SELCHI : Travel Profiling

Technical Report · November 2013

DOI: 10.13140/RG.2.2.35156.91529

CITATIONS
0

READS
89

6 authors, including:

Nisansa de Silva
University of Moratuwa
62 PUBLICATIONS **348** CITATIONS
SEE PROFILE

Danaja Maldeniya
University of Michigan
16 PUBLICATIONS **50** CITATIONS
SEE PROFILE

Final Year Project Report

SELCHI

Travel Profiling

Department of Computer Science and Engineering
University of Moratuwa

Group Members

Chiran Chathuranga (090067P)

Eranga Mapa (090321P)

Lasitha Wattaladeniya (090545F)

Samith Dassanayake (090076R)

Project Supervisors

Mr. Nisansa de Silva

Mr. Danaja Maldeniya

Abstract

In present days, with the evolution social media, people has become addicted to it social media. Due to that large amount textual information related to human activities are available in social network repositories. If these data can be used in a proper way, the outcome would be very powerful. In this project we are planning to use those data to structure and update travel domain data. The basic problems are to identify the trending places within time periods and trending geo locations. A system where it suggests users about trending travel locations according to their preferences within given time period would be the final outcome of this project.

When considering the research, we have two major research areas. Main research area is domain specific information extraction from social networks. Other area is data mining. Under domain specific information extraction from social networks, we are referring the area of Natural Language Processing. Under that we are focused on POS tagging, Sentimental Analysis and Pattern-based extraction.

Acknowledgements

First and foremost we would like to thank our internal supervisor of this project. Mr. Nisansa De Silva, who linked us with this project which is under our areas of interests. He motivated and guided us on initiating this project.

Mr. Danaja Maldeniya from Codegen International, being our external supervisor, and Dr. Upali Kohomban from Codegen International helped us realizing the value of this idea and directed us on correct path. We would like to convey our gratitude to them.

Finally, we would like to thank our families and friends who were behind us and motivated us to perform well.

Table of Content

List of Figures

List of Tables

1. Introduction

1.1 Motivation

With the advancement of technology the way of interaction with people have changed, there is a lot of new technologies involved in day to day life of humans'. Some may say there are positive impacts and negative impacts to the society. It is not a secret that the technology has changed the normal human life. Technology has enabled different sort of ways to interact with other humans. Most importantly the information flow has come to a level where anyone can access anything from anywhere.

Currently one of the trending aspects of the technology is that to enable social networking everywhere. Apart from Mobile conversation, now there are new platforms where people can interact with other people. New concepts have been taken into this social networking domain and there are a lot of various applications. For an example Facebook, Twitter, Foursquare, Instagram and other applications. Statistics show how much people are active on social networks. For an example there are 400 million tweets per day [1].

Consider this example to get an idea about how social media has involved with human life. Nowadays when people think of going to a restaurant for a meal, what they first do is, they will put a status or tweet saying what they are going to do. In the meantime, other people can get into the conversation by commenting about their experiences so that users can get an idea about different standards of places. After a person selects a place to go and when he reaches that place he normally uses Foursquare and check-in there. Sometimes after ordering food or while having a conversation with others, he will take a snap through Instagram and share it in other social networks. Finally they can even comment about their satisfaction. This scenario clearly describes how social media has involved with human lives.

When billions of people shares their experience daily in social media there are enormous amount of information aggregate daily. This information are not static, it grows rapidly second by second, day by day and because of that it has higher value.

1.2 Idea

With the enormous amount of information lies in social media we can try to use it as a resource for our research project where we can make use of them in travel domain. What if we can build an application where it will use social data to determine trending places. From those places, it will suggest users what places they might be interested in going based on users' personal preferences and activities published in social media. In this application we are focusing on two main research problems.

The first problem is identifying and understanding the relevant information from unstructured social media feeds by using natural language processing. Generally social media feeds are not structured at all since social media networks never bother about structuring that information rather than interpreting them in real time. Apart from that they need to put an extra effort to structure those massive amounts of data. Our goal is to extract the travel related data among those large data sets from social feeds such as Facebook, twitter and Foursquare. To achieve our goal we need to use natural language processing (NLP) since all most all the social feeds are in natural language texts either tweet, Facebook status or any other feed. In this project we only focus on English language text feeds. The meaning of the text can be extracted by processing those texts. These processes become harder as people use slang or urban language while express their ideas in social media. Changing of slung language with the time is another issue which we have to face during this project. Several NLP techniques will be used to process social feeds and extract the meaning .Only after getting the idea of the texts, we can determine whether it is related to traveling or not. Next problem is converting extracted information into a Traveler and Destination Profiles through data mining and other statistical techniques into a formal Ontology.

1.3 Destination profiling

Using the information extracted from social networks we can decide, currently what places are most trending and also for what type of activity those places are famous. This is called location profiling.

2. Literature Review

2.1 Natural Language Processing

Natural Language Processing will be playing a major role in information extraction from social media under travel domain.

2.1.1 Parsing

A natural language parser is a program that understands the grammatical structure of sentences. For instance, which groups of words go together and which words are the subject or object of a verb. Probabilistic parsers use knowledge of language gained from hand-parsed sentences to try to produce the most likely analysis of new sentences [2]. These statistical parsers still make some mistakes, but commonly work rather well. There are different parsers developed for different languages. Since this project is only focused on English text data on social networks, we are only interested in English language parsers. Following are the parsing tools available.

2.1.1.1 Stanford parser

Stanford parser is a Java based typed dependency parser. Phrase structure tree can also be extracted at an intermediate step of the parsing. Development of phrase structure is done based on probabilistic context free grammar. These are necessarily context free grammar rules which have associated probabilities. Dependencies identified by the parser are triplets which consist of the name of the relation, governor and the dependent. A total of 53 typed dependencies are identified by the parser. Five different representations of typed dependencies are supported by the parser which is intended to give different level of information and different access methods to efficiently manipulate the output of the parser.

2.1.1.2 RelEx

RelEx [3] is a relation extracting tool which was initially developed by mainly targeting bioinformatics field. Getting details about some biological aspect was difficult, because large scaled biomedical publications need to refer for that. Also they needed a way to get relations between some interrelated biological aspects e.g. physical or regulatory interactions between genes and proteins. In case of both above reasons they need a structured way to manage their free text resources. RelEx is the methodology used for that by extracting relationship from free text. But now RelEx is a part of openCog‘s main project. It is creating an open source Artificial General Intelligence framework (A thinking machine).

2.1.2 Part of Speech Tagging (POS)

POS tagging is one of the main NLP tasks which can be used for many other NLP tasks such as Named Entity Recognition and Information Extraction. Basically it's the process of marking a particular word in a corpus (Text) depending on its definition and context. There are many standard POS tags that is been used in many applications over years. Most popular one is Penn Treebank POS tags. To illustrate let's consider following sentence "Little Brown Kitten Jumped from fence". After tagging, sentence will be "Little/RB Brow/NNP Kitten/NNP Jumped/NNP form/NN fence/NN" [4]. Tag NN stands for Noun, singular or mass and tag RB stands for Adverb according to Penn Treebank POS tag definitions.

Most common approach for POS tagging is using a machine learning approach. One of the major difficulties is a particular word in a word sequence can have more than one tag. Tagging depends on the context of the word. Two features can be considered to overcome this.

1. Some tag sequences are more likely than others. For instance, AT JJ NN is quite common, while AT JJ VBP is unlikely. ("A new book")

2. A word may have multiple possible POS, but some are more likely than others, e.g., "attack" is more often a noun than a verb.

Let's consider Hidden Markov Model approach.

An HMM have the following components:

- K states (e.g., tag types)
- Initial distribution $\pi \equiv (\pi 1. \ . \ . \ \pi K)$, a vector of size K.
- Emission probabilities p(x|z), where x is an output symbol (e.g., a word) and z is a state (e.g., a tag). p(x|z) can be a multinomial for each z. Note it is possible for different states to output the same symbol – this is the source of difficulty. Let $\varphi = \{p(x|z)\}$.
- state transition probabilities $p(zn = j|zn-1 = i) \equiv Aij$, where A is a $K \times K$ transition matrix. This is a first-order Markov assumption on the states.

The parameters of an HMM is $\theta = \{\pi, \varphi, A\}$. An HMM can be plotted as a transition diagram

(note it is not a graphical model! The nodes are not random variables). However, its graphical model is a linear chain on hidden nodes z1: N , with observed nodes x1:N .

Features can be extracted using training data. When adopting this to a specific domain we have to make sure to train the system using domain specific training data. With those feature, by using above Hidden Markov Model approach, we can identify tags for sentences which we are feeding.

For POS tagging there are many tools available now. Most of the major tools can be considered as results from long term researches under NLP. We have tried several popular tools such as Stanford NLP, NLTK toolkit and Opencog RelEx. But when considering the data from social network, we can't expect accurate results if we are going to use them as they are. Most of sentences coming from Facebook status updates or Tweets will not contain grammatical text and will contain misspelled or shorten words. Let's consider the word "Please". The word "please" will contain in training data tagged in various ways. But text from Social Feeds may contain "Please" as "Pls" or "Plz". But the system is not trained to handle these kinds of situations. Therefore we need take a different approach for this.

2.1.3 Sentiment Analysis

Sentiment analysis is the process of identifying the meaning of the sentence. It is actually an application of natural language processing. In our application after identifying travel related social text feeds, we need to identify whether they express a positive comment or a negative comment. Only after identifying the meaning of the text, we can use that data to log about travel locations. During the research we did, for sentiment analysis we found that the best tool for this requirement is the Natural language toolkit [5] which has been implemented using python.

There are many ways of implementing sentiment analysis using NLTK toolkit. The most popular implementation is to implement the analysis tool using a classifier. For an example let's say we have to analyze a set of sentences and output the result whether each sentence means positive or negative. In that case first we have to train the classifier with sample data. The training happens by feeding predefined sets of positive and negative sentences in large numbers to the classifier. In the sample set of data, there are some redundant data such as words with less than or equal to two letters. Before feeding to the classifier, the sample data has to be cleaned by removing those redundant words. After that, using the existing knowledge, classifier can predict the probability of a given sentence whether it is positive or negative. The probability calculation depends on the size of the knowledge base it has. When the size increases the accuracy increases.

In NLTK there are various methods to calculate the probability of a given sentence whether it is positive or negative. Those classifying methods are Naive Bayes Classifier, Maximum Entropy Classifier and Support Vector Machines.

2.1.4 Named Entity Recognition

Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, dates, and so on. Below are some of the more commonly used types of NEs.

NE type	**Example**
ORGANIZATION	Georgia-Pacific Corp., WHO
PERSON	Eddy Bonte, President Obama
LOCATION	Murray River, Mount Everest
DATE	June, 2008-06-29
TIME	two fifty a m, 1:30 p.m.
MONEY	175 million Canadian Dollars, GBP 10.40
PERCENT	twenty pct, 18.75 %
FACILITY	Washington Monument, Stonehenge
GPE	South East Asia, Midlothian

Table 1 NER Tag Meanings

These should be self-explanatory, except for "FACILITY" (human-made artifacts) in the domains of architecture and civil engineering and "GPE" (geo-political entities) such as city, state/province, and country. The goal of Named Entity Recognition (NER) system is to identify all textual mentions of the named entities. This can be broken down into two subtasks. Identifying the boundaries of the NE, and identifying its type. While named entity recognition is frequently a prelude to identifying relations in Information Extraction, it can also contribute to other tasks.

How do we go about identifying named entities? One option would be to look up each word in an appropriate list of names. For example, in the case of locations, we could use a gazetteer, or geographical dictionary, such as the Alexandria Gazetteer or the Getty Gazetteer. Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, dates, and so on.

2.1.5 Chunking

Chunking is also called shallow parsing and it's basically the identification of parts of speech and short phrases (like noun phrases). Part of speech tagging tells you whether words are nouns, verbs, adjectives and much more. But it doesn't give you any clue about the structure of the sentence or phrases in the sentence. Sometimes it's useful to have more information than just the parts of speech of words. But you don't need the full parse tree that you would get from parsing.
An example of when chunking might be preferable is Named Entity Recognition. In NER, your goal is to find named entities, which tend to be noun phrases (though aren't always). Chunking a text means segmenting it into an unstructured sequence of syntactically organized text units called "chunks", which display the range of the relations holding between their internal words. It is a textual unit of adjacent word tokens and, discontinuous chunks are not allowed. A chunk is always a maximal, non-recursive text unit. For example, a sentence such as "the interested photographer could always observe the animals visible in the rain forests" will be chunked as follows:

A. [the interested photographer]
B. [could always observe]
C. [the animals]
D. [visible]
E. [in the rain forests]

The sentence is segmented into five chunks. Each chunk includes a sequence of adjacent word tokens (a text substring) which are mutually related through dependency links of some specifiable kind. For example, in chunks A and B the following dependency chains are observed:

the <------watcher ------->interested
could <-----observe------->always
Simply, on the basis of the knowledge available to the chunker, it is impossible to state unambiguously what chunk relates to its neighboring chunks and what the nature of this relationship.

A chunk contains at most one potential governor, which is an element internal to the chunk on which neighboring chunks can syntactically depend either as arguments or adjuncts. A potential governor is always the rightmost element of a chunk. It represents a kind of syntactic "handle" externally available to other chunks for them to hang onto syntactically. The potential governors in A and C above are watcher and observe respectively. Incidentally, in the chunked sentence above, there is no other chunk external element depending on watcher, while observe takes the chunk [the animals] as a direct object. As mentioned above, all these inter-chunk dependencies are invisible to the chunker.

2.1.6 Training data selection

Each of the processes described above are dependent on the data set used to train them. For an example Named Entity Recognition depends on the set of domain words used to train it. It is really hard to get an accurate output from a NER trained to work in a different domain. Since in this application we are extracting data from social networks, we have to select the training data set in social network domain. The words and phrases used in social networks are somewhat different. Therefore we have to take that into consideration.

2.1.7 Language Detection

Language detection is needed when text processing as text processing tools are trained or configured for a specific language. Mainly they have different characters. Even two languages have same characters like Italian and English, still features are different from each other. To detect whether a given text or a phrase belongs to a particular language, we use most common words in that language [6]. They are called as stop words [7]**.** In English we have several stop words such as the, then, these, there, an, after, next, out, proper. Wordnet contains a list of stop words. By using that list we can get the stop word frequency from a given text and depending on that value we can determine the language.

2.1.8 Stemming

Stemming is reducing the derived words in a particular language. Stemming algorithm reduces a word to its root word. To illustrate let's take the word “Stemming”. We can have “Stemmer”, “Stemming” and “Stemmed”. We can reduce all these three words to just one word “Stemming”. For some languages, concept of stemming is not applicable.

Let’s look at a stemming algorithm. The variable part of a word that can be stemmed is the `ending', or `suffix'. Taking these endings off is called `suffix stripping' or `stemming', and the residual part is called the stem.

Another way of looking at endings and suffixes is to think of the suffix as being made up of a number of endings. For example, the French word confirmative can be suffixed with -atif, -e or, -f. Endings fall into two classes, grammatical and morphological [8]. The addition of -s in English to make a plural is an example of a grammatical ending. The word remains of the same type. There is usually only one dictionary entry for a word with all its various grammatical endings. Morphological endings create new types of word. In English -ise or -ize makes verbs from nouns (`demon', `demonize'), -ly makes adverbs from adjectives (`foolish', `foolishly'), and so on. Usually there are separate dictionary endings for these creations. By considering each of these cases proper algorithm can be implemented for stemming.

2.1.9 Natural language understanding

In our research project our scope is to analyze social feeds in English language. In order to accomplish that, we need to filter out feeds (tweets, Facebook feeds) which are in other languages and choose English feeds only. First off, to detect the language of a tweet, the updated API uses Twitter's machine language detection algorithm, which should help developers better serve its users (especially to an international crowd) with language specific aggregation, analysis, and duration of tweets. For instance, developers can manually pick out the types of languages that they'd like to have surface in their app while burying tweets from unspecified languages. In one scenario, the parameter "language=ja" would display only tweets in the Japanese language. If you're wondering where the "ja" abbreviation comes from, Twitter uses the two-letter code, which corresponds to certain languages.

JLangDetect is a pure Java implementation of a language detector. It provides a toolkit for training language recognition, and a simple implementation of a detector. JLangDetect is based on n-gram tokenization. Basically, texts are tokenized with different token sizes. For example, given the text "cat", n-gram tokenization for 1 to 3 token sizes will produce the following tokens

c, a, t, ca, at, cat

The idea is to tokenize a large set of documents in a given language and record token statistics. When you need to identify a language, then you'll tokenize it the same way, and you'll be able to score the input string against several token stats.

We could find an open source Java language detection library which calculates language probabilities from features of spelling (Naive Bayes with character n-gram). It generates language profiles from training corpus and returns candidates and their probabilities for the given text supporting 49 languages.

2.1.10 Anaphora resolution

While analyzing social feeds it is very common that people uses several sentences to express their idea instead of using single sentence. In this case we have to face a very important research problem called Anaphora resolution. Anaphora refers to the phenomenon where a word or phrase in a sentence is used to refer to an entity introduced earlier into the discourse, and the word or phrase is said to be an anaphor, or anaphoric.

In the following example, 1) and 2) are utterances; and together, they form a discourse.

1) We went to Hikkaduwa.
2) It was an awesome place for surfing.

As human, readers and listeners can quickly and unconsciously work out that the pronoun "It" in utterance 2) refers to "Hikkaduwa" in 1). The underlying process of how this is done is yet unclear especially when we encounter more complex sentences. An example involving Noun phrases

1a) I traveled around France twice.
1b) They were both wonderful.
2a) I took two trips around France.
2b) They were both wonderful.

It is hard to understand the reference of "they" here by a computed though it can be easily done by human mind. Since this type of understanding is still poorly implemented in software, automated anaphora resolution is currently an area of active research within the province of natural language processing. Accordingly, anaphora resolution is the process of identifying an anaphor's antecedent(s) thus to conceptually link it with its referent. Given that anaphora is resolved correctly, it can significantly augment the performance of downstream NLP applications. While there have been many approaches to anaphora resolution in the literature, the comparative evaluation of such algorithms has been hampered by a number of factors, including the lack of publicly available reference implementations.

RAP is an algorithm for identifying both inter-sentential and intra-sentential antecedents of third person pronouns (in their nominative, accusative or possessive case) and lexical anaphors (including reflexives -- pronouns like "myself", "yourself", etc. which are used when the complement of the verb is the same as the subject or to emphasize the subject or object, and reciprocals -- phrases like "each other" and "one another" showing that an action is two-way).

Using RAP algorithm, JavaRAP [9] resolves the anaphora by first extracting a list of all noun phrases in the input and a list of resolvable anaphors -- third person pronouns and lexical anaphors. Each anaphor is paired with noun phrases within a small sentence window. The resulting anaphor/antecedent-candidate pairs are then checked for agreement (gender, person and number) and filtered through the anaphor binding algorithm or syntactic filter, whichever applies.
These results align with our research project's sub area of extracting travel related data from social feeds using NLP. According to our perspective locations in the feeds can be traced with considerable accuracy where they mostly noted as pronouns while describing. Nevertheless It only handles third person pronouns.

