):
    # Binarize the output
    y = label_binarize (y, classes =sorted(y.unique()))
    n_classes = y.shape[1]

    # shuffle and split training and test sets
    X_train, X_test, y_train, y_test = train_test_split (X, y,
                                                         test_size =settings.
                                                             CLASSIFY.TEST_SIZE
                                                             ,
                                                        random_state=settings.
                                                             CLASSIFY.
                                                             RANDOM_SEED)

    # Learn to predict each class against the other
    classifier = OneVsRestClassifier(LogisticRegression())
    y_score = classifier . fit (X_train, y_train). decision_function (X_test)

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
```

```
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
         ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
output_filename = settings.CLASSIFY.ROC_FILENAME
```

```
    plt . savefig (output_filename, format='svg', dpi=2000)
    print("ROC file:", output_filename)


def cli () :
    """
    The entry point function.
    """

    from datetime import datetime
    start = datetime.now()

    command = sys.argv[1]
    if command == 'cluster':
        input_filename = sys.argv [2]

        # cluster corpus by geocoordinates
        coords, tree , centroids = _cluster (input_filename)

        # serialize results
        joblib .dump(coords, settings.CLUSTER.COORDS_FILENAME)
        joblib .dump(tree, settings . CLUSTER.TREE_FILENAME)
        joblib .dump(centroids, settings . CLUSTER.CENTROIDS_FILENAME)
    elif command == 'classify':
        # learn model and predict geolabels by text content

        coords = joblib . load( settings . CLUSTER.COORDS_FILENAME)
        tree = joblib . load( settings . CLUSTER.TREE_FILENAME)
        centroids = joblib . load( settings . CLUSTER.CENTROIDS_FILENAME)

        accuracy, kappa, y_values = _learn( settings . CLUSTER.
            OUTPUT_FILENAME,
                                           centroids ,
                                           settings . CLASSIFY.DISTANCE)
        print("k-fold CV accuracy:", accuracy)
        print("Cohen Kappa score:", kappa)

        # draw a map
        #_visualize (coords, y_values, tree ,
        #              settings .MAP.BOUNDS, settings.MAP.OUTPUT_FILENAME)
    elif command == 'roc':
        X = sparse.load_npz(settings.ROC.XINPUT)
        y = pd.Series.from_csv(settings.ROC.YINPUT, sep='\t', header=0)
        _roc(X, y)
    else:
        print('Wrong command:', command)

    time_elapsed = datetime.now() − start
    print('Time elpased (hh:mm:ss.ms) {}'.format(time_elapsed))


if __name__ == "__main__":
    cli ()
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−


"""
This module is aimed to cluster tweets location based on its content into grids.
"""

from __future__ import print_function

import itertools
import numpy as np
import pandas as pd
import settings
import preprocess



def _get_bounds(coords):
    """
    Return a bounding box coordinates for a collection of points from dataframe.
    """
    return ((coords.min()[settings.CSV.INPUT.LATITUDE],
             coords.max()[settings.CSV.INPUT.LATITUDE]),
            (coords.min()[settings.CSV.INPUT.LONGITUDE],
            coords.max()[settings.CSV.INPUT.LONGITUDE]))


def in_bounds(dataframe, bounds):
    """
    Return the dataframe of the points that are in the map bounding box.
    """
    precision = 1e-5
    return dataframe.query(settings.CSV.INPUT.LATITUDE + '>=' + str(bounds
        [0][0] - precision)) \
                    .query(settings.CSV.INPUT.LATITUDE + '<=' + str(bounds
                        [0][1] + precision)) \
                    .query(settings.CSV.INPUT.LONGITUDE + '>=' + str(
                        bounds[1][0] - precision)) \
                    .query(settings.CSV.INPUT.LONGITUDE + '<=' + str(
                        bounds[1][1] + precision))


def _to_cells(coords, bounding_box, pattern):
    """
    Split a dataframe with geocoordinates into four cells.
    """
    batches = []
    cells = []

    ((lat_0, lat_n), (lon_0, lon_n)) = bounding_box

    lats = np.linspace(lat_0, lat_n, pattern + 1)
    lons = np.linspace(lon_0, lon_n, pattern + 1)

    cells = list(itertools.product(
        zip(lats[:], lats[1:]),
        zip(lons[:], lons[1:])
    ))
    batches = [in_bounds(coords, cell) for cell in cells]
    return batches, cells


# stop criteria
```

```
def _stop_by_depth(self, _):
    return self.depth >= self.max_depth


def _stop_by_point_number(self, coords):
    return len(coords) <= self.max_per_grid


def _to_dec(number, base):
    """
    Convert a number to decimal from 'base'
    """
    assert type(number) is list
    return sum([(int(v) * base**i) for i, v in enumerate(number[::-1])])


class QuadTree:
    """Quad-tree class which recursively subdivide the space into quadrants"""

    def __init__(self, coords, depth=0,
                 max_per_grid=settings.CLUSTER.MAXIMUM_PER_GRID,
                 max_depth=settings.CLUSTER.MAXIMUM_DEPTH,
                 pattern=settings.CLUSTER.PATTERN,
                 bounds=None, stop_criteria=None):
        coords = coords.sort_values([settings.CSV.INPUT.LATITUDE, settings.
            CSV.INPUT.LONGITUDE])
        self.children = []
        self.coords = pd.DataFrame([])
        self.bounds = _get_bounds(coords) if not bounds else bounds
        self.depth = depth
        self.max_depth = max_depth
        self.max_per_grid = max_per_grid
        self.stop_criteria = stop_criteria
        self.pattern = pattern
        self.height = self.depth

        # stop if no point anymore
        if len(coords) == 0:
            return
        # stop by chosen criteria
        if stop_criteria(self, coords):
            self.coords = coords
            return
        else:
            batches, cells = _to_cells(coords, self.bounds, pattern)
            for batch, cell in zip(batches, cells):
                self.children.append(QuadTree(
                    batch, depth=depth + 1, bounds=cell, stop_criteria=
                        stop_criteria, pattern=pattern))

        # calculation the height of the whole tree
        self.height = max(self.height, max(child.height for child in self.children)
            )


    def assign_labels(self, grid_labels, centroids, path):
        """
        DFS procudure to assign labels to every point.
        """

        if len(self.coords) > 0:
            class_id = "G" + str(_to_dec(path, self.pattern ** 2))
            for key in self.coords.index:
                grid_labels[key] = class_id

            centroids[class_id] = self.coords.mean()
        else:
            for i, child in enumerate(self.children):
                child.assign_labels(grid_labels, centroids, path + [i])


def by_grid(input_filename, output_filename, map_bounds, by_depth):
    """
    The entry point of the module.
    """

    dataframe = preprocess.read_dataframe(input_filename)
    # fliter points that are not in the bounds
    dataframe = in_bounds(dataframe, map_bounds)
    coords = dataframe[[settings.CSV.INPUT.LATITUDE, settings.CSV.INPUT.
        LONGITUDE]]

    stop_criteria = _stop_by_depth if by_depth else _stop_by_point_number
    tree = QuadTree(coords, stop_criteria=stop_criteria)

    # write output csv file
    centroids = {}
    grid_labels = {key: None for key in list(dataframe.index)}
    tree.assign_labels(grid_labels, centroids, [])
    label_values = [grid_labels[key] for key in dataframe.index]
    dataframe[settings.CSV.OUTPUT.ACTUAL_GRID] = label_values
    output_dataframe = \
        dataframe.loc[:, (settings.CSV.INPUT.TEXT,
                          settings.CSV.INPUT.LATITUDE,
                          settings.CSV.INPUT.LONGITUDE,
                          settings.CSV.OUTPUT.ACTUAL_GRID)]

    output_dataframe = output_dataframe.rename(index=str, columns={
        settings.CSV.INPUT.LATITUDE: settings.CSV.OUTPUT.
            ACTUAL_LATITUDE,
        settings.CSV.INPUT.LONGITUDE: settings.CSV.OUTPUT.
            ACTUAL_LONGITUDE
    })
    output_dataframe.to_csv(output_filename, sep='\t', header=True)
    return coords, tree, centroids


if __name__ == "__main__":
    raise RuntimeError("This module is not supposed to be called.")
%------------------------------------------------------------

%------------------------------------------------------------

"""
This module is aimed to classify and predict tweets label based on its content.
"""
```

```
from __future__ import print_function

import os
import joblib
import numpy as np
from sklearn.naive_bayes import MultinomialNB
from sklearn import tree
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from gensim.models import word2vec
from ctypes import cdll, POINTER, c_uint64, c_size_t, c_double
import settings


jaccard_lib = None

def _jaccard(x, y):
    """
    Generalized Jaccard index of similarity
    """
    xdata_p = x.ctypes.data_as(POINTER(c_double))
    ydata_p = y.ctypes.data_as(POINTER(c_double))
    return jaccard_lib.jaccard(xdata_p, ydata_p, len(x))


def _normalize(model):
    """
    Normalize word embeddings
    """
    result = {}
    for word in model.wv.vocab:
        result[word] = model.wv[word] + abs(min(model.wv[word]))
    return result


class TextEmbedder:
```

```
"""
Word2Vec embedding for tweet words, and reducing the dictionary
"""

def __init__(self, distance):
    self.distance = distance
    self.model = None
    self.normalized_wv = None

def __call__(self, corpus, labels):
    """
    Reduce total count of words used in corpus by testing their
    word2vec embeddings on high similarity
    """
    self.model = self._embed(corpus)
    self.normalized_wv = _normalize(self.model)
    corpus = self._dimreduce(corpus)
    return corpus, labels

def _embed(self, corpus):
    """
    Create word2vec embeddings for words from the corpus
    """

    model = word2vec.Word2Vec(corpus,
                              min_count=settings.CLASSIFY.
                                  W2V_MIN_COUNT,
                              size=settings.CLASSIFY.W2V_SIZE,
                              window=settings.CLASSIFY.W2V_WINDOW,
                              workers=settings.CLASSIFY.W2V_WORKERS)
    model.save(settings.CLASSIFY.W2V_FILENAME)
    return model

def _replace_condition(self, word, synonim):
    """
    Create a boolean condition of word replacement
    """
    cosin_sim = self.model.similarity(word, synonim)
    jaccard_sim = _jaccard(self.normalized_wv[word], self.normalized_wv[
            synonim])

        if self.distance == settings.CLASSIFY.COSINE:
            condition = cosin_sim >= settings.CLASSIFY.COSINE_THRESHOLD
        elif self.distance == settings.CLASSIFY.JACCARD:
            condition = jaccard_sim >= settings.CLASSIFY.
                JACCARD_THRESHOLD
        else:
            condition = cosin_sim >= settings.CLASSIFY.COSINE_THRESHOLD
                and \
                jaccard_sim >= settings.CLASSIFY.JACCARD_THRESHOLD

        condition = condition and self.model.wv.vocab[synonim].count > self.model.
            wv.vocab[word].count
        return condition

    def _most_similar(self, word):
        """
        Find a most similar word for a given word
        """
        if self.distance == settings.CLASSIFY.JACCARD:
            synonim, _ = min([(x, _jaccard(self.normalized_wv[word], self.
                normalized_wv[x])) for x in self.model.wv.vocab],
                             key = lambda x: x[1])
        else:
            synonim, _ = self.model.wv.most_similar(word)[0]
        return synonim

    def _dimreduce(self, corpus):
        """
        Replace similar words by their synonyms
        """

        to_replace = {}
        for word in self.model.wv.vocab:
            synonim = self._most_similar(word)
            if self._replace_condition(word, synonim):
                to_replace[word] = synonim
        for i in range(len(corpus)):
            corpus[i] = [to_replace[word]
                         if word in to_replace else word for word in corpus[i]]

