MODULE CODE: CM3203

Cardiff University: Final Year Project Report

Recognising Place Names in Text Documents

Supervisor: Prf. Chris B Jones

Craig Harris: c1335098
5-6-2016

Abstract

Due to the growing amount of information, created by online social media, there are opportunities to study and understand the usage of natural language throughout the world. The human brain can evaluate and extract information from seemingly random data with ease, however, computers are still unable to process data with little to no context efficiently. This project will attempt to develop a series of machine learning methods to aid in the computer automated recognition of place names within Twitter posts. By examining the usage of toponyms within social media, it may be possible to identify notable patterns that may aid computer recognition systems. Using this approach could allow the possibility of applying co-ordinates (geocoding) to social media information allowing for systems to efficiently identify and provide information on the toponym being discussed. This project is focused around the ambiguity and context issues that are found in non-structured and informal language patterns.

Table of Contents

Figure Listing

Introduction

Project Overview

The primary objective of this project is to develop machine learning methods to correctly identify place names within documents. Due to the difficulty of distinguishing place names from other terms, such as names of people, objects and organisations, the machine learning methods will need to be capable of distinguishing the names of locations in an absolute form.

To attempt to overcome the ambiguous nature of place names in text documents, the machine learning process will have to utilize various pieces of evidence to aid in the correct categorisation and recognition of such names. Examples of the sort of evidence that will be employed in the machine learning process include: whether the name occurs in a gazetteer (a list of place names of the concerning region), if the name is preceded by spatial prepositions such as “near to” or “towards” and whether it is associated with place type terms, such as “town” or “river.”

The focus of this project will be aimed primarily in the areas of Wales and the United Kingdom as a whole. Therefore, the gazetteer used will be locally focused, such as the National Gazetteer of Wales and the Ordnance Survey OpenNames gazetteer product.

The use of spatial relationships will be key in determining if a name is a place or something else. This will be achieved by examining the qualitative and quantitative information that may precede or follow a place name within the particular document. For this particular project it will be more common to find qualitative spatial relationships mentioned within the documents, such as relative locations using proximal relations (“near”, “close”) and orientation based relations (“north”, “south”). Parsing this information will be key in developing the machine learning methods required to correctly identify absolute place names in their correct context.

It has been decided that rather than using traditional text documents to perform analysis on, the project will focus on recognising place names in a pre-made, database, of Twitter posts. This will mean that the documents being dealt with are smaller and somewhat easier to process. However, due to the nature of social media, it would also mean that there is more variation within the text documents. The use of Twitter documents will also introduce several interesting focus points within the project, including the analysis of place name trends, such as when an event happens in a particular area. Also, the way in which places are defined by individuals can change from person to person.

Main Project Objectives

The primary identified objectives of this project are as follows:

- Develop a series of machine learning methods to correctly identify place names within a Twitter post.

- Create a stable accuracy rate regarding place name recognition by developing and expanding the machine learning methods.
- Word recognition should be as accurate as possible to ensure proper place name indexing.
- Attempt to perform geocoding on the results for the place name recognition function.

Project Outcomes

The following points are the desired outcomes for the project. These outcomes include what are required as deliverables for this project and any features that could be used in further development of the project:

- A system that contains appropriate identifiers that will help locate place names within Twitter posts.
    - The system must be mutable, allowing for further development of the identifiers that are in place to locate place names. This will allow for further development to increase system accuracy.
- A system that is able to work with a database or document containing a large amount of text in a variety of formats.
- The system output after the parsing of a body of text will need to include:
    - An appropriate Waikato Environment for Knowledge Analysis (WEKA) file to analyse the performance of the system.
    - A word frequency document for further development of the machine learning methods.

Identified Difficulties for Toponym Recognition

Upon initial review of the project overview and objectives, several obstacles were identified that may be encountered when attempting to retrieve correct place names from the Twitter documents.

The first obstacle identified, was the ambiguous nature of Twitter posts. Twitter posts can be comprised of up to 140 characters. Due to the restrictive character limit, posts are often written in shorthand or abbreviated forms. This could cause issue with understanding what post relates to and if a reference to a place was actual made in the post. The abbreviated format of the posts may make it difficult to identify spatial prepositions that precede or spatial terms that follow a specific place name.

A second obstacle identified is acquiring place names within the correct context. Twitter users employ the use of ‘hashtags’ within their posts. Hashtags are generally short links that are preceded by an octothorpe (#). By using hashtags, users are able to turn single or groups of words into a searchable link, allowing users to search content and keep track of ongoing discussions (Hiscott 2013). Hashtags can relate to a variety of different information; this could potentially cause confusion when parsing the posts for place names being used in the correct context.

Following this obstacle, a second issue in regards to the Twitter post format is '@' symbols. The '@' symbol is used to direct a message to a specific user and usually precedes the users name. This symbol could cause the same issue as the hashtag if a user was referring to a particular company or organisation. In this particular case, the user *would* be referring to a place, but not in the context in which the system is seeking.

Generally, the overall obstacle for this project is the high state of ambiguity and difficulty of establishing context within the Twitter post format.

Toponym Recognition and Geocoding: Background

Related Work

In January 2016, there were a total of 2.307 billion social media users worldwide, this equates to a global penetration of 31% and has risen by 10% since 2015 (Kemp 2016). This is a staggering number of social media users, when considering the world's population in January 2016 was 7.395 billion (Kemp 2016). The large user count for social media has led to a massive amount of usable data in which to explore ambiguous toponym recognition and geocoding. Unlike traditional text, social media is difficult to understand when evaluating context. Users are not bound to the standard rules and typical stylistic approaches that form natural language, this can lead to confusion and misunderstanding (Habib & Keulen 2014).

There have been several, prominent, approaches towards extracting information and correctly geocoding found toponyms within social media text. To better understand how to proceed with this project it was appropriate to examine and outline some key works in this field.

Toponym Recognition

The correct recognition of toponyms in social media can be considered the key approach to attempting to geocode any information found. The basis of geocoding relies on the correct identification of a place name, furthermore, the place name is required to be in the correct context. The underlying problem for toponym recognition in social media text is ambiguity (Overell 2011). The removal of ambiguity from a named entity in text is considered crucial to the success of correct extraction and recognition of toponyms (Leidner & Lieberman 2011). Attempts have been made to disambiguate place names from within a text, however, it is considered a difficult problem when working with data that can reference any global region (Overell 2011).

Alongside ambiguity, there are several other issues that can be identified when reviewing works relating to toponym recognition such as spelling and punctuation issues and constantly changing / new instances of names. Twitter posts are not always written with the most accurate punctuation or correct spelling, this generally leads to 'fuzzy' searches being implemented as an initial, toponym recognition step (Balaji & Gelernter 2013). A 'fuzzy'

search is usually implemented as a pattern match algorithm (often using Regular Expressions), allowing place names not spelt correctly to be matched to the correctly spelt toponym within the lexicon / gazetteer. New and changing place names affect the way in which gazetteer lookups can be used within an NER (Named Entity Recognition) project. The model chosen cannot rely solely on whether the name appears in a gazetteer or dictionary and will need to be couple with other features in order to provide and accurate match.

Toponym recognition is considered a subproblem of information extraction, usually consisting of two phases: identification and classification (Mamat et al 2008). Identification simple refers to the correct recognition of named entities; such as a place name, person or organization. Classification refers to the process of assigning the found names into an appropriate catergory (Mamat et al 2008). According to Mamat et al 2008, there are considered to be three main approaches that can be used to attempt named entity recognition; rule-based, machine Learning-based and hybrid NER. These approaches have been explored and the pros and cons have been considered for each.

- **Rule-based NER**
    - Hand-made rule set that considers grammatical, syntactic and orthographic features
    - Usually paired with dictionaries
    - Rely on manually coded rules and corpora

    - **Pros**
        - Useful when dealing with complex entities
        - Can contain a large rule-set
        - Generally accurate in focused data sets
        - High precision
    - **Cons**
        - Rules need to be changed when examining different domains
        - Rule set can be time consuming to produce

- **Machine Learning-based NER**
    - Use either supervised or unsupervised approaches
    - Statistical analysis for clustering
    - Labelling of entities via classification

    - **Pros**
        - Supervised learning allows for a multitude of approaches
        - Statistical models examine relationships of labelled text
    - **Cons**
        - Supervised learning needs a large amount of data to work accurately
        - Unsupervised learning is mostly unused for NER

- **Hybrid NER**
    - Combination of both rule-based and machine learning-based systems

    - **Pros**

        - Creation of new methods using the strongest aspects of both approaches
    - **Cons**
        - Alongside using the best features of both approaches, Hybrid NER generally suffers from the same cons as rule-based NER

(Mamat et al 2008)

Machine learning models that have been commonly applied to solving the problem of Named Entity Recognition include the Hidden Markov Model, Bayesian Networks, Maximum Entropy Models and Decision Trees. As this project is focused on producing a machine learning solution to NER and geocoding; these approaches have been further explored in the following sections.

Geocoding

Geocoding refers to the process of applying co-ordinates to an address or place name (Gelernter and Zhang 2014). To apply this process effectively the named entity recognition system must be fairly accurate; this is due to the difficulties highlighted above such as place name ambiguity. To ensure the accuracy of geocoding it is important to ensure that the 'feature' set used to classify each word is able to discriminate between a variety of proper nouns (including a name of a person or organisation used in the wrong context). This process will require the experimentation of features to find which provide the most accurate results when run through a classification algorithm.

The most straightforward approach to geocoding a toponym is by using a gazetteer or atlas lookup. Once a toponym in the data set is found, the name identified is compared to an existing gazeteer or atlas, longitude and latitude can be looked up and append to the place name, effectively geocoding the toponym located (Balaji & Gelernter 2013). Again, the problem with this approach is the ambiguity of the text being examined, potentially the place found will be incorrect or simply will not exist within the gazeteer. The ambiguity of Twitter posts is what fuels the need for a more extensive feature list to classify place names and then geocode them effectively.

Machine Learning Approaches

As mentioned in the previous section, there are several options when exploring the machine learning approaches for correct toponym recognition. The options outlined below are seen to be the most popular and explored approaches when focusing on supervised (or semi-supervised) machine learning (Nadeau and Sekine 2007). To better approaches available this section explores and evaluates these models to better understand the options available to during the design and implementation of the proposed system.

ID3 Decision Tree Algorithm

The ID3 Decision Tree Algorithm is a common machine learning approach when working with a set of data that can fall into 'if-then' rules. This approach to machine learning will create a tree based on tests of specified attributes, this will start with some key value represented as a *root node* and proceed to develop itself based on the attribute tests producing some *leaf nodes* as it classifies each case. Each leaf node is joined by *branches* that corresponds to one option from the attribute testing.

Decision Trees allow the users to fully understand and analyse all possible outcomes of a set of decisions. In essence, the ability to discern place names from an unknown test revolves around the ability to *decide* if a word is a place name via a set of tests or attributes in which they are filtered.

A basic example of the use of the Decision Tree algorithm for use within place name recognition can give us insight on whether this method is appropriate in our system. In this example, the word being evaluated is in bold and any preceding phrase is taken note of for use as an attribute in the process.

Figure 1: ID3 Training Data

5 Yes / 5 No

Word / Phrase	**In Gazetteer?**	**Capital Letter?**	**Spatial Preposition?**	**Is Place Name?**
Going to **Cardiff**	True	True	False	**Yes**
North of **Natwest**	False	True	True	**Yes**
Found in **Gabalfa**	True	True	False	**Yes**
At **Home**	False	True	False	**No**
Travel to **shops**	False	False	False	**No**
Here they **are**	False	False	False	**No**
We're **Safe**	False	True	False	**No**
Newport	True	True	False	**Yes**
This is **Swansea**	True	True	False	**Yes**
Near **James**	False	True	True	**No**

To start the decision tree process, it is required to work out the entropy of each possible attribute to find the purest set.

Entropy:

$$H(S) = -p(+)\ log2\ p(+) - p(-) log2\ p(-)$$

Applying this algorithm to the three potential attributes for this data we generate the following entropy for each (Figure 2: ID3 Entropy)

Figure 2: ID3 Entropy

	Entropy (True)	**Entropy (False)**
In Gazetteer?	0 *(Pure Set)*	0.65
Capital Letter?	0.95	1 *(Impure Set)*
Spatial Preposition?	1 *(Impure Set)*	1 *(Impure Set)*

The ID3 algorithm will perform this function recursively for each unused attribute and select the one the possess the lowest entropy. This function can also work using the information gain algorithm, selecting the attribute with the highest information gain of all attributes.