Apart from RAP algorithm Expectation Maximization Algorithm (EM) has been used by several researchers as a solution for anaphora resolution. Some of them have presented a generative model of pronoun anaphora in which virtually all of the parameters are learned by expectation maximization. We find it of interest first as an example of one of the few tasks for which EM has been shown to be effective, and second as a useful program to be put in general use. Nevertheless the current system has several obvious limitations. It does not handle anaphora (antecedents occurring after the pronoun), only allows antecedents to be at most two sentences back, does not recognize that a conjoined NP can be the antecedent of a plural pronoun.

Among other options available for anaphora resolution, following three are more general in that they handle all NP anaphora. The GuiTAR system [10] is designed to work in an "off the shelf" fashion on general text GUITAR resolves pronouns using MARS pronoun resolution algorithm which filters candidate antecedents and then ranks them using morphosyntactic features. GuiTAR has been designed to be as independent as possible from the specifics of the modules used to extract certain information from the text – e.g., POS-taggers and parsers – and to be as modular as possible, allowing for the possibility of replacing specific components (e.g., the pronoun resolution component). These two goals are achieved in part by designing clear interfaces between the modules, in part by making the code modular. Due to a bug in version 3, GUITAR does not currently handle possessive pronouns. GUITAR also has an optional discourse new classification step, which cannot be used as it requires a discontinued Google search API.

OpenNLP uses a maximum-entropy classifier to rank potential antecedents for pronouns. However despite being the best-performing (on pronouns) of the existing systems, there is a remarkable lack of published information on its innards. NLTK also has also addressed the problem of anaphora resolution.

2.2 Ontology

2.2.1 Introduction

Ontologies are used to represent knowledge as a set of concepts within a domain and the relationships between pairs of concepts. It can be used to model a domain and to support reasoning about entities. Ontologies are consist with following components,

1. Individuals
2. Classes
3. Attributes
4. Relationships

2.2.1.1 Individuals

Individuals are the basic ground level components of ontology. Individuals may include concrete components such as Locations, People, Vehicles and planets, as well as abstract individuals such as Numbers and Words.

2.2.1.2 Classes

Classes – concepts that are also called type, sort, category, and kind – can be defined as an extension or an intention. According to an extensional definition, they are abstract groups, sets, or collections of objects. According to an intentional definition, they are abstract objects that are defined by values of aspects that are constraints for being member of the class. The first definition of class results in ontologies in which a class is a subclass of collection. The second definition of class results in ontologies in which collections and classes are more fundamentally different. Classes may classify individuals, other classes, or a combination of both.

2.2.1.2 Attributes

Objects in ontology can be described by relating them to other things, typically aspects or parts. These related things are often called attributes, although they may be independent things. Each attribute can be a class or an individual. The kind of object and the kind of attribute determine the kind of relation between them. A relation between an object and an attribute express a fact that is specific to the object to which it is related.

2.2.1.2 Relationships

Relationships (also known as relations) between objects in ontology specify how objects are related to other objects. Typically a relation is of a particular type (or class) that specifies in what sense the object is related to the other object in the ontology. Basically relationships are ways in which classes and individuals can be related to one another.

2.2.2 Why an Ontology?

In our TravelProfiling application, we have to store the extracted knowledge from social media, after that the storage should be capable of outputting the knowledge as we request. Before output the knowledge, the storages will have to verify whether it outputs the correct knowledge or not. Therefore the storages should have reasoning capabilities. After considering all the requirements for knowledge representation in TravelProfiling application, the best option is to use Ontology.
There are various tools available for create Ontologies. One of the best tools is Protégé [11], it supports modeling ontologies via web client or desktop client.

2.2.3 Protégé

Protégé is an open source platform that provides tools to construct domain models and knowledge base applications with ontologies. Protégé platform supports two main ways of modeling ontologies.

1. Protégé-frames editor: enables users to build and populate ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol (OKBC). In this model, ontology consists of a set of classes organized in a subsumption hierarchy to represent a domain's salient concepts, a set of slots associated to classes to describe their properties and relationships, and a set of instances of those classes - individual exemplars of the concepts that hold specific values for their properties.

2. Protégé-owl editor: enables users to build ontologies for the Semantic Web, in particular in the W3C's Web Ontology Language (OWL). "An OWL ontology may include descriptions of classes, properties and their instances. Given such ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms".

2.3 Social Data Mining

2.3.1 Introduction

Data mining researches carried out over the years has successfully produced numerous methods, tools, and algorithms for handling large amounts of data to solve real world problems. Traditional data mining has become a part of various application domains like bioinformatics, data warehousing, business intelligence, predictive analytics, decision support systems and social marketing. Primary objectives of the data mining process are to effectively handle large-scale data, extract actionable patterns, and gain insights. Social media is widely used for various purposes. Huge amounts of user-generated data exist in Social Networks and can be made available for data mining. Data mining of social media is a valuable research area as information that can mine from it can be directed to improve business intelligence to provide better services and develop innovative opportunities. For example, data mining techniques can help to detect implicit or hidden groups with specific preferences like surfing. Thereafter we can target that group and promote surfing equipment and destinations. Thus, mining social media is an awakening multi-disciplinary area where researchers from different backgrounds can make important contributions that matter for social media research and development. Our attempt to achieve an accurate solution for travel

domain specific social media mining tasks based on our experience and research. This section brings out how the data mining knowledge that we gathered can be adopted for this solution.

2.3.2 Data Mining

Data mining is a process of discovering useful knowledge from huge amounts of data [12]. Data mining also means knowledge discovery from data which describes the processes of extracting useful information from raw data. Indeed data itself has the process. Data mining process can be divided into three parts as preprocessing, data mining, and post processing. These steps need not be separate tasks and can be combined together. Data mining is an integral part of many related theories like statistics, machine learning, pattern recognition, database systems, visualization, data warehouse, and information retrieval. Data mining algorithms are classified as follows

2.3.2.1 Supervised

For supervised learning algorithms, a given data set is typically divided into two parts. Training and testing data sets with known class labels. Supervised algorithms build classification models from training data and use the learned models for prediction. To evaluate a classification model's performance, the model is applied to test data to obtain classification accuracy. Typical supervised learning methods include decision tree induction, k-nearest neighbors, Naive Bayes classification, and support vector machines.

2.3.2.2 Unsupervised

Unsupervised learning algorithms are designed for data without class labels. Clustering is a common example of unsupervised learning. For a given task, unsupervised learning algorithms build the model based on the similarity or dissimilarity between data objects. Similarity or dissimilarity between data objects can be measured using proximity measures including Euclidean distance, Minkowski distance, and Mahalanobis distance. Other proximity measures such as simple matching coefficient, Jaccard coefficient, cosine similarity, and Pearson's correlation can also be used to calculate similarity or dissimilarity between
The data objects. K-means, hierarchical clustering (agglomerative or partition methods), and density-based clustering are typical examples of unsupervised learning.

2.3.2.3 Semi-supervised

Semi-supervised learning algorithms are most applicable where there exist small amounts of labeled data and large amounts of unlabeled data. Two typical types of semi-supervised learning are semi-supervised classification and semi-supervised clustering. The semi-supervised classification uses labeled data to make classification and unlabeled data to refine the classification

boundaries further, and semi-supervised clustering uses labeled data to guide clustering. Co training is a representative semi-supervised learning algorithm. Active learning algorithms allow users to play an active role in the learning process via labeling. Typically, users are domain experts and their skills are adapted to label some data instances for which a machine learning algorithm are confident about its classification. Minimum marginal hyperplane and maximum curiosity are two popular active learning algorithms.

2.3.3 Mining Social Media

Social media is denied as a group of Internet-based applications that build on the ideological and technological foundations of Web 2.0 and that allow the creation and exchanges of user generated content. Social media is conglomerate of different types of social media sites including traditional media such as newspaper, radio, and television and nontraditional media such as Facebook, Twitter, etc. Following table gives characteristics of different types of social media content.

Type	Characteristics
Online social networking	Online social networks are Web-based services that allow individuals and communities to connect with real-world friends and acquaintances online. Users interact with each other through status updates, comments, media sharing, messages, etc. (e.g., Facebook, Myspace, LinkedIn).
Blogging	A blog is a journal-like website for users, aka bloggers, to contribute textual and multimedia content, arranged in reverse chronological order. Blogs are generally maintained by an individual or by a community (e.g., Hungton Post, Business Insider, Engadget)
Wikis	wiki is a collaborative editing environment that allow multiple users to develop Web pages (e.g., Wikipedia, Wikitravel, Wikihow)
Social news	Social news refers to the sharing and selection of news stories and articles by community of users (e.g., Digg, Slashdot, Reddit).
Media Sharing	Media sharing is an umbrella term that refers to the sharing of variety of media on the Web

	including video, audio, and photo (e.g., YouTube, Flickr, UstreamTV)
Opinion, reviews, and ratings	The primary function of such sites is to collect and publish user submitted content in the form of subjective commentary on existing products, services, entertainment, businesses, places, etc. Some of these sites also provide products reviews (e.g., Epinions, Yelp, Cnet)

Table 2 Social Media in Brief

Vast amounts of user-generated content are collected into social media sites every day. Amount of that is exponentially growing. When there are more users more data will be added to Social Media. Following graph will illustrate usage of social networks which is great evidence that states mining Social Media is valuable.

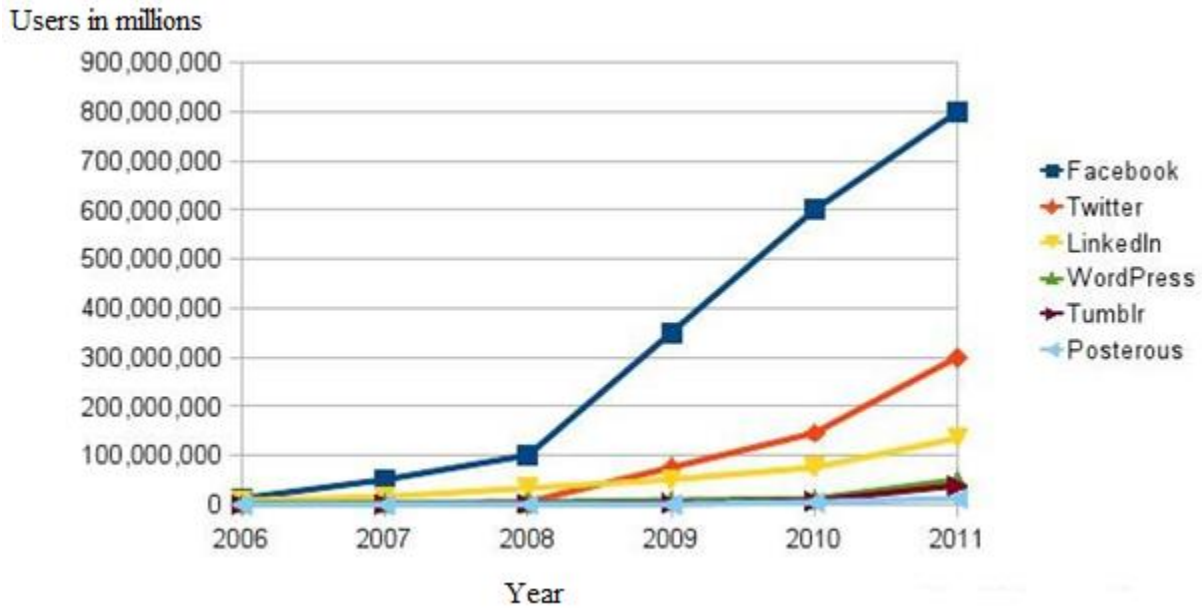


Figure 1 User Growth in Social Media

Hence, it is critical for producers, consumers, and service providers to figure out management and utility of massive user-generated data. Social media growth is driven by following challenges.

1. How can a user be heard?
2. Which source of information should a user use?
3. How can user experience be improved?

Answers to these questions are hidden in the social media data. These challenges bring opportunities for data miners to develop new algorithms and methods for social media.

Data generated on social media sites comprise a different style from conventional attribute-value data for classic data mining. Social media data are largely user-generated content on social media sites. Social media data are vast, noisy, distributed, unstructured, and dynamic. These characteristics pose challenges to data mining tasks to invent new efficient techniques and algorithms. For example, On average in one year, we will share 415 pieces of content on Facebook, we'll spend an average of about 23 minutes a day on Twitter, tweeting a total of around 15,795 tweets, we'll check in 563 times on Foursquare, upload 196 hours of video on YouTube, and send countless emails [13].

As mentioned above there are three things Data preprocessing, data mining, and post processing. It is for social media, it is mandatory to apply data preprocessing. Depending on social media platforms, social media data can often be very noisy. They also differ from writing styles. Twitter is having a different writing style when compared with Facebook. For our solution, NLP is responsible for preprocessing.

2.3.4 Community Analysis

A community is formed by individuals such that members within a group interact with each other more frequently than with those outside the group. Based on the context, a community is also referred to as a group, cluster, cohesive subgroup, or module. Communities can be observed via connections in social media because social media allows people to expand social networks online. Social media enables people to connect friends and new users of similar interests. In Social Media there exist large numbers of groups with similar interests implicitly. Explicit groups are formed with subscriptions. Therefore we don't need to involve Data Mining to recognize those. But when there are implicit groups, Data Mining comes to play.

Social media networks are highly dynamic. Communities can expand, shrink, or dissolve in dynamic networks. Community evolution aims to discover the patterns of a community over time with the presence of dynamic network interactions. More friends you have in a group, the more likely you are to join, and communities with cliques grow more slowly than those that are not tightly connected.

2.3.5 Opinion Mining

Textual information in the world can be broadly classified into two main categories, facts and opinions. Facts are objective statements about entities. Opinions are subjective statements that reflect personal preferences. Most of the existing researches on text information processing is been focused on mining and retrieval of factual information. As examples we can take information retrieval, Web search, and many other text mining and natural language processing tasks. Not much work has been carried out under the area of processing opinions until only recently. But still opinions are so important that whenever one need to make a decision one wants to hear others' opinions. Definition of opinion mining is as follows.

Given a set of evaluative text documents D that contain opinions (or sentiments) about an object, opinion mining aims to extract attributes and components of the object that have been commented on in each document $d \in D$ and to determine whether the comments are positive, negative or neutral [14].

2.3.5.1 Model of Opinion Mining

Generally, opinions can be expressed on any object. It can be a product, an organization, an individual or an event. The general term object is used to denote the entity that has been commented on. An object has a set of components and a set of attributes. Each component may also have its sub-components or sub concepts and its set of attributes, likewise. Thus, the object can be hierarchically decomposed based on the part-of relationship. As an example, let's say there is a mountain. Opinions can be expressed as this mountain is not good for cycling but better for hiking. Here, cycling and hiking are two concepts related to mountain. By considering this component and subcomponent, we can generate a tree.

In this tree, the root is the object itself. Each non-root node is a component or subcomponent of the object. Each link is a part-of relationship. Each node is associated with a set of attributes. An opinion can be expressed on any node and any attribute of the node. Following diagram illustrates an opinion tree in travel domain.

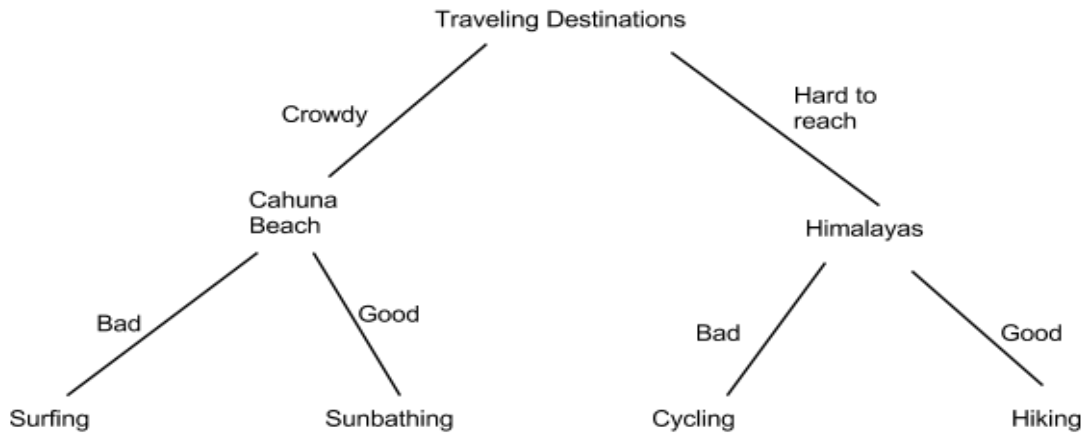


Figure 2 Opinion Tree on Travel Preferences

2.3.5.2 Model of Feature-Based Opinion Mining

An object O is represented with a finite set of features, F= {f1, f2, ..., fn}, which includes the object itself. Each feature fi∈F can be expressed with a finite set of words or phrases Wi, which are synonyms. That is, there is a set of corresponding synonym sets W= {W1, W2, ..., Wn} for the n features. In an evaluative document d which evaluates object O, an opinion holder j comments on a subset of the features Sj⊆F. For each feature fk∈ Sj that opinion holder j comments on, he/she chooses a word or phrase from Wk to describe the feature, and then expresses a positive, negative or neutral opinion on fk. The opinion mining task is to discover all these hidden pieces of information from a given evaluative document d. Mining output: Given an evaluative document d, the mining result is a set of quadruples. Each quadruple is denoted by (H, O, f, SO), where H is the opinion holder, O is the object, f is a feature of the object and
SO is the semantic orientation of the opinion expressed on feature f in a sentence of d. Neutral opinions are ignored in the output as they are not usually useful. Given a collection of evaluative documents
D containing opinions on an object, three main technical problems can be identified.

1. Extracting object features that have been commented on in each document d∈D.
2. Determining whether the opinions on the features are positive, negative or neutral.
3. Grouping synonyms of features (as different opinion holders may use different words or phrase to express the same feature).
4. Identifying patterns in opinion.

Solution for first two problems can be achieved by NLP. For rest of the problems, we need to use opinion mining.

2.3.6. Trend Mining.

The identification of trends has been an important activity in many application domains such as business intelligence, demography and epidemiology. Trend mining is concerned with the application of data mining techniques to extract trends from time Stamped data collections. For our purpose we are concerned about trend mining within the context of social networks. Mainly following two issues associated with it.

1. The large amount of data that has to be processed, social network datasets tend to be substantial.
2. Trend mining techniques typically generate large numbers of trends which are consequently difficult to analyze.

2.3.6.1 Trend Analyzing.

In order to analyze trends, we have to define what a trend is. With regard to the trend analysis based on time series, the analysis process consists of four major components or movements for characterizing time-series data. We can refer to the long-term movements that can be visualized by a trend curve. Based on the trend curve generated over quantitative data, we can identify time segments for those long-term movements that can have positive or negative trend values ("ups" and "downs" on a product). When analyzing social media texts, we are concentrating on trend indicating language structure. After preprocessing those texts and binding them to a structure, we can apply trending discovery algorithms effectively.