        # save fixed words
        with open("dictionary.txt", 'w') as g:
            print("\n".join("{}\t{}".format(k, v)
                          for k, v in to_replace.items()), file=g)

        print("Words eliminating:", len(self.model.wv.vocab), "->",
              len(self.model.wv.vocab) - len(to_replace))
        return corpus


def learn(X, y):
    """
    Learn logit model to predict labels by vectorized corpus
    """

    with open('ylog.txt', 'w') as ylog:
        print(y, file=ylog)

    with open('Xlog.txt', 'w') as xlog:
        print(X, file=xlog)

    # Fit Logistic Regression model to the dataset
    logreg = MultinomialNB()
    #logreg = LogisticRegression()
    #logreg = tree.DecisionTreeClassifier(random_state=0, max_depth=10)
    logreg.fit(X, y)

    # save the logit model
    joblib.dump(logreg, settings.CLASSIFY.LOGIT_FILENAME)
    return logreg


def test(logreg, X, y):
    """
    Test accuracy of prediction
    """
    # Applying k-Fold Cross Validation
    accuracies = cross_val_score(
        estimator=logreg,
        X=X, y=y,
        cv=settings.CLASSIFY.CROSS_VALID_K)

    with open(settings.CLASSIFY.CV_ACCURACY_FILENAME, 'w') as output:
        print(accuracies.mean(), file=output)

    # calculate accuracy of class predictions
    pre_rec_fm = metrics.classification_report (y, logreg.predict(X))
    with open(settings.CLASSIFY.F1_FILENAME, 'w') as output:
        print(pre_rec_fm, file=output)

    return accuracies.mean(), pre_rec_fm


def find_lib(directory: str, prefix: str) -> str:
    for f in os.listdir(directory):
        fullname = os.path.join(directory, f)
        if os.path.isfile(fullname) and f.startswith(prefix) and f.endswith("so"):
            return fullname

    raise ValueError("Library '%s' not found" % prefix)



if __name__ == "__main__":
    raise RuntimeError("This module is not supposed to be called.")
else:
    libdir = os.path.dirname(os.path.abspath(__file__))
    jaccard_lib = cdll.LoadLibrary(find_lib(libdir, "jaccard"))
    jaccard_lib.jaccard.argtypes = [POINTER(c_double), POINTER(c_double),
        c_size_t]
    jaccard_lib.jaccard.restype = c_double
%-----------------------------------------------------------------

%-----------------------------------------------------------------
```

```
#This module does text pre-processing of the tweets
"""
I/O functions
"""


import re
import pandas as pd
import nltk
import settings
import joblib
import csv
from sklearn.feature_extraction.text import TfidfVectorizer, \
    TfidfTransformer, \
    CountVectorizer


def get_stop_words():
    """
    Setup nltk on import
    """
    nltk.download('stopwords')
    return set(nltk.corpus.stopwords.words('english'))


STOP_WORDS = get_stop_words()


def read_dataframe(input_filename):
    """
    Read a csv file as pandas dataframe.
    """
    with open(input_filename, 'r'):
        return pd.read_csv(input_filename, sep='\t', quoting=csv.QUOTE_NONE)


def _tokenize(tweet):
    """
    Split a tweet text into tokens, and remove stop words
    """
    tweet = re.sub(r"http\S+", "", tweet)
    tokens = re.findall(r'\w+', tweet.lower())
    tokens = [token for token in tokens if not token in STOP_WORDS]
    return tokens if len(tokens) > 1 else tokens * 2


def read_corpus(input_filename):
    """
    Read and preprocess tweet dataset.
    """
    dataset = read_dataframe(input_filename)
    dataset[settings.CSV.OUTPUT.TEXT] = dataset[settings.CSV.OUTPUT.
        TEXT].apply(
        _tokenize)
    return dataset[settings.CSV.OUTPUT.TEXT], dataset[settings.CSV.OUTPUT.
        ACTUAL_GRID]


class Vectorizer:
    """
    Vectorize corpus by transforming into TF-IDF matrix
    """

    def __init__(self):
        self.vocabulary = None

    def _load_vocabulary(self):
        self.vocabulary = joblib.load(settings.CLASSIFY.TFIDF_FILENAME)

    def _join_words(self, corpus):
        return [' '.join(tokens for tokens in tweet) for tweet in corpus]

    def fit(self, corpus):
        """
        Fit and transform new data to TF-IDF matrix
        """
        vectorizer = CountVectorizer(decode_error="replace", min_df=1)
        corpus = self._join_words(corpus)
        vec_train = vectorizer.fit_transform(corpus)
        # save vocabulary
        self.vocabulary = vectorizer.vocabulary_
        joblib.dump(vectorizer.vocabulary_, settings.CLASSIFY.
            TFIDF_FILENAME)

        return vec_train

    def transform(self, corpus):
        """
        Transform to TF-IDF matrix using only word vocabulary of previous fit
        """
        transformer = TfidfTransformer()
        if not self.vocabulary:
            self._load_vocabulary()

        loaded_vec = CountVectorizer(decode_error="replace",
                                     vocabulary=self.vocabulary)

        corpus = self._join_words(corpus)
        return transformer.fit_transform(loaded_vec.fit_transform(corpus))
%------------------------------------------------------------

%------------------------------------------------------------

#This module contains the Python requirements for running the codes
numpy==1.13.1
pandas==0.20.3
gensim==2.3.0
geopy==1.11.0
joblib==0.11
nltk==3.2.4
scikit_learn==0.19.0
cycler==0.10.0
matplotlib==2.0.2
olefile==0.44
Pillow==4.2.1
pyparsing==2.2.0
python-dateutil==2.6.1
```

```
pytz==2017.2
scipy==0.19.1
six==1.10.0
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

#This module contains all the settings for the running the software
"""
Settings and constant values of the project.
"""

from bunch import Bunch

# Geographic map parameters
MAP = Bunch({

    # Including Alaska
    #'BOUNDS': ((5.49955, 83.162102), (−167.27641, −52.23304)),

    # Only continetal part of the USA
    'BOUNDS': ((20, 50), (−127, −64)),

    # Output map colors
    'WATER_COLOR': '#9db8d3',
    'LAND_COLOR': '#f8f7f0',

    'OUTPUT_FILENAME': 'map.svg',

    'TRAIN_COLOR': 'gray',
    'FALSE_COLOR': 'r',
    'TRUE_COLOR': 'g',
    'BOUNDS_COLOR': 'b',

})

# Quadtree clustering parameters
CLUSTER = Bunch({
    # Change this boolean value to use maximum−depth stop criteria
    'BY_DEPTH': True,

    # Cell pattern for  splitting  (i.e. 2x2 for value  of  2)
    'PATTERN': 2,

    'MAXIMUM_PER_GRID': 5000,
    'MAXIMUM_DEPTH': 4,
    'OUTPUT_FILENAME': 'clustered.csv',
    'COORDS_FILENAME': 'coords.dat',
    'TREE_FILENAME': 'tree.dat',
    'CENTROIDS_FILENAME': 'centroids.dat',
})

# Word2Vec and logit parameters
CLASSIFY = Bunch({
    'W2V_FILENAME': 'w2v.dat',
    'TFIDF_FILENAME': 'vocabulary.dat',
    'LOGIT_FILENAME': 'logit.dat',
    'CV_ACCURACY_FILENAME': 'accuracy.txt',
    'F1_FILENAME': 'pre_rec_fm.txt',
    'ROC_FILENAME': 'roc.svg',
    'OUTPUT_FILENAME': 'predicted.csv',

    # Word2Vec learning paramters
    'W2V_MIN_COUNT': 1,
    'W2V_SIZE': 200,
    'W2V_WINDOW': 3,
    'W2V_WORKERS': 4,



    # Change this to use only cosine distance as  criteria
    # of words similarity
    #'DISTANCE': 'cosine',
    #'DISTANCE': 'jaccard',
    'DISTANCE': 'hybrid',

    # possible DISTNACE values: cosine, jaccard, hybrid
    'COSINE': 'cosine',
    'JACCARD': 'jaccard',
    'HYBRID': 'hybrid',

    # Similarity thresholds for words eliminating
    'COSINE_THRESHOLD': 0.99,
    'JACCARD_THRESHOLD': 0.95,

    # Size of test part for logit learning
    'TEST_SIZE': 0.2,
    'CROSS_VALID_K': 3,
    'RANDOM_SEED': 42
})


# CSV input/output file column naming convention
CSV = Bunch({
    'INPUT' : Bunch({
        'INDEX': 'index_tweet',
        'TEXT': 'text',
        'LATITUDE': 'geocoordinate0',
        'LONGITUDE': 'geocoordinate1',
    }),

    'OUTPUT' : Bunch({
        'INDEX': 'index_tweet',
        'TEXT': 'text',

        'ACTUAL_GRID': 'ActualGrid',
        'PREDICTED_GRID': 'PredictedGrid',

        'ACTUAL_LATITUDE': 'ActualLatitude',
        'ACTUAL_LONGITUDE': 'ActualLongitude',

        'PREDICTED_LATITUDE': 'PredictedLatitude',
        'PREDICTED_LONGITUDE': 'PredictedLongitude',
        'ERROR': 'GreatCircle(km)'
    })
})

ROC = Bunch({
    'XINPUT': 'roc_x.npz',
    'YINPUT': 'roc_y.csv',
})
%====================================
%====================================
```
## Description and READ ME file for running the application

This project **is** aimed at predicting tweet geolocation by its contents only.

### Clustering

1. At the beginning it performs quadtree clustering of tweets by their latitude **and** longitude coordinates.

2. Every tweet gets a grid label, which **is** a unique label of tree node it was clustered **in**.

### Reducing words space dimensionality

3. Then the dictionary of the words used **in all** tweets **is** embedded by word2vec CBOW model.

4. Embedded word vectors are used to determine words similarity using cosine distance, **or** combination of cosine distance **and** generalized jaccard similarity .

5. Most similar words **from** the step 3 are eliminated by replacing them to their synonims.

### Learning model to predict location

6. Reduced tweets are embedded using TF−IDF model.

7. Vectors **from** the previous step are used **in** learning logit regression **for** predict grid labels **from** the step 1.