Information Gain:

$$Gain(S, A) = H(S) - \sum(\left(\frac{|Sv|}{|S|}\right) * H(Sv))$$

The ID3 algorithm, in this case, would find the '**In Gazetteer?'** attribute to be the most suitable root node. The algorithm would then continue to process all attributes in this manner until all were used or a perfect fit was found. Examining this example, of which the training set is extremely small, it is clear that no perfect fit would be found before the algorithm exhausted all possible attributes.

The ID3 algorithm is worth considering as a machine learning approach as it adopts an 'If-then' approach which is ideal when attempting to classify named entities such as toponym. However, exploring this method may lead to issues when evaluating Twitter posts due to the ambiguous nature of the language. It would be reasonable to assume that given the three attributes above (Fig 1) a toponym could still be found false in all three cases and incorrectly classified. This problem may occur when the toponym is referred to in a way other than the administrative name given, such as spelling errors or shortening of the name (or not present in the gazetteer), incorrectly capitalised and is found at the beginning of the post (or after a 'hashtag').

Markov Models

The following section looks at both Hidden Markov Models and Maximum Entropy Markov Models as a way to find named entities in text. The two approaches are similar; however, each model differs when dealing with the identification of current states.

Hidden Markov Model (HMM)

The Hidden Markov Model is often used in conjunction with Named Entity Recognition (NER) systems and Part-of-speech Tagging. These systems are designed to classify named entities when provided with a body of text via sequence labelling.

A brief summary of the generative Hidden Markov Model is as follows:

> Given a set of training examples *(x(1), y(1))…(x(n), y(n))*
> Assuming that each *x* refers to a sequence (sentence or post) and each *y* refers to a tag sequence (Noun, Proper Noun etc.).
>
> Once a word is seen in the training data the system can attempt to find the most probable following word in the sequence using the joint probability *p(x,y)*.

The Hidden Markov Model can provide a generative model for sequences (in this case a sentence or Twitter post). This will enable the model to move from each entity in a sentence via a probabilistic approach. The model will decide what the most likely word to follow its current state is, based on the probabilities inferred from the training set. The training set is usually a pre-defined corpus.

The Hidden Markov Model could be a useful approach to locating toponyms within Twitter documents. The model allows for a probabilistic approach to the problem which, when paired with manual tagging and analysing of word trends, may provide a reliable method in which to locate proper nouns.

However, there are several issues with this model in regards to the analysing of Twitter posts. The natural language of Twitter does not always follow the normal structure of the English language. Many words are ambiguous in nature and tagged terms within Twitter (‘hashtags’, ‘@’) have a high likelihood to confuse the probabilities determined by the Markov model. A further issue is finding a place name in the correct context; an organization may be mentioned as oppose to the place in the context of the sentence. The final issue found with the Hidden Markwov Model is that each word is examined as a singular entity. This means that a toponym with a multi-word name such as “St Fagans” would be tagged incorrectly. These problems would provide incorrect results in regard to the correct gathering of toponyms.

Maximum Entropy Markov Model (MEMM)

The Maximum Entropy Markov Model is similar to the Hidden Markov Model. The model relies on probabilities, generated from the current state. The model then employs a list of features to determine the correct tag for the state it is at.

The set of features that the MEMM employs is pre-determined by the user. The features can contain items such as: what the previous tag was, what the following tag is (if any), if the word is alpha or numerical, if the word contains a capital letter or if the previous word contained a capital letter. This process is known as Feature Extraction and helps classify the particular article in the sequence and label it correctly.

This approach is more suited to the locating of toponyms within Twitter posts. The ability to pre-define a list a features and enable the system to check previous and following words would help catergorise the current word being examined, while maintaining the context in

which it is used. The features could also help narrow down ambiguous terms, potentially finding vernacular names that are not defined in gazetteers.

The issues with this approach are less prominent than the Hidden Markov approach. However, difficulties in catergorising ambiguous names or retrieving toponyms in the correct context may still be challenging.

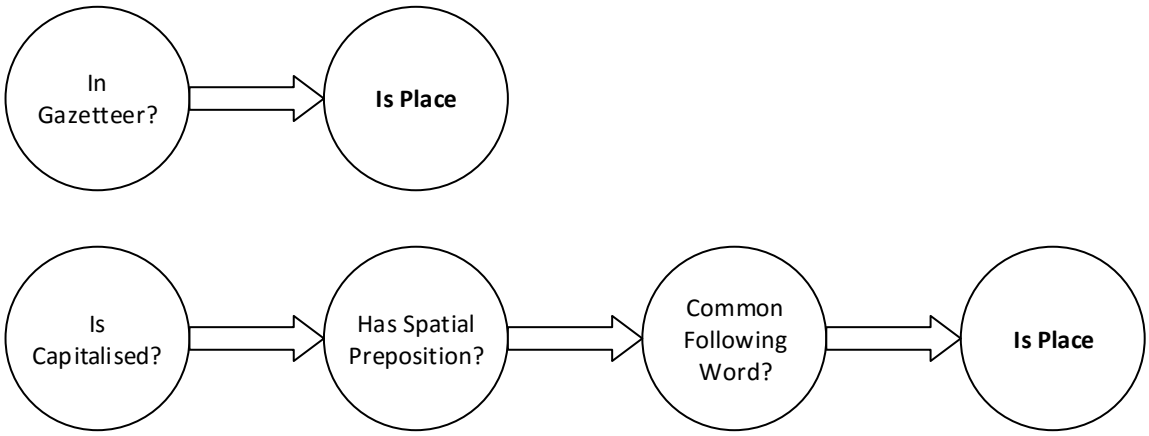
Bayesian Networks

Bayesian Networks are a second, probabilistic approach that could be used to explore the named entity recognition problem. The Markov Models (described in the previous sections) fall under a similar category as the Bayesian Network, however, Bayesian Networks allow for a more complex dependency set. The basis of this dependency is that each variable in the network is conditionally independent of any other variable given its Markov Blanket. Using the Bayesian Network model, it is possible to discover relationships in sequences, if the structure of the sequence (much like a sentence) is unknown.

In the context of named entity recognition, this would allow the inclusion of a set of attributes that are independent of each other, such as a simple check to see if the current word is present in a gazetteer. The model would also allow for a *string* of attributes that are conditionally, only dependent on its parent to reach a conclusion.

To apply this model to the named entity recognition problem, it would be useful to assign each node as an attribute, such as "*Capitalised?*" or "*Spatial Preposition Present?*", which could then be used to find the probability of whether or not the word being examined is a place name or not. A basic, visual representation of this can be seen in Figure 3.

Figure 3: Hidden Markov Model



As seen in Figure 2, "*In Gazetteer?*" is completely independent of the other nodes, this is because if it exists in the gazetteer, it is most likely to be a place name. The other nodes link to each other, creating a flow of attribute checks that will narrow down the probability that the current state, is in fact, a place name.

Overall, the Bayesian Network model appears to be a reasonable approach to finding place names within Twitter posts. Examining a set of data and building a probability table to predict the existence of a toponym may be fairly reliable. The issues with this approach is somewhat similar to the Hidden Markov Model, despite the extended relationships found within Bayesian Networks, a false negative could still be identified.

Word Frequency Analysis

Word frequency analysis in regards to this project would be an *aid* to the machine learning process as oppose to being the sole way in which to achieve the main objectives. The basis of the word analysis approach would assist in the creation of lists that could be used to create *filters* or *attributes* in which to narrow down toponyms.

Word frequency analysis, at its core, is a simple concept. Each word in a body of text is placed into a data structure with a corresponding number based on the amount of appearances within the text. If the word has not been seen before, the word is added to the data structure as a new entry. If the word *has* been seen, then an attribute associated with that word is incremented. Essentially this process is a simple matter of counting words, however, when working with a large amount of text, using a system to perform the word analysis is very efficient.

Within the pre-populated Twitter database that was supplied at the start of the project, there are a total of 219446 Twitter entries for the Cardiff area. Twitter (2016) states that the maximum character length of a post is **140** characters and Wolfram|Alpha (2016) suggests that the average length of an English word is **5.1** characters long. Considering this information, at the high end of the scale, if each Twitter post within the database was a maximum of 140 characters and all written in English, there would be a rough total of **4,817,019** words. This calculation assumes that there is a space after each word except the final word in each the post. This is an extremely large set of data and the use of the whole database within this project may prove to be difficult. Applying some simple word frequency analysis techniques would help make use of every entry within this database table, aiding with the creation of training data for the primary objectives of the project.

Using a basic data structure (such as Python's Dictionary) I would be able to create a simple tally of word frequencies. This paired with a gazetteer lookup function could help prepare a list of the most frequent words that precede and follow a place name which exists within the gazetteer. This may not be an optimal approach as many place names may not exists within the gazetteer itself, such as vernacular variants of administrative names. However, this simple implementation could still provide the project with a good starting point in which to develop a list of *features* to aid in the identification of all toponyms.

Supervised vs Unsupervised Learning

To better understand what approach would best suit a named entity recognition problem it is worth exploring the differences between supervised and unsupervised machine learning.

Within supervised learning models, training data is supplied to the model with both the input data *and* the expected results. The training data must be comprised of negative and positive results in order to train the model on what is expected and what is not. The model then uses the training data to make an educated estimation on what the correct and incorrect results of a set of unannotated data will be. These methods are usually quicker to produce results and are fairly well accurate. The drawbacks to using supervised models are that it requires the data to be annotated with the correct outputs beforehand, this can be time consuming when working with large datasets.

Unlike supervised learning models, unsupervised learning is not supplied with the correct results in the training data. This means that the training data is examined by the model and then (usually) using statistical probability, the model is able to infer the correct output. The input data can be clustered based on statistical properties, the clusters can then be examined and labelled by the user. This model requires no user annotation which can save time on the user's behalf. However, unsupervised models tend to need a much larger amount of data to correctly produce results. The time taken by the system is also increased compared to that of the supervised models.

Considering the main objective of the project, there are benefits of using both unsupervised and supervised learning techniques. Due to the ambiguity of the text being examined and the possibility of different vernacular names for toponyms, I feel that supervised learning (such as the ID3 algorithm) may be the most appropriate approach. Using this approach, I will be able to create a list of testable attributes to filter the data until correct place names are identified. I feel that manually tagging the data, or using a part-of-speech tagger will speed up this process significantly.

System Specification

System Requirements

The areas outlined in this section are key system requirements which have been selected, through the identification, of the main system objectives and project overview.

Functional Requirements

Figure 4 shows the functional requirements that have been identified. These requirements are core features for the system that are desired to achieve the project objectives.

Figure 4: Functional Requirements

Requirement	MoSCoW	Acceptance Criteria
The system is able to correctly distinguish between proper nouns. The system will identify that an initial capital letter for words within the database of tweets can refer to organisations, peoples names or simple incorrect grammar.	**Must Have** (Key functionality for system and end user)	System can correctly identify a place name from a text document or Tweet. The system will use capitalisation as an early baseline to establish the type of word being used and then proceeded to determine if the word is a place name.
The System can develop appropriate datasets to use within the machine learning process.	**Must Have** (Key functionality for system and end user)	The system will be able to produce helpful datasets that can be used in further iterations to improve the correct identification of desired data. This can be achieved via appropriate output once the system has run on supplied test data.
The system can appropriately, and correctly, index and geocode locations found within the Twitter database.	**Must Have** (Key functionality for system and end user)	With the aid of a Gazetteer, the system will correctly identify and geocode locations found with Twitter posts and text documents. This will require a gazetteer lookup function upon implementation.
The System is able to correctly identify vernacular / colloquial place names within the Twitter database.	**Should Have** (High priority functionality for system and end user)	The system will be able to correctly determine if, and what, location is being referred to within a Tweet and output the appropriate location. This may be achieved through the use of ‘fuzzy’ word matching through regular expressions.
The system should be able to deal with datasets from other database formats. It	**Should Have**	The system will be compatible with multiple database formats and not

should also be able to work with various text formats.	(High priority functionality for system and end user)	force the user into using a particular format for the dataset. Using specific software that are designed to work with a wide variety of file formats and structures should enable this requirement.
The system can create mapped space via data gathered throughout the machine learning process. The mapped space will show borders that are often referred to as part of a town or city locally, but administratively are actually not part of that region.	**Could Have** (Possible functionality for system (Desired))	The system will output a visual, or text based, report showing two bordered regions of a particular location. One of these borders will refer to the official administrative borders and the other will use data gathered from parsed Twitter posts to determine where the local population consider part of their town / city. This requirement will be dependent on time constraints but may be achievable during the geocoding step of the system.