2.3.6.2 Non Symantec Trending Patterns

Since we analyze a given social media text corpus that is divided in trend classes, the very simple method for identifying trend patterns is the counting of the most frequent keywords. Different methods from text mining can be successfully applied in order to identify keywords or simple statements from the social media text corpus. However, we assume that not every keyword or statement extracted from the given trend class in text corpus is the trend-indicating one. Interesting feature is how to recognize whether given keywords or statements are trend-indicating or not. In particular we rely on the observation that there are characteristic words used in different domains describing the customer's opinion or sentiment (Good or bad). In an opinion extracted from Social Media, there should be an adjective which describes the quality and a noun where quality belongs. We can identify these pairs in the Social Media text corpus. Regardless, search for trend patterns requires more complex text analysis than the POS. We assume that we should refer POS tags to

structure them, and then by using their meaning we can use several approaches to get the accuracy improved.

2.3.6.2 Trend Pattern Ontology.

The non-semantic trend feature extraction provides a basis for a trend pattern structure. This can be useful for both, analyzing trend patterns from non- semantic level and creating a trend knowledge base that provides insight into the general characteristic of the trend patterns. A knowledge base can be represented as a classic ontology.

2.3.6.3 Learning Trends

Regarding different possibilities of learning methods from machine learning, firstly, we can use the supervised learning approach. Therefore we can work with text classes which don't have any common features. The texts with positive trend indicating patterns cannot belong to the neutral or negative trend category at the same time standard classification seems to be an appropriate learning form for the trend recognition problem, particularly where the trend classes' ranges are well separable. We can use Firstly Decision trees to visualize the learning model. Learning trends with decision trees means learning trend indicating language patterns after structuring.

3. Design

3.1. Overview Diagram

Following diagram represents an overview of our system. Data flow from social media APIs to common data store through specific filter for each social media to common data store. Data table formats in the table are for the comfort of trend analyzer. Trend analyzer will push its data stream to travel profiling service. It will expose an API which is used in web application (Dashboard) and for other users.

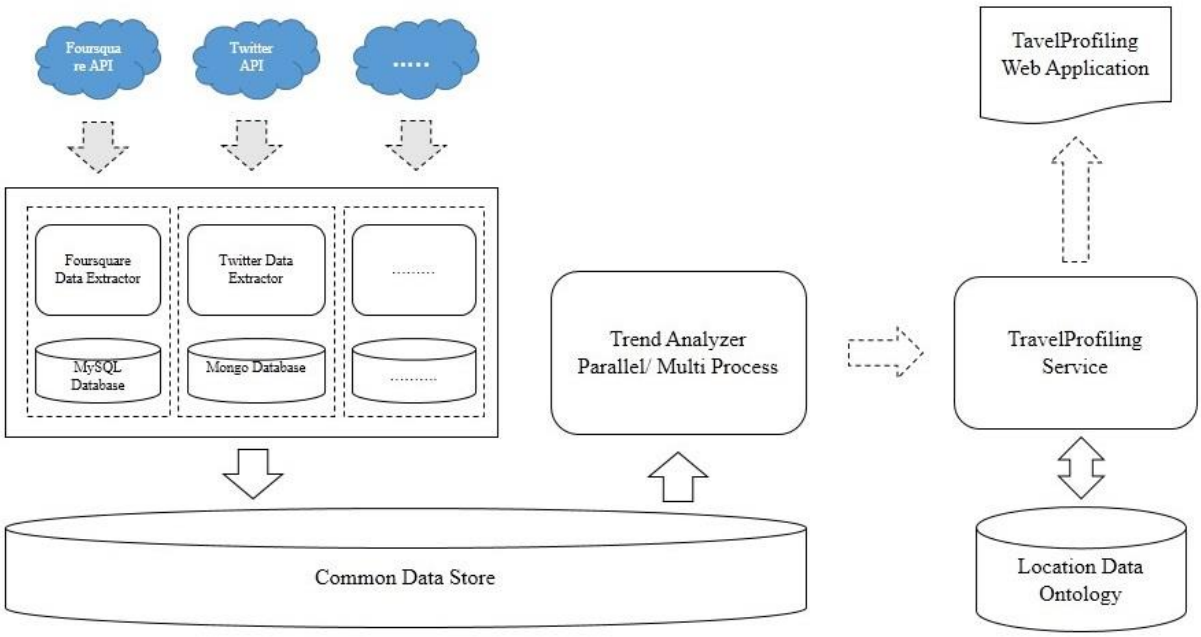


Figure 3 System Overview

3.2. Twitter Preprocessor Component

From Twitter API, public stream of tweets are retrieved through REST 1.1v Resources. The resource we used here is POST statuses/filter which returns public statuses that match one or more filter predicates. Multiple parameters may be specified which allows most clients to use a single connection to the Streaming API. Both GET and POST requests are supported, but GET requests with too many parameters may cause the request to be rejected for excessive URL length. Use a POST request to avoid long URLs. The default access level allows up to 400 track keywords, 5,000 follow user ids and 25 0.1-360 degree location boxes. Preprocessor component architecture diagram is given below.

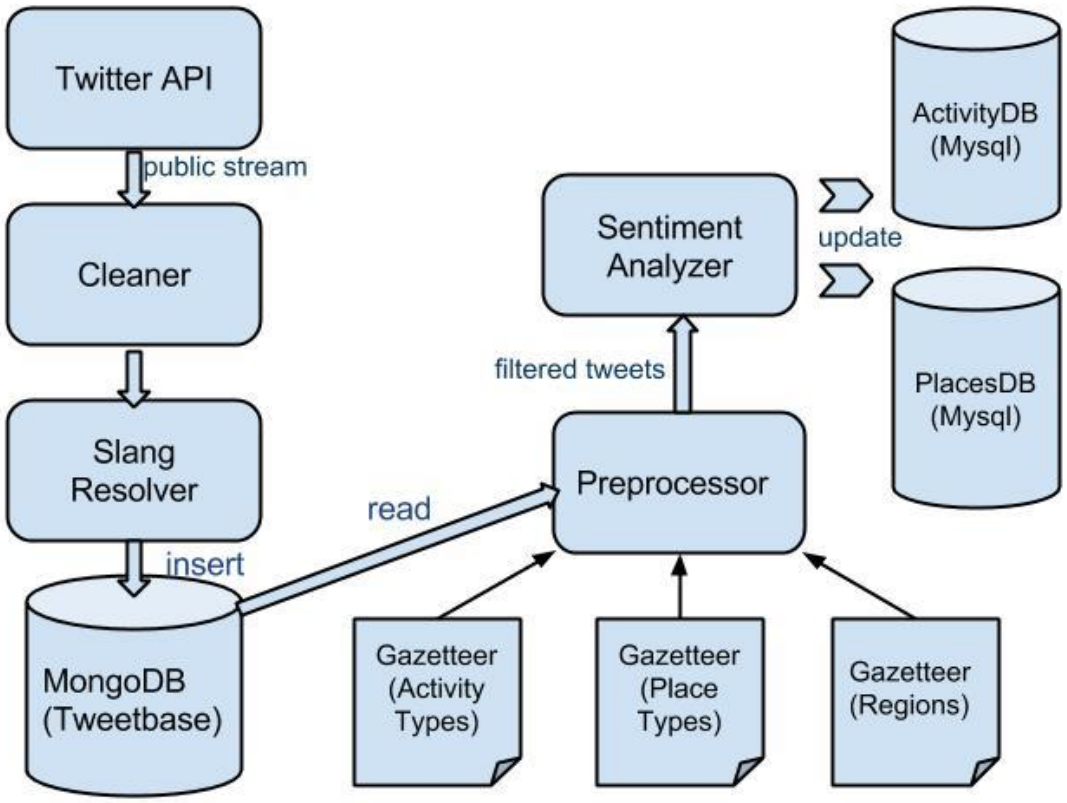


Figure 4 Twitter Preprocessor Architecture

Example Request

POST	**https://stream.twitter.com/1.1/statuses/filter.json**
POST Data	track=travel

3.2.1. Cleaner for Raw Tweets

Since tweets come with many unnecessary data which are not useful in our process has to be ignored using cleaner script.raw tweet response comes as in Appendix 1.

But we only care about text, and the created date, language is ignored since we check it with language likelihood which will be discussed in next chapter.

The cleaner removes its extracted text from retweets (RT), @mentions # signs and URLs which won't be useful in text processing. Furthermore we tokenize each tweet word for the convenience of further processing.

3.2.2. Slang word detection

Then check whether each token/word contains in the slang word gazetteer which maps slang word to the formal word. Our slang word gazetteer contains more than 1000 words including most common slangs used in social media. Internet Slang Dictionary & Translator (No slang) and Urban Dictionary is used to populated the gazetteer. Before check with slang words we first check each word with Standard English dictionary. All the words which don't have hits with that particular English dictionary are checked with slang word gazetteer. We use Enchant spell checking library which appears to be a generic spell checking library and we can request dictionaries from it, ask if a word is correctly spelled, get corrections for a misspelled word in order to achieve above mentioned task. Finally if the word exists in slang list it is replaced with the word with its formal meaning.

3.2.3. MongoDB Storage

Then cleaned and slang free tweets are stored in a MongoDB. Text and the created date will be inserted. First we thought of keeping track of users in order to consider user preferences and give suggestions. But we limited our scope and decided not to do user profiling. Therefore each tweet is stored in Mongo database with attributes text and date. In order to make our life easy in trend analysis using time series analysis we set the date such that it is represented from the Monday of that particular week. Since we are giving trends in weekly basis. iso week module in python is used to accomplish that task.

The strength of MongoDB is in its unstructured collection of data and so MongoDB doesn’t provide joins to collection various of data but because of its dynamic structure, so the child data should be collected in every row of data in the parent table. MongoDB has good insert speed than Mysql which has better speed for reading. We thought of using MongoDB to store tweets which has unstructured collection .MongoDB also has good insert speed.

3.2.4. Extract travel related tweets

Now we have the most important part which s to extract travel related tweets from the MongoDB collection. We read the tweets one by one and process it. Here each tweet is popped and kept no more in database after processing. By this we were able to save the space, not keeping redundant data.we also had to encode the data which was stored in unicode format. When database has no more data to processed by another preprocessing component Mongo reader has to sleep till data is available by MongoDB.

3.2.4. Using Gazetteers

While preprocessing we refer several gazetteers including region name gazetteer, place type gazetteer and activity type gazetteer. They were generated using Travel wiki data dumps which is the same source used to populate the ontology with activity types place types and regions. We first attempted to use POS (Part of Speech) taggers, NER (Named Entity Recognition) taggers to tag the tweets and distinguish particular patterns and track travel related tweets. But Tweets exemplify the phrase "language in the wild." Users abbreviate words, freely use web slang ("OMG", "SMH", "LOL", etc.), capitalize unpredictably, and frequently misspell words. Complete, well-formed sentences are also not the norm. Named entity recognizers (NER) are usually trained on traditional corpora such as newswire reports that represent much more formal uses of written language. Obtaining reliable annotations of text for training NER taggers is expensive and difficult even for traditional corpora, much less noisy social media text. Domain shift issues also exist for text corpora, making the challenge of using supervised techniques even greater. Robust unsupervised techniques, are required that can work in many cases without the requirement of training data.

Extracting entity mentions referencing a location is only a first step, however, and techniques for disambiguating location mentions, or toponyms, are needed. Most approaches for toponym disambiguation rely on the context of the mention in relation other mentions in a document. Such context may not be present in the very short documents found on Twitter. Some context may be available across a user's tweet history, but such data may not be available. Other techniques, are needed for this data stream. Therefore we thought of using aforesaid set of gazetteers. Each tweet text is tokenized and checked whether it contains a region in our region gazetteer which contains more than 30,000 regions at the first place. Rest of the tweets are ignored. If it hit a region then only we go further and check whether it contains a activity type or place type using relevant gazetteers. If it contains either of each or both then it is send to sentiment analyzer. While checking for matches tweets may contain misspelled words or phonetically similar words. Therefore we give a score using edit distance and phonetic similarity. Candidate which has the most score is taken. Furthermore place type can be unigram, bigram trigram or etc. We consider up to trigram. Recursively we check the longest match separating each sentence to bigrams and trigrams. fuzzywuzzy and difflib python libraries are used to set scores according to edit distance and phonetic similarity.

3.2.5. Mysql database for data storage

After getting a sentiment value from sentiment analysis each data associated with each tweet has to be inserted to database. For the convenience we use separate two Mysql databases as

- Activity DB Structure

Activity_Type	Date	frequency_twitter	frequency_foursquare
excursion	2013-09-02	0	178.73
trekking	2013-09-02	0	21.23
picnic	2013-09-02	0	230.82
horse riding	2013-09-02	0	105.74
camel riding	2013-09-02	0	69.34
golf	2013-09-02	0	40.46
fishing	2013-09-02	0	12.38
bowling	2013-09-02	0	239.82
bird watching	2013-09-02	0	68.4
siteseeing	2013-09-02	0	131.1

Table 3 Activity DB Structure

- Places DB Structure

Place_Type	Date	frequency_twitter	frequency_foursquare
airport	2013-09-02	0	100.18
amusement park	2013-09-02	0	165.46
animal park	2013-09-02	0	140.19
aquarium	2013-09-02	0	163.85
arena	2013-09-02	0	195.92
beach	2013-09-02	0	73.53
casino	2013-09-02	0	30.99
children park	2013-09-02	0	91.1
cinema	2013-09-02	0	144.42
club	2013-09-02	0	11.11

Table 4 Places DB Structure

If a table is not exist for particular region it is created at preprocessing time and update the entries. If a duplicate entry found twitter frequency is increased or decreased by the sentiment score value. Primary key of Activity DB would be Activity Type and Date while the primary key of PlacesDB would be Place_Type and Date. Then each table is read on request of the time series analyzer.First all the tables which have data is tracked and unique activity types and place types from each region-tables are listed. Then it selects all entries for each activity type for each region-tables and each place type for each region-tables on request. Finally listed result is sent to the time series analyzer at the end of each week. MySQL is used since it has more reading speed than the MongoDB. FourSquare data is also entered by Foursquare data extractor module appropriately to the same Mysql database.

3.3.Foursquare Data Collection Component

For foursquare data collection, we are consuming foursquare limited API strategically using this component. Following is the design diagram for Foursquare Data Collector. Since we need to run this task weekly to collect data, we have deployed this whole component in Red Hat Cloud [15].

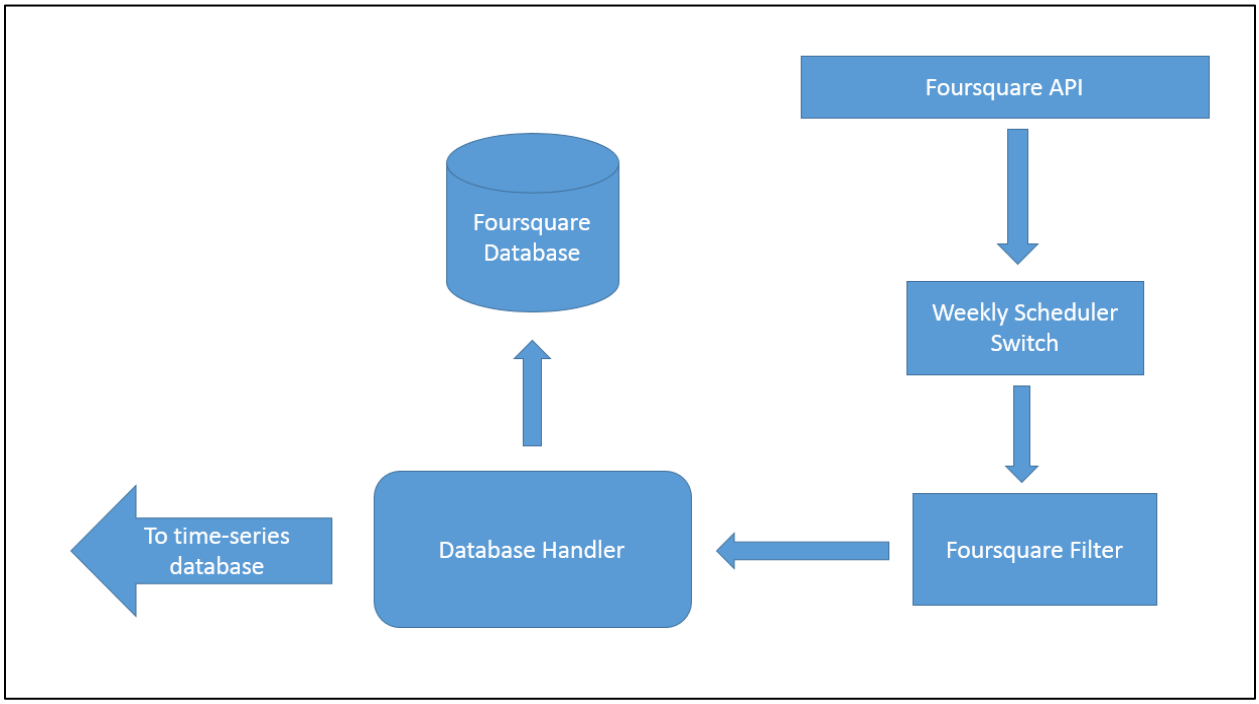


Figure 5 Foursquare Data Collector

3.3.1 Foursquare API

For our domain we are only concerned on travel destinations. Thus we are refereeing Foursquare Venues Platform(API) [16]. The Foursquare Venues Platform makes it easy to build location information into applications without requiring deeper Foursquare integration or authentication. Select partner mappings are retained in Foursquare's database and made available to other developers, making it easy for third parties to link to them.

The Venues Database lets us to use Foursquare as their location layer. We can use their API as the interface you use to interact with it. The API lets us search their database and find information including tips, photos, check-in counts, and here now. Searches can be done near a point or through a whole city, and they can be restricted to trending or recommended places. The platform offers all of this without requiring end user authentication and is available at high rate limits. Their normal limit is 5000 requests per hour. But for our purpose we wanted a higher rate. Thus we sent them a request to increase the limit and meanwhile we deployed this system strategically to handle API limits and collect data. The platform enables a range of applications that previously required developers to contract with data providers and maintain a geospatial database. Foursquare's results are constantly updated by their users, and they're ranked by social relevance.

3.3.2 Weekly Scheduler Switch

This component get the use of Red Hat cloud cron job scheduling [17]. After we implement our component, we configured our Red Hat Cloud gear to launch the data collection process on Monday on every week.

3.3.3 Foursquare Filter

This filter is used to filter out Venue data retrieved from Foursquare Venues Platform. We are comparing venues name and their categories against ontology tags. For matching ontology tags with foursquare categories, we specifically created a semantic mapping. As an example in our ontology we have Bowling Spot as a place type. But in Foursquare, they provide Bowling Alley. So we had to create a mapping for that.