~~~~
### Visualisation

8. Origin and predicted geocoordinates are used to draw tweet points on the map.

9. Origin (from clustering stage) and predicted (from learning + testing stage)
   grid labels are used to colorize tweet points. Green and red points correspond
    to correct and incorrect label predictions respectively.

## Settings
You can tune clustering, embedding, learning stages by changing following
   parameters in ```settings.py```:

- ```settings.CLUSTER.BY_DEPTH``` [Boolean]: use maximum-depth as stop
   condition of quadtree clustering

- ```settings.CLUSTER.PATTERN'``` [Int]: cell pattern for splitting step. Every
   split node will contains (Pattern x Pattern) number of child nodes.

- ```settings.CLUSTER.MAXIMUM_PER_GRID``` [Int]: maximum tweets per grid
   allowed (if default criteria is used)

- ```settings.CLUSTER.MAXIMUM_DEPTH``` [Int]: maximum tree depth allowed
   (if BY_DEPTH criteria is used)

- ```settings.CLASSIFY.USE_HYBRID``` [Boolean]: use both cosine and
   generalized jaccard similarity thresholds of word similarity as condition for
   eliminating similar words

- ```settings.CLASSIFY.*_THRESHOLD``` [Float]: similarity threshold for cosine/
   generalized jaccard distance, used in words eliminating stage

- ```settings.CLASSIFY.W2V*```: word2vec CBOW parameters. See gensim
   word2vec documentation for more detailed description.



## Usage

Input csv format must be formatted as following, using tab as separate character:
~~~~

```
‘‘‘
index_tweet text  geocoordinate0 geocoordinate1
 ...   ...   ...   ...
‘‘‘
When geocoordinate0, geocoordinate1 are latitude, longitude respectively .

To install the application via pip (you may want to do this under virtualenv):
‘‘‘
pip −e install  .
‘‘‘

Run the application using ‘tweetmap‘ command:
‘‘‘
tweetmap cluster INPUT_FILE
tweetmap classify
tweetmap roc
‘‘‘

%====================
%====================
”””
#Module for drawing tweets and quadtree on the map.
”””

import numpy as np
from mpl_toolkits.basemap import Basemap
from matplotlib import pyplot as plt
import settings


def draw_bounds(bmap, tree, max_depth):
    ”””
    Draw a proper bounding rectangle of quadtree on basemap
    ”””

    bounds_lons = [tree.bounds[1][0], tree.bounds[1][0],
                   tree.bounds[1][1], tree.bounds[1][1], tree.bounds[1][0]]
    bounds_lats = [tree.bounds[0][0], tree.bounds[0][1],
                   tree.bounds[0][1], tree.bounds[0][0], tree.bounds[0][0]]
    bmap.plot(bounds_lons, bounds_lats, linewidth=0.2,
              color=settings.MAP.BOUNDS_COLOR, zorder=3)

    if not tree.children or tree.depth >= max_depth:
        return
    else:
        for child in tree.children:
            draw_bounds(bmap, child, max_depth)


def draw_result(coords, labels, tree, max_depth, map_bounds):
    """
    Draw a geographic map and the resulting tree with tweets on it.
    """

    bmap = Basemap(projection='cyl', llcrnrlon=map_bounds[1][0], llcrnrlat=
        map_bounds[0][0],
                   urcrnrlon=map_bounds[1][1], urcrnrlat=map_bounds[0][1],
                   resolution='l')

    # draw continents, countries, states
    bmap.drawcoastlines()
    bmap.drawcountries(linewidth=1.5)
    bmap.fillcontinents(
        color=settings.MAP.LAND_COLOR, lake_color=settings.MAP.
            WATER_COLOR)
    bmap.drawmapboundary(fill_color=settings.MAP.WATER_COLOR)

    y_test, y_pred = labels

    # draw tweet points
    dataframe = coords.join(y_pred == y_test)
    test_df = dataframe.dropna()
    train_df = dataframe.ix[ dataframe[settings.CSV.OUTPUT.ACTUAL_GRID].
        isnull().nonzero() ]
    train_points = np.array([point for _, point in train_df.iterrows()])
    true_df = test_df.loc[test_df[settings.CSV.OUTPUT.ACTUAL_GRID] ==
        True]
    false_df = test_df.loc[test_df[settings.CSV.OUTPUT.ACTUAL_GRID] ==
        False]
    true_points = np.array([point for _, point in true_df.iterrows()])
    false_points = np.array([point for _, point in false_df.iterrows()])

    handle_0 = bmap.scatter(train_points[:, 1], train_points[:, 0],
                    marker='o', color=settings.MAP.TRAIN_COLOR, zorder=2, s
                        =0.5)
    handle_1 = bmap.scatter(false_points[:, 1], false_points[:, 0],
                    marker='o', color=settings.MAP.FALSE_COLOR, zorder=2, s=1)
    handle_2 = bmap.scatter(true_points[:, 1], true_points[:, 0],
                    marker='o', color=settings.MAP.TRUE_COLOR, zorder=2, s=1)



    plt.xlabel('')
    plt.ylabel('')
    plt.title('')
    plt.legend((handle_0, handle_1, handle_2),
               ('Train points', 'False prediction', 'True prediction'),
               ncol=3, loc=8)

    # draw quadtree regions layout
    draw_bounds(bmap, tree, max_depth)
%=============================================================================

%=============================================================================

#include <cstddef>
#include <vector>

#include <iostream>

typedef double num_t;

double _jaccard(const num_t* x_val, const num_t* y_val, const size_t len)
{
    std::vector<num_t> x(x_val, x_val + len);
    std::vector<num_t> y(y_val, y_val + len);

    num_t minsum = 0, maxsum = 0;
    for ( size_t  i  = 0; i  < len;  ++i)
    {
        minsum += std::min(x[i], y[i]) ;
        maxsum += std::max(x[i], y[i]);
    }
    return minsum / maxsum;
}


extern "C" {
    double jaccard(const num_t∗ x_val, const num_t∗ y_val, const size_t len)
    {
        return _jaccard(x_val, y_val, len);
    }
}


int main()
{
    std::vector<double> x = {1.0, 2.0, 3.0, 4.0, 5.0};
    std::vector<double> y = {3.0, 1.0, 2.0, 5.0, 4.0};
    std::cout << jaccard(&x[0], &y[0], 5) << std::endl;
    return 0;
}
```

Appendix B

Python Codes for Fake News Detection - Text only

This section contains the source codes and algorithms used in the implementation of the methods and techniques for the Fake News detection using text only.