Non-Functional Requirements

Figure 5 shows the non-functional requirements that have been identified. These requirements are system based and provide an outline analysis on how the system should perform, as well as desirable system performance features.

Figure 5: Non-Functional Requirements

Requirement	MoSCoW	Acceptance Criteria
Efficient code design that allows the system to perform the word lookup in an appropriate time scale.	**Must Have** (Key functionality for system)	The system will perform the text parsing and supply correct result (in this case, a place name) in an appropriate and workable time frame.

The system must be secure. Data being used within the system, despite being publicly broadcast online, should be kept out of reach of third parties.	**Must Have** (Key functionality for system)	Any information stored online must be appropriately secured via standard means (password protected).
The system must be scalable and allow for project sizes both larger and smaller than the current dataset being used.	**Must Have** (Key functionality for system)	The system must perform appropriate for all dataset sizes. Single and multiple database entries should be processed at appropriate speeds and with the same results. The system performance will be analysed during implementations.
The system must follow appropriate error handling procedures and produce correct and informative user errors.	**Must Have** (Key functionality for system)	The system must display any errors in a readable format, allowing users to recognise machine or user error and troubleshoot.

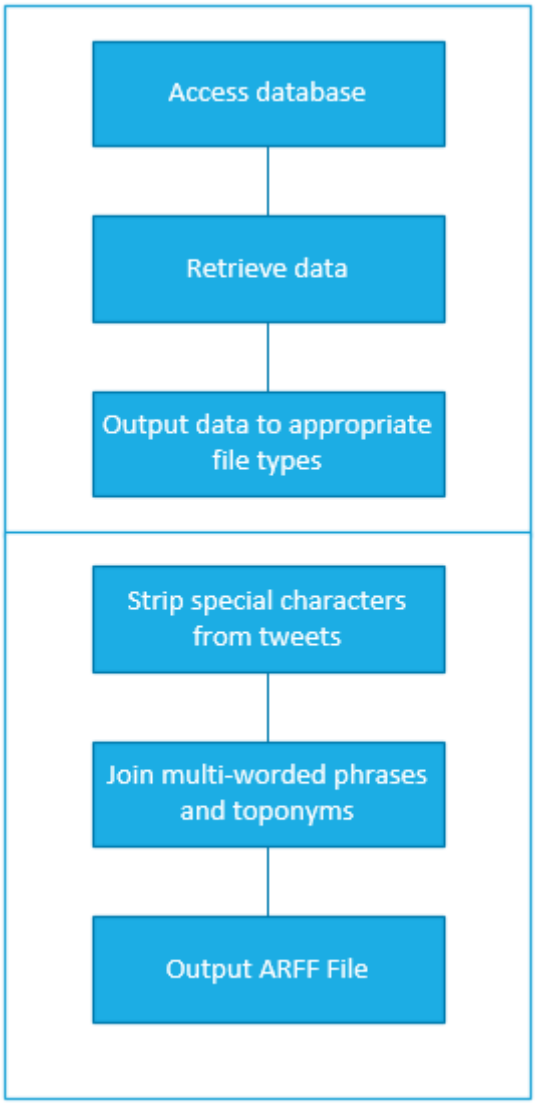
System Design

Following the evaluation of machine learning approaches and system requirements, it is necessary to design the system. This process involves mapping out the system both structurally and associating correct components within the system. The design has been split into sections that cover the main processes which need to be addressed within the system.

System Structure Overview

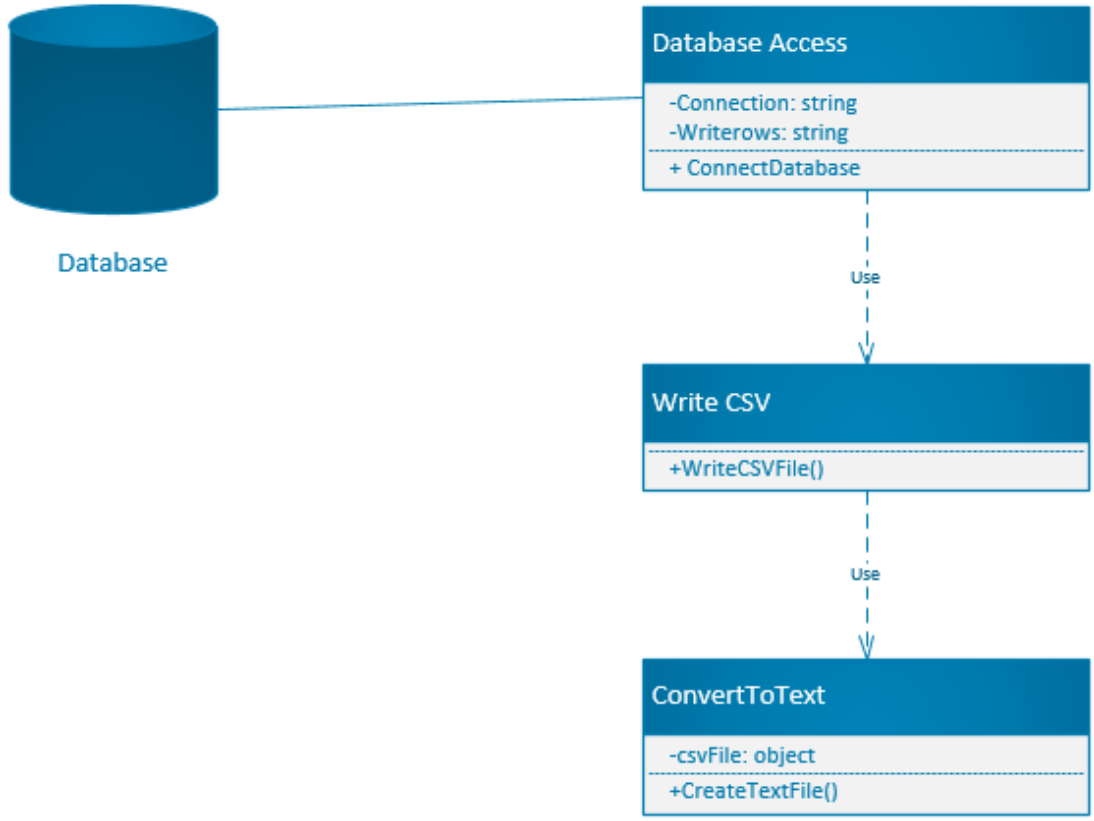
The following diagram shows a general overview of the system structure. Including the main process used by the system and the connection between the relevant sections (Figure 6).

Figure 6: System Structure



Database Access and CSV / Text File Creation

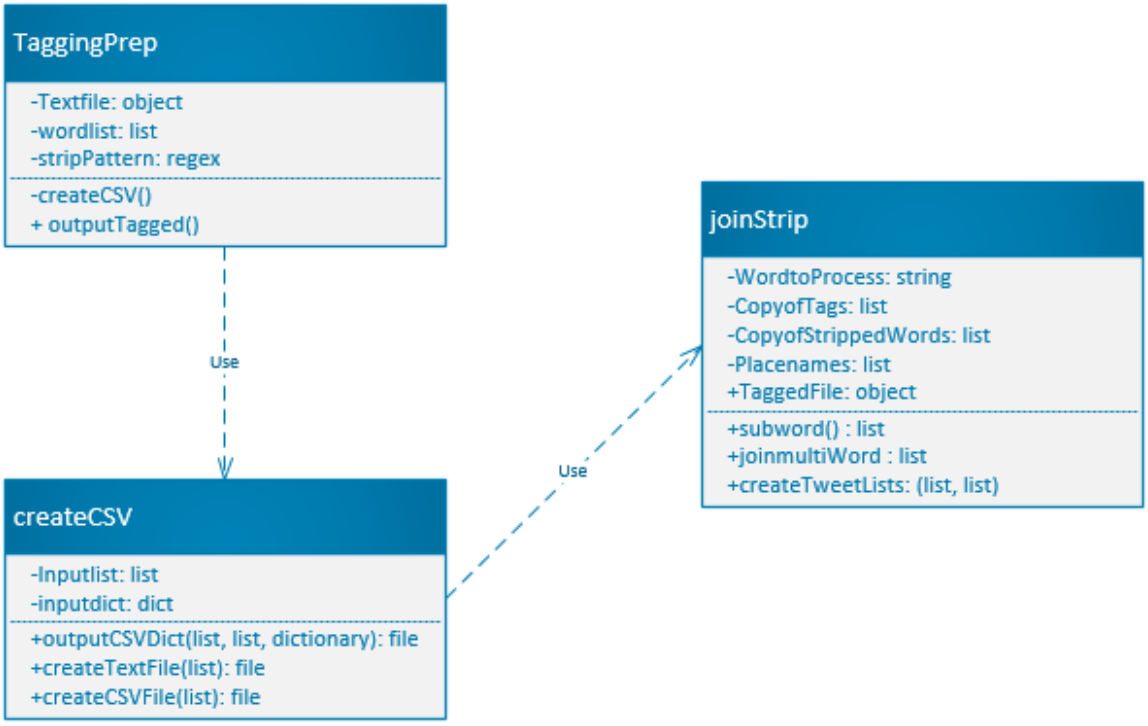
Figure 7: Database Access Class Diagram



This class diagram displays the basic way in which the system accesses and outputs a Comma Separated File from the information stored within the database. The information is accessed through the ConnectDatabase() function and then fed to the WriteCSVfunction(). This functionality is used to create both the gazetteer and Twitter post CSV files. The CSV file can then be exported to a further function to create a standard text file.

Tagging Preparation and Processing of Tagged Words

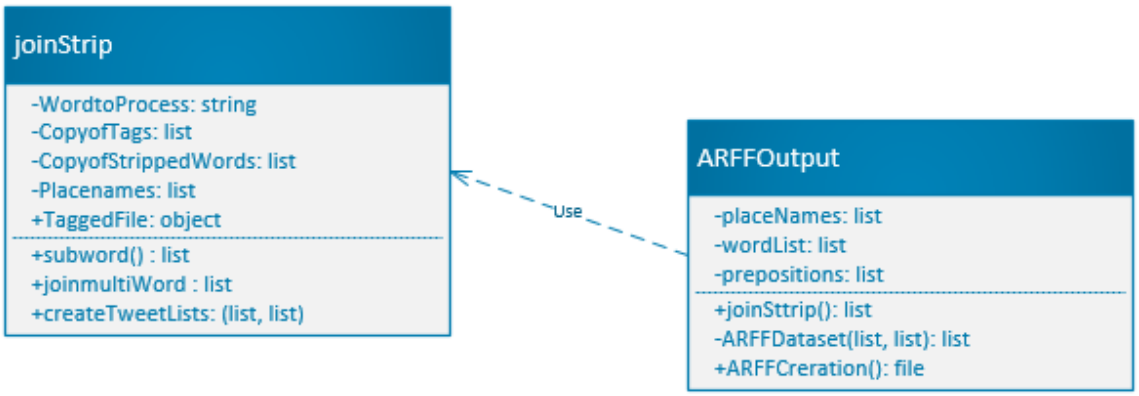
Figure 8: Tagging Preparation and Joining of multi-worded toponyms Class Diagram



This class diagram shows the relationship of the main classes used within the stripping of Twitter posts and joining of multi-worded phrases / place names. The system prepares the file within the TaggingPrep class, the file is converted to a list and stripped of special characters. The createCSV file then uses the list processed in TaggingPrep to create a text file of the stripped text. The joinStrip class then uses a manually tagged file, alongside a copy of the tagset to join multi-worded toponyms and spatial phrases and output them as a list.

ARFF File Output

Figure 9: ARFF File Output Class Diagram



This class diagrams highlights the relationships between the joinStrip and ARFFOutput classes. Within the joinStrip class, the createTweetsList() function outputs both a stipped and non-stripped version of the Twitter posts, in the form of lists. The joinStrip class then provides the ARFFOutput with the appropriate lists required by the ARFFDataset() function. The ARFFDataset() function is called by the ARFFCreation() function, ultimately producing the ARFF file used for WEKA analysis.

Implementation

In this section of the report I will explain the choices I have made in regards to my implementation approach. This will review the language, modules and libraries used as well as each process I have implemented into my code to achieve the desired objectives for the project.

Python

For the project I chose to use Python 2.7 as the programming language. There are several reasons behind this choice which I will briefly examine. The modules used in the project will be evaluated in more depth later in this section.