3.3.4 Database Handler

We will be using a Database Handler to store data to two databases. One database is dedicated only for foursquare data. We will be saving weekly records related to foursquare venues for each activity. Following table shows a set of records that we collected from foursquare for a week.

id	time	week_date	region	name	onto_tag	lat	lng	category_id	category_name	checkins_count
4bb84cac98c7ef3b55053102	2013-11-25 05:01:06	2013-11-18	TOKYO	TOKYO SKY TREE BEST VIEW POINT	Scenic Natural Feature	35.7095	139.812	4bf58dd8d48988d165941735	Scenic Lookout	843
4c88c655ed0aa14399c5abf3	2013-11-25 05:01:07	2013-11-18	TOKYO	Pond at Kyu Yasuda Garden	Beach	35.6984	139.794	4bf58dd8d48988d1e2941735	Beach	19
4d2b297bb818a35d539dad8a	2013-11-25 05:01:07	2013-11-18	TOKYO	The BEACH	Beach	35.641	139.585	4bf58dd8d48988d1f2941735	Sporting Goods Shop	4
50ad6dfee4b01d0e29f42fae	2013-11-25 05:01:07	2013-11-18	TOKYO	The coast	Coast	35.7595	139.474	none	none	0
4f366a78e4b0d26b9fdfa588	2013-11-25 05:01:07	2013-11-18	TOKYO	BAR WEST COAST	Coast	35.9779	139.754	none	none	5
4f4750d0e4b0e4755b85829f	2013-11-25 05:01:08	2013-11-18	TOKYO	Fountain Plaza	Fountain	35.6259	139.778	none	none	3
51595e84e4b00f5fa65f807a	2013-11-25 05:01:11	2013-11-18	TOKYO	Port of Yokohama	Harbor	35.442	139.67	4bf58dd8d48988d1e0941735	Harbor / Marina	57
4b5baaa4f964a520bc0e29e3	2013-11-25 05:01:12	2013-11-18	TOKYO	TOYOTA MEGA WEB	Theme Park	35.6261	139.781	4bf58dd8d48988d182941735	Theme Park	5410
4f3ab175e4b05505712b7d5a	2013-11-25 05:01:12	2013-11-18	TOKYO	Park Luxe	Park	35.6994	139.759	none	none	0
5252344411d276269e7f3fc3	2013-11-25 05:01:12	2013-11-18	TOKYO	Odaiba park	Park	35.6821	139.737	none	none	0
4fb7643de4b0abf74f054259	2013-11-25 05:01:13	2013-11-18	TOKYO	Scating Plaza	Skiing Facility	35.6973	139.704	4bf58dd8d48988d1e9941735	Ski Area	2

Table 5 Foursquare Venue Data Sample

Then we are reading the table and updating common shared time-series table. Then both foursquare and Tweeter values will appear in the same tables in common format which makes it easier to feed to sentiment analyzing and times rise analysis components.

3.4.Sentiment Analyzer

Find the subjectivity and the sentiment value of a given tweet.

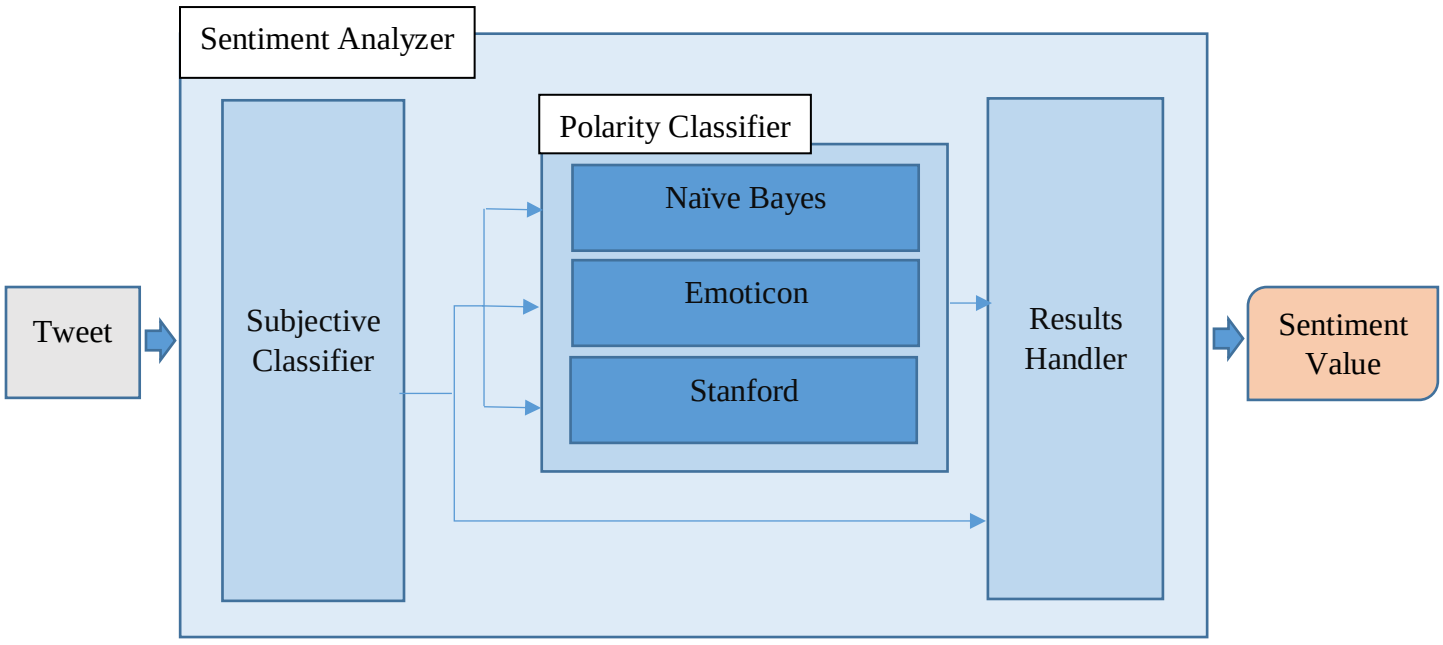


Figure 6 Sentiment Analyzer Overview

3.4.1. Subjective Classifier

The subjectivity classifier determines whether the text is objective or subjective.

3.4.2. Polarity Classifier

Polarity classifier contains with three polarity classifying models.

3.4.2.1. Naïve Bayes

Naïve Bayes classifier for polarity detection

3.4.2.2. Emoticon

Classifier which determines the polarity based on emoticons present in the tweet

3.4.2.3. Stanford Classifier

Stanford Sentiment classifier based on neural network

3.4.3. Results Handler

Determines the final sentiment value for the text, based on output from the classifiers.

3.5. Trend Analyzer

Forecast the future (next week and month) frequencies of twitter and foursquare data.

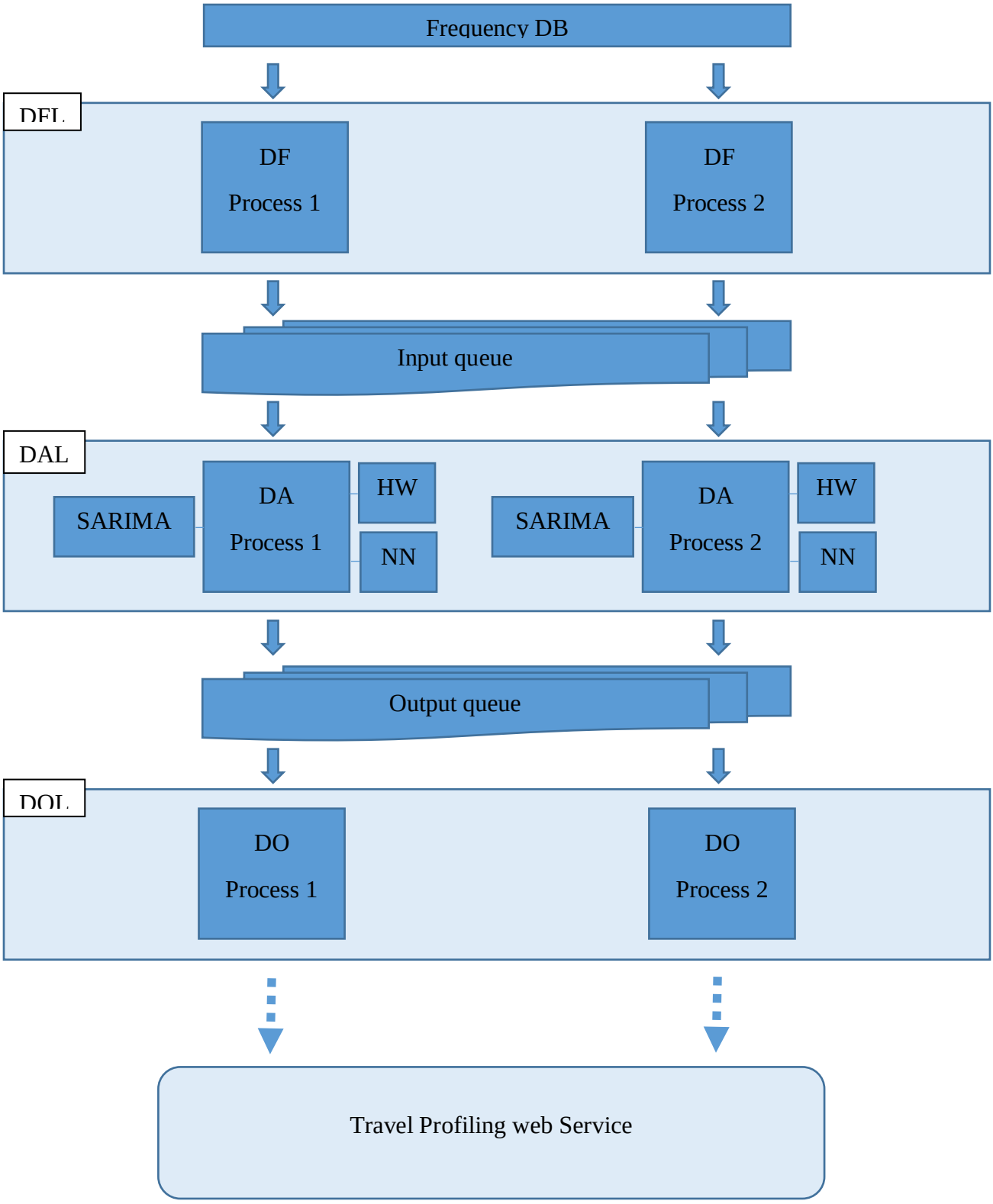


Figure 7 Trend Analyzer Overview

3.5.1. Data Fetching Layer (DFL)

Handles data fetching from database. Fetched data will fed in into input queue time series analysis. Contains with parallel processes to improve accuracy

3.5.2. Data Analysis Layer (DAL)

Analyze the fetch data using time series analysis models. Three time series analysis models are used.

3.5.2.1. Holt-Winters Model (HW)

Statistical method for time series analysis.

3.5.2.2. Seasonal Autoregressive Integrated Moving Average (SARIMA)

Statistical method which aim to describe the autocorrelations in the data.

3.5.2.3. Neural Network (NN)

Neural network for forecast future values based on past time series data.

3.5.3. Data Output Layer (DOL)

Update the ontology with forecasted future values using Travel profiling web service

3.5.4. Input/Output Queues

Shared FIFO queue for communication between adjacent layers

3.6. Travel Profiling Service

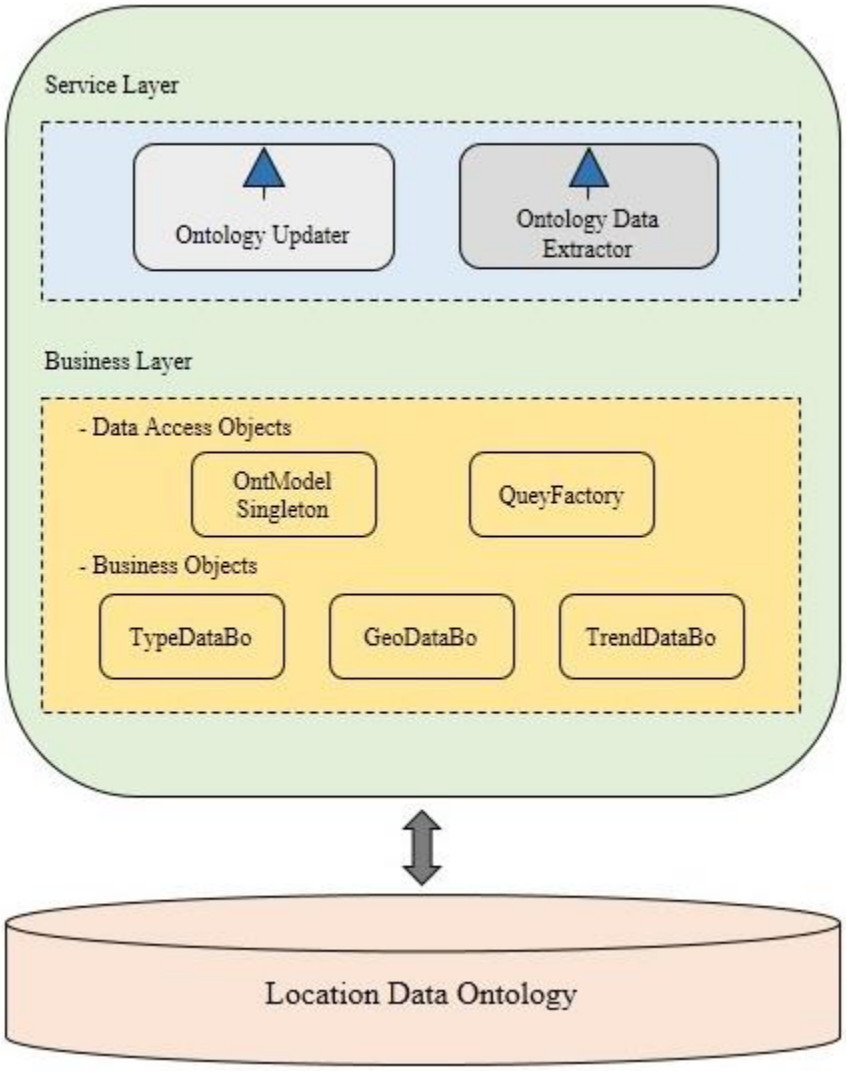


Figure 8 Travel Profiling Web Service Overview

Travel Profiling Service is used to update the Ontology and query results from it. It was developed as a RESTFUL [18] service because the libraries and technologies which were used to implement the service are based on Java programing language and the component's which are responsible to collect information from social media is written using Python. Communication between python and Java also better to implement via a web service. One of another reasons to implement this as a web service is that the third party applications can access the application easily.

3.6.1. Location Data Ontology

This is the data store that we used to store the knowledge information that we extracted from social media. The Data Store is an OWL [19] file which was written in RDF/XML [20] syntax. Once the application is started the OWL file is read in the memory and it create an in memory model, which will be later used to query and update the ontology. The in memory model must be write back to the file to persist the changes made by the application.

3.6.2. Business Layer

Business layer has two main parts, Business Objects and Data Access Objects.

- Business Objects
  Business Objects is used to transfer data within the service and as a return object.

- Data Access Objects

  OntoModelSingleton - This object is used to generate singleton Ontology Model to all the objects which required the Ontology Model.

  QueryFactory – This is used to query data from Ontology. We used SPARQL [21] as the query language.

3.6.3. Service Layer

This layer has two interfaces, Ontology Updater and Ontology Data Extractor. "Trend Analyzer" component uses the services in the Ontology Updater interface to update the Ontology. Third Party application and the TravelProfiling web application uses the methods in Ontology Data Extractor interface to query data.

4. Implementation

4.1 Collecting social media information - Twitter

Main intention of our research project is to collect social media information which is related to traveling. Therefore as the first step of the implementation phase we thought of targeting the twitter social network to gather information. Since Twitter allows getting public streams to users it aligned best with our requirements. Such public streaming mechanism has not yet been introduced by Facebook. The set of streaming APIs offered by Twitter give developers low latency access to Twitter's global stream of Tweet data. A proper implementation of a streaming client will be pushed messages indicating Tweets and other events have occurred, without any of the overhead associated with polling a REST endpoint.
Twitter offers several streaming endpoints, each customized to certain use cases.

Public Streams

Streams of the public data flowing through Twitter. Suitable for following specific users or topics, and data mining.

User Stream

Single-user streams, containing roughly all of the data corresponding with a single user's view of Twitter.

Site Stream

The multi-user version of user streams. Site streams are intended for servers which must connect to Twitter on behalf of many users.

But we only considered about public stream which suits best for our purpose.
code for maintaining the Streaming connection is typically run in a process separate from the process which handles HTTP requests

The streaming process gets the input Tweets and performs any parsing, filtering, and/or aggregation needed before storing the result to a data store. The HTTP handling process queries the data store for results in response to user requests.

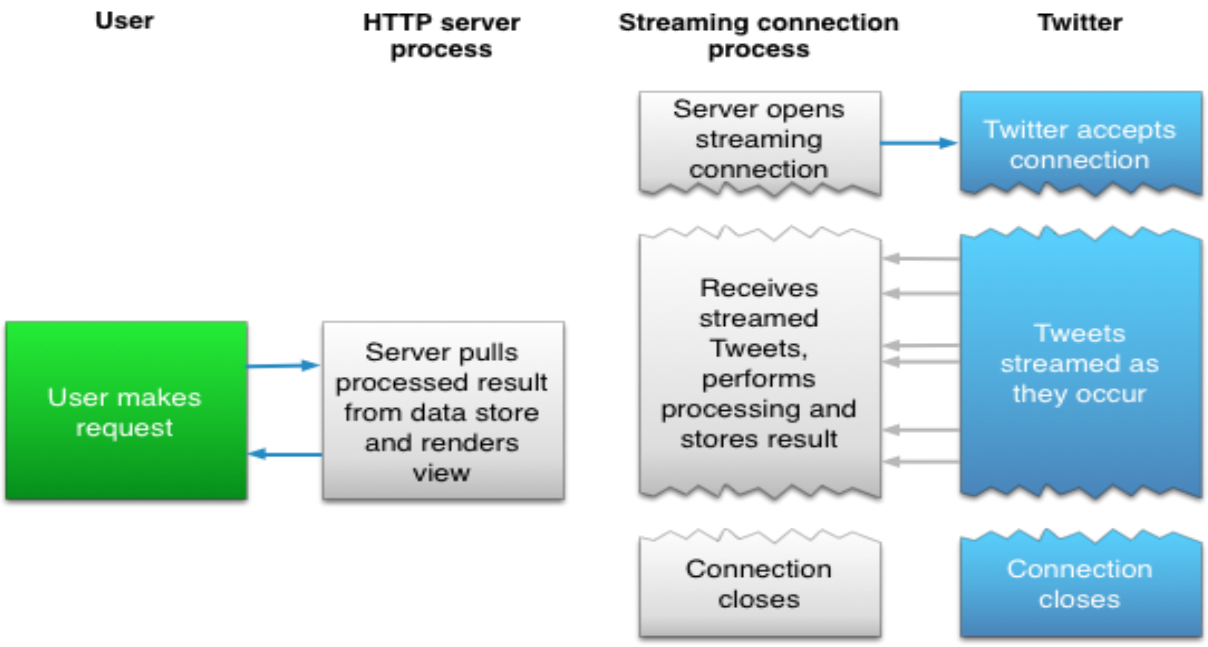


Figure 9 Twitter Streaming API

Establishing a connection to the streaming APIs means making a very long lived HTTP request, and parsing the response incrementally.