The python libraries used include: Keras (`https://keras.io/`) Scikit Learn (`www.scikit-learn.org`) Pandas (`https://pandas.pydata.org/`) Numpy (`https://numpy.org/`)

```
# LSTM classification model of the Rumor-non Rumor Dataset
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import tensorflow as tf
from metrics import precision
from metrics import recall
from metrics import fmeasure
from sklearn.model_selection import StratifiedKFold
import keras.backend as k
```

```
# fix random seed for reproducibility
seed = 0
np.random.seed(seed)

# Importing the training set
data_set = pd.read_csv('RNR.tsv', delimiter = '\t', quoting = 3, header = None)
X = data_set.iloc [:, 1]
Y = data_set.iloc [:, 6]

# tokenize the training texts and make it sequential
top_words = 50
tokenizer = Tokenizer(num_words=top_words)
tokenizer. fit_on_texts (X)
sequences_train = tokenizer. texts_to_sequences (X)

# tokenize the testing texts and make it sequential
tokenizer. fit_on_texts (X)
sequences_test = tokenizer. texts_to_sequences (X)

#word_index_train = tokenizer.word_index_train
#print('Found %s unique tokens.' % len(word_index_train))
Y = Y.values.reshape(2600,)
print(Y.shape)
# truncate and pad input sequences
max_tweet_length = 300
X = sequence.pad_sequences(sequences_train, maxlen=max_tweet_length)

# define 10-fold cross validation test harness
kfold = StratifiedKFold( n_splits =10, shuffle =True, random_state=seed)
cvscores = []
for train , test in kfold . split (X, Y):

    # create the model
    embedding_vector_length = 32
    model = Sequential()
    # load the dataset with word embedding but only keep the top n words, zero the
        rest
    model.add(Embedding(top_words, embedding_vector_length, input_length=
        max_tweet_length))

    model.add(LSTM(100))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy
        ',precision,recall, fmeasure])
    print(model.summary())
    model.fit(X[train], Y[train], epochs=50, batch_size=64)

    # Final evaluation of the model
    scores = model.evaluate(X[test], Y[test], verbose=0)
    print("Accuracy: %.2f%%" % (scores[1]*100))
    print("Precision: %.2f%%" % (scores[2]*100))
    print("Recall: %.2f%%" % (scores[3]*100))
    print("Fmeasure: %.2f%%" % (scores[4]*100))
    print(scores)
    cvscores.append(scores[1] * 100)
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
k.clear_session()
%--------------------------------------------------------------

%--------------------------------------------------------------

# LSTM classification model with CNN of the Rumor-non Rumor Dataset
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import tensorflow as tf
from metrics import precision
from metrics import recall
from metrics import fmeasure
from sklearn.model_selection import StratifiedKFold
import keras.backend as k
# fix random seed for reproducibility
seed = 0
np.random.seed(seed)

# Importing the training set
data_set = pd.read_csv('RNR.tsv', delimiter = '\t', quoting = 3, header = None)
X = data_set.iloc [:, 1]
Y = data_set.iloc [:, 6]

# tokenize the training texts and make it sequential
top_words = 50
tokenizer = Tokenizer(num_words=top_words)
tokenizer. fit_on_texts (X)
sequences_train = tokenizer.texts_to_sequences(X)

# tokenize the testing texts and make it sequential
tokenizer. fit_on_texts (X)
sequences_test = tokenizer.texts_to_sequences(X)

#word_index_train = tokenizer.word_index_train
#print('Found %s unique tokens.' % len(word_index_train))
Y = Y.values.reshape(2600,)
print(Y.shape)
# truncate and pad input sequences
max_tweet_length = 300
X = sequence.pad_sequences(sequences_train, maxlen=max_tweet_length)

# define 10-fold cross validation test harness
kfold = StratifiedKFold( n_splits=10, shuffle=True, random_state=seed)
cvscores = []
for train, test in kfold. split (X, Y):

    # create the model
    embedding_vector_length = 32
    model = Sequential()
    # load the dataset with word embedding but only keep the top n words, zero the
        rest
    model.add(Embedding(top_words, embedding_vector_length, input_length=
        max_tweet_length))

    # include the conolution and maxpooling layers
    model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(LSTM(100))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy
        ', precision, recall ,fmeasure])
    print(model.summary())
    model.fit(X[train], Y[train], epochs=50, batch_size=64)

    # Final evaluation of the model
    scores = model.evaluate(X[test], Y[test], verbose=0)
    print("Accuracy: %.2f%%" % (scores[1]*100))
    print("Precision: %.2f%%" % (scores[2]*100))
    print("Recall: %.2f%%" % (scores[3]*100))
    print("Fmeasure: %.2f%%" % (scores[4]*100))
    print(scores)
    cvscores.append(scores[1] * 100)
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
%--------------------------------------------------------------------

%--------------------------------------------------------------------

# LSTM classification model with Dropout Regularization of the Rumor-non Rumor
    Dataset
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import tensorflow as tf
```

```
from metrics import precision
from metrics import recall
from metrics import fmeasure
from sklearn.model_selection import StratifiedKFold
import keras.backend as k
# fix random seed for reproducibility
seed = 0
np.random.seed(seed)

# Importing the training set
data_set = pd.read_csv('RNR.tsv', delimiter = '\t', quoting = 3, header = None)
X = data_set.iloc [:, 1]
Y = data_set.iloc [:, 6]

# tokenize the training texts and make it sequential
top_words = 50
tokenizer = Tokenizer(num_words=top_words)
tokenizer. fit_on_texts (X)
sequences_train = tokenizer.texts_to_sequences(X)

# tokenize the testing texts and make it sequential
tokenizer. fit_on_texts (X)
sequences_test = tokenizer.texts_to_sequences(X)

#word_index_train = tokenizer.word_index_train
#print('Found %s unique tokens.' % len(word_index_train))
Y = Y.values.reshape(2600,)
print(Y.shape)
# truncate and pad input sequences
max_tweet_length = 300
X = sequence.pad_sequences(sequences_train, maxlen=max_tweet_length)

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
cvscores = []
for train, test in kfold.split(X, Y):

   # create the model
   embedding_vector_length = 32
```

```
    model = Sequential()
    # load the dataset with word embedding but only keep the top n words, zero the
        rest
    model.add(Embedding(top_words, embedding_vector_length, input_length=
        max_tweet_length))

    # include a dropout layer
    model.add(Dropout(0.2))
    model.add(LSTM(100))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy
        ',precision,recall, fmeasure])
    print(model.summary())
    model.fit(X[train], Y[train], epochs=50, batch_size=64)

    # Final evaluation of the model
    scores = model.evaluate(X[test], Y[test], verbose=0)
    print("Accuracy: %.2f%%" % (scores[1]*100))
    print("Precision: %.2f%%" % (scores[2]*100))
    print("Recall: %.2f%%" % (scores[3]*100))
    print("Fmeasure: %.2f%%" % (scores[4]*100))
    print(scores)
    cvscores.append(scores[1] * 100)
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
k. clear_session ()
%------------------------------------------------------------

%------------------------------------------------------------


import keras.backend as K
def precision(y_true, y_pred):
    """Precision metric.

    Only computes a batch-wise average of precision.

    Computes the precision, a metric for multi-label classification of
    how many selected items are relevant.
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision


def recall(y_true, y_pred):
    """Recall metric.

    Only computes a batch-wise average of recall.

    Computes the recall, a metric for multi-label classification of
    how many relevant items are selected.
    """
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall


def fbeta_score(y_true, y_pred, beta=1):
    """Computes the F score.

    The F score is the weighted harmonic mean of precision and recall.
    Here it is only computed as a batch-wise average, not globally.

    This is useful for multi-label classification, where input samples can be
    classified as sets of labels. By only using accuracy (precision) a model
    would achieve a perfect score by simply assigning every class to every
    input. In order to avoid this, a metric should penalize incorrect class
    assignments as well (recall). The F-beta score (ranged from 0.0 to 1.0)
    computes this, as a weighted mean of the proportion of correct class
      assignments vs. the proportion of incorrect class assignments.

    With beta = 1, this is equivalent to a F-measure. With beta < 1, assigning
    correct classes becomes more important, and with beta > 1 the metric is
    instead weighted towards penalizing incorrect class assignments.
    """
    if beta < 0:
        raise ValueError('The lowest choosable beta is zero (only precision).')

    # If there are no true positives, fix the F score at 0 like sklearn.
    if K.sum(K.round(K.clip(y_true, 0, 1))) == 0:
        return 0

    p = precision(y_true, y_pred)
    r = recall(y_true, y_pred)
    bb = beta ** 2
    fbeta_score = (1 + bb) * (p * r) / (bb * p + r + K.epsilon())
    return fbeta_score


def fmeasure(y_true, y_pred):
    """Computes the f-measure, the harmonic mean of precision and recall.

    Here it is only computed as a batch-wise average, not globally.
    """
    return fbeta_score(y_true, y_pred, beta=1)
%----------------------------------------------------------------

%----------------------------------------------------------------
```

Appendix C

Python Codes for Fake News Detection - Text with sentiments / emotions

This section contains the source Python codes and algorithms used in the implementation of the methods and techniques for the Fake News detection using text with sentiments. It also includes topic modelling methods adopted (LSA and LDA).

The python libraries used include: Keras (`https://keras.io/`) Scikit Learn (`www.scikit-learn.org`) Pandas (`https://pandas.pydata.org/`) Numpy (`https://numpy.org/`) Gensim (`https://radimrehurek.com/gensim/`) GloVe (Pennington (2014), `https://nlp.stanford.edu/projects/glove/`) Bing-Liu Opinion Lexicon (Liu 2012) Hierarchical Attention Networks(Yang et al 2016)