- **Pre-defined Objects**
    - Python Lists
        - Easily traversable.
        - Due to my approaches to dealing with words in this project, lists were an appropriate data structure to use.
    - Python Dictionaries
        - Simple data structure to use for word frequency analysis.
        - Can be easily output to a 'Comma Separated Value' file format (.CSV).

- **Existing Modules**
  - CSV Module
    - CSV module in Python implements classes to easily read and write CSV files.
    - Useful for the project for storage of 'tweets' and frequency analysis.
  - Python WEKA Wrapper
    - Allows for the use of WEKA classifiers within Python
    - Useful for the output of classified toponyms in which to geocode
  - Python DB
    - DB module allows for easy access to query MySQL databases from within Python.
    - Necessary for accessing the pre-defined Twitter database.
  - NLTK (Natural Language Toolkit)
    - NLTK is a part-of-speech tagger written for Python.
    - Will be used for the tagging of Twitter posts for analysis of appropriate word context.
  - LIAC-ARFF
    - This module helps to create an ARFF file used by WEKA within Python
  - Regular Expressions Operations
    - Used to perform regular expressions in Python

Module and Program Overview

During the creation and implementation of the code I was able to find several Python Modules (briefly explained in previous section) and programs that would make the implementation process easier. This section will highlight the key uses of the modules and programs within the project, along with any additional information that will help understand how they aid reaching the project objectives.

Comma Separated Value (CSV) Module

The CSV module introduced several classes to aid in the creation and reading of the CSV file format. This module was selected to assist in the storage of the majority of data used within this project.

The main functions used within the project were the **reader** / **writer** and **Dictreader / Dictwriter** functions. Using the module API, I was able to effectively implement these functions for use in my project. These functions allowed for the storage of tweets contained within the Twitter database used for the project. The functions were also used for creation of the word frequency analysis graphs which were used to aid the creation of the system feature list.

Python WEKA Wrapper Module

The Python WEKA Wrapper module allows for the use of WEKA classifiers within the Python environment. I attempted to use this module to help with the creation of geocoding function, however, I was unable to utilise the module to great effect. When trying to import the Naïve Bayes classifier model that I had created, the datasets were not processed correctly, resulting in incorrect and incomplete data.

I have used this module with the J48 classifier as an example of how I would solve the geocoding problem within this project.

Python DB Module

The Python DB module was used to access the MySQL database where the Twitter posts were stored. This module was used as an appropriate way to obtain the tweets stored within the Twitter database provided to use directly within the Python platform. Once the data had been queried from the database I was able to output the information into usable formats for the continuation of the project. This module was primarily used in the initial stages of the implementation process.

Natural Language Toolkit (NLTK) Module

Using the NLTK module I was able to part-of-speech tag the existing tweets. This allowed for analysis of the word types found within each post. NLTK allowed me to tagged tweets using a corpus, I was then able to review the data and look for any patterns to use in a statistical approach. Due to the ambiguous and non-structured format of Twitter posts the NLTK part-of-speech tagger identified many, incorrect proper nouns within the text. However, this was useful for reference during the implementation of my own, manual tagging procedure.

Regular Expressions Operations (re) Module

The '.re' module was used as part of the character stripping process within the system. The module allows for the use of regular expressions within Python. This module enabled me to build pattern matching expressions for use within the feature list within the system. The module also allowed me to remove unwanted punctuation and Twitter tags with ease.

Using the 're' module to pattern match words within the Twitter document made the gazetteer lookup function trivial to implement.

LIAC-ARFF Module

The LIAC-ARFF module was added to the implementation to help trivialise the creation of the ARFF file for Weka analysis. This module simply allows for the creation of an object within Python that, when provided with the correct data, formats the object to allow for direct writing to an ARFF file.

Windows, Apache, MySQL, PHP (WAMP) Program and phpMyAdmin

WAMP was used to host a local server containing a version of phpMyAdmin. This allowed me to reliable utilise the database containing the primary data (Twitter posts). I chose to use WAMP over an online version of phpMyAdmin to ensure reliable access to the database at all times. This method worked well throughout the implementation process, however, it meant that the system was less portable. With modifications to the database access functions the system could be used in conjunction with an online version of the database fairly easily.

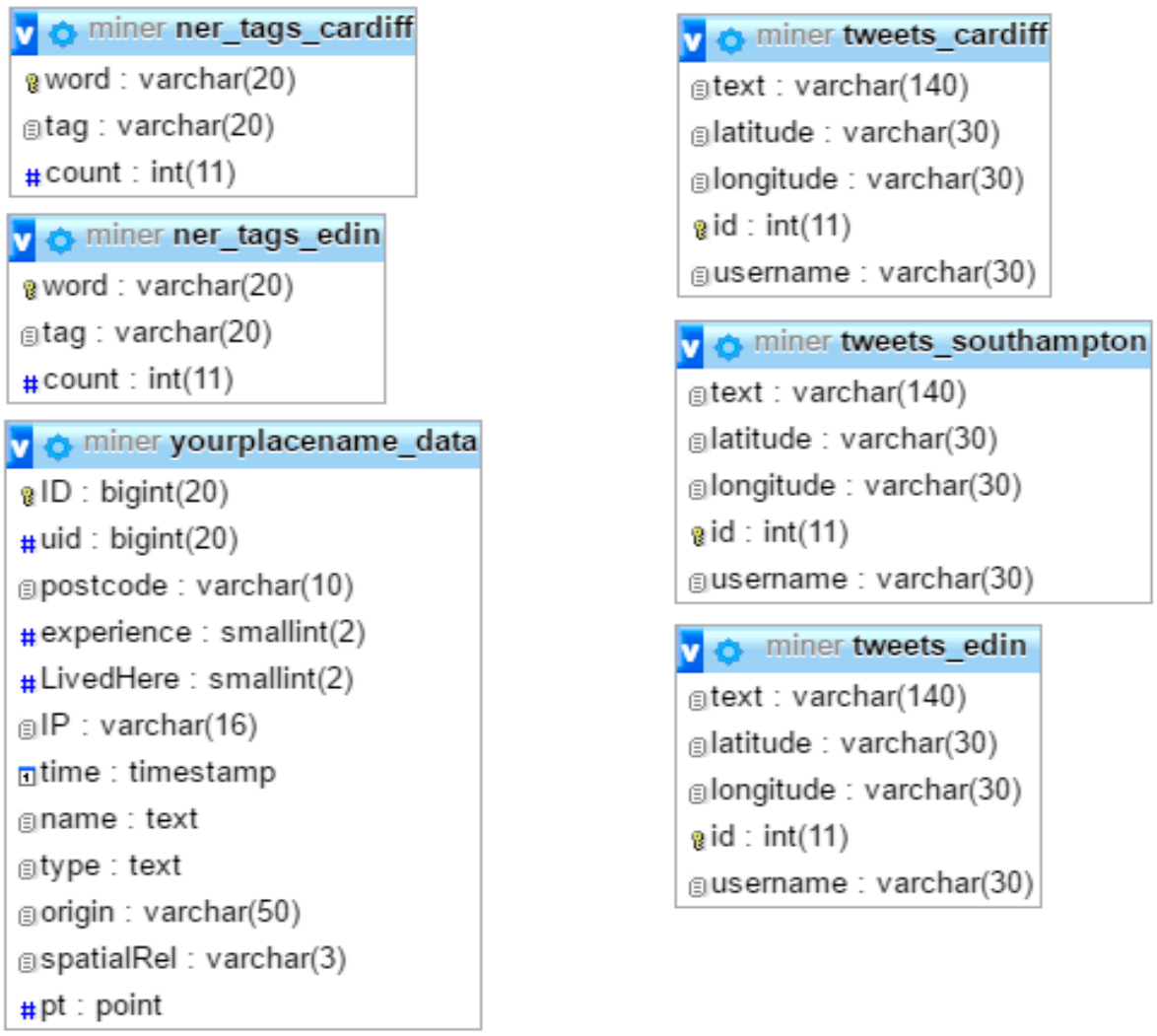
Overview of Implementation

Database Access

The logical starting point for this particular project is being able to effectively access and store the pre-defined Twitter database. This would be an optimal way in which to use the large amount of data effectively within the rest of the system.

The first step was accessing the database manually. To do this I felt that the WAMP software would be most appropriate. WAMP creates a virtual environment on the system the user is working from, emulating a local server. The WAMP software includes the most recent version of phpMyAdmin, this allows for the creation or import of database files to create a functional MySQL database. Due to the size of the database file supplied at the start of the project, using a local host was a more reliable option than using an actually web server. After importing the miner.db file supplied to the virtual server, I was able to view the database and review where I would be acquiring the data for this project from.

Figure 10: Database Overview



The Twitter posts that I would be using for this project were found within the *tweets_cardiff* tables of the *miner* database. The 'tweets' themselves were located within the *text* column. The next step was creating a small function within Python that would allow me to extract the information from this table and store it in a mutable format.

Using the Python DB module API, I was able to easily create a function that would allow me to query the database effectively. Initially this function was implemented with a simple 'print' statement for testing purposes, later the function was modified using the CSV module so that all resulting rows of the query were stored in a CSV file format (Figure 11, Figure 12)

Figure 11: Example of implemented code for Database query and CSV storage of data

```
import MySQLdb as mdb
import csv

# Open database connection
db = mdb.connect("localhost","root","","miner" )

# prepare a cursor object
cursor = db.cursor()

# SQL query construction
sql = "SELECT text FROM tweets_cardiff \
        LIMIT 100"

try:
   # Execute command
   cursor.execute(sql)

   rows = cursor.fetchall()

   # Open a .csv file
   fp = open('cardiff100.csv', 'w')

   # Write database rows to .csv file
   myFile = csv.writer(fp)
   myFile.writerows(rows)

   fp.close()

# Error handling
except:
   print "Error: unable to fetch data"

# disconnect from server
db.close()
```

Figure 12: Example of resulting CSV file format containing Twitter posts

@FA @england Did Villa have any entry? Probs because I've packed 7pairs of trainers LOL @nathanhurn1 aw nothing butt ?? FIFA?

Once this basic function was implemented I was able to easily manipulate the data within the rest of my system.

CSV and Text File Creation

I found that an appropriate solution to the storage of data was using a simple CSV file format. As the majority of data would need to be stored to some degree I produced a series of simple functions that could be called to ease this process.

Figure 13: Example of Dictionary to CSV

```
import csv

def createAfter(list1, dict1, dict2):

    for i in range(0, len(list1)):

        if list1[i] in dict1: #Check key in Dict
            dict1[list1[i]] += 1 #Increment Value

        elif list1[i] in dict2:
            dict2[list1[i]] += 1

        else:
            dict2[list1[i]] = 1

    with open('.\CreatedCSVs\preListAfter.csv', 'wb') as f:
        w = csv.DictWriter(f, dict1.keys())
        w.writeheader()
        w.writerow(dict1) #Write Dict to CSV

    with open('.\CreatedCSVs\unDictAfter.csv', 'wb') as f:
        w = csv.DictWriter(f, dict2.keys())
        w.writeheader()
        w.writerow(dict2) #Write Dict to CSV
```

Figure 13 shows the approach taken to write a dictionary data structure to a CSV file within Python. This particular function was produced to track spatial prepositions in a pre-populated dictionary. The function also created a new dictionary which would populate itself with any words that were not found to be in the spatial preposition dictionary, for use with word frequency analysis and refinement of the pre-populated spatial preposition list. This was one of two functions, one would populate based on words that precede the word being examined; the other would track words that followed.

The other functions within this file simply produced text files for use with tagging tweets (Figure 14).

Figure 14: Example of basic file writing function

```
def createStrippedFileList(twitterList):


    with open('.\strippedTweetsList.txt', 'a') as f:
        f.writelines(str(twitterList)+'\n')
```

Regular Expression Character Stripping

Now that the raw Twitter posts are stored in a mutable format, I felt the next logical approach was to remove all special characters from the posts themselves. Due to the nature of Twitter posts, as mentioned in previous sections, there are many special characters that do not appear frequently in normal text documents. These characters include the frequent use of the octothorpe or 'hashtag' (#) and the (@) symbol. Removing these characters create both negatives and positives when dealing with the problem of ambiguity of toponyms within the post.

'Hashtags' and '@' tags in Twitter can refer to multitude of different things, varying from popular quotes or groups/users within Twitter, to organisations and place names. For example, if a user is talking about watching Cardiff City FC on the television, they may post a tweet similar to:

"Loving the football!! #CardiffCity"

This example highlights the potential ambiguity problem with removing *all* special characters from a Twitter post. 'CardiffCity' is in fact a place name, however, in the context of the post, the user is not referring to the actual city of Cardiff, but more likely the football team Cardiff City FC. With the special characters intact, we can make an educated assumption that the 'CardiffCity' mentioned is referring to the football club, based on the overall context of the post. Once the special characters are removed, this becomes less obvious:

"Loving the football CardiffCity"

With all the special characters removed it is less obvious to what context 'CardiffCity' is being used in. This user may in fact be saying that he is enjoying the football *in* the city of Cardiff.