4.1.1 Authentication Problems faced

We started to collect tweets using twitter public from February 2013 to MongoDB with the help of server machine provided by the department. But suddenly in March we found that problem has occurred with authentication and data stream had been blocked. We could find that the old Twitter API had been stopped working on March 5th. There were several changes in Auth protocol. The new API required OAuth for all requests, even for searches. OAuth is an authentication protocol that allows users to approve application to act on their behalf without sharing their password.

4.1.2 Application-user authentication

This is the most common form of resource authentication in Twitter's OAuth 1.0A implementation to date. Your signed request both identifies your application's identity in addition to the identity accompanying granted permissions of the end-user you're making API calls on behalf of, represented by the user's access token.

4.1.3 Application-only authentication

Application-only authentication is a form of authentication where your application makes API requests on its own behalf, without a user context. API calls are still rate limited per API method, but the pool each method draws from belongs to your entire application at large, rather than from a per-user limit. API methods that support this form of authentication will contain two rate limits in their documentation, one that is per user (for application-user authentication) and the other is per app (for this form of application-only authentication). Not all API methods support application-only authentication

We used Application user authentication and modified our code such that we can access the public data stream.

```
API_ENDPOINT_URL = 'https://stream.twitter.com/1.1/statuses/filter.json'
USER_AGENT = 'TwitterStream 1.0'


OAUTH_KEYS = {'consumer_key': 'xxxxxxxxxxxxxxxxxxx',
              'consumer_secret': 'xxxxxxxxxxxxxxxxxxxxxxxxx',
              'access_token_key': 'xxxxxxxxxxxxxxxxxxxxxxx',
              'access_token_secret': 'xxxxxxxxxxxxxxxxxxxxxxx'}


POST_PARAMS = {'include_entities': 0,
               'stall_warning': 'true',
               'track': 'travel'}

class TwitterStream:
    def __init__(self, timeout=False):

        self.oauth_token = oauth.Token(key=OAUTH_KEYS['access_token_key'], secret=OAUTH_KEYS['access_token_secret'])
        self.oauth_consumer = oauth.Consumer(key=OAUTH_KEYS['consumer_key'], secret=OAUTH_KEYS['consumer_secret'])
        self.conn = None
        self.buffer = ''
        self.timeout = timeout
        self.setup_connection()
        self.cleaner = Cleaner()
```

Figure 10 Twitter 1.1 API Authentication Protocol

While retrieving the tweets we only considered about the following four attributes.

1. User name
2. Language
3. Text
4. Time Stamp

We only deal with English language tweets within our scope. But from the attribute entailed in tweet only gives the user preferred language. We can't guarantee that all English users always put English tweets. On the other hand other language users may also have tweeted in English which we must take into our account while collecting travel related data. Therefore we thought of distinguishing English tweets using the text.

Text is extracted tokenized and checked with NLTK stop word corpus which language has most matching stop words after sorting. We filter only the language likelihood with English and rest of the tweets are rejected.

```
def get_language_likelihood(input_text):
        """Return a dictionary of languages and their likelihood of being the
        natural language of the input text
        """

        input_text = input_text.lower()
        input_words = wordpunct_tokenize(input_text)

        language_likelihood = {}

        for language in stopwords._fileids:
            language_likelihood[language] = len(set(input_words) &
                    set(stopwords.words(language)))

        return language_likelihood

def get_language(input_text):
        """Return the most likely language of the given text
        """

        likelihoods = get_language_likelihood(input_text)
        return sorted(likelihoods, key=likelihoods.get, reverse=True)[0]
```

Figure 11 Detect Language Likelihood

Twitter provides filters for their streams by setting up "track parameters" as bunch of keywords. A comma-separated list of phrases which will be used to determine what Tweets will be delivered on the stream. A phrase may be one or more terms separated by spaces, and a phrase will match if all of the terms in the phrase are present in the Tweet, regardless of order and ignoring case. By this model, you can think of commas as logical ORs, while spaces are equivalent to logical ANDs (e.g. 'the twitter' is the AND twitter, and 'the, twitter' is the OR twitter).

The text of the Tweet and some entity fields are considered for matches. Specifically, the text attribute of the Tweet,expanded_url and display_url for links and media, text for hashtags, and screen_name for user mentions are checked for matches.

Keywords are taken from gazetteers containing activities and places. Gazetteers will be explained later in this report. Before feeding into our pipeline, Twitter texts need to be preprocessed. There are various types of entities in a tweet which needs to be cleared. Following is an example tweet.

"RT @LanaParrilla: Will u b there? http://t.co/sey0eq9MG9 #thegracies YES!!! :) I will b !!"

In above tweet, you can identify the URLs, hash tags, at mentions, emotions and some punctuation symbols mixed with words. With those items in the scene, it is hard to identify slang terms. Hence we implemented a cleaning component based on python regex engine to eliminate above mentioned entities. Then we tokenized tweets using NLTK word punkt tokenizer.

With some words, people use extra characters to accentuate them. People use word ‘love’ as ‘loveeeeee’ and words like ‘please’ as ‘plzzzzzzz’. In English language, it’s rare to find words with same character three times repeated consecutively. Hence to resolve those words, we reduced characters to 2 from them if there are more than 2 repetitive characters. Then we search in both dictionary and slang list for a hit. If it does not exist, we again reduced repetitive characters to 1 and searched again. Again if it’s not there, we forwarded the word to next step.

Next we had the problem of distinguishing between slang words and formal words. Initially we were focused on identifying non vocabulary terms with the aid of Part of Speech (POS) tagging in NLTK. If NLTK can’t determine the POS tag for a particular word, it will tag that word as ‘None’. Then they became slang candidates. To evaluate the accuracy, we decided to do an experiment. For our experiment, we prepared a corpus of 1000 tweets from twitter public stream. First we POS tagged those 1000 tweets manually. Then we trained NLTK POS tagger with Brown corpus. After that we obtained POS tagging results for all 1000 tweets. After careful inspection, we found out two things. In some cases slang terms are tagged incorrectly. They were supposed to get tagged as “None”. But at the end, some of them were tagged as NNP (Proper Noun). We also found that system judged few names as slang terms. Following table represents the results from our experiment.

	Predicted Class		
Actual Class		Slangs	Names
	Slangs	771	131
	Names	469	184

Table 6 Confusion Matrix

Total number of terms = 1555
Overall error percentage = 38%
Probability of false negatives = 0.41

With the 0.37 of false positive probability, NLTK POS tagger will predict many names as a slang terms. It also has a 0.41 value for false negative. Therefore slang prediction error is high we may lose some slangs during detection. Thereafter we trained the tagger with Treebank corpus. But end results gave a similar effect. Then we decided to use dictionary comparison. We used English dictionary from PyEnchant [22] which is a spell checking library for Python. But by only comparing against a dictionary, we can’t distinguish between slangs and names. Consequently we will have a mix of slangs. To overcome that, we decided to compile a list of common names.

If you consider tweets, there are various types of names they accommodate. Mainly, people names, location names and brand names. For people names, we adopted the names corpus provided by NLTK. It has more than 8000 names categorized as male, female and pet names. Then we found a solution described in Name Development web site which has a nice list of brand names. We implemented a spider and crawled it in Name Development to extract this list. But in Twitter, people don’t always use brand names as they are. If you consider the brands like ‘Coca Cola’ and ‘Mcdonalds’, they use them as ‘coke’ and ‘mac’. Therefore we manually

went through the brand list that we have and added different derivations of them which we suspects that people will use. For location names we used the list in. The list does not contain a list of locations that we can directly add to our dictionary. It has a location hierarchy where locations linked to each other. Therefore we implemented spider that can crawl recursively in through that list and collected location names. Finally we combined all of the above mentioned name lists and created a name dictionary. Then we used it to filter out names. Ultimately, we decided to use a combination of POS tagging and dictionary comparison to achieve a fair amount of accuracy. We highly considered the words which got tagged as 'None' in POS tagging process. Also we considered the words which got tagged as NNP (proper nouns). That is because according to the results we got from previous experiment, there is a possibility for a word which got tagged as NNP to be a slang word.

4.2 Slang Words Frequency Analysis

In the world of social networks, people tend to use slang words in various ways. Some of them uses slang words with different other meanings. For an example "cut it off" slang is used in some contexts to mean "stop". Sometimes people use words without vowels or by removing some set of insignificant letters from the word, identifying the correct word from these terms has to be done with the past experience and knowledge of the context. Sometimes users tend to write words as how they pronounce, not the actual word. Some people uses abbreviations such as 'lol' to mean something like "Laugh Out Loud". Some people use repeated characters to emphasize the meaning. For an example we can take word "looooooong". There is another way of using slang words in social media. That is when new trends comes in, users start to make new slangs out of it. For an example the TV series such as "How I Met Your Mother" are referred to as 'HIMYM' by some of social media users. To understand the meanings of those words uses should have an understanding about the context. As mentioned earlier there are different types of slang words. Thus, understanding the frequency of different types of slang words used in Social Media is very important in this research.

As one of the steps of understanding the degree of how many OOV words are used in twitter type of social networks, we did a frequency analysis. In that we used a twitter corpus with one million tweets. What we did in the analysis was, first we cleared the tweets by removing unwanted URLs, hash tags and white spaces. Python and NLTK were used in the above task. After cleaning the data, we started to run the script on those pre-processed data to calculate the frequency of OOV words. What the script basically did was, it checked each word with python PyEnchant spellchecking dictionary (dictionary which we used was "en_US") and if word doesn't contain in the dictionary, it was added as OOV term to a python dictionary. Since there were one million tweets to process, script ran nearly 48 hours. Followings are the OOV words with highest frequencies out of the one million tweets. By looking at the results we can observe that those are the very commonly used slang terms.

Words	Frequency
im	68090
dont	39218
lol	33885
youre	11680
haha	9104
aint	7829
lmao	6946
omg	6386
didnt	6022

Table 7 Selected Slang Frequency List

In the results set there were more than twelve thousand OOV terms, more than half of them were garbage words with frequency less than 60. The statistics showed that the words with some kind of spelling deviations also has some weight in the results, we can't just ignore them. Hence all the different types of slang word terms have to be considered when preprocessing the twitter dataset. Apart from the well-known meaningful slang words, there are other types of words tending to appear at the bottom of the list with some less frequency. The deviations of the same word also appear with similar number of frequency. Following are the list of those words.

Apart from the well-known meaningful slang words, there are other types of words tending to appear at the bottom of the list with some less frequency. The deviations of the same word also appear with similar number of frequency. Following are the list of those words.

Word	Frequency
aww	970
hmm	595
ahh	560
duhhh	28
shh	30
ewwww	27
yaya	25
grrrr	23
ehh	20
grr	18

Table 8 Interjection Frequencies

These types of words are called interjections. They are used to express feeling and they are not grammatically related to the rest of the sentence. These words are also a type of slang words. But we don't need to consider them when filtering slang words. We can just ignore them. But if we are planning to do Natural Language Processing (NLP) tasks like sentiment analysis, these interjections plays a vital role.

4.2.1 Slang Candidates

Nowadays people use various slang words across the world. We have much common slangs used across internet. Then we have various types of slangs like American Slang and Australian Slang which are considered as native. We can consider the slangs used in internet as a mix of these. In Social Media, daily there will be a large number of new slangs added. For our purpose we need to find slang words along with their proper meaning. But there are no solid definitions for slang words. Most of the slang definitions available are made up by people who use them in their day today communication.

For our research, we were in the need of finding a proper source to compile a list of slang words and their meanings. Then we came across urbandictionary.com, the most
Popular and informative slang word dictionary. According to urbandictionary.com is a community which updates frequently with new and trending slang words. Almost all the words in Urban Dictionary have more than single meaning for a word. As of march 2013, Urban Dictionary possesses more than 7 million definitions. They have opened a voting system for users in order to rank the meanings according to popularity.

To compile a list of slang words, we needed to access the urbandictionary.com API. But they haven't provided an API. urbandictionary.com has a view to show list of words alphabetically. So we implemented a spider to extract words from it. After running spider for two hours, we found 1300251 words. It's a quite large number to handle with our mapping process. Since meanings are not formatted into a structure, it's hard to extract the correct meaning that can be replaced with to place of the slang. It not only contains slang words, but also the words with proper spelling and with slang sense depending on their usage. For our solution, we are concerned on converting slang words to their meaningful form. We discovered some other sources which have a small list compared to urbandictionary.com. Among them we found noslang.com which we considered as a suitable web source for our task. We also found that they have well defined meanings with a proper structure for their wordlist. Their structure is well supportive for scraping using a spider. Again we wrote a spider and extracted their words and meanings. With that we were able to collect 1047 words.

As described in section V, we created a frequency list. That list also contained 11278 entries with person names and other names. So we used a frequency threshold value of 60 to remove names and obtain a solid slang word list. With that we formed a list of 1324 most frequent slang words. Then we compared both lists and merged them to get a common list. With the comparison, we removed different slang words which ultimately maps to same formal word. To illustrate consider the forms '2moro', '2mrw' and 'tomrw'. All these forms map to the same word tomorrow. Among all the forms, we got the most frequent form '2mrw'. Slang words

coming after slang detection will be first compared with the list that we created by merging. If there is a hit, we are replacing it with what we have in our direct list. If not, we will then pass them to a normal spell checker first. With that we identified incorrectly spelled words in tweets.

4.2.2 Handling Non-Hits

Going forward with the research we faced another problem. That is to automatically include the newly originating slang words to the system. When we get a newly originated slang word, it will not get a hit from our slang word list. But when we calculate the minimum edit distance of that term, we'll get some results with some high edit distance values. After analyzing those edit distance values we can come to a conclusion that if the minimum edit distance is greater than a threshold, the unidentified word as a newly originated slangs word. We put that new slang word to a new list called non-hit list.

Think about a word that is passed through both normal spell checker and modified spell checker described later on.. If it ended up without getting resolved, we update that word into the non-hit list. If the word already exists in that list, we update its frequency. With it, we can identify new trending slang words by using a threshold frequency. In twitter, there can be some entities that will get popular in a certain period and those will be referred in tweets as slang words. To illustrate we can consider the musician 'Justin Bieber'. When he got popular, tweets referred him as 'jb'. Slang 'jb' will gain a higher frequency in our list. Likewise we can make aware the system about new trending slangs. By creating a human interface, we can allow users to enter meanings of those trending slangs. When a user enters a meaning for a possible slang word (word with a frequency higher than the threshold) in non-hit list, that word entry will be removed from it and added to the main slang word list along with its meaning. But in order to pick trending words, they need to have significant frequency values. For that we can run the system for a quite long time period and pick trending words. Therefore we can do this periodically to update the system with upcoming slang words. If a word belongs to non-hit list, there is no meaning discovered for that word yet. If that same word appears again in a tweet, we have to pass it again through our pipeline. But it's costly. Instead we decided to compare it against our non-hit list as soon as we detect it as a slang word. If we find it in non-hit list, we increments its frequency and discontinues processing with it.

4.2.3 Spell Checker Approach

In this stage we are contending about a spell checker divergent from a normal spell checker. With various researches carried out for spell checking over the years, we decided to adopt one of them for our solution. With that focus, we improved our solution to resolve slang words to formal words not only by looking at the characters, but also considering contexts. We are using edit distance to figure out character confusions. To identify how suitable is a particular candidate for replacing, we are using context based spell checking. Minimum edit distance is a technique used in identifying the different between two words. Literary it's the minimum number of edit operations needed to transform one string to another. Basically these edit operations are "Deletion" , "Insertion" and "Substitution". When calculating the value we can use marking schema for each operation. For an example, for both insertion, deletion, cost is 1 and for substitution cost is 2. Likewise we can calculate a value for transforming each word to

another word. This value is useful in spell correction applications. In spellchecking application we can calculate the minimum edit distance of misspelled word. Using those values we can guess the correct word of misspelled word as the word with minimum edit distance value. This simple technique already covers 80% of all spell errors.

In our research we used minimum edit distance to identify wrongly interpreted slang words. Sometimes uses make mistakes when they type correct English words, same as that they also make mistake in writing slang words. The result of a misspelled slang word will be less frequent slang word of the parent word. Text from social media contains formal words, bslang words and misspelled slang words. For an example consider the word 'tomorrow', the most frequently used slang word of it is 'tmrw', therefore we can consider any misspelling of the root word 'tomorrow' as a misspelling of its most frequently used slang word 'tmrw'. Thus, when a user mistakenly types a word, we can find the most possible correct word of it using minimum edit distance.

In our research we did an experiment to identify how good this approach is. For that we referred Peter Norvig simple spell correction algorithm. We modified that algorithm to cater our purposes. Given a slang word, we are trying to identify the most relevant slang word where the given slang word is a derivation of suggesting slang word. In other words, we are providing the best possible parent slang word for given slang words. Consider the word 'whatever'. With our frequency analysis, we found 'watevr', 'wotevr', 'watev' and 'watevs' as a set of slang words used for 'whatever'. Among them 'watevr' as the most frequent slang word used. Therefore we call it as the parent slang word. All the other derivations mentioned are child slang words. Following are the calculations we considered for our algorithm.

P(PSW) – Probability that a parent slang word (PSW) that will appear on a Tweet.

P(DSW) – Probability that a detected slang word (DSW) that will appear on a Tweet.

P(DSW | PSW) – Probability that slang word DSW appears in a Tweet where author meant slang word PSW.

P(PSW | DSW) - Probability that parent slang word being PSW when detected slang word DSW appears in text.

To find the best parent slang for a detected slang word, we need to find the maximum value of P(PSW | DSW). By Baye's Theorem.

$$P(PSW \mid DSW) = \frac{P(DSW \mid PSW)\, P(PSW)}{P(DSW)}$$

$$\text{argmax } P(PSW \mid DSW) = \text{argmax } \frac{P(DSW \mid PSW)\, P(PSW)}{P(DSW)}$$

P(DSW) is the same value for all probabilities of S. Therefore we can ignore P(DSW). Now we are left with

$$\text{argmax } P(PSW \mid DSW) \propto \text{argmax } P(DSW \mid PSW)\, P(PSW)$$

If we calculate and find argmax P(DSW | PSW) P(PSW) , PSW in it is the correct parent slang for DSW. For all the parent slang words PSW, we calculate argmax P(PSW | DSW) and get the PSW as the parent which maximises it. To take this into action, we used the text corpus used in. We created a dictionary with all possible words that can have a slang word. Then we added there frequencies to the dictionary. Consider

The word 'the'. We found that the word 'the' appears 66327 times in the corpus. Which means slang 'da' of the proper word 'the' will also appear 66327 times in a corpus which only contains slangs for the words which have slangs. In this scenario, we are only concerned about mapping a slang word to its parent slang. So we only need to train the model for slangs. Model does not need to have the knowledge on the word 'the'. It should only have to know 'da' which is the parent slang for the word 'the' according to our slang frequency list. Our next challenge was to incorporate minimum edit distance for this. If you consider the slang words 'wotevr', 'watev' and 'watevs', most of them are not more than two edit units away with parent term 'watevr'. So we decided to use minimum edit distance threshold as two. Then we prepared a set of parent slang words which have two or less than two minimum edit distances compared to detected slang word. Then by using the elements in this set, we got the highest P(PSW) value among the frequencies available for the elements in the dictionary. That is our parent slang for the detected slang. Before conclude this experiment, we tested the accuracy of this. For that we used 76 different derivations of various slang terms and tagged them with their parent slangs. After that we passed those items to our spell checker and compared the results. Out of 76 we got 47 words correctly tagged with their parent. That means 62% accuracy. Following table shows some slangs and number of derivations used.