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Topic Modelling and Sentiment Classification using ML/DL methods\n",
    "---\n",
    "Author – Oluwaseun Ajao                              #"
   ]
  },
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "### Load the required libraries"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "ExecuteTime": {
   "end_time": "2018-10-05T10:08:49.807417Z",
   "start_time": "2018-10-05T10:06:46.143970Z"
  }
 },
 "outputs": [],
 "source": [
  "import os\n",
  "import csv\n",
  "import sys\n",
  "\n",
  "\n",
  "import numpy as np\n",
  "import pandas as pd\n",
  "from string import punctuation\n",
  "\n",
  "# XgBoost\n",
  "import xgboost\n",
  "from xgboost import XGBClassifier\n",
  "\n",
  "# Logistics Regression\n",
  "from sklearn.linear_model import LogisticRegression\n",
  "# SVM\n",
  "from sklearn.svm import SVC\n",
  "# Decision Tree and Random Forest\n",
  "from sklearn.tree import DecisionTreeClassifier\n",
  "from sklearn.ensemble import RandomForestClassifier,
      GradientBoostingClassifier, AdaBoostClassifier\n",
"from sklearn.metrics import confusion_matrix,accuracy_score, precision_score ,
    classification_report \n",
"\n",
"from keras.models import Sequential\n",
"from keras.layers import Dense\n",
"from keras.layers import LSTM\n",
"from keras.layers.embeddings import Embedding\n",
"from keras.preprocessing import sequence\n",
"from keras.preprocessing.text import Tokenizer\n",
"from keras.preprocessing.sequence import pad_sequences\n",
"from keras.utils import to_categorical\n",
"from keras.layers import (Input, Dense, Embedding, LSTM, Flatten, \n",
"                         SpatialDropout1D, MaxPooling1D, Concatenate, \n",
"                         Conv1D, Dropout, BatchNormalization, Activation)\n"
    ,
"from keras.callbacks import ModelCheckpoint\n",
"from keras.optimizers import Adam, SGD, Nadam\n",
"\n",
"\n",
"import tensorflow as tf\n",
"from sklearn.model_selection import StratifiedKFold\n",
"import keras.backend as k\n",
"# For sentiment analysis\n",
"from textblob import TextBlob\n",
"import itertools\n",
"from sklearn import metrics\n",
"from sklearn.cross_validation import train_test_split \n",
"\n",
"# fix random seed for reproducibility\n",
"seed = 0\n",
"np.random.seed(seed)\n",
"import matplotlib.pyplot as plt\n",
"plt.switch_backend('agg')\n",
"plt.style.use('bmh')\n",
"# For text processing\n",
"import gensim\n",
"from gensim.utils import simple_preprocess\n",
"from gensim.parsing.preprocessing import STOPWORDS\n",
"from nltk.stem import WordNetLemmatizer, SnowballStemmer\n",
   "from nltk.stem.porter import *\n",
   "\n",
   "import nltk\n",
   "#nltk.download('wordnet')\n",
   "# TF−IDF\n",
   "from gensim import corpora, models\n",
   "%matplotlib inline\n",
   "# Define Helper Functions\n",
   "import keras.backend as K"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "## Define Helper Functions"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "ExecuteTime": {
    "end_time": "2018−10−05T10:08:57.203349Z",
    "start_time": "2018−10−05T10:08:57.182286Z"
   }
  },
  "outputs": [],
  "source": [
   "print(\"Defining Helper Functions\")\n",
   "\n",
   "def precision(y_true, y_pred):\n",
   "    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))\n",
   "    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))\n",
   "    precision = true_positives / (predicted_positives + K.epsilon())\n",
   "    return precision\n",
   " \n",
   "def recall(y_true, y_pred):\n",
   "    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))\n",
"    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))\n",
"    recall = true_positives / (possible_positives + K.epsilon())\n",
"    return recall\n",
" \n",
"def fbeta_score(y_true, y_pred, beta=1):\n",
"    if beta < 0:\n",
"        raise ValueError('The lowest choosable beta is zero (only precision)
   .')\n",
"\n",
"    # If there are no true positives, fix the F score at 0 like sklearn.\n",
"    if K.sum(K.round(K.clip(y_true, 0, 1))) == 0:\n",
"        return 0\n",
" \n",
"    p = precision(y_true, y_pred)\n",
"    r = recall(y_true, y_pred)\n",
"    bb = beta ** 2\n",
"    fbeta_score = (1 + bb) * (p * r) / (bb * p + r + K.epsilon())\n",
"    return fbeta_score\n",
" \n",
"def fmeasure(y_true, y_pred):\n",
"    return fbeta_score(y_true, y_pred, beta=1)\n",
"\n",
"def lemmatize_stemming(text):\n",
"    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))\n",
"\n",
"def preprocess(text):\n",
"    result = []\n",
"    for token in gensim.utils.simple_preprocess(text):\n",
"        if token not in gensim.parsing.preprocessing.STOPWORDS and len(
   token) > 3:\n",
"            result.append(token)\n",
"    return result"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "ExecuteTime": {
    "end_time": "2018-10-05T10:13:28.494097Z",
    "start_time": "2018-10-05T10:13:28.414117Z"
   }
  },
  "outputs": [],
  "source": [
   "filename = 'dataset.txt'\n",
   "print(\" Initiating  Topic Modelling\")\n",
   "# Topic Modelling\n",
   "\n",
   "new_posts = []\n",
   "df = pd.read_csv(filename, sep='\\t', header=None)\n",
   "label = df.iloc [:,1]\ n",
   "# Import the dataset\n",
   "with open(filename, 'r', encoding='utf-8') as f:\n",
   "     posts = csv.reader(f)\n",
   "     #posts = [x.lower() for x in posts]\n",
   "     #posts.pop(0)\n",
   "     for row in posts:\n",
   "          new_posts.append(row[0])"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "ExecuteTime": {
    "end_time": "2018-10-05T10:13:45.233006Z",
    "start_time": "2018-10-05T10:13:44.561177Z"
   }
  },
  "outputs": [],
  "source": [
   "filename = 'dataset.txt'\n",
   "print(\" Initiating  Topic Modelling\")\n",
   "# Topic Modelling\n",
   "new_posts = []\n",
   "#Import the dataset\n",
   "with open(filename, 'r', encoding='utf-8') as f:\n",
"      posts = csv.reader(f)\n",
"      #posts = [x.lower() for x in posts]\n",
"      #posts.pop(0)\n",
"      for row in posts:\n",
"            new_posts.append(row[0])\n",
"\n",
"stemmer = SnowballStemmer('english')\n",
"doc_sample = new_posts[1000]\n",
"\n",
"# Bag of words on the dataset\n",
"processed_docs = map(preprocess, new_posts)\n",
"dictionary = gensim.corpora.Dictionary(processed_docs)\n",
"count = 0\n",
"for k, v in dictionary.iteritems():\n",
"      #print(k, v)\n",
"      count += 1\n",
"      if count > 5:\n",
"            break\n",
"\n",
"dictionary.filter_extremes(no_below=15, no_above=0.5, keep_n=100000)\n",
"bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]\n",
"\n",
"\n",
"tfidf = models.TfidfModel(bow_corpus)\n",
"corpus_tfidf = tfidf[bow_corpus]\n",
"lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=10, id2word
     =dictionary, passes=2, workers=2)\n",
"\n",
"with open('lda_output.txt', 'w') as f:\n",
"      for idx, topic in lda_model.print_topics(-1):\n",
"            f.write('Topic:' + str(idx) + '\\n')\n",
"            f.write('Words: ' + topic + '\\n')\n",
"            print('Topic: {} \\nWords: {}'.format(idx, topic))"
]
},
{
"cell_type": "code",
"execution_count": null,
"metadata": {
  "ExecuteTime": {
   "end_time": "2018−10−05T10:21:33.745828Z",
   "start_time": "2018−10−05T10:21:17.388142Z"
  }
 },
 "outputs": [],
 "source": [
  "print(\"Visualizaing Topic using PyLDAvis\")\n",
  "import pyLDAvis.gensim\n",
  "from gensim.models import LdaModel\n",
  "pyLDAvis.enable_notebook()\n",
  "\n",
  "import pyLDAvis\n",
  "import pyLDAvis.sklearn\n",
  "from sklearn.feature_extraction.text import CountVectorizer\n",
  "from sklearn.decomposition import LatentDirichletAllocation\n",
  "tf_vectorizer = CountVectorizer(strip_accents = 'unicode',stop_words = '
     english', lowercase = True,token_pattern = r'\\b[a−zA−Z]{3,}\\b',max_df =
      0.5,min_df = 10)\n",
  "dtm_tf = tf_vectorizer.fit_transform(new_posts)\n",
  "lda_tf = LatentDirichletAllocation(n_components=20, learning_method='online
     ')\n",
  "lda_tf.fit(dtm_tf)\n",
  "vis_data = pyLDAvis.sklearn.prepare(lda_tf, dtm_tf, tf_vectorizer)\n",
  "pyLDAvis.display(vis_data)\n",
  "\n",
  "print(\"Topic Modelling Completed\")\n",
  "print()\n"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "### Feature Engineering and Classifiation"
 ]
},
{
 "cell_type": "code",
"execution_count": null,
"metadata": {
 "ExecuteTime": {
  "end_time": "2018−10−05T10:24:04.193496Z",
  "start_time": "2018−10−05T10:23:49.223395Z"
 }
},
"outputs": [],
"source": [
 "print(\"Creating Features and Labels for Fake News Classfication\")\n",
 "print()\n",
 "path = os.path.join(filename)\n",
 "data_set = pd.read_csv(path, delimiter = '\\t', quoting = 3, header = None)\
    n",
 "\n",
 "print(\"Extracting Emoratio using Bing Lui Lexicon\")\n",
 "try:\n",
 "     new_post = open(filename, encoding='ISO−8859−1').read()\n",
 "     new_posts=new_post.split('\\n')\n",
 "     \n",
 "     pos_sent = open(\"positive−words.txt\", encoding='ISO−8859−1').read()\
    n",
 "     positive_words=pos_sent.split('\\n')\n",
 "     positive_counts=[]\n",
 "     \n",
 "     neg_sent = open('negative−words.txt', encoding='ISO−8859−1').read()\n",
 "     negative_words=neg_sent.split('\\n')\n",
 "     negative_counts=[]\n",
 "\n",
 "except IOException as e:\n",
 "     print('File(s) not found')\n",
 "     sys.exit(1)\n",
 "\n",
 "     \n",
 "for tweet in new_posts:\n",
 "     positive_counter=0\n",
 "     negative_counter=0\n",
 "\n",
 "     tweet_processed=tweet.lower()\n",
    "      \n",
    "      for  p in  list (punctuation):\n",
    "          tweet_processed=tweet_processed.replace(p,'') \n",
    "\n",
    "      words=tweet_processed.split('  ') \n",
    "      word_count=len(words)\n",
    "      for  word in words:\n",
    "          if  word in positive_words:\n",
    "              positive_counter=positive_counter+1\n",
    "          elif  word in negative_words:\n",
    "              negative_counter=negative_counter+1\n",
    "          \n",
    "      positive_counts .append(positive_counter/word_count)\n",
    "      negative_counts.append(negative_counter/word_count)\n",
    "      \n",
    "negative_count_ = negative_counts\n",
    "positive_count_ = []\n",
    "for  item in  positive_counts :\n",
    "      if  item == 0:\n",
    "          item = 1\n",
    "      positive_count_.append(item)\n",
    "emoratio = np.divide(negative_count_, positive_count_)\n",
    "print(emoratio)\n",
    "\n",
    "labels_ = {'emoratio': emoratio}\n",
    "labels_df  = pd.DataFrame(labels_, columns = ['emoratio'])\n",
    "labels_df"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "ExecuteTime": {
     "end_time": "2018-10-05T10:25:02.087287Z",
     "start_time": "2018-10-05T10:25:02.071291Z"
    }
   },
   "outputs": [],
 "source": [
  "#labels_df['fake']  = np.where(labels_df['emoratio']<=1, 0, 1)\n",
  "\n",
  "X = data_set.iloc [:,  0]\n",
  "X_ = labels_df.emoratio\n",
  "\n",
  "print(X.head(10))\n",
  "print(X_.head(10))\n",
  "\n",
  "print(X.shape, X_.shape)"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "ExecuteTime": {
   "end_time": "2018−10−05T10:25:41.521965Z",
   "start_time": "2018−10−05T10:25:40.267524Z"
  }
 },
 "outputs": [],
 "source": [
  "# tokenize the training  texts  and make it sequential\n",
  "top_words = 500\n",
  "tokenizer  = Tokenizer(num_words=top_words)\n",
  "tokenizer. fit_on_texts (X)\n",
  "sequences_train  = tokenizer. texts_to_sequences (X)\n",
  "# tokenize the testing  texts  and make it sequential\n",
  "tokenizer. fit_on_texts (X)\n",
  "sequences_test  = tokenizer. texts_to_sequences (X)\n",
  "#word_index_train = tokenizer.word_index_train\n",
  "Y = label.values.reshape(X.shape[0],)\n",
  "\n",
  "print(\"Shape of Features :  \"  ,  X.shape)\n",
  "print(\"Shape of Labels :  \",  Y.shape)"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "ExecuteTime": {
   "end_time": "2018−10−05T10:26:03.701397Z",
   "start_time": "2018−10−05T10:26:02.748489Z"
  }
 },
 "outputs": [],
 "source": [
  "from sklearn.feature_extraction.text import TfidfVectorizer as tfidf\n",
  "from nltk.corpus import stopwords\n",
  "stopword= set(stopwords.words('english'))\n",
  "# using TfIdf to make words as features by making word vectors\n",
  "#vectorizer= tfidf(stop_words= stopword)        \n",
  "#X = vectorizer.fit_transform(X)\n",
  "vectorizer = tfidf(max_features = 2000) \n",
  "vectorizer.fit (X) \n",
  "X = vectorizer.transform(X)\n",
  "\n",
  "print(\"Features Shape : \", X.shape)\n",
  "print(\"Label Shape : \", Y.shape)"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "ExecuteTime": {
   "end_time": "2018−10−05T10:30:15.618293Z",
   "start_time": "2018−10−05T10:30:15.606298Z"
  }
 },
 "outputs": [],
 "source": [
  "print(\"Shape of Feature Matrix before adding Emoratio\", X.shape)\n",
  "from scipy.sparse import hstack\n",
  "X = hstack((X ,np.array(X_)[:,None]))\n",
  "print(\"Shape of Feature Matrix after adding Emoratio\", X.shape)"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "### Define Helper Function for Machine learing Modelling"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "ExecuteTime": {
   "end_time": "2018-10-05T10:32:21.105494Z",
   "start_time": "2018-10-05T10:32:21.061506Z"
  },
  "collapsed": true
 },
 "outputs": [],
 "source": [
  "def plot_confusion_matrix(cm, classes = ['Negative', 'Positive '],\ n",
  "                          normalize=False,\n",
  "                           title ='Confusion matrix',\n",
  "                          cmap=plt.cm.Blues):\n",
  "    if normalize:\n",
  "        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]\n",
  "        print(\"Normalized confusion matrix\")\n",
  "    else:\n",
  "        print ('Confusion matrix, without normalization')\n",
  "    plt.imshow(cm, interpolation='nearest', cmap=cmap)\n",
  "    plt. title ( title )\n",
  "    plt.colorbar()\n",
  "    tick_marks = np.arange(len(classes))\n",
  "    plt.xticks(tick_marks, classes , rotation=45)\n",
  "    plt.yticks(tick_marks, classes )\n",
  "\n",
  "    fmt = '.2f' if normalize else 'd'\n",
  "    thresh = cm.max() / 2.\n",
"    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):\n",
"        plt.text(j, i, format(cm[i, j], fmt),\n",
"                 horizontalalignment=\"center\",\n",
"                 color=\"white\" if cm[i, j] > thresh else \"black\")\n",
"\n",
"    plt.tight_layout()\n",
"    plt.ylabel('True label')\n",
"    plt.xlabel('Predicted label')     \n",
"    \n",
"def plot_roc_curve(y_test, y_pred_proba):\n",
"    fpr, tpr, thresholds =metrics.roc_curve(y_test, y_pred_proba[:, 1])\n",
"    roc_auc = metrics.auc(fpr, tpr)\n",
"    plt.figure(figsize=(15,7))\n",
"    plt.plot(fpr, tpr, label='ROC curve (area = %0.3f)' % roc_auc)\n",
"    plt.plot([0, 1], [0, 1], 'k--')\n",
"    plt.xlim([0.0, 1.0])\n",
"    plt.ylim([0.0, 1.0])\n",
"    plt.xlabel('False Positive Rate or (1 - Specificity)')\n",
"    plt.ylabel('True Positive Rate or (Sensitivity)')\n",
"    plt.title('Receiver Operating Characteristic')\n",
"    plt.legend(loc=\"lower right\")\n",
"    plt.show()\n",
"\t\n",
"def plot_roc_curve_nn(y_test, y_pred_proba):\n",
"    fpr, tpr, thresholds =metrics.roc_curve(y_test, y_pred_proba)\n",
"    roc_auc = metrics.auc(fpr, tpr)\n",
"    plt.figure(figsize=(15,7))\n",
"    plt.plot(fpr, tpr, label='ROC curve (area = %0.3f)' % roc_auc)\n",
"    plt.plot([0, 1], [0, 1], 'k--')\n",
"    plt.xlim([0.0, 1.0])\n",
"    plt.ylim([0.0, 1.0])\n",
"    plt.xlabel('False Positive Rate or (1 - Specificity)')\n",
"    plt.ylabel('True Positive Rate or (Sensitivity)')\n",
"    plt.title('Receiver Operating Characteristic')\n",
"    plt.legend(loc=\"lower right\")\n",
"    plt.show()\n",
"\t\n",
"\t\n",
"    \n",
"def fit_model(model, model_name):\n",
"    print(\" Initiating  Model...\")\n",
"    print(model_name)\n",
"    print(\"\")\n",
"    print('Fitting on Training Data ...')\n",
"    model.fit(X_train, y_train)\n",
"    print('Model Trained... Predicting on Test Data ...')\n",
"    train_preds = model.predict(X_train)\n",
"    test_preds = model.predict(X_test)\n",
"    train_accuracy = accuracy_score(y_train,train_preds)\n",
"    test_accuracy = accuracy_score(test_preds, y_test)\n",
"    \n",
"    print('Train accuracy :', train_accuracy)\n",
"    print('Test accuracy :', test_accuracy)\n",
"\n",
"    # Confusion Matrix\n",
"    cm = confusion_matrix(y_test, test_preds)\n",
"    print(\"Confusion Matrix\")\n",
"    print(\"\\n\")\n",
"    print(cm)\n",
"    \n",
"    # Precision/Recall/F1-Score\n",
"    print(\" Classification  Report\")\n",
"    print(\"\\n\")\n",
"    print( classification_report (y_test, test_preds, target_names=['Non-
    Rumour', 'Rumour']))\n",
"\n",
"    plot_confusion_matrix(cm, classes = ['Non-Rumour', 'Rumour'],\n",
"                          normalize=False,\n",
"                           title ='Confusion matrix',\n",
"                          cmap=plt.cm.Blues)\n",
"    \n",
"\n",
"    # ROC Curve\n",
"    print(\"ROC(AUC) Curve\")\n",
"    print(\"\\n\")\n",
"    test_pred_proba = model.predict_proba(X_test)\n",
"    plot_roc_curve(y_test, test_pred_proba)\n",
"    print
```