After considering this issue, I decided to proceed with the character stripping of all special characters from the Twitter post. This decision was made following a meeting with my supervisor regarding the manual text tagging of the tweets. Once it was decided that the manual tagging of tweets would be a good step towards the removal of ambiguity within the text, I was able to safely remove all special characters as I would have both stripped and non-stripped versions of each Tweet as reference material.

To strip all special characters within the Twitter posts I used the Python module; Regular Expressions Operations (.re). This module allows for the use of regular expressions within the Python environment, replacing all characters following the pattern provided. The example code in figure 15 shows the basic structure of the code used to perform the removal of special characters.

Figure 15: Example of implemented code for the removal of special characters

```
import createCSV
import csv
import re

def taggingPrep():

    # Open file as f
    with open('.\InputFiles\cardiff100.csv', 'rb') as f:
        reader = csv.reader(f)
        # Create list 'cardList' from file
        cardList = list(reader)

    for i in range(0, len(cardList)):

        # Convert current index to string
        y = str(cardList[i])

        # Regular Expression to replace all special
        characters except Alphanumeric and Whitespace
        with a blank entry ''
        subString = re.sub('[^a-zA-Z\d\s]', '', y)

        #Join stipped string to remove multiple
        whitepaces
        subString = ' '.join(subString.split())

        #Call function to create text file of stripped
        Tweets
        createCSV.createStrippedFile(subString)
```

Once the function is complete, a createCSV function is called to output the stripped Twitter posts to a text document, ready for manual part-of-speech tagging.

Preparing Spatial Preposition List

To create a functional feature list to properly identify place names within a Twitter post; I implemented a spatial preposition list for the feature list to cross reference. This would allow the implementation of a feature to check if a special preposition such as 'North' or multi-word preposition such as 'close to' was found to precede a word. Using a combination of word frequency analysis, online resources (grammar.yourdicitionay.com) and general knowledge I was able to construct a list to use as reference during the feature check.
If a word or phrase from the list was found to match a word or phrase that preceded the current item being examined, a flag was set in the feature list, creating another filter for the classifier to use to determine toponym existence.

Gazetteer Lookup

To create a gazetteer lookup function, I first needed a reliable source for the gazetteer to be acquired. At the start of the project I was provided with information on two different gazetteers that could be used to cross reference place names found from my system, these gazetteers were the OS OpenNames and National Gazetteer of Wales.

I chose to focus on correctly identifying the toponyms found in tweets within the Cardiff table of the database, however it was necessary to include a large gazetteer as references may be made to other areas of Britain. As finding correct toponyms would be a difficult task, I felt it was worth focusing on larger areas (villages, towns, cities, districts) for the gazetteer and ignoring locations such as street names and addresses. This focus may result in a loss of accuracy within the final system, however, the dataset is extremely large if street names are included, making it difficult to analyse the system performance appropriately.

To implement the gazetteer, it was first uploaded to the working database system. To reduce load on the database and speed up query time, the data was then exported to a CSV file and ultimately converted to a text file for quick, toponym comparison. Using the text file gazetteer, I was able to compare the current word being examined with the gazetteer list easily within the feature list.

Word Frequency Analysis and NLTK

While developing the feature list for my classifier I felt it would be useful to apply some basic word frequency analysis to the Twitter database, allowing me review any instances of common following words after a place name found in the gazetteer. I felt that this process would be helpful in identifying opportunities to implement new features to my feature list.

Figure 16 shows the top 5 results for words that follow a gazeteer place name. This information was generated by the word frequency function.

Figure 16: Word Frequency Table

To	**Count**: 2130
You	**Count**: 1019
In	**Count**: 974
And	**Count**: 911
From	**Count:** 555

This information was gathered and examined to attempt to create an accurate 'following word' feature for the classifier to work. Upon the examination of the data, it was clear that many of the words would not be include as they are just general words that are common to follow any type of word (not just toponyms). Ultimately, I decided that words such as 'street' and 'avenue' were more appropriate for the following word list.

NLTK was used within this project to attempt basic part-of-speech tagging for the collection of tweets pulled from the database. The speech tagging that NLTK attempted to perform was fairly reasonable given the state of the data. However, due to the difficulty in interpreting non-structured language found in Twitter posts, many words were mislabeled. To perform the tagging, I used the Brown Corpus and applied the part-of-speech tagger to the entire Twitter list that had been compiled. NPP is the tag given to proper nouns through NLTK tagging and the tagger chose to mark any instance of a capital letter as such. As there is no need for a spell or grammar checker when writing Twitter posts, much of this tagging was incorrect. Once I had started to develop the plan for the feature list, using word frequency analysis and preposition lists the NLTK data acquired became less relevant. It became fairly clear that part-of-speech taggers could not perform well on ambiguous text, therefore I opted to manually tagging the Twitter posts myself.

Manual Part-of-Speech Tagging

Considering the evaluation of text tagging using automated means (NLTK) and following a meeting with my supervisor, I decided that manual tagging of the Twitter posts would provide the most accuracy when deal with ambiguous toponyms. During the initial word frequency analysis, it became apparent that multiple worded place names and spatial prepositional phrases would be difficult to track, even when exploring the use of a 'moving window' approach. To solve this problem and to reduce the ambiguity of certain toponyms, a custom based tagging system that would be appended to the text manually seemed like an appropriate solution (Figure 17).

Figure 17: Key for custom made part-of-speech tagging

Tag Type	**Tag**
End of Word / Phrase	[#]
Proper Noun	[np]
Noun	[n]
City	[c]

Spatial Preposition Phrase	[spp]
Spatial Preposition Single	[sp]
Organisation	[o]
Generic Place	[pl]
County	[cu]
Town	[t]
District	[d]
Country	[cr]
Village	[v]

To allow for future proofing of the project, I included a more in-depth tagging system than required for this project. This breakdown of the types of places mentioned could be useful for further development with the current system, allowing for a more specific search for named entities.

Joining of Multi-Worded Toponyms and Phrases (Tokeniser)

With the creation of the tag set, I was able to mark multiple word toponyms and spatial prepositions, allowing the system to identify them and combined them into a single index within the list. When paired with the preposition lists and gazetteer lookup function created previously, the system would be able to classify a larger array of place names.

Figure 18: Example of a tagged Twitter Post

Non-stripped / Non-Tagged:
366. Cardiff #Spoons (@ The Ernest Willows (Wetherspoon) in Cardiff, South Glamorgan) https://t.co/EvAVDNctBf

Stripped / Tagged:
366 [c]Cardiff[#] Spoons [pl]The Ernest Willows[#] [o]Wetherspoon[#] in [c]Cardiff[#] [cu]South Glamorgan[#] httpstcoEvAVDNctBf

In the example shown in Figure 18, each of the appropriate place names within the post have been tagged. However, in the context of this Tweet, the classifier should be able to extract only the toponym that is actually relevant; in this case that would be '[c]Cardiff[#] [cu]South Glamorgan[#]' at the end of the post.

Using the end tag that I have defined [#], the initial function that processes the 'tweets' will be able to identify that because there is no end tag after 'South' then it must be a multi-worded toponym. The function will parse the text, looking for the tags and end tags and then merge any multi-worded entries into their own index within the list. This will result in the following Python list:

[[366], [Cardiff], [Spoons], [The Ernest Willows], [Wetherspoon], [in], [Cardiff], [South Glamorgan], [httpstcoEvAVDNctBf]]

The function outputs both a tagged and completely stripped version (as shown in the above example) for processing by the feature list defined. Having the non-tagged version allows for direct and simple gazetteer lookups, where the tagged list allows for the identification and reduced ambiguity of what *is* actually the place name in the context of the post.

ARFF File Creation

To create the ARFF file I implemented an 'if-then-else' statement which would utilise the data obtained throughout the implementation process to output a set of Boolean results. This process makes use of multiple outputs created throughout the system including: Gazetteer List, Preposition List, Following Words List. The purpose of this function is to create a set of data which will be useable in WEKA to allow for the comparison and analysis of the best classifier to use for the ultimate goal of geocoding toponyms found in Twitter posts.

All produced lists are opened within this function and then converted to a Python list. In many cases, the '.lower()' method is called on the lists to allow for a more accurate comparison between the tweets and the lists created. This is due to Twitter posts not requiring correct capitalisation and therefore certain place names may be found in a lowercase format. Likewise, prepositions may be found to be incorrectly capitalised (Fig 19).

Figure 19: Lower Case Conversion

```
with open('.\InputFiles\gazList.txt', 'rb') as f:

    for line in f:
        k = line
        placenamesLower.append(k.strip().lower())
```

The process of applying True or False values to each word is performed through multiple check phases for each word being examined. The check phases are comprised of 'if-else' statements which evaluate each word based on my identified list of features. The feature list examines the following:

1. Is the word present in the Gazetteer?
2. Is the word alphabetic and does it start with a capital letter?
3. Does a preposition precede the word?
4. Does a unique following word come after the currently examined word?

Using this list of features, the systems examines each word and appends the True or False result to a list of lists. This data is then placed into an object which allows LIAC-ARFF to format the data correctly and then be written to an ARFF file (Fig 20).

Figure 20: ARFF File Creation

```
def ARFFCreation():

    dataSet = ARFFDataset(Stripped, nonStripped)

    attList = [
    ('Gazetteer', ['TRUE', 'FALSE']),
    ('CapitalLetter', ['TRUE', 'FALSE']),
    ('Preposition', ['TRUE', 'FALSE']),
    ('FollowingWord', ['TRUE', 'FALSE']),
    ('Place', ['yes', 'no'])
    ]

    obj = {
        'description': u'',
        'relation': 'PlaceNames',
        'attributes': attList,
        'data': dataSet,
    }

    with open('.\CreatedCSVs\Data.arff', 'a') as f:
        f.write(arff.dumps(obj))
```

As seen in Figure 20, the feature list chosen is provided with a Boolean option, when each word is run through the feature list it outputs an appropriate Boolean value for each attribute. An example of this can be seen in Figure 21. The ‘Place’ attribute is given a ‘yes’ or ‘no’ option to choose from. This attribute is not assigned within the feature list. The ‘Place’ attribute is assigned manually to each set of data to create the training data that will be used in WEKA.

Figure 21: Feature List Example

Word to be examined: 'cardiff'
Preceding words: 'north of'
Following word: 'here'

Is in gazetteer?	True
Has capital letter?	False
Preposition?	True
Following Word?	False

Data = [True, False, True, False, yes]

Geocoding Function

I was unable to fully implement the geocoding function as I initially planned. Using WEKA I was able to create a model to help classify toponyms correctly. Despite the progress made with regards to classifying the toponyms in Twitter posts I was unable to get the model working correctly using the Python WEKA Wrapper module.

Despite not having fully implemented the geocoding functionality, I was able to prepare code to display how I planned on approaching this problem. By using the Python WEKA Wrapper module, I was able to import classifiers used in WEKA, directly into the Python code. This would allow me to step through each classification result, alongside the actual Twitter posts. By accessing the label for each instance, I would be able to make an index search within the Twitter post list and output the relevant word.

Figure 22: Geocoding function

```
import weka.core.jvm as jvm

import joinStrip

tempList = list()

jvm.start()

data_dir = "C:\Users\Softmints\Desktop\Diss\Code\WEKA"

from weka.core.converters import Loader
#Prepare ARFF Loader
loader = Loader(classname="weka.core.converters.ArffLoader")
#Assign and load ARFF data file
data = loader.load_file(data_dir +
"\TestDataEleventoTwentyTwo.arff")
data.class_is_last()

from weka.classifiers import Classifier
#Assign classifier
cls = Classifier(classname="weka.classifiers.trees.J48")
cls.build_classifier(data)

#For each item in the dataset, output label index and distribution
for index, inst in enumerate(data):
    pred = cls.classify_instance(inst)
    dist = cls.distribution_for_instance(inst)
    print(str(index) + ": label index=" + str(pred) + ", class
      distribution=" + str(dist))

    if str(pred) == "0.0":
        tempList.append(str(index))
```

Once the correctly classified toponyms have been output to a list or dictionary I could assign co-ordinates based on the gazetteer list stored on the database. This could then be compared to the co-ordinates provided within the Twitter post database to attempt to narrow down the ambiguity of the place name.