Parent slang	Derivations
2moro (tomorrow)	5
somthin (something)	6
watevr (whatever)	5
srsly (seriously)	3
sht (shit)	6

Table 9 Possible Derivations from Slangs

When considering resolving slang words, contexts will also matter. As an example consider the tweet "at least im not a prvt lyk someone here". In this sentence 'prvt' is a slang term which can be mapped to both 'private' and 'pervert'. Both have minimum edit distance value of 3. By looking at the sentence we can say that the correct mapping is 'pervert'. Therefore to pick the most suitable word, we also need to give the context knowledge to our system. In order to achieve this, we thought of using an N-gram model to map slang word with the correct context.

4.2.4 Context Based Replacement

As explained at the end of section VII, there can be situations where what the author of a tweet meant may be different from what we get using minimum edit distance. The reason is there are different character sequences which can be transformed in to the same word with same edit distance value. To resolve those kinds of situations, we have to understand the context of the writing. When suggesting for spell errors, understanding the context is not implemented in most of the spell correction tools. We found that there are different approaches to identify words according to the context of the sentence. One of the best approaches is N-gram modelling combined with minimum edit distance.

N-Gram model is a statistical technique used to calculate the probability of the next word given sequence of words. For an example,

"swallowed the large green _____"
pill? Broccoli?

"large green _____-"
tree? Mountain? Frog? Car?

Given words are w1, w2, w3, w4, ..., wn-1, we should be able to calculate the wn. That is a problem of conditional probability. Therefore to get results with fair accuracy implementing a N- gram model requires a lot of corpus data. In real word usages of N-gram modelling, they have used thri-gram modelling, four-gram modelling where N is quite small.

Larger N : more information about the context of the specific instance (greater discrimination)

Smaller N : more instances in training data, better statistical estimates (more reliability)

The main idea of this is to find correct term of misspelled words. This problem has already been solved formal English language context. But in the context of social networks, we have some additional complexity to deal with. As we have stated in Section VI, we will first check each misspelled word in Urban Dictionary and will replace it with its original word. For an example consider the sentence,

"fnd a good art glari"

Here we can identify that "fnd" and "glari" are out of vocabulary terms. First we have to search the meaning of these terms in an Urban Dictionary. Word "fnd" will get a hit since it's the most frequently used slang term of 'Find'. Even though there are other possible formal words such as 'Friend' for the slang term 'fnd' we replace it with the word 'Find' by analysing the context. We can identify the most frequently used slang terms of the formal words from the results in section V. The term "glari" will not get any hit. The next step is to find what does actually mean by the word "glari" using a N-gram model. In our experiments we used maximum of 4-grams to achieve the results. For an example to resolve the given example, we take the correct 3 words before the term "glari" and feed it to the four-gram model with list of possible candidates for the next word. The candidate words list is calculated by using the minimum edit

distance of 'glari' with respect to the slang words list and then mapping those to its formal word. Finally the N- gram model will output the list of probabilities of those candidate words. We can identify the word with highest probability as the correctly spelled word.

If there are no correct 4 words before the slang term, we have to consider three-gram modelling. Likewise we have to consider different N-gram models depending on the available number of correct words. We can achieve a good understanding about the context when we use a higher N-gram model (ex : four-gram model). In our research we played more attention on training an N-gram model. The problem with standard N-gram corpora is that their content is not specific to social media context. Having a context specific corpus is important because it will affect directly to the results. In order to build a context specific corpus, we collected tweets which have only the most
frequently used slang words and formal English words, over a 1week period. Then we replaced those slang words with its formal English word to build the corpus.

4.2.5 System Overview

In this section we are describing how a tweet with slang words will get resolved to its formal version. We implemented this system using python. For language processing tasks, we used NLTK library. When we feed a tweet, initially it will get cleaned by Regex base cleaner. It will eliminate all the redundant entities like URLs, hash tags and at mentions. Then tweet will be moved to tokenizer and POS tagger. It will tokenize words by using spaces and ignoring punctuation symbols. Thereafter it will tag resulted tokens with POS tags. After that those tagged words will get moved to comparator. Firstly, comparator will compare them in non-hit list. Then it will get relevant POS tagged words and compare with pyEnchant dictionary. Finally, it will compare with Names dictionary. As next step, words will be moved to spell checker where they will be checked for spelling errors. After that both words and tweet will enter the analyzer. Then analyzer, with the aid of modified spell checker and slang list, will check what the possible parent slang candidates for detected slang are. Finally analyzer will access N-gram model and select the most suitable meaning among the meanings that we have for identified parent slang candidates. Likewise analyzer will do this for all detected slang words and give the most possible meaning depending on context.

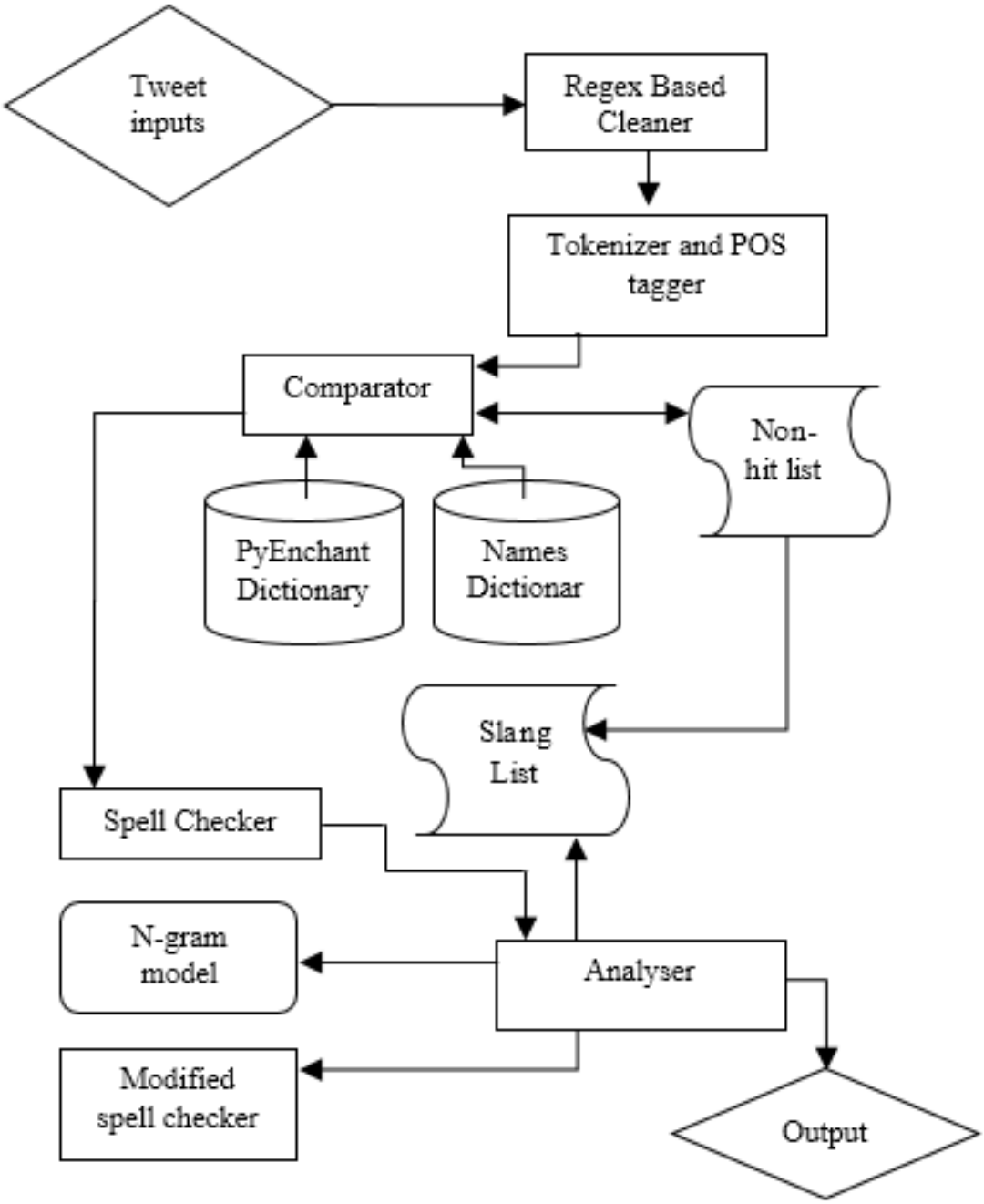


Figure 12 Slang Detector Overview

After completing preprocessing next step was the filtering travel related tweets among them. Cleaned tweets were again run through further filtering process.

Main Assumption:

There should be a Location and Activity entailed with a tweet to consider it as a travel related tweet.

Each tweet is tokenized and then POS tagged using NLTK POS tagger and Treebank Corpus. From previous research we came to know that Places names are often tagged as NNP (proper nouns) or sometimes as NN (Nouns). But person names can also be tagged as NNP. Then we compare each NN or NNP tagged word with our gazetteers. Here we use places , regions and place types gazetteers for comparison. We thought of giving a score by matching the query. This is a complicated process since gazetteer may not contain the exact word in the query. Instead query may have misspelled word or slang word.
Eg:

Tweet: “I wanna go n watching da Clausn Memrial Musem.”

POS tagged:
[('I', 'PRP'), ('wan', 'VBP'), ('na', 'NNS'), ('go', 'VBP'), ('n', 'NN'), ('watching', 'VBG'), ('da', 'NN'), ('Clausn', 'NNP'), ('Memrial', 'NNP'), ('Musem', 'NNP'), ('.', '.')]

Query : “n da Clausn Memrial Musem”

Matching place in gazetteer: Clausen Memorial Museum.

Score: 0.8333

Following methods considering edit distance and longest match are taken in to account while calculating the score.

find_longest_match(alo, ahi, blo, bhi)
Find longest matching block in a[alo:ahi] and b[blo:bhi].

get_matching_blocks()
Return list of triples describing matching subsequences.

get_opcodes()
Return list of 5-tuples describing how to turn a into b.

ratio()
Return a measure of the sequences' similarity (float in [0,1]).

We also used NLTK NER chunker to do named entity recognition after POS tagging tweets. It distinguishes the GPE location using nltk.ne_chunker which is the. recommended named entity chunker to chunk the given list of tagged tokens.

RegexpChunkParser is an implementation of the chunk parser interface that uses regular-expressions over tags to chunk a text. Its parse() method first constructs a ChunkString, which encodes a particular chunking of the input text. Initially, nothing is chunked. parse.RegexpChunkParser then applies a sequence of RegexpChunkRule rules to the ChunkString, each of which modifies the chunking that it encodes. Finally, the ChunkString is transformed back into a chunk structure, which is returned. RegexpChunkParser can only be used to chunk a single kind of phrase. For example, you can use an RegexpChunkParser to chunk the noun phrases in a text, or the verb phrases in a text; but you can not use it to simultaneously chunk both noun phrases and verb phrases in the same text.

Like above procedure activities are normally tagged as VBZ in NLTK POS tagger. All the words set which are tagged as VBZ set as a query and compared with Activities and Activity types gazetteers .(Qualifiers and Qulifier types) . Same thing we did continued to find a sore considering edit distance and longest matches.

We set threshold of the score as 0.75
Eg:
Input:surfing in Hikkaduwa is awesome.
(S surfing/VBG in/IN (GPE Hikkaduwa/NNP) is/VBZ awesome/VBN)
Tweet contains a GPE location

Tweet contains a "Activity" called Surfing

Tweet contains both location and Activity

score: 0.8

There would be some conditions for a tweet to satisfy before doing sentimental analysis and time series analysis. It can be either one of the following.

- Tweet containing Place and Activity.
- Tweet containing GPE location and Activity.
- Tweet containing Region and Activity.

4.2.6 Where to Store the tweets collection

Then we had the problem where we store the data we extracted from twitter stream. First we tried MongoDB .Since the project we are involved in deals with the collection and storage of huge amount of structured data, we needed to have a database that is flexible and performance in storing and retrieving dynamic content and form data. We feel that MongoDB is the best option because of its document-oriented and schema less nature.
But there is no data source component in Weka for MongoDB yet. Therefore we cannot continue our research on time series analysis which is totally done using WEKA.

Then we thought of using mysql databases for storage since it is compatible with WEKA. Tables for each entry in places gazetteer is stored which contains tuples of tweet , time stamp ,activity ,place and region. Finally these information is pumped to sentimental analysis and then time series analysis process.

4.3 Information Extraction

4.3.1 Information Extraction From Semi Structured Sources

Internet consist of various documents mainly web pages and other document formats like pdfs, word docs etc. Think that you are writing a document. If you do not have much content for that or if its going to be a one page document then you just write the document with several paragraphs. But if it is a complex document, you will decide a structure for it most likely before start writing it. You will break the content of it to subtopics and write. Its the same case even for single page web page. When we are going to create a website with multiple web pages, we need to prepare navigation or define a site map. This should be done explicitly. So always there is a structure for a web site. But they are not structured into micro level such as named entity tagging for word by word. Hence we identify travel related web sites as semi structured web sources for our project.

4.3.2 Discovered Semi structured Sources and their importance and credibility.

There are hundreds of travel related web site published in the internet. But we need to consider web sites which are credible, updated and highly interactive users. Therefore we are considering following web sites.

4.3.2.1 WikiTravel

Though WikiTravel does not have interaction with users, we selected WikiTravel as our main web source. That is because we wanted a static source that we can use to populate our ontology as the base. WikiTravel only contains articles about travel locations and those articles contain well formatted reliable information. It is based on MediaWiki and MediaWiki also add a proper structure to it.

WikiTravel is a project to create a free, complete, up-to-date and reliable world-wide travel guide. It is built in collaboration by Wiki Travelers from around the globe.
WikiTravel is built with the spirit of sharing knowledge that makes travel so enjoyable. Whenever travellers meet each other on the road, they swap info about the places they came from and ask questions about places they're going. We want to make it easy to share that knowledge and let others share it; our copy left license means that the facts you know can spread far and wide.
The project was begun in July 2003 by the two founders, Evan and Maj and inspired by Wikipedia, the free encyclopedia, and by the needs of travelers for timely information that long book-publishing cycles can't seem to meet. In April 2006, Wikitravel was acquired by Internet Brands, Inc., an operator of consumer information Web sites.

To create Wikitravel [23] we use a tool (or a process, or a technology) called Wiki which lets any Internet reader create, update, edit, and illustrate any article on the Web site. We all share our pieces of knowledge, edit them, distill them, and assemble them into a pleasing and cohesive whole. The more people that use the Edit link, the better Wikitravel becomes.

4.3.2.2 Virtual Tourists

Virtual Tourist [24] is different from Wikipedia mainly because it has a higher user interaction. It is like a Social Media site for tourism. As one of the largest sources of unbiased, user-generated travel content in the world, Virtual Tourist is the premier resource for travelers seeking an insider's perspective. Real travel tips, reviews and photos from real people who have actually been there and done that and this is what makes the travel content on Virtual Tourist so useful for our project.

Since every tip on Virtual Tourist is linked to a member's profile, with just a click you can learn more about travelers details like hometown, travel interests, where they've been, hobbies, even what they look like-and then read about more of their travel experiences.

On Virtual Tourist, tips and reviews are organized by destination-from Sydney to New York and everywhere in between-and then into 13 separate categories, like Hotels, Things to Do, Local Customs, Shopping, Tourist Traps and more. This makes it easy for members to contribute content and for other users to find it.

4.3.2.3 Trip Advisor

Trip Advisor [25] is considered as the world's largest travel site, enabling travelers to plan and have the perfect trip. Trip Advisor can be used to extract hot travel locations from real travelers and a wide variety of travel choices and planning features with seamless links to booking tools. Trip Advisor branded sites make up the largest travel community in the world, with more than 260 million unique monthly visitors, and over 125 million reviews and opinions covering more than 3.1 million accommodations, restaurants and attractions. But for our purpose, we are only concerned on travel attractions.

4.3.2.4 Nature of Semi Structured Sources

Semi structured sources have structuring up to some level. For instance if you take a web article it will have topics, subtopics and bullet points etc. With those we can identify a hierarchy. Then we can identify the inheritance with it. But there are paragraphs or text chunks under these sub topics without categorizing. With these text, it’s hard to identify relationships with these. For that we need to process this text to identify the key words and their categorization tags.