```
    (\"−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
    \n",
"    \n",
"def fit_model_nn(model, model_name):\n",
"    print(\" Initiating  Model...\")\n",
"    print(model_name)\n",
"    print(\"\")\n",
"    print ('Fitting  on Training Data ...') \n",
"    history  = model.fit(X_train,  y_train ,  epochs=1, batch_size=64,
    validation_data=(X_test, y_test),  verbose = 2)\n",
"    print ('Model Trained... Predicting on Test Data ...') \n",
"    train_preds  = model.predict_classes(X_train)\n",
"    test_preds  = model.predict_classes(X_test)\n",
"    train_accuracy  = accuracy_score(y_train,train_preds)\n",
"    test_accuracy  = accuracy_score(test_preds,  y_test )\n",
"    \n",
"    print ('Train accuracy  :',  train_accuracy)\n",
"    print ('Test accuracy  :',  test_accuracy)\n",
"\n",
"    # Confusion Matrix\n",
"    cm = confusion_matrix(y_test, test_preds)\n",
"    print(\"Confusion Matrix\")\n",
"    print(\"\\n\")\n",
"    print(cm)\n",
"    \n",
"    # Precision/Recall/F1−Score\n",
"    print(\" Classification  Report\")\n",
"    print(\"\\n\")\n",
"    print( classification_report  (y_test ,  test_preds ,  target_names=['Non−
    Rumour', 'Rumour']))\n",
"\n",
"    plot_confusion_matrix(cm, classes  = ['Non−Rumour', 'Rumour'],\n",
"                                    normalize=False,\n",
"                                     title ='Confusion matrix',\n",
"                                    cmap=plt.cm.Blues)\n",
"    \n",
"\n",
"    # ROC Curve\n",
"    print(\"ROC(AUC) Curve\")\n",
"      print(\"\\n\")\n",
"      test_pred_proba = model.predict(X_test)\n",
"      plot_roc_curve_nn(y_test, test_pred_proba)\n",
"      # Model History\n",
"      print(\"Model History\")\n",
"      plot_history(history)\n",
"      print
     (\"--------------------------------------------------------------
     \n",
"\n",
"     \n",
"def plot_history(history):\n",
"       loss_list  = [s for s in history.history.keys() if 'loss' in s and 'val'
     not in s]\n",
"       val_loss_list  = [s for s in history.history.keys() if 'loss' in s and 'val
     ' in s]\n",
"       acc_list  = [s for s in history.history.keys() if 'acc' in s and 'val' not
     in s]\n",
"       val_acc_list  = [s for s in history.history.keys() if 'acc' in s and 'val'
     in s]\n",
"      \n",
"      if len( loss_list ) == 0:\n",
"          print('Loss is missing in history')\n",
"          return \n",
"      \n",
"      ## As loss always exists\n",
"      epochs = range(1,len(history.history[ loss_list [0]]) + 1)\n",
"      \n",
"      ## Loss\n",
"      plt.figure(1)\n",
"      for l in  loss_list :\n",
"          plt.plot(epochs, history.history[l], 'b', label='Training loss (' +
     str(str(format(history.history[l][-1],'.5 f'))+')'))\n",
"      for l in  val_loss_list :\n",
"          plt.plot(epochs, history.history[l], 'g', label='Validation loss (' +
     str(str(format(history.history[l][-1],'.5 f'))+')'))\n",
"      \n",
"      plt.title('Loss')\n",
"      plt.xlabel('Epochs')\n",
 "      plt.ylabel('Loss')\n",
 "      plt.legend()\n",
 "      \n",
 "      ## Accuracy\n",
 "      plt.figure(2)\n",
 "      for l in  acc_list :\n",
 "            plt.plot(epochs, history.history[l],  'b',  label='Training accuracy ('
      + str(format(history.history[l][-1],'.5 f'))+')')\n",
 "      for l in  val_acc_list :      \n",
 "            plt.plot(epochs, history.history[l],  'g',  label='Validation accuracy
      (' + str(format(history.history[l][-1],'.5 f'))+')')\n",
 "\n",
 "      plt.title('Accuracy')\n",
 "      plt.xlabel('Epochs')\n",
 "      plt.ylabel('Accuracy')\n",
 "      plt.legend()\n",
 "      plt.show()"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "collapsed": true
 },
 "outputs": [],
 "source": [
  "# Train Test Split\n",
  "X_train, X_test, y_train, y_test = train_test_split (X,Y, test_size = 0.3)"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "ExecuteTime": {
   "end_time": "2018-10-05T10:33:36.390199Z",
   "start_time": "2018-10-05T10:32:25.903025Z"
  }
  },
  "outputs": [],
  "source": [
   "model_logreg = LogisticRegression()\n",
   "model_svm_linear = SVC(kernel='linear', probability=True)\n",
   "model_svm_radial = SVC(kernel='rbf', probability=True, gamma=0.1, C = 10)
       \n",
   "model_svm_poly = SVC(kernel='poly', probability=True, gamma=0.1, C = 10,
       degree=3)\n",
   "model_dt = DecisionTreeClassifier()\n",
   "model_rf = RandomForestClassifier(n_estimators=500)\n",
   "model_xgb = XGBClassifier(colsample_bytree = .6, max_depth = 10, subsample
       =.7,n_estimators = 500,n_jobs=4)\n",
   "\n",
   "fit_model(model= model_logreg, model_name='Logistic Regression')\n",
   "fit_model(model= model_svm_linear, model_name='Support Vector Machine
       with linear kernel')\n",
   "fit_model(model= model_svm_radial, model_name='Support Vector Machine
       with Radial kernel')\n",
   "fit_model(model= model_svm_poly, model_name='Support Vector Machine with
        Polynomial kernel')\n",
   "fit_model(model= model_dt, model_name='Decision Tree Classifier')\n",
   "fit_model(model= model_rf, model_name='Random Forest Classifier')\n",
   "fit_model(model= model_xgb, model_name='Xtreme Boosting Classifier')"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "## Neural Networks"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  ”outputs”: [],
  ”source”: [
   ”# Define Embedding\n”,
   ”embedding_vector_length = 32”
  ]
 },
 {
  ”cell_type”: ”code”,
  ”execution_count”: null,
  ”metadata”: {
   ”collapsed”: true
  },
  ”outputs”: [],
  ”source”: [
   ”import os\n”,
   ”os.chdir(’/home/paperspace/ImageSentimentAnalysis/’)\n”,
   ”embeddings_index = dict()\n”,
   ”f = open(’glove.6B.100d.txt’)\n”,
   ”for line in f:\n”,
   ”    values = line.split()\n”,
   ”    word = values[0]\n”,
   ”    coefs = np.asarray(values[1:], dtype=’float32’)\n”,
   ”    embeddings_index[word] = coefs\n”,
   ”f.close()”
  ]
 },
 {
  ”cell_type”: ”code”,
  ”execution_count”: null,
  ”metadata”: {
   ”collapsed”: true
  },
  ”outputs”: [],
  ”source”: [
   ”vocabulary_size = X.shape[1]\n”,
   ”embedding_matrix = np.zeros((vocabulary_size, 100))\n”,
   ”for word, index in tokenizer.word_index.items():\n”,
   ”    if index > vocabulary_size − 1:\n”,
   ”        break\n”,
 "       else:\n",
 "           embedding_vector = embeddings_index.get(word)\n",
 "            if  embedding_vector is not None:\n",
 "                 embedding_matrix[index] = embedding_vector"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "collapsed": true
 },
 "outputs": [],
 "source": [
  "## create model\n",
  "model_glove = Sequential()\n",
  "model_glove.add(Embedding(vocabulary_size, 100, input_length=X.shape[1],
      weights=[embedding_matrix], trainable=False))\n",
  "model_glove.add(Dropout(0.2))\n",
  "model_glove.add(Conv1D(64, 5, activation='relu'))\n",
  "model_glove.add(MaxPooling1D(pool_size=4))\n",
  "model_glove.add(LSTM(100))\n",
  "model_glove.add(Dense(1, activation='sigmoid'))\n",
  "model_glove.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
      accuracy',precision, recall , fmeasure])"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "print(model_glove.summary())"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "fit_model_nn(model = model_glove, model_name = \"LSTM−CNN Model with
      Dropout and Pre−trained Features\")"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "## Model 2"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "#################################\n",
  "# LSTM−CNN Model with Dropout #\n",
  "#################################\n",
  "\n",
  "model=Sequential()\n",
  "model.add(Embedding(X.shape[1], 100, input_length=X.shape[1], weights=[
      embedding_matrix], trainable=False))\n",
  "model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3, return_sequences=
      True))\n",
  "model.add(LSTM(64, dropout=0.3, recurrent_dropout=0.3, return_sequences=
      True))\n",
  "model.add(SpatialDropout1D(0.2))\n",
  "model.add(Conv1D(64, kernel_size=3, padding='same', activation='relu'))\n",
  "model.add(MaxPooling1D(pool_size=2))\n",
  "model.add(Conv1D(32, kernel_size=3, padding='same', activation='relu'))\n",
  "model.add(MaxPooling1D(pool_size=2))\n",
  "model.add(Flatten())\n",
  "#Dense/Output\n",
  "model.add(Dense(16))\n",
  "model.add(Dense(8))\n",
  "model.add(Dense(1,activation='sigmoid'))\n",
  "model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy
     ', precision , recall , fmeasure])\n",
  "print(model.summary())\n",
  "\n",
  "fit_model_nn(model = model, model_name = \"LSTM−CNN Model with
     Dropout\")"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "#####################\n",
  "# LSTM − CNN Model #\n",
  "#####################\n",
  "\n",
  "model = Sequential()\n",
  "model.add(Embedding(X.shape[1], embedding_vector_length, input_length=X.
     shape[1]))\n",
  "model.add(SpatialDropout1D(0.2))\n",
  "model.add(Conv1D(64, kernel_size=3, padding='same', activation='relu'))\n",
  "model.add(MaxPooling1D(pool_size=2))\n",
  "model.add(Conv1D(128, kernel_size=3, padding='same', activation='relu'))\n",
  "model.add(MaxPooling1D(pool_size=2))\n",
  "model.add(Flatten())\n",
  "model.add(Dense(1, activation='sigmoid'))\n",
  "model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
     accuracy',precision, recall , fmeasure])\n",
  "print(model.summary())\n",
  "\n",
  "fit_model_nn(model = model, model_name = \"LSTM − CNN Model without
     Dropout\")"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "##################### \n",
  "# Simple LSTM Model #\n",