Issues found during Implementation

The main issues I discovered during the implementation section of the system were the processing of features for the ARFF file creation section. The issues stemmed from deciding upon an appropriate model to use to create the feature list filter. Initially it seemed the 'if-then-else' approach would suffice, however, upon the evaluation of the WEKA results, it seems this approach may have aided in skewing the results to a certain extent (this is mainly in regards to the gazetteer lookup attribute holding so much weight in the classifier testing).

The second issue I encountered was the implementation of the geocoding function. Although the basic functionality is present (as shown in the previous section), I was unable to achieve full implementation of this function into the system. This was mainly due to problems encountered when using the Python Weka Wrapper module. The problems

encountered involved not being able to instantiate my Bayes classifier model into the Wrapper module.

Weka File Processing

When classifying data through WEKA it is useful to examine certain aspects of the output report to create a comprehensive and correct analysis of the system performance. Using the WEKA explorer, I was able to run my ARFF training data through several different classifiers, allowing me to evaluate the way in which the data is handled and identify any issues within the feature list used to create the dataset.

Analysis of Accuracy (Naïve Bayes)

The first classifier examined on the dataset was **Naïve Bayes**. The data was run using 10-fold cross validation. The first dataset consisting of 111 words was ran within the WEKA explorer. The results for this classifier are shown in Figure 23.

Figure 23: Naive Bayes Output 1

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          99               89.1892 %
Incorrectly Classified Instances        12               10.8108 %
Kappa statistic                          0.5644
Mean absolute error                      0.1427
Root mean squared error                  0.2839
Relative absolute error                 49.5292 %
Root relative squared error             75.3171 %
Total Number of Instances              111

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.526     0.033      0.769     0.526     0.625      0.835    yes
                 0.967     0.474      0.908     0.967     0.937      0.835    no
Weighted Avg.    0.892     0.398      0.884     0.892     0.883      0.835

=== Confusion Matrix ===

  a  b   <-- classified as
 10  9 |  a = yes
  3 89 |  b = no
```

Upon initial review of the accuracy of the **NaiveBayes** classifier, the process seems to have performed well. However, upon further inspection of the results it is evident that the correctly classified instances have a skewed accuracy. Due to large amounts of 'no' (or words that are considered to be non-place names), the classifier looks as if it has performed well. Looking at the **Correctly Classified Instances** we see that 99 of 11 are correctly classified and only 12 incorrectly classified, resulting in an accuracy of 89.1892%. Upon further inspection, if we examine the confusion matrix, we can see that in fact, 9 of the place names labelled as 'yes' (were place names) within the training data were incorrectly

considered to not be place names and classified as such. This information is known as false negatives and can be further examined within the **Recall** percentage of the WEKA output. Recall (sensitivity) of the classified instances displays what fraction of the data that was actually positive was predicted positive. In figure 23 we can see that the recall for the classification of 'yes' is very low, resulting in 0.526 which is little over half of the correctly classified positives.

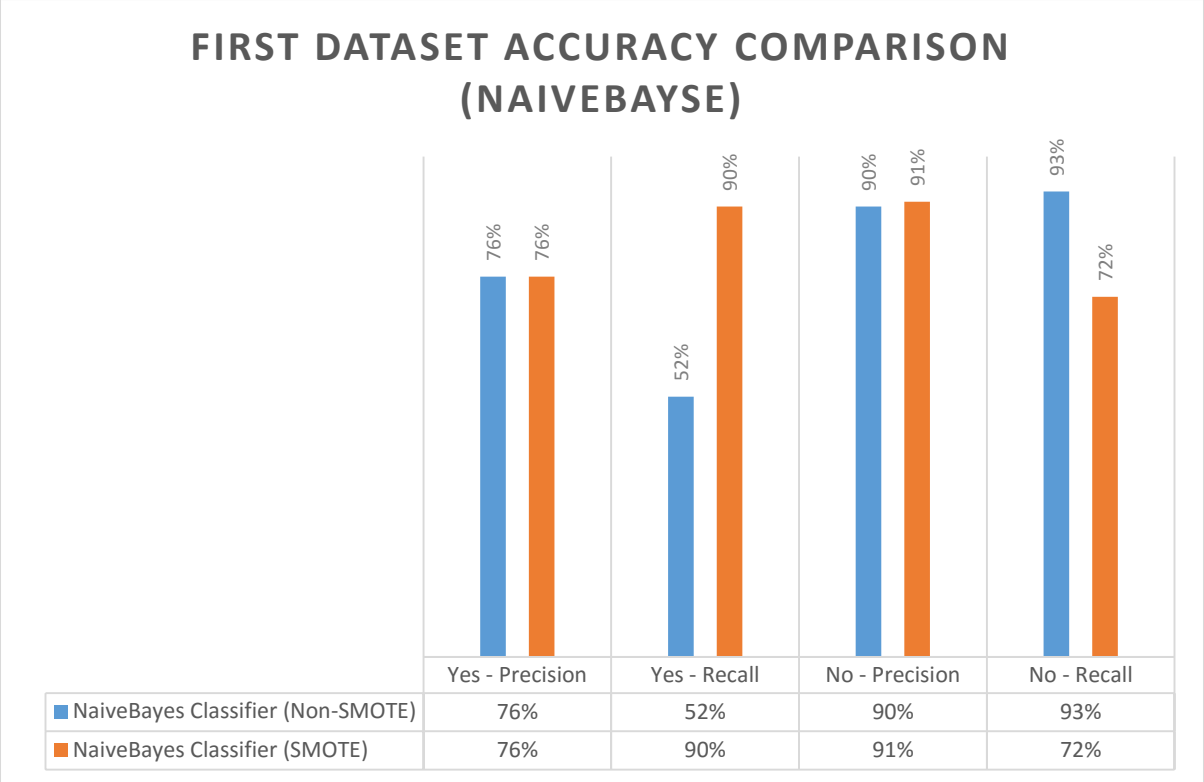
As the accuracy seemed to be skewed from the high percentage of 'no' results in the training data, I chose to apply a filter to attempt to balance the results. The filter used is known as SMOTE (Synthetic Minority Over-sampling TEchnique). SMOTE attempts to balance datasets by artificially creating more instances of the lowest found attribute, in the case of this particular dataset the attribute found to be the lowest was 'Place: Yes'. I applied the SMOTE filter to increase the amount of positive results in the dataset by 200%. Once the SMOTE method had been invoked on the dataset I then applied another filter to randomized the ordering of the data in the ARFF file. The randomization was to ensure that when 10 fold cross-validation was performed, data would not be grouped together (large amounts of no's and yes's next to each other). The results of the SMOTE filtered dataset are displayed in Figure 24.

Figure 24: Naive Bayes Output SMOTE

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         140               83.3333 %
Incorrectly Classified Instances        28               16.6667 %
Kappa statistic                          0.6689
Mean absolute error                      0.2155
Root mean squared error                  0.3252
Relative absolute error                 43.4751 %
Root relative squared error             65.3105 %
Total Number of Instances              168

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.908     0.228      0.767     0.908     0.831      0.888    yes
                 0.772     0.092      0.91      0.772     0.835      0.888    no
Weighted Avg.    0.833     0.154      0.845     0.833     0.833      0.888

=== Confusion Matrix ===

  a  b   <-- classified as
 69  7 |  a = yes
 21 71 |  b = no
```

The results of the dataset once SMOTE had attempted to balance the results were more positive, however, I believe that this is due to the replication of instances in the dataset that contain a 'True' label for if it is contained within the gazetteer listing.

Figure 25: Database Accuracy Comparison



Precision: The precision accuracy relates to the what fraction of the data that was predicted positive was *actually* positive. In the comparison shown in figure 25, there is little to no difference in the cases of true positives classified.

Recall: The recall accuracy refers to what fraction of the data that was actually positive was *predicted* positive. The recall values vary somewhat for the SMOTE and non-SMOTE datasets. The number of instances of actual positives being predicated as positive in the SMOTE dataset is roughly 38% higher than that of the Non-SMOTE dataset. However, as mentioned previously, this may be due to the way in which SMOTE has synthetically generated its extra datasets.

Once I had analysed the accuracy of the initial SMOTE dataset, I created a model using the dataset as the training data. Using a second set of data, consisting of 125, I once again applied the SMOTE filter to the dataset to help balance the 'yes' and 'no' classes respectively. I then ran the saved model from the initial training set on the new test set resulting in the figures shown in figure 26. Much like the original dataset, I found that the high Precision and Recall rates may be explained by the existence of the gazetteer lookup attribute.

Figure 26: Naive Bayes Model Results

```
=== Summary ===

Correctly Classified Instances         141               82.9412 %
Incorrectly Classified Instances        29               17.0588 %
Kappa statistic                          0.6635
Mean absolute error                      0.1322
Root mean squared error                  0.2336
Total Number of Instances              170

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0.264      0.674     1         0.805      0.993    yes
                 0.736     0          1         0.736     0.848      0.993    no
Weighted Avg.    0.829     0.093      0.885     0.829     0.833      0.993

=== Confusion Matrix ===

  a  b   <-- classified as
 60  0 |  a = yes
 29 81 |  b = no
```

Analysis of Accuracy (J48)

The second classifier I chose to evaluate was a decision tree model known as J48. This model as it applies a different approach compared to that of the Naïve Bayes classifier by attempting to construct a tree to determine correct data classification. Using the same datasets as previously used I performed an initial classification of the dataset using 10 fold cross-validation as before figure 27. I then continued to perform the same SMOTE and model creation as used with the Naïve Bayes classifier (Fig 28, 29 and 30)

Upon examination of the J48 classifier I found that when using the non-SMOTE filtered dataset, the tree created by J48 was pruned to just the gazetteer attribute (Figure 27). Once the dataset had grown through the use of the SMOTE method the tree size increased, resulting in the inclusion of the preposition attribute to classify the dataset.

Once the trained classifier model had been created I was able to review the J48 classifier's performance. Ultimately, the results of the model suggest that the J48 classifier was a weaker choice than that of the Naïve Bayes model. The Recall and Precision values found within Naïve Bayes were more consistent in correctly identifying actual and predicted positives. Due to the amount of words being classed as 'not' a place name was higher, the correct identification of the negative values is of less interest to me.

Figure 27: J48 Results Output 1

```
=== Classifier model (full training set) ===

J48 pruned tree
------------------

Gazetteer = TRUE: yes (11.0/2.0)
Gazetteer = FALSE: no (100.0/10.0)

Number of Leaves  :     2

Size of the tree :  3


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          97               87.3874 %
Incorrectly Classified Instances        14               12.6126 %
Kappa statistic                          0.4918
Mean absolute error                      0.2083
Root mean squared error                  0.342
Relative absolute error                 72.2839 %
Root relative squared error             90.7277 %
Total Number of Instances              111

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.474     0.043      0.692     0.474     0.563      0.628    yes
                 0.957     0.526      0.898     0.957     0.926      0.628    no
Weighted Avg.    0.874     0.444      0.863     0.874     0.864      0.628

=== Confusion Matrix ===

  a  b   <-- classified as
  9 10 |  a = yes
  4 88 |  b = no
```

Figure 28: Results Output SMOTE

```
=== Classifier model (full training set) ===

J48 pruned tree
------------------

Gazetteer = TRUE: yes (20.0/2.0)
Gazetteer = FALSE
|   Preposition = TRUE: yes (8.0/2.0)
|   Preposition = FALSE: no (102.0/14.0)

Number of Leaves  :     3

Size of the tree :  5


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         107               82.3077 %
Incorrectly Classified Instances        23               17.6923 %
Kappa statistic                          0.5404
Mean absolute error                      0.2596
Root mean squared error                  0.3859
Relative absolute error                 62.4977 %
Root relative squared error             84.7999 %
Total Number of Instances              130

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.579     0.076      0.759     0.579     0.657      0.77     yes
                 0.924     0.421      0.842     0.924     0.881      0.77     no
Weighted Avg.    0.823     0.32       0.817     0.823     0.815      0.77

=== Confusion Matrix ===

  a  b   <-- classified as
 22 16 |  a = yes
  7 85 |  b = no
```