4.3.2.5 Distinguishing unstructured and structured content.

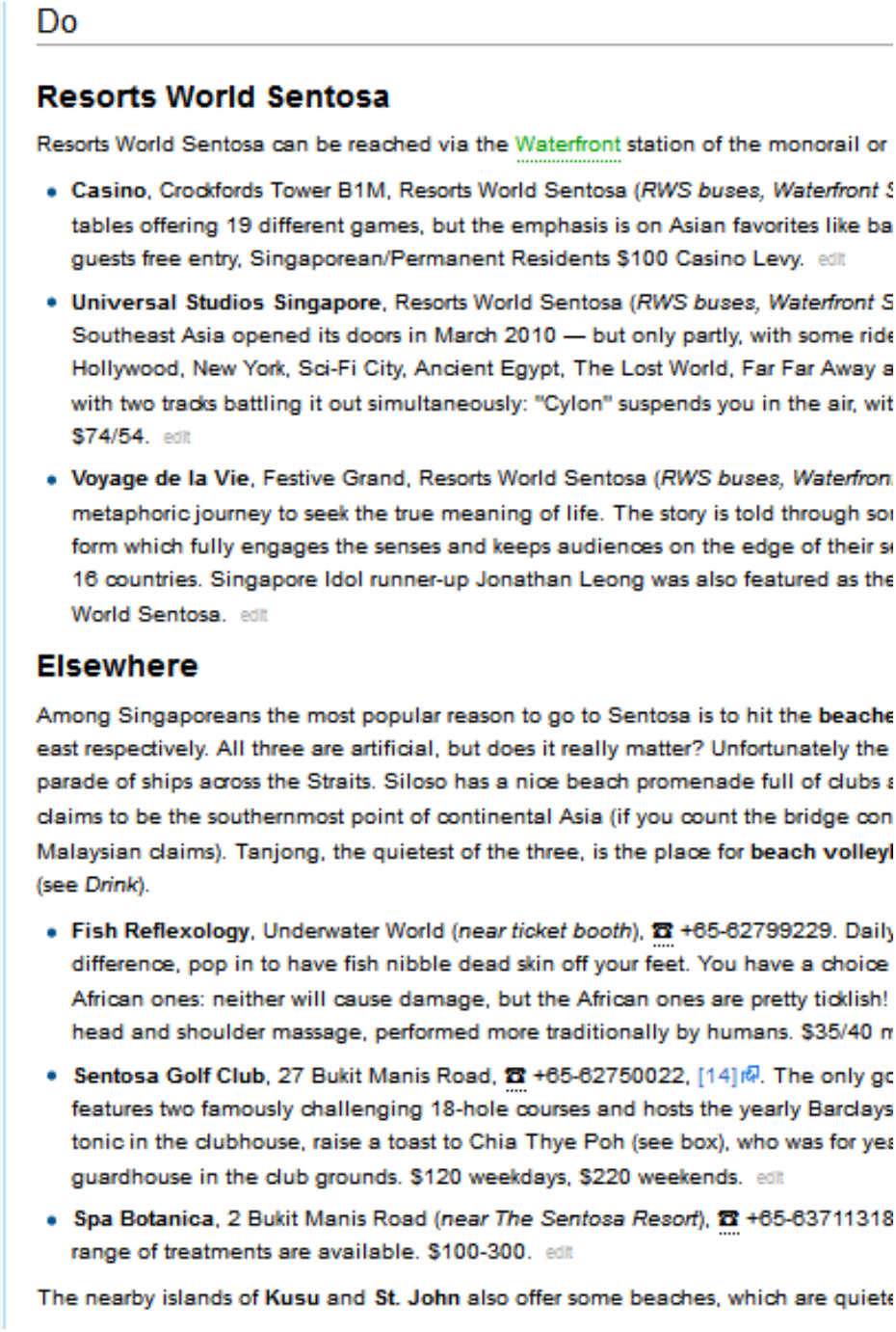
As mentioned previously, with these web pages, there are two parts of content. Unstructured content and structured content. First we need to distinguish structured and structured content. As mentioned previously, it’s obvious that we can take topic structure as structured content. Let’s consider following image from TravelWiki.

Figure 13 WikiTravel Article Anchors

This is a topic table in Travel Wiki article. Almost all the Travel Wiki articles have this table. Each article have the same structure with this topic hierarchy. Then, If you read more into these listed topics, there are more sub topics. Among these we can directly use structured information. Then there are unstructured information under these structured information where we need to find out.

4.3.2.6 Extraction Structured Information

In above image you can see that list has an entry called “do”. Under that there are many sub items under it as shown in following figure.



Do

Resorts World Sentosa

Resorts World Sentosa can be reached via the Waterfront station of the monorail or

- **Casino**, Crockfords Tower B1M, Resorts World Sentosa (*RWS buses, Waterfront S* tables offering 19 different games, but the emphasis is on Asian favorites like ba guests free entry, Singaporean/Permanent Residents $100 Casino Levy. edit
- **Universal Studios Singapore**, Resorts World Sentosa (*RWS buses, Waterfront S* Southeast Asia opened its doors in March 2010 — but only partly, with some ride Hollywood, New York, Sci-Fi City, Ancient Egypt, The Lost World, Far Far Away a with two tracks battling it out simultaneously: "Cylon" suspends you in the air, wit $74/54. edit
- **Voyage de la Vie**, Festive Grand, Resorts World Sentosa (*RWS buses, Waterfron* metaphoric journey to seek the true meaning of life. The story is told through so form which fully engages the senses and keeps audiences on the edge of their s 16 countries. Singapore Idol runner-up Jonathan Leong was also featured as the World Sentosa. edit

Elsewhere

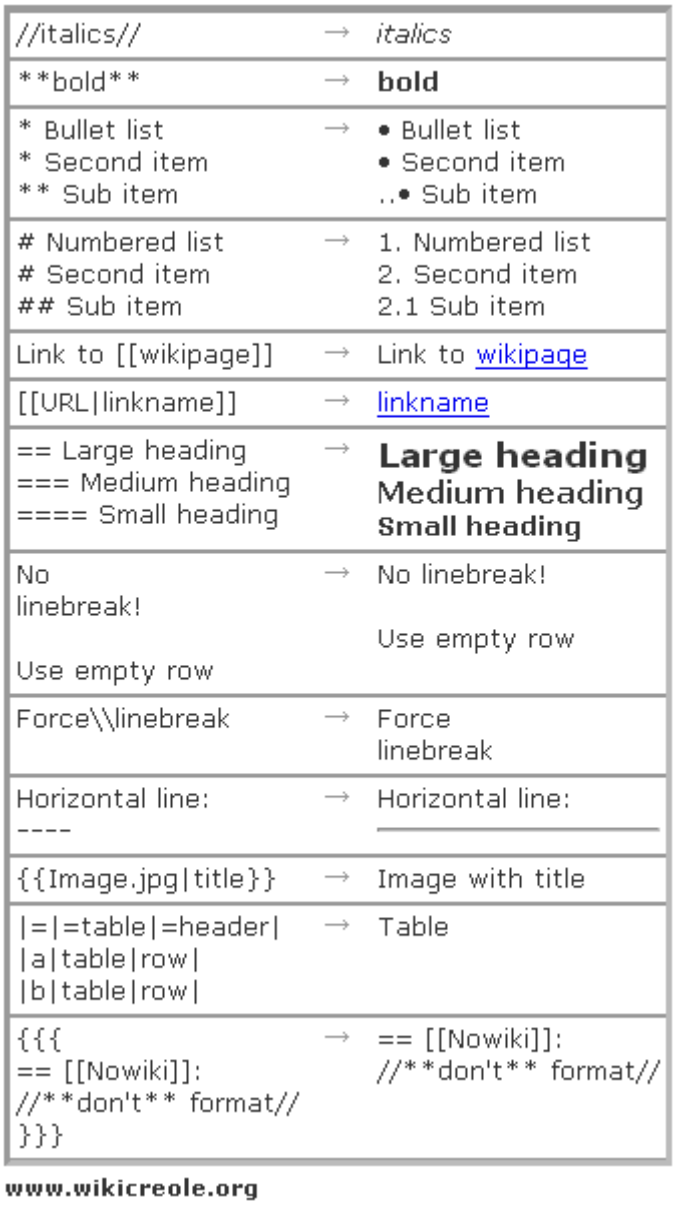
Among Singaporeans the most popular reason to go to Sentosa is to hit the **beache** east respectively. All three are artificial, but does it really matter? Unfortunately the parade of ships across the Straits. Siloso has a nice beach promenade full of clubs a claims to be the southernmost point of continental Asia (if you count the bridge con Malaysian claims). Tanjong, the quietest of the three, is the place for **beach volleyl** (see *Drink*).

- **Fish Reflexology**, Underwater World (*near ticket booth*), ☎ +65-62799229. Daily difference, pop in to have fish nibble dead skin off your feet. You have a choice African ones: neither will cause damage, but the African ones are pretty ticklish! head and shoulder massage, performed more traditionally by humans. $35/40 m
- **Sentosa Golf Club**, 27 Bukit Manis Road, ☎ +65-62750022, [14]. The only go features two famously challenging 18-hole courses and hosts the yearly Barclays tonic in the clubhouse, raise a toast to Chia Thye Poh (see box), who was for yea guardhouse in the club grounds. $120 weekdays, $220 weekends. edit
- **Spa Botanica**, 2 Bukit Manis Road (*near The Sentosa Resort*), ☎ +65-63711318 range of treatments are available. $100-300. edit

The nearby islands of **Kusu** and **St. John** also offer some beaches, which are quiete

Figure 14 TravelWiki Article Structure Sample

As shown in the image topics for sub sections and place names are written in bold. But since this is a wiki, this not an ordinary text where we can do direct string processing. TravelWiki is using MediaWIki. Thus we need to consider parsing MediaWiki markup. Following image illustrates how Mediawiki markup is used.



//italics//	→	*italics*
bold	→	**bold**
* Bullet list * Second item ** Sub item	→	• Bullet list • Second item ..• Sub item
# Numbered list # Second item ## Sub item	→	1. Numbered list 2. Second item 2.1 Sub item
Link to [[wikipage]]	→	Link to wikipage
[[URL|linkname]]	→	linkname
== Large heading === Medium heading ==== Small heading	→	**Large heading** **Medium heading** **Small heading**
No linebreak! Use empty row	→	No linebreak! Use empty row
Force\\linebreak	→	Force linebreak
Horizontal line: ----	→	Horizontal line: ----------
{{Image.jpg|title}}	→	Image with title
|=|=table|=header| |a|table|row| |b|table|row|	→	Table
{{{ == [[Nowiki]]: //**don't** format// }}}	→	== [[Nowiki]]: //**don't** format//

www.wikicreole.org

Figure 15 Wiki Markup Meanings

From above markup table we can identify following types of markups to identify topic hierarchy in articles.

- Bold
- Bullet List
- Second Item
- Sub Item
- Numbered List
- Large Heading
- Medium Heading
- Small Heading

By using simple regular expressions, we can identify all the keywords, names, topics and relationships between them. For instance a particular article may contain a medium heading as “Park”. Then it will have a small heading called “Amusement Park”. So we can identify that “Park” as parent and “Amusement Park” as child.

4.3.2.7 Extracting and relating unstructured information with structured information.

Next we have paragraphs or word chunks under these keywords or topics. Those paragraphs may also contain special keywords or names which we should concern about. For that we need to take paragraph by paragraph and process them using NLP. Explicitly saying we can use Named Entity Recognition and POS tagging for this. Thereafter we can combine both methods to relate structured and unstructured keywords. To illustrate let's consider a paragraph under the topic "Park". If we can find a name by feeding that paragraph to NLP and NER, we may be able to identify a name and most probably it will be a park. If that name contains name "Park" to the end or at the beginning of that name, it is surely a park. In this manner we can relate unstructured information with structured information.

4.3.3 Populating Ontology

Since we are using an automated data collection and semantic mapping process, we need to ensure the reliability of our structure data represented by the ontology. Thus we wanted to populate the ontology using a reliable source. We decided to choose WikiTravel for this as it has reliable information and a proper structure as described above.

4.3.3.1 Bridging the gap between ontology and extracted information.

When considering extracted information, there are hidden clues that we can use to map with real text. In order to identify relations we need to apply various techniques like NLP and differencing. Not only that, but also we need to focus on ontology structure as well. First of all we need to identify the keywords relevant to our purpose. For that we can use simple rules as we have a semi structured Wiki text as described in section 4.3.2.6. Then for each text under these topics, we can use NLP to identify hidden keywords. Refer the ontology structure is described in section 4.4.1. We are using class labels and hierarchy in the ontology to match with keywords from text. For text chunks under each keyword, we need to apply NLP and identify keywords. Then we are relating those to the ontology as well.

4.3.3.2 Similarities between ontology and extracted information

When considering the similarities, there are two types of similarities that we have identify. Direct similarities and indirect similarities. Direct similarities exist when ontology keyword or its derivation appears in the Wiki text with special markup as described in Figure 8 . Consider the following word derivation tree on travel activity diving.

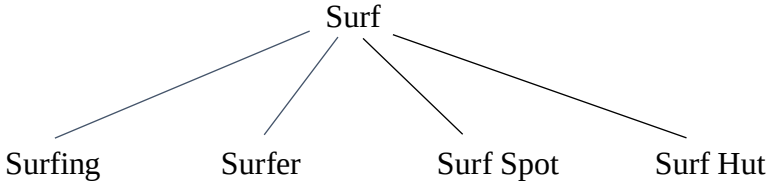


Figure 16 Word Tree for WikiTravel Keywords

Parent word surf in the diagram is the stem of both words surfing and surfer. Then we have surf spot and surf hut words. These are not related with stem. They are just words related to travel domain. But in the ontology we have a qualifier type Surfing and a place type Surf Spot. Thus we need to identify and map these words ontology class names. This is a one type of

indirect mapping. By identifying weather both ontology class name and identified keyword from wiki text has a similarity, we can check weather both words getting stemmed to the same term. For that we are using Porters Stemming Algorithm. For other words like Surf Spot and Surf Hut, we can't identify unless we have a mapping defined. So we manually defined that type of mapping and added relations accordingly.

All the above descriptions about mapping considered keywords that can be identified by using Wiki markup. But for the keywords that are inside text chunks under each section of wiki markup, we had to use NLP and string and pattern matching to recognize. First we POS tagged each text chunk. Then we used POS tag patterns to identify keywords. To illustrate consider the following example.

Sentence exist in wiki text

Dubai is great for shopping.
London is awesome for Skating.

After POS tagging

Dubai/NNP is/VBZ great/JJ for/IN Skydiving/VBG
London/NNP is/VBZ nice/JJ for/IN Skating/VGB

POS tag pattern

<NNP>*<JJ>*<VBG>

By using above POS tag pattern, we can identify what the key words are in and relationships between them. By tracing NNP(Dubai) and VBG(skydiving) after a JJ(great) tag, we can map Dubai to Skydiving in ontology. For this POS tagging purpose we are using NLTK POS tagger with Treebank tagger.

4.3.3.3 Rule Engines

When considering the above mentioned mapping methods we need to write many if else conditions for stemming, POS tag patterns mapping while referring lists of words. But there are various class names and ways that a particular keyword can be related to. So we decided to go with a rule engine we explored two rule engines and explored following details about them.

- **Gate**

  GATE [26] stands for General Architecture for Text Engineering. GATE is over 15 years old and is in active use for all types of computational task involving human language. GATE excels at text analysis of all shapes and sizes. From large corporations to small startups and to undergraduate projects, GATE user community is the largest and most diverse of any system of this type, and is spread across all but one of the continents. This is one of the main reasons we decided to choose GATE.

  GATE is open source free software and users can obtain free support from the user and developer community. They are the biggest open source language processing project with a development team more than double the size of the largest comparable projects.

On top of the core functions GATE includes components for diverse language processing tasks like parsers, morphology, tagging, Information Retrieval tools, Information Extraction components for various languages, and many others. GATE Developer and Embedded are supplied with an Information Extraction system (ANNIE) which has been adapted and evaluated very widely. ANNIE is often used to create RDF or OWL for unstructured content. Thus ANNIE becomes a key component for our project. From GATE we used XML document format from document reader, JAPE for handling annotations and ANNIE for writing rules.

- **JAPE**

JAPE [27], which stands for Java Annotation Patterns Engine, played and important role when annotating WikiTravel text. JAPE provides finite state transduction over annotations based on regular expressions. JAPE is a version of CPSL – Common Pattern Specification Language.

JAPE allows you to recognize regular expressions in annotations on documents. Regular expressions are applied to character strings, a simple linear sequence of items, but here we are applying them to a much more complex data structure in WikiTravel. The result is that in certain cases the matching process is non-deterministic (i.e. the results are dependent on random factors like the addresses at which data is stored in the virtual machine): when there is structure in the graph being matched that requires more than the power of a regular automaton to recognize, JAPE chooses an alternative arbitrarily. However, this is not the bad news that it seems to be, as it turns out that in many useful cases the data stored in annotation graphs in GATE (and other language processing systems) can be regarded as simple sequences, and matched deterministically with regular expressions.

A JAPE grammar consists of a set of phases, each of which consists of a set of pattern/action rules. The phases run sequentially and constitute a cascade of finite state transducers over annotations. The left-hand-side (LHS) of the rules consist of an annotation pattern description. The right-hand-side (RHS) consists of annotation manipulation statements. Annotations matched on the LHS of a rule may be referred to on the RHS by means of labels that are attached to pattern elements. Consider the following example.

```
Phase: QualiferType
Input: Lookup
Options: control = appelt debug = true

Rule: QualiferType1
(
 {Lookup.majorType == qualifertype }
 (
 {Lookup.majorType == qualifertype }
 )?
)
: qualifertype
-->
 : qualifertype. QualiferType = {rule = " QualiferType1"}
```

The LHS is the part preceding the '-->' and the RHS is the part following it. The LHS specifies a pattern to be matched to the annotated GATE document, whereas the RHS specifies what is to be done to the matched text. In this example, we have a rule entitled 'Qualifiertype1', which will match text annotated with a 'Lookup' annotation with a 'majorType' feature of 'Qualifiertype', followed optionally by further text annotated as a 'Lookup' with 'majorType' of 'Qualifiertype'. Once this rule has matched a sequence of text, the entire sequence is allocated a label by the rule, and in this case, the label is 'Qualifiertype'. On the RHS, we refer to this span of text using the label given in the LHS; 'Qualifiertype'. We say that this text is to be given an annotation of type 'Qualifiertype' and a 'rule' feature set to 'Qualifiertype1'.

We began the JAPE grammar by giving it a phase name, e.g. 'Phase: Qualifiertype'. JAPE grammars can be cascaded, and so each grammar is considered to be a 'phase' . The phase name makes up part of the Java class name for the compiled RHS actions. Because of this, it must contain alphanumeric characters and underscores only, and cannot start with a number.

We also provide a list of the annotation types we will use in the grammar. In this case, we say 'Input: Lookup' because the only annotation type we use on the LHS are Lookup annotations. If no annotations are defined, all annotations will be matched.Several options can be set as follows.

By using control defines the method of rule matching. Then we have debug option. When it is set to true, if the grammar is running in Appelt mode and there is more than one possible match, the conflicts will be displayed on the standard output. A wide range of functionality can be used with JAPE, making it a very adaptive for out WikiTravel text. But JAPE was unable to give mappings for words in Wiki markup as it was not considering semantics and context of a word in a text. Hence, we had to search for another solution. Then we found information extraction system built into GATE. It was called ANNIE.

- **ANNIE**

ANNIE [28] stands for Nearly New Information Extraction. ANNIE consist of a pipeline which include components which we have described in Section 4.3.3.2. ANNIE consist of set of resources. The document reset resource enables the document to be reset to its original state, by removing all the annotation sets and their contents, apart from the one containing the document format analysis (Original Markups). An optional parameter, keepOriginalMarkupsAS, allows users to decide whether to keep the Original Markups AS or not while resetting the document. The parameter annotationTypes can be used to specify a list of annotation types to remove from all the sets instead of the whole sets.

Alternatively, if the parameter setsToRemove is not empty, the other parameters except annotationTypes are ignored and only the annotation sets specified in this list will be removed. If annotationTypes is also specified, only those annotation types in the specified sets are removed. In order to specify that you want to reset the default annotation set, we can add without entering a name – this will add a null value which denotes the default annotation set. This resource is normally added to the beginning of an application, so that a document is reset before an application is rerun on that document.

ANNIE tokenizer splits the text into very simple tokens such as numbers, punctuation and words of different types. For example, we distinguish between words in uppercase and lowercase, and between certain types of punctuation. The aim is to limit the work of the tokenizer to maximize efficiency, and enable greater flexibility by placing the burden on the grammar rules, which are more adaptable. A rule has a left hand side (LHS) and a right hand side (RHS). The LHS is a regular expression which has to be matched on the input; the RHS describes the annotations to be added to the AnnotationSet. The LHS is separated from the RHS by '>'. The following operators can be used on the LHS. Since our WikiTravel text is from English and we are only recognizing English text for our project we had to use English tokenizer.