  "#####################\n",
  "\n",
  "model = Sequential()\n",
  "model.add(Embedding(X.shape[1], embedding_vector_length, input_length=X.
      shape[1])) \n",
  "model.add(LSTM(100))\n",
  "model.add(Dense(1, activation='sigmoid'))\n",
  "model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
      accuracy',precision, recall , fmeasure])\n",
  "print(model.summary())\n",
  "\n",
  "fit_model_nn(model = model, model_name = \"Simple LSTM Model\") \n",
  "#k.clear_session()"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "---"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "Trying the next step - HAN"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
  "source": [
   "# HAN"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "MAX_SENT_LENGTH = 200\n",
   "MAX_SENTS = 15\n",
   "MAX_NB_WORDS = 20000\n",
   "EMBEDDING_DIM = 100\n",
   "VALIDATION_SPLIT = 0.3"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "collapsed": true
  },
  "outputs": [],
  "source": [
   "from bs4 import BeautifulSoup\n",
   "import nltk\n",
   "from nltk import tokenize\n",
   "def clean_str(string):\n",
   "    string = re.sub(r\"\\\\\", \"\", string)\n",
   "    string = re.sub(r\"\\'\", \"\", string)\n",
   "    string = re.sub(r\"\\\"\", \"\", string)\n",
   "    return string.strip().lower()\n",
   "\n",
   "reviews = []\n",
   "labels = []\n",
   "texts = []\n",
 "df.columns = ['message', 'class ']\ n",
 "macronum=sorted(set(df['class']))\n",
 "macro_to_id = dict((note, number) for number, note in enumerate(macronum))\
     n",
 "\n",
 "def fun(i):\n",
 "    return macro_to_id[i]\n",
 "\n",
 "df[' class']=df[' class ']. apply(fun)\n",
 "\n",
 "for i in range(df.message.shape[0]):\n",
 "    text = BeautifulSoup(df.message[i])\n",
 "    text=clean_str(str(text.get_text().encode()).lower())\n",
 "    texts.append(text)\n",
 "    sentences = tokenize.sent_tokenize(text)\n",
 "    reviews.append(sentences)\n",
 "\n",
 "for i in df[' class ']:\ n",
 "    labels.append(i)"
]
},
{
"cell_type": "code",
"execution_count": null,
"metadata": {
 "collapsed": true
},
"outputs": [],
"source": [
 "from keras.preprocessing.text import Tokenizer,text_to_word_sequence\n",
 "tokenizer = Tokenizer(num_words=MAX_NB_WORDS)"
]
},
{
"cell_type": "code",
"execution_count": null,
"metadata": {
 "collapsed": true
},
 "outputs": [],
 "source": [
  "tokenizer. fit_on_texts (texts)"
 ]
},
{
 " cell_type": "code",
 "execution_count": null,
 "metadata": {
  "collapsed": true
 },
 "outputs": [],
 "source": [
  "data = np.zeros((len(texts), MAX_SENTS, MAX_SENT_LENGTH), dtype='
      int32')\n",
  "\n",
  "for i, sentences in enumerate(reviews):\n",
  "     for j, sent in enumerate(sentences):\n",
  "          if j< MAX_SENTS:\n",
  "               wordTokens = text_to_word_sequence(sent)\n",
  "               k=0\n",
  "               for _, word in enumerate(wordTokens):\n",
  "                    if k<MAX_SENT_LENGTH and tokenizer.word_index[word]<
      MAX_NB_WORDS:\n",
  "                         data[i,j,k] = tokenizer.word_index[word]\n",
  "                         k=k+1"
 ]
},
{
 " cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "word_index = tokenizer.word_index\n",
  "print('No. of %s unique tokens.' % len(word_index))"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "scrolled": true
 },
 "outputs": [],
 "source": [
  "labels = to_categorical(np.asarray(labels))\n",
  "print('Shape of data tensor:', data.shape)\n",
  "print('Shape of label tensor:', labels.shape)\n",
  "\n",
  "\n",
  "indices = np.arange(data.shape[0])\n",
  "np.random.shuffle(indices)\n",
  "data = data[indices]\n",
  "labels = labels[indices]\n",
  "nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "print(\"Shape of Feature Matrix before adding Emoratio\", data.shape)\n",
  "\n",
  "data_ = np.concatenate((data, \n",
  "                        np.broadcast_to(np.array(X_)[:, None, None], data.shape
     [:-1] + (1,))), \n",
  "                       axis = -1)\n",
  "\n",
  "print(\"Shape of Feature Matrix after adding Emoratio\", data_.shape)"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "collapsed": true
 },
 "outputs": [],
 "source": [
  "x_train = data[:-nb_validation_samples]\n",
  "y_train = labels[:-nb_validation_samples]\n",
  "x_val = data[-nb_validation_samples:]\n",
  "y_val = labels[-nb_validation_samples:]"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "embeddings_index = {}\n",
  "f = open('/home/paperspace/ImageSentimentAnalysis/glove.6B.100d.txt',
      encoding='utf8')\n",
  "for line in f:\n",
  "    values = line.split()\n",
  "    word = values[0]\n",
  "    coefs = np.asarray(values[1:], dtype='float32')\n",
  "    embeddings_index[word] = coefs\n",
  "f.close()\n",
  "\n",
  "print('Total %s word vectors.' % len(embeddings_index))"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "collapsed": true
 },
 "outputs": [],
 "source": [
  "embedding_matrix = np.random.random((len(word_index) + 1,
      EMBEDDING_DIM))\n",
   "for word, i in word_index.items():\n",
   "     embedding_vector = embeddings_index.get(word)\n",
   "     if embedding_vector is not None:\n",
   "          # words not found in embedding index will be all-zeros.\n",
   "          embedding_matrix[i] = embedding_vector\n",
   "\n",
   "embedding_layer = Embedding(len(word_index) + 1,\n",
   "                            EMBEDDING_DIM,\n",
   "                            weights=[embedding_matrix],\n",
   "                            input_length=MAX_SENT_LENGTH,\n",
   "                            trainable=True)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "scrolled": true
  },
  "outputs": [],
  "source": [
   "from keras.preprocessing.text import Tokenizer,text_to_word_sequence\n",
   "from keras.preprocessing.sequence import pad_sequences\n",
   "from keras.utils.np_utils import to_categorical\n",
   "from keras.layers import Embedding\n",
   "from keras.layers import Dense, Input, Flatten\n",
   "from keras.layers import Conv1D, MaxPooling1D, Embedding, Dropout, LSTM
       , GRU, Bidirectional, TimeDistributed\n",
   "from keras.models import Model\n",
   "from keras.callbacks import ModelCheckpoint\n",
   "import matplotlib.pyplot as plt\n",
   "plt.switch_backend('agg')\n",
   "from keras import backend as K\n",
   "from keras.engine.topology import Layer, InputSpec\n",
   "from keras import initializers \n",
   "%matplotlib inline\n",
   "\n",
   "sentence_input = Input(shape=(MAX_SENT_LENGTH+1,), dtype='int32')\n",
   "embedded_sequences = embedding_layer(sentence_input)\n",
  "l_lstm  = Bidirectional(LSTM(100))(embedded_sequences)\n",
  "sentEncoder = Model(sentence_input, l_lstm)\n",
  "\n",
  "review_input = Input(shape=(MAX_SENTS,MAX_SENT_LENGTH), dtype='
      int32')\n",
  "review_encoder = TimeDistributed(sentEncoder)(review_input)\n",
  " l_lstm_sent  = Bidirectional(LSTM(100))(review_encoder)\n",
  "preds = Dense(len(macronum), activation='softmax')(l_lstm_sent)\n",
  "model = Model(review_input, preds)\n",
  "\n",
  "model.compile(loss='categorical_crossentropy ',\ n",
  "                  optimizer='rmsprop',\n",
  "                  metrics=['acc ']) \n",
  "\n",
  "print(\"Hierachical  LSTM\")\n",
  "model.summary()"
 ]
},
{
 " cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "cp=ModelCheckpoint('model_han_.hdf5',monitor='val_acc',verbose=1,
      save_best_only=True)\n",
  "history=model.fit(x_train,  y_train,  validation_data=(x_val,  y_val),\n",
  "              epochs=100, batch_size=64,callbacks=[cp])"
 ]
},
{
 " cell_type": "code",
 "execution_count": null,
 "metadata": {
  "collapsed": true
 },
 "outputs": [],
 "source": [
  " fig1  = plt. figure ()\n",
  "plt.plot(history.history['loss'],'r',linewidth=3.0)\n",
  "plt.plot(history.history['val_loss'],'b',linewidth=3.0)\n",
  "plt.legend(['Training loss', 'Validation Loss'], fontsize=18)\n",
  "plt.xlabel('Epochs',fontsize=16)\n",
  "plt.ylabel('Loss', fontsize=16)\n",
  "plt.title('Loss Curves :HAN',fontsize=16)\n",
  "fig1.savefig('loss_han.png')\n",
  "plt.show()"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "collapsed": true
 },
 "outputs": [],
 "source": [
  "fig2=plt.figure()\n",
  "plt.plot(history.history['acc'],'r',linewidth=3.0)\n",
  "plt.plot(history.history['val_acc'],'b',linewidth=3.0)\n",
  "plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)\n",
  "plt.xlabel('Epochs',fontsize=16)\n",
  "plt.ylabel('Accuracy',fontsize=16)\n",
  "plt.title('Accuracy Curves : HAN',fontsize=16)\n",
  "fig2.savefig('accuracy_han.png')\n",
  "plt.show()"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "collapsed": true
 },
 "outputs": [],
 "source": [
  "import pydot\n",
  "from keras.utils.vis_utils import plot_model\n",
   "plot_model(model,  to_file ='han_model.png', show_shapes=True,
        show_layer_names=True)"
  ]
 },
 {
  " cell_type ":  "code",
  "execution_count":  null ,
  "metadata": {
   "collapsed":  true
  },
  "outputs":  [],
  "source":  [
   "from PIL import Image\n",
   "display(Image.open('han_model.png'))"
  ]
 },
 {
  " cell_type ":  "markdown",
  "metadata": {},
  "source":  [
   "−−−"
  ]
 }
],
"metadata": {
 "kernelspec":  {
  "display_name": "Python 3",
  "language": "python",
  "name": "python3"
 },
 "language_info":  {
  "codemirror_mode": {
   "name": "ipython",
   "version":  3
  },
  " file_extension ":  ".py",
  "mimetype": "text/x−python",
  "name": "python",
  "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.6.3"
  },
  "notify_time": "5"
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```