Figure 29: J48 Dataset Accuracy

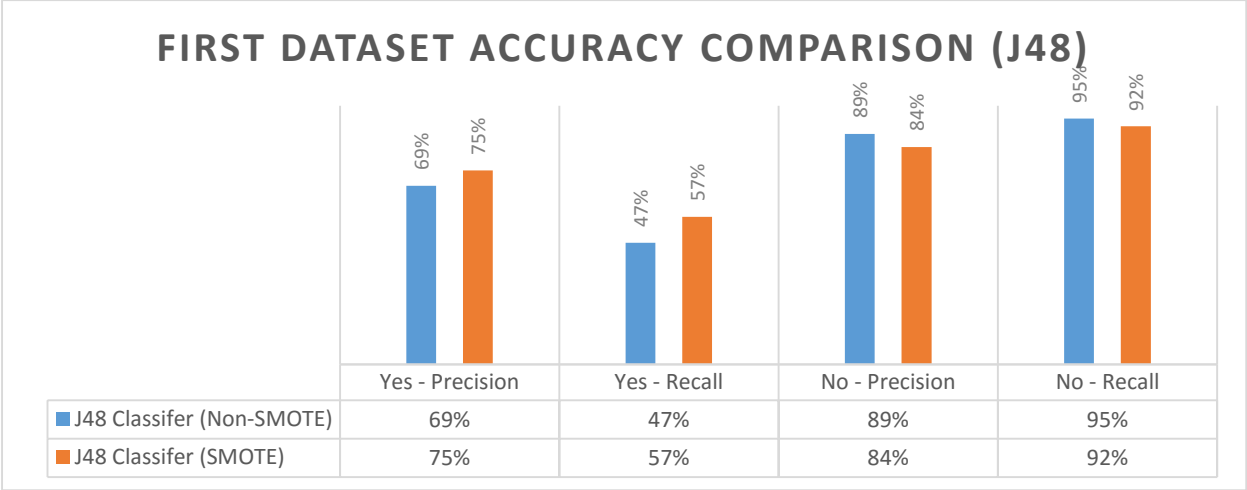


Figure 30: J48 Model Output

```
=== Summary ===

Correctly Classified Instances         160               94.1176 %
Incorrectly Classified Instances        10                5.8824 %
Kappa statistic                          0.8741
Mean absolute error                      0.1778
Root mean squared error                  0.2356
Total Number of Instances              170

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.967     0.073      0.879     0.967     0.921      0.971    yes
                 0.927     0.033      0.981     0.927     0.953      0.971    no
Weighted Avg.    0.941     0.047      0.945     0.941     0.942      0.971

=== Confusion Matrix ===

   a   b   <-- classified as
  58   2 |   a = yes
   8 102 |   b = no
```

Evaluation

Overview of WEKA Results

Following on from the previous section, the first areas to evaluate are the results of the classifiers trained with WEKA. In the Accuracy Analysis I was able to identify some possible difficulties within the training and test data that may have caused some issues with the accuracy of the models.

After the initial accuracy comparison of the two main classifiers I decided to evaluate, Naïve Bayes seemed like the most reasonable choice. Once the classifier had been trained using the supplied with the training data the accuracy of the classifications seemed to be fairly reasonable. However, as mentioned within the Accuracy Analysis, I believe that the accuracy is fairly skewed. Due to the way in which I process each word within the ARFF output function, it seems that the most definitive attribute to if a word is a place name or not is the existence of a 'true' Boolean value within the '**Is in Gazetteer?'** attribute. This suggests that the feature list is not strong enough to properly determine an ambiguous place name and relies on the gazetteer lookup as its main deciding factor.

Once the data had been expanded using the SMOTE filter within WEKA the results became slightly less skewed. The decisions made by the classifier seemed to extend beyond just the existence of 'true' within the gazetteer attribute. This was made more apparent when exploring the J48 classifier. The J48 classifier, using the expanded dataset, produced a tree which also included leaf nodes that checked the value within the preposition attribute. This suggests that with a larger data set the results may be less skewed in favor of the gazetteer lookup attribute.

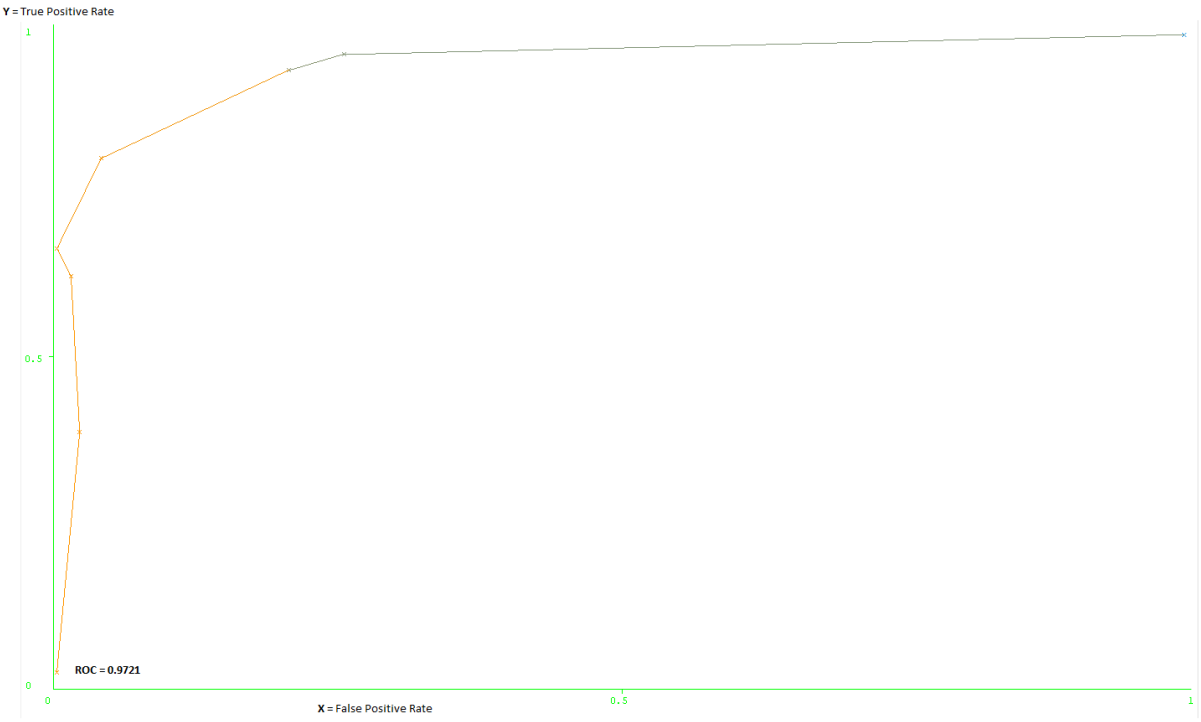
Despite the difficulties with the implementation of a working classifier, it is worth reviewing the general performance and accuracy of the selected NaiveBayes classifier. Once supplied

with enough data, the classifier is able to correctly label the vast majority of the dataset, this provides a solid foundation for the further development of the geocoding function.

ROC Curves (Receiver Operating Characteristic Curve)

In order to evaluate the trained Naïve Bayes classifier further I have produced the ROC curves for several instances of test data. A ROC curve is plotted against the False Positive and True Positive results of the model being run on the test data. The information plotted can be used to examine the classifiers ability to correctly predict the correct way in which to classify the dataset provided to it. All of the ROC curves shown were created through the WEKA explorer upon completion of classifying a given dataset.

Figure 31: ROC Curve Naive Bayes



This curve was plotted under the ROC value of 0.9721 and is examining the results of the 'yes' class within the dataset. The Y axis represents the True Positive Rate; in this case this was equal to **1**. The X axis represents the False Positive Rate; in this particular example this was equal to **0.264**.

The ROC curve suggests that the classifier performed well within this field. However, upon further inspection of the model summary we can see that the system incorrectly classified 29 instances in the 'no' class (Figure 32).

Figure 32: Naive Bayes Model Summary

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0.264      0.341     1         0.508      0.972    yes
                 0.736     0          1         0.736     0.848      0.972    no
Weighted Avg.    0.768     0.032      0.921     0.768     0.807      0.972

=== Confusion Matrix ===

  a  b   <-- classified as
 15  0 |  a = yes
 29 81 |  b = no
```

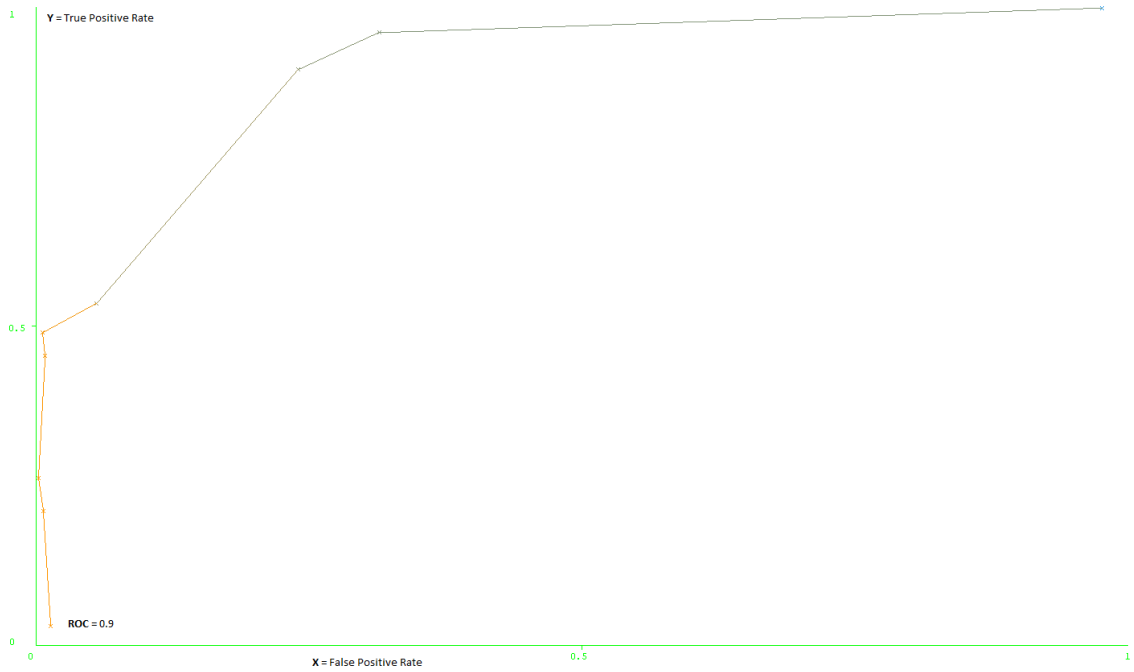
As mentioned in the WEKA Accuracy analysis section of this report, despite what seem like positive results from the classifier, the system is only able to correctly identify the Gazetteer attributes effectively. After a review of the test data the model was run on, I found that the 15 True Positives that are shown within the results section contain a mix of True and False values within the gazetteer attribute. Despite this, I feel that 29 False Positives are caused by the fact that several of the correctly classified instances contain a False value in the gazetteer attribute. The classifier is pattern matching to attempt to correctly identify the True Positives and failing when a similar match is found but is actually labeled as not being a place name. This may be considered an issue with the data size, the total number of instances used is 125. The issue with increasing the instance size is the difficulty in obtaining useful tweets that can provide varied attributes from the existing database.

I felt it necessary to investigate any possibility of incorrect or weighted results further. To attempt to resolve this issue I created a file with, close to, an even mix of yes and no classed instances (25 yes, 24 no). I then randomised the ordering of the data to ensure that there was a reduced possibility of association based on neighbours weighting the results. The ROC curve for the mix data set is shown in Figure 34. The summary of the results are shown in Figure 33.

Figure 33: Mix Data Summary

```
Correctly Classified Instances          41               83.6735 %
Incorrectly Classified Instances         8               16.3265 %
Kappa statistic                          0.6717
Mean absolute error                      0.226
Root mean squared error                  0.3328
Total Number of Instances               49

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.96      0.292      0.774     0.96      0.857      0.9      yes
                 0.708     0.04       0.944     0.708     0.81       0.9      no
Weighted Avg.    0.837     0.168      0.858     0.837     0.834      0.9

=== Confusion Matrix ===

  a  b   <-- classified as
 24  1 |  a = yes
  7 17 |  b = no
```

Figure 34: ROC Curve Mix Data



This curve was plotted under the ROC value of 0.9 and is examining the results of the 'yes' class within the dataset. The Y axis represents the True Positive Rate; in this case this was equal to **0.96**. The X axis represents the False Positive Rate; in this particular example this was equal to **0.292**.

The Precision and Recall contained in Figure 33 are fairly high once again. I have examined the dataset and have not identified any striking patterns that would suggest severe interference with the results of the classifier. Therefore, it is of my opinion that the accuracy figures are appropriate.

Based on the examination of this evenly split dataset, I feel that the accuracy of identifying toponyms within the Twitter posts examined are reported slightly too high, however still within a reasonable margin of error. The classifier is correctly labelling place name instances that have varying attribute values, suggesting that the output is correct to a certain degree. To ensure that the classifier was correctly labeling the place names, I matched the outputs of all instances labelled 'yes' to the number that existed within the dataset.

I feel that further development of the training data supplied to the classifier would produce a more conclusive result. The difficulty I have encountered when attempting to apply this to the system is keeping the Twitter posts intact, in order to retain the context of how the words are used within the post itself. To correctly classify the toponyms within the system, to a higher degree of accuracy, would require a more selective approach to the datasets used. This would allow for a greater balance of False Positives and True Positives within the classifier.

Areas to Improve in WEKA Results

I have evaluated the key areas which I believe need to be improved within the system to create a more specific and accurate result set.

The first area that needs improving (namely expanding) is the feature list. I feel that despite selecting several key features in which to classify the toponyms from, there is room for expansion. The first, clear attribute that needs to be expanded or established further is the 'Following Word' feature. This feature was implemented based on the results of the word frequency analysis performed throughout the project. The issue with this particular feature is that there are no, immediately obvious, following words that help define the context of a place name. Street names and addresses within the Twitter posts mostly became incorporated into the place name, resulting in entries such as "Park Street" becoming a singular entry within the token list. Furthermore, the features list could have been expanded, creating more areas in which to help the classifier find a toponym within the correct context. The feature list stayed at its current size as finding other, strong features, to implement was proved difficult when working with an unstructured format such as Twitter.

I feel that the overall datasets that were supplied to WEKA could also have been improved. A more selective process of instances (words) and overall tweets may have resulted in a more comprehensive result, possibly removing the uncertainties that were found when analysing the classifier accuracy. I chose to use the current datasets as I wanted to keep the tweets within the correct structure, allowing easier reference when attempting the geocoding process and evaluating the place names found.

Evaluation of Requirements

This section will review the functional requirements outlined at the start of the project and provide an overview on what was achieved and what was left to improve upon or implement. Some of the requirements found with the labels 'Could Have' or 'Should Have' did not make it into the final project.

Requirement:

The system is able to correctly distinguish between proper nouns. The system will identify that an initial capital letter for words within the database of tweets can refer to organisations, peoples names or simple incorrect grammar.

MoSCoW: 'Must Have'
Implemented: Yes

This requirement has been expanded since the initial creation of the requirements at the start of the project. The requirement should be changed to include the implementation of a feature list in order to identify toponyms as opposed to the mention of only proper noun recognition based on capital letters.

This requirement was the backbone of the project and I believe that it has been implemented to a good standard. The system is capable of identifying toponyms based on a pre-defined feature set, examining both the word itself and its neighbours. The WEKA analysis shows the successful retrieval of toponyms from a dataset and performance to a reasonable standard.

Requirement:

The System can develop appropriate datasets to use within the machine learning process.

MoSCoW: 'Must Have'
Implemented: Yes

As outlined in the Implementation and System Design sections of this project, the system implemented is able to produce ARFF files in the correct format to use within WEKA. The implementation of this requirement was necessary to help deal with the dataset sizes that I was working with. Manually creating ARFF files and applying a feature list to each instance would be time consuming and an example of poor preparation.

Requirement:

The system can appropriately, and correctly, index and geocode locations found within the Twitter database.

MoSCoW: 'Must Have'
Implemented: Partial

Unfortunatley, due to difficulties using the Python Weka Wrapper module and a large amount of project time spent creating the toponym recognition process, I was unable to fully implement the geocoding portion of this project. I have supplied the progress I have made within the implementation section and I believe that, if given more time, the implementation of this function would be complete.

My main focus during this project was to ensure that the toponym recognition was working correctly. I considered this to be the main focus of the project, as without the correct identification of place names, geocoding would have been impossible to perform.

Requirement:

The System is able to correctly identify vernacular / colloquial place names within the Twitter database.

MoSCoW: 'Should Have'
Implemented: Yes (Indirectly)

This requirement was achieved through the implementation of the toponym recognition function. Vernacular names are found within the datasets via the various features used to identify place names.

The vernacular names are not directly categorised with their appropriate relationship to the administrative names. However, providing the classifier is supplied with the instance within the dataset, it can be recognised as a place name within this system. I purposely did not include direct pattern matching to the feature list to assist with this requirement.

Requirement:

The system should be able to deal with datasets from other database formats. It should also be able to work with various text formats.

MoSCoW: 'Should Have'
Implemented: Yes

I approached this requirement by selecting the Python DB Module to communicate with the database system. Python DB allows access to many different database platforms, therefore the system is portable enough to be applied to any database protocols supported within the module.

Requirement:

The system can create mapped space via data gathered throughout the machine learning process. The mapped space will show borders that are often referred to as part of a town or city locally, but administratively are actually not part of that region.

MoSCoW: 'Could Have'
Implemented: No

This was a desired feature but due to the time constraints of the project and incomplete implementation of the geocoding function I was unable to implement this feature.

Conclusion

Overall, after reviewing the results from the toponym recognition process and achieved system requirements, I believe the project was a good example of the implementation of machine learning to find toponyms within social media data. By evaluating the WEKA results, it is clear that the toponyms *are* being located from the datasets provided, however, I feel that the feature list and approach to producing the data set is the ultimate weakness of the system.

The system suffered from many issues that, given more time and a greater understanding of the machine learning processes, I believe I could rectify. A recurring issue I found during the evaluation of the classifiers was that the system was focusing its results on the gazetteer attribute and not considering the weightings of the other features. I believe this is due to the dataset that I was using, many of the place names identified *were* present within the gazetteer list and therefore the attribute was often flagged with 'true'. When the classifier evaluated the training data, it frequently found that if the gazetteer attribute was set to true then the word was in fact a place name. When balanced against the high proportion of words found as being *not* a place name this led to an overall skewed result.

My main focus within this project was to ensure that the toponym recognition was as correct as possible. Due to the focus on this segment of the project, I found that the implementation of the geocoding system to be difficult within the remaining timeframe. With better time management within this project and a better understanding of the ways in which to approach the classification of the datasets, I believe that the geocoding functionality may have been completed.

Future Work

There are many opportunities to expand the system I have produced. The most obvious area for future development would be the full implementation of the geocoding feature, as this was considered a main project objective. Given the opportunity to further develop this system, I would focus on applying the trained classifier model on the appropriate datasets from within Python, rather than relying on the WEKA explorer. This process would allow me to then cross-reference the output with the co-ordinates found within the gazetteer.

A further area of development could include the addition of a wider array of social media sources. There are many popular social media sites that could provide data for the toponym recognition and geocoding process (Facebook). These data sources could also include the geocoding of photos uploaded to the social media sites (Flikr). This process would involve reviewing the metadata stored within the photograph and developing a system to extract relevant information. I believe that this would be a great opportunity to develop valuable, geocoding data.

I believe that the accuracy of the system would benefit from a larger feature list. For a future development of the system I believe that a more extensive feature list could be implemented, reviewing more standardised features found within plain text analysis (Length of word and tag pattern matching). I felt that given the time frame to produce the system, these features would not be necessary to produce a relatively accurate working system. However, based on the accuracy of the WEKA results, it would have been useful to have implemented more features.

The following requirement highlights a great opportunity for development within the system.

Requirement: The system can create mapped space via data gathered throughout the machine learning process. The mapped space will show borders that are often referred to as part of a town or city locally, but administratively are actually not part of that region.

I believe that implementation of this feature would have been an excellent example of the geocoding system working correctly, allowing the user to review the areas marked on a mapped space. Using boundaries for each grid reference it would also be possible to include clustering analysis to find the most popular areas referenced with the social media data.

Reflection and Learning

There are many changes that I would make if I was to work on this project again. I feel that I have developed a greater understanding and expanded my skill set in regards to machine learning. I feel I have also developed a variety of technical approaches to resolve issues experienced when working with a project of this size.

Reflection

I believe that my greatest weakness during this project was my understanding on how to implement the machine learning methods within the system I had created. I spent a lot of time attempting to create a system which managed the data to use within different processes, however, this ultimately led to difficulty with time when it approached the end of the project. I understood how I would approach the problems the main objectives of the project, though my lack of experience within this field resulted in what I consider a lower standard of work than I am capable of producing.

Upon review of the functionality of the system, I found several areas that I could have focused less upon, this would have created a larger amount of time to work on the implementation of the geocoding functionality and the overall system results. I feel that I have greatly increased my understanding of machine learning methods and how to approach the classification and training of data sets. However, I feel that there is still a large amount of development I could pursue within this area.

A greater time frame allocated to experimentation of the system outputs may have given me the further knowledge needed to develop a more functional and complete project. Upon review of the initial report written before the start of the main project I noticed that I had not allocated an appropriate time scale to the testing of the system. Allowing myself a larger time frame to test and experiment with the system results would have helped developed the system and allowed for a more expansive listing of results.

The system was designed to work mostly independently, with the majority of functions ultimately producing the ARFF files for training and evaluation of the chosen classifier. I feel that If I had structured the system differently, allowing for greater communication between the functions implemented, the production of data to train and test classifiers would have been quick. This would have resulted in more time to develop the geocoding portion of the system.

Learning

During the process of this project I have learnt a great deal about the systems used to correctly identified toponyms in an ambiguous dataset. At the start of the project I had made many assumptions about the way in which I would approach the development of the system. Many of my assumptions quickly became redundant, as I researched deeper into the problems regarding implementation and retrieval of the appropriate data. Generally, I

feel that this project has increased my technical skill set in regards to the WEKA environment and Python based programming.

The greatest learning experiences I have gained from this project are that of an administrative nature. Throughout the course of the degree process I have developed my programming skills from having zero knowledge to being able to develop appropriate systems. However, until this module, my experience with project management was somewhat limited. By attempting a project based on a subject that I had a limited knowledge of and developing a project plan on how to approach and ultimately create a system, has expanded my understanding of the project development lifecycle dramatically. I feel that given the opportunity to attempt a project of a similar size, I would be able to apply my knowledge gained from this project in order to create a more structured time frame to work within. This would allow for a more focused approach on the areas of this particular project that I feel show clear weaknesses in my understanding of appropriate time scales and planning.

When approaching projects in the future, I will allow more time for initial research and knowledge gathering before creating a project plan. I will also assign more realistic timescales to projects in future.

Reference List

Balaji, S. & Gelernter, J. 2013. *An algorithm for local geoparsing of microtext*. School of Computer Science, Carnegie Mellon University. New York: Springer Science+Business Media.

Gelernter, J. & Zhang, W. 2014. *Geocoding location expressions in Twitter messages: A preference learning method*. Available at: http://www.josis.org/index.php/josis/article/viewFile/170/129. [Accessed 22nd April 2016].

Hiscott, R. 2013. *The Beginner's Guide to the Hashtag*. Available at: http://mashable.com/2013/10/08/what-is-hashtag/#u07hwOBf4uqg. [Accessed 20th April 2016].

Habib, M. & Keulen, M. 2014. *Information Extraction for Social Media*. Available at: http://www.aclweb.org/anthology/W14-6202. [Accessed 20th April 2016].

Kemp, S. 2016. *Digital in 2016*. Available at: http://wearesocial.com/uk/special-reports/digital-in-2016. [Accessed 20th April 2016].

Leidner, J. L. & Lieberman, M. D. 2011. *Detecting Geographical References in the Form of Place Names and Associated Spatial Natural Language*. Available at: http://www.umiacs.umd.edu/~codepoet/pubs/recognition-special.pdf. [Accessed 22nd April 2016].

Mamat, A. & Mansouri, A. & Suriani, L. *Named Entity Recognition Approaches*. Available at: http://paper.ijcsns.org/07_book/200802/20080246.pdf. [Accessed 22nd April 2016].

Nadeau, D. & Sekine, S. 2007. *A survey of named entity recognition and classification*. Available at: http://nlp.cs.nyu.edu/sekine/papers/li07.pdf. [Accessed 23rd April 2016].

Overell, S. 2011. *The Problem of Place Name Ambiguity*. Department of Computing, Imperial College London. London.

Twitter. 2016. *Counting Characters*. Available at: https://dev.twitter.com/overview/api/counting-characters. [Accessed 20th April 2016].

Wolfram|Alpha. 2016. *Average English word length*. Available at: http://www.wolframalpha.com/input/?i=average+english+word+length. [Accessed 20th April 2016].