Appendix D

PHEME Rumor Non-Rumor Dataset

The initial PHEME dataset given in JSON format by Zubiaga et al. (2016) and is publicly available in .

```
https:// figshare .com/articles/PHEME_dataset_of_rumours_and_non−rumours
    /4010619
```

The URL link is an online repository to download the PHEME dataset used in this work after the dataset was converted to a tab-delimited format

```
https://drive.google.com/open?id=1E5rKCvOCMDsH432BtOEOPjAro7HSR5KO
```

Appendix E

UTGEO (Small) and GEOTEXT Datasets

```
This is the download link to access the UTGEO−Small dataset
https://drive.google.com/file/d/14qLdOdBnsQbL_HD5M7ydhCX0cSrYoFpY/view?
    usp=sharing

This is the download link to access the GEOTEXT dataset
https://drive.google.com/file/d/16−l−BPZalynK4W2ps−N6ahQLzp4Tyt9w/view?
    usp=sharing
```

Appendix F

A Tweet in JSON Format

```
{
   "contributors":null,
   "truncated":false,
   "text":"Now 10 dead in a shooting there today RT \"@BBCDanielS: Charlie
       Hebdo became well known for publishing the Muhammed cartoons two years
       ago\u201d",
   "in_reply_to_status_id":552784600502915072,
   "id":552785249420447745,
   "favorite_count":0,
   "source":"<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">
       Twitter for iPhone</a>",
   "retweeted":false,
   "coordinates":null,
   "entities":{
      "symbols":[

      ],
      "user_mentions":[
         {
            "id":331658004,
            "indices":[
               42,
               53
            ],
            "id_str":"331658004",
            "screen_name":"BBCDanielS",
            "name":"Daniel Sandford"
        }
    ],
    "hashtags":[

    ],
    "urls" :[

    ]
},
"in_reply_to_screen_name":"BBCDanielS",
" id_str ":"552785249420447745",
"retweet_count":0,
" in_reply_to_user_id ":331658004,
"favorited": false ,
"user":{
    " follow_request_sent ": false ,
    "profile_use_background_image":true,
    "  profile_text_color  ":"333333",
    " default_profile_image ": false ,
    "id":18370911,
    "profile_background_image_url_https":"https://pbs.twimg.com/
        profile_background_images/578554964/clrvcuc60cp6ce3hqosb.jpeg",
    " verified ": false ,
    "  profile_location  ": null ,
    " profile_image_url_https ":"https://pbs.twimg.com/profile_images
        /378800000320937958/abf98da1430f224cbea0c75c027a178c_normal.jpeg",
    "   profile_sidebar_fill_color   ":"DDEEF6",
    " entities ":{
        "description":{
            "urls" :[

            ]
        }
    },
    "followers_count":4671,
    "  profile_sidebar_border_color ":"C0DEED",
    " id_str ":"18370911",
    "profile_background_color":"C0DEED",
    "listed_count":118,
    "is_translation_enabled":false,
    "utc_offset":−21600,
    "statuses_count":5064,
    "description":"agricultural  commodity options/futures trader in CBOT corn
        options pit, student of markets, former meat marketer, renewable energy
        supporter, duke blue devil",
    "friends_count":4954,
    "location":"Chicago",
    "profile_link_color":"0084B4",
    "profile_image_url":"http://pbs.twimg.com/profile_images
        /378800000320937958/abf98da1430f224cbea0c75c027a178c_normal.jpeg",
    "following":false,
    "geo_enabled":false,
    "profile_banner_url":"https://pbs.twimg.com/profile_banners
        /18370911/1398141023",
    "profile_background_image_url":"http://pbs.twimg.com/
        profile_background_images/578554964/clrvcuc60cp6ce3hqosb.jpeg",
    "name":"Rob Levy",
    "lang":"en",
    "profile_background_tile":false,
    "favourites_count":300,
    "screen_name":"robbylevy",
    "notifications":false,
    "url":null,
    "created_at":"Thu Dec 25 05:12:43 +0000 2008",
    "contributors_enabled":false,
    "time_zone":"Central Time (US & Canada)",
    "protected":false,
    "default_profile":false,
    "is_translator":false
},
"geo":null,
"in_reply_to_user_id_str":"331658004",
"lang":"en",
"created_at":"Wed Jan 07 11:14:08 +0000 2015",
"in_reply_to_status_id_str":"552784600502915072",
"place":null
```

```
}
```

Appendix G

Typical Tweet Variables

Table G.1: Typical Tweet Variables and Components

1	_id	21	entitiesuser_mentions1id_str	41	entitieshashtags3text	61	entitiesmedia0sizesmediumh	81	userdefault_profile_image
2	contributors	22	entitiesuser_mentions1screen_name	42	entitieshashtags4indices0	62	entitiesmedia0sizesmediumresize	82	userid
3	truncated	23	entitiesuser_mentions1name	43	entitieshashtags4indices1	63	entitiesmedia0sizesmediumw	83	userprofile_background_image_url_https
4	text	24	entitiesuser_mentions2id	44	entitieshashtags4text	64	entitiesmedia0sizesthumbh	84	userverified
5	in_reply_to_status_id	25	entitiesuser_mentions2indices0	45	entitiesurls0url	65	entitiesmedia0sizesthumbresize	85	userprofile_text_color
6	id	26	entitiesuser_mentions2indices1	46	entitiesurls0indices0	66	entitiesmedia0sizesthumbw	86	userprofile_image_url_https
7	favorite_count	27	entitiesuser_mentions2id_str	47	entitiesurls0indices1	67	entitiesmedia0indices0	87	userprofile_sidebar_fill_color
8	source	28	entitiesuser_mentions2screen_name	48	entitiesurls0expanded_url	68	entitiesmedia0indices1	88	userentitiesurlurls0url
9	retweeted	29	entitiesuser_mentions2name	49	entitiesurls0display_url	69	entitiesmedia0type	89	userentitiesurlurls0indices0
10	coordinates	30	entitieshashtags0indices0	50	entitiesmedia0expanded_url	70	entitiesmedia0id	90	userentitiesurlurls0indices1
11	entitiessymbols	31	entitieshashtags0indices1	51	entitiesmedia0display_url	71	entitiesmedia0media_url	91	userentitiesurlurls0expanded_url
12	entitiesuser_mentions0id	32	entitieshashtags0text	52	entitiesmedia0url	72	entitiesmedia0source_status_id_str	92	userentitiesurlurls0display_url
13	entitiesuser_mentions0indices0	33	entitieshashtags1indices0	53	entitiesmedia0media_url_https	73	entitiesmedia0source_status_id	93	userentitiesdescriptionurls0url
14	entitiesuser_mentions0indices1	34	entitieshashtags1indices1	54	entitiesmedia0id_str	74	in_reply_to_screen_name	94	userentitiesdescriptionurls0indices0
15	entitiesuser_mentions0id_str	35	entitieshashtags1text	55	entitiesmedia0sizessmallh	75	id_str	95	userentitiesdescriptionurls0indices1
16	entitiesuser_mentions0screen_name	36	entitieshashtags2indices0	56	entitiesmedia0sizessmallresize	76	retweet_count	96	userentitiesdescriptionurls0expanded_url
17	entitiesuser_mentions0name	37	entitieshashtags2indices1	57	entitiesmedia0sizessmallw	77	in_reply_to_user_id	97	userentitiesdescriptionurls0display_url
18	entitiesuser_mentions1id	38	entitieshashtags2text	58	entitiesmedia0sizeslargeh	78	favorited	98	userentitiesdescriptionurls1url
19	entitiesuser_mentions1indices0	39	entitieshashtags3indices0	59	entitiesmedia0sizeslargeresize	79	userfollow_request_sent	99	userentitiesdescriptionurls1indices0
20	entitiesuser_mentions1indices1	40	entitieshashtags3indices1	60	entitiesmedia0sizeslargew	80	userprofile_use_background_image	100	userentitiesdescriptionurls1indices1