The English Tokenizer is a processing resource that comprises a normal tokenizer and a JAPE transducer. The transducer has the role of adapting the generic output of the tokenizer to the requirements of the English POS tagger. One such adaptation is the joining together in one token of constructs like " '30s", " 'Cause", " 'em", " 'N", " 'S", " 's", " 'T", " 'd", " 'll", " 'm", " 're", " 'til", " ve", etc. Another task of the JAPE transducer is to convert negative constructs like "don't" from three tokens ("don", " ' " and "t") into two tokens ("do" and "n't"). The English Tokenizer should always be used on English texts that need to be processed afterwards by the POS Tagger.

We are using Gazetteer to enforce Travel Domain specific keywords to the system. The role of the gazetteer is to identify entity names in the text based on lists. The gazetteer lists used are plain text files, with one entry per line. Each list represents a set of names, such as names of places, travel activity names etc. Below is a small section of the list for units of currency.

An index file (lists.def) is used to access these lists, for each list, a major type is specified and, optionally, a minor type. It is also possible to include a language in the same way, where lists for different languages are used, though ANNIE is only concerned with monolingual recognition. By default, the Gazetteer PR creates a Lookup annotation for every gazetteer entry it finds in the text. One can also specify an annotation type specific to an individual list.

These lists are compiled into finite state machines. Any text tokens that are matched by these machines will be annotated with features specifying the major and minor types. Grammar rules then specify the types to be identified in particular circumstances. Each gazetteer list should reside in the same directory as the index file. So, for example, if a specific travel activity 'racing' needs to be identified, the minor type 'horse racing' should be specified in the grammar, in order to match only information about specific 'racing' activity.

In addition, the gazetteer allows arbitrary feature values to be associated with particular entries in a single list. ANNIE does not use this capability, but to enable it for your own gazetteers, by setting the optional gazetteerFeatureSeparator parameter to a single character (or an escape sequence such as \t or \uNNNN) when creating a gazetteer. In this mode, each line in a .lst file can have feature values specified.

It is possible to have more than one pattern and corresponding action in a single JAPE grammar rule, as shown in the TwoPatterns.jape rule below. On the LHS, each pattern is enclosed in a set of round brackets and has a unique label; on the RHS, each label is associated with an action. In this example, the Lookup annotation Title is labelled "QualifierType" and is given the new annotation Title. Activity annotation is labelled "ActivityType" and is given the new annotation ActivityType.

```
Input: Lookup Token
Options: control = brill
Rule: TwoPatterns
Priority: 20

(
{Lookup.majorType == "QualiferType"}
): qualifertype

(
{Lookup.majorType == " ActivityType "}
): activitytype

-->

:title.Title ={rule= "TwoPatterns" },
: qualifertype. QualiferType = {rule= "TwoPatterns" }
```

.

4.3.3.4 Implementation of inferencing and populating component

Though there were many rule engines we have tried as explained above, we colulddnt find a proper rule engine to match our purpose. Because WikiTravel had its own Wiki markup and we were unable to recognize keywords that we can be recognized by using Wiki markup when joining with a rule engine. Thus we used a set of tools namely python regex engine, Porter stemmer and developed our own inference engine to populate ontology with WikiTravel instances.

Next challenge was to query to real ontology file. These ontology files will have various formats like .owl and .rdf. Both these file formats are having a xml structure. Though we could have write to those ontology files by implementing a xml writer, we decided to go for an external library to populate the ontology from WikiTravel instances. That was mainly because we had to populate the ontology adhering to ontology concepts. In other words we wanted to add instances and assert data type properties and object properties to instances. So we decided to go with a third part library. We found and tried POC's with three libraries in combination with our system. Details as follows.

- **Seasame**

  Sesame [29] is an open source Java framework for storage and querying of RDF data. The framework is fully extensible and configurable with respect to storage mechanisms, inferences, RDF file formats, query result formats and query languages. Sesame offers a JBDC-like user API, streamlined system APIs and a RESTful HTTP interface supporting the SPARQL Protocol for RDF. A framework isn't very useful without implementations of the various APIs. Out of the box, Sesame supports SPARQL and SeRQL querying, a memory-based and a disk-based RDF store and RDF Schema inferencers. It also supports most popular RDF file formats and query result formats. Various extensions are available or are being worked at elsewhere.

The reason that we decided to choose Seasame beacause, we wanted to use a python as it will be more compatible with our text processing tool Open NLTK. Then Seasame turns out to be the only API that had a Python wrapper. But with further analysis, we figured out that we may need to query a large ontology and most of the optimized ontology querying techniques are available in java. There were many other lightweight Java only libraries that we can use to query ontology. So we decide to use try following libraries.

- **Java OWL API**
  The Java OWL API [30] is a Java API for creating, parsing, manipulating and serialising OWL Ontologies. This include following components.

  1. An API for OWL 2.
  2. In-memory reference implementation for this API.
  3. RDF/XML, OWL/XML, OWL Functional Syntax, Manchester OWL Syntax, Turtle, KRSS and OBO Flat file parsers and renderers.
  4. Reasoner interfaces for external reasoners.

As you can see from above components, we have what we need in component in 4th item. So we decided to go for Java OWL api. Following code samples illustrates how we have used Java OWL API in our populating component.

Following code chunk illustrates on how to accept and preparing a ontology to populate.

```
/**
  * @param fileName .owl file name we are going to populate
  * @param baseIRI Information Resource Indicator(IRI)
  *
*/
public Ontopop(String fileName, String baseIRI) {
                try {
                        f = new File("res/ontologies/" + fileName + ".owl");
                        this.baseIRI = baseIRI;
                        manager = OWLManager.createOWLOntologyManager();
                        ontology = manager.loadOntologyFromOntologyDocument(f);
                        ontologyIRI = manager.getOntologyDocumentIRI(ontology);
                        dataFactory = manager.getOWLDataFactory();
                        this.extractClasses();
                } catch (OWLOntologyCreationException e) {
                        e.printStackTrace();
                }
}
```

Figure 17 Initiating Ontology

Following figure shows how to add an individual using Java OWL API to the ontology.

```
/**
  * @param className name of the class that we need to add the instance
  * @param individualName name of the individual that we need to add
  *
*/
public void addInstance(String className, String individualName){
                try {
                        OWLIndividual myIndividual =
                        dataFactory.getOWLNamedIndividual(
                        IRI.create(this.baseIRI + "#" + individualName));
                        OWLClass myClass =
                        dataFactory.getOWLClass(IRI.create(baseIRI + "#" + className));
                        OWLClassAssertionAxiom classAssertion =
                        dataFactory.getOWLClassAssertionAxiom(myClass, myIndividual);
                        manager.addAxiom(ontology, classAssertion);
                        manager.saveOntology(ontology);
                } catch (OWLOntologyStorageException e) {
                        e.printStackTrace();
                }
}
```

Figure 18 Populate Instance Method

Next code chunk shows how we added a object property to our ontology to our ontology.

```
/**
  * @param propertyName name of the property we need to add
  * @param subject name of the subject that we need to relate to
  * @param object name of the object that we need to relate with
  *
*/
public void addObjectProperty(String propertyName, String subject, String object){
                try {
                        OWLIndividual mySubject =
                        dataFactory.getOWLNamedIndividual(IRI.create(this.baseIRI + "#" + subject));
                        OWLIndividual myObject =
                        dataFactory.getOWLNamedIndividual(IRI.create(this.baseIRI + "#" + object));
                        OWLObjectProperty myProperty =
                        dataFactory.getOWLObjectProperty(IRI.create(this.baseIRI + "#" + propertyName));
                        OWLObjectPropertyAssertionAxiom assertion =
                        dataFactory.getOWLObjectPropertyAssertionAxiom(myProperty, mySubject, myObject);
                        AddAxiom addAxiomChange =
                        new AddAxiom(ontology, assertion);
                        manager.applyChange(addAxiomChange);
                        manager.saveOntology(ontology);
                } catch (Exception e) {
                        e.printStackTrace();
                }
}
```

Figure 19 Property Adding Method

4.4. Sentiment analysis

Sentiment analysis is the process of identifying the meaning of the sentence. It is actually an application of natural language processing. In Travel Profiling application, after identifying the travel related social text feeds, it is needed to identify whether they express a positive comment or a negative comment on that place/activity. Only after identifying the meaning of the text, we can use that data to log about travel locations. For example system should identify tweets like “Had an awesome fun at Hikkaduwa beach” as a positive comment and tweets such as “Hikkaduwa is not good for Surfing” as negative.

The approach used to sentiment analysis is basically two levels of hierarchical classification by combining a subjectivity classifier and polarity classifiers. As the figure below, it contains two levels of classifications. The subjectivity classifier is first, and determines whether the text is objective or subjective. If the text is objective, then a label of neutral is returned, and the polarity classifier is not used. However, if the text is subjective (or polar), then the polarity classifiers are used to determine if the text is positive or negative.

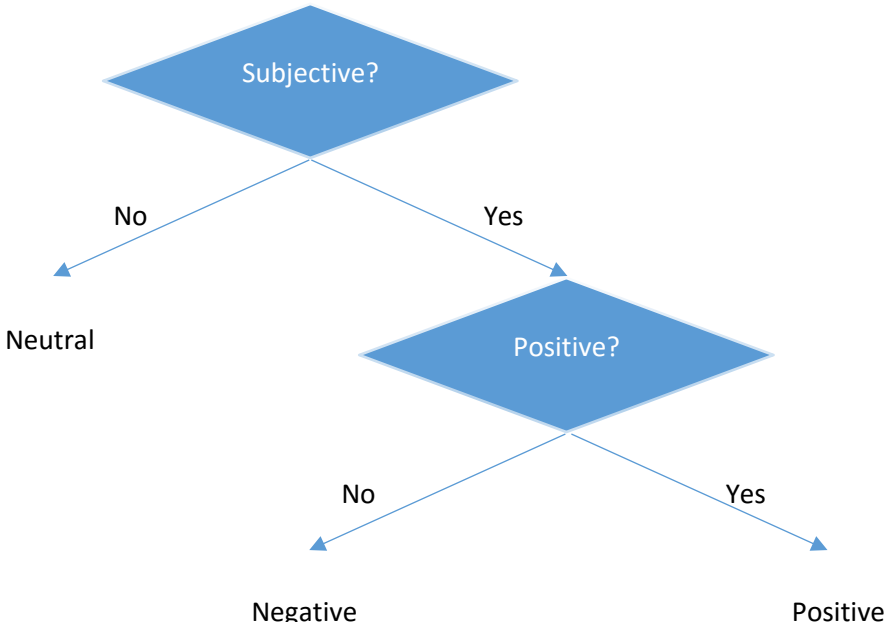


Figure 20 Hierarchical Classification

For classification Naive Bayes and Maximum Entropy classifiers are used. First classifiers are trained using training datasets which contains labelled tweets with relevant classification classes according to its sentiment. Before training the classifier all the data is preprocessed using preprocessing engine to make the consistency with training data and actual data.

4.4.1. NLTK Classifiers

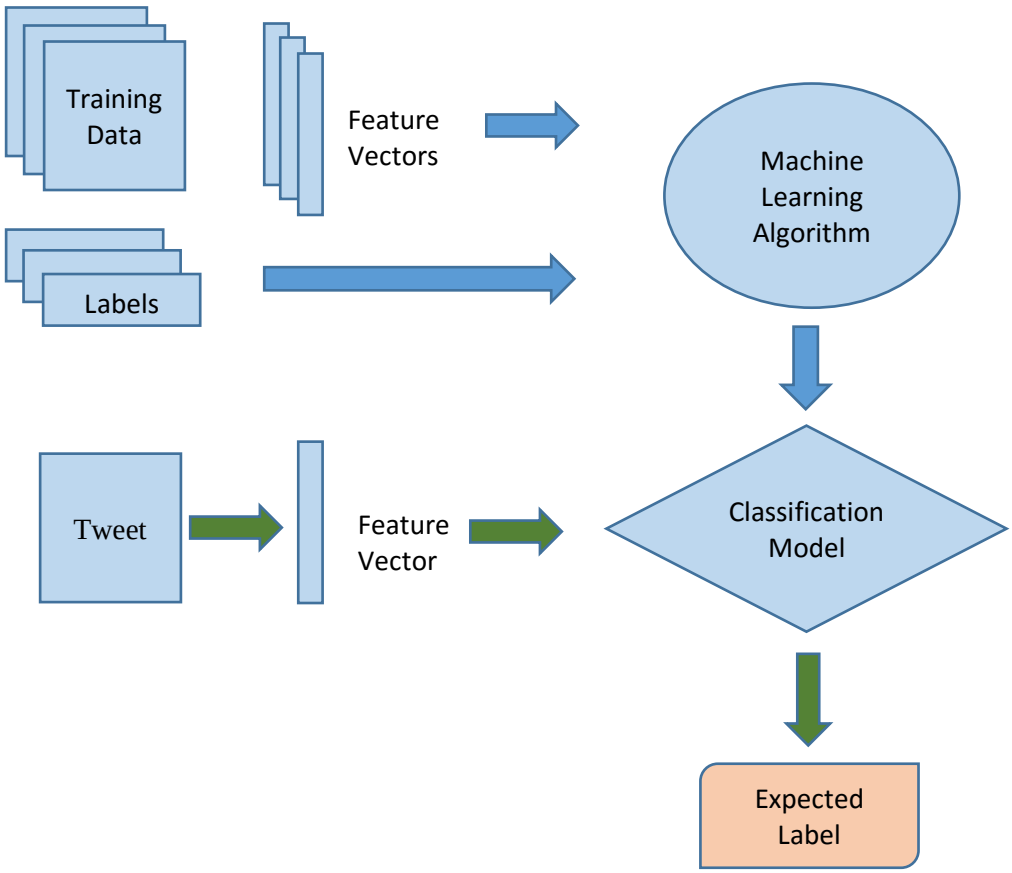


Figure 21 NLTK Classifier Model

The first step in training a classifier is deciding what features of the input are relevant, and how to encode those features. All of the NLTK classifiers [31] work with "featstructs" [32], which can be simple dictionaries mapping a feature name to a feature value. During training, a feature extractor is used to convert each input value to a feature set. These feature sets, which capture the basic information about each input that should be used to classify it. Pairs of feature sets and labels are fed into the machine learning algorithm to generate a model. During prediction, the same feature extractor is used to convert unseen inputs to feature sets. These feature sets are then fed into the model, which generates predicted labels. Therefore selecting relevant features and deciding how to encode them for a learning method can have an enormous impact on the learning method's ability to extract a good model.

4.4.1.1 Feature sets selection

For feature set selection following techniques are used.

- **Stopword Filtering**
  Stopwords are words that are generally considered useless. These words are ignored because they are so common that including them would greatly increase the size of the index without improving precision or recall. NLTK comes with a stopwords [33] corpus that includes a list of 128 English stopwords.

- **Bag of words**
  Simplified bag of words model is used for text where every word is feature name with a value of True. For improve accuracy and remove unwanted indexes and complexity, significant feature selection method (discussed below) is used.

- **N-gram Collocation**
  Including n-grams will improve classification accuracy. The hypothesis is that people say things like "not great", which is a negative expression that the bag of words model could interpret as positive since it sees "great" as a separate word. So using n-grams for feature selection will avoid those problems. Therefore bigrams and trigrams are used for feature selections.

- **High information Feature Selection**
  If all the n-grams are used as features, many of the features will have low information since these are features that are common across all classes, and therefore contribute little information to the classification process. Individually they are harmless, but in aggregate, low information features can decrease performance. Eliminating low information features gives our model clarity by removing noisy data. It can save from over fitting and the curse of dimensionality. We used only the higher information features, since it increases the performance while also decreasing the size of the model, which results in less memory usage along with faster training and classification.
  To find the highest information features, information gain for each feature is calculated. A word that occurs primarily in positive tweets and rarely in negative tweets is high information. Chi-square method is used for get high information features. NTLK includes with the built in classes for get information gain for each n-gram. In our models only the 10,000 most informative words are used and discarded the rest.
  After considering several classification algorithms and after doing some accuracy testing Naive Bayes classifier is selected as the nltk machine learning classification model for our sentiment classifier. Two Naive Bayes classifiers are used, where one for subjectivity classifier and other one for polarity classifier.

4.4.2. Emoticon Classifier

When people express their feeling in social media, it is customary to use emoticons as a way of expressing their feeling. For identify those sentiments, a classifier using regular expressions is implemented to identify positive and negative emoticons. This improve the accuracy of the classifier.

4.4.3. Stanford Sentiment Analyzer

In the latter part of the project we found The Stanford Natural Language Processing Group [34] has released a new sentiment analysis tool which uses new type of Recursive Neural Network [35] that builds on top of grammatical structures which computes the sentiment based on how words compose the meaning of longer phrases. After some testing it is integrated to our solution.

4.4.3. SentiWordNet Classifier

WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets). SentiWordNet [36] is a lexical resource for opinion mining. SentiWordNet assigns to each synset of WordNet three sentiment scores: positivity, negativity, objectivity. This has added to feature selection of bag of words model of nltk classifier for more accuracy.

As the full solution the NLTK Classifier, Emoticon Classifier and Stanford Sentiment Analyzer are combined with boosting aggregation to improve the accuracy.

4.5. Trend Analysis

Trend analysis is an essential part of the project. For trend analysis we used statistical approach which is called time series analysis [37]. Time series analysis is the process of using statistical techniques to model and explain a time-dependent series of data points. Time series forecasting is the process of using a model to generate predictions (forecasts) for future events based on known past events. Time series data has a natural temporal ordering - this differs from typical data mining/machine learning applications where each data point is an independent example of the concept to be learned, and the ordering of data points within a data set does not matter.

Basic strategy for trend analysis is to collect data from social media as time series data analyze them for future values. Therefore data is collected on weekly basis, with sentiment scores (for twitter data) and number of weekly mentions (for foursquare data) about the places and activities as frequencies. Then data for each place activity pair will fed into forecaster module which will predict and forecast the next week and next month frequencies and update the ontology. Those values will be used to identify the current trending traveling locations.

4.5.1. Preprocessing

Since data may contain missing values for some weeks preprocessing step for handle missing data is needed. Python module pandas [38] which is a data analyzing tool is used to preprocess time series data. First weekly data for place, activity pair will interpolate to handle missing data. Then again resample data on weekly and monthly basis for weekly and monthly predictions.

4.5.2. Time Series analysis models

There are three models are been used in Travel profiling for time series analysis.

4.5.2.1. Holt-Winters seasonal method (HW)

The Holt-Winters seasonal method [39] comprises the forecast equation and three smoothing equations one for the level ℓ_t, one for trend b_t, and one for the seasonal component denoted by s_t, with smoothing parameters α, β^* and γ. Holt-Winters exponential smoothing estimates the level, slope and seasonal component at the current time point. There are two variations to this method that differ in the nature of the seasonal component. The additive method is preferred when the seasonal variations are roughly constant through the series, while the multiplicative method is preferred when the seasonal variations are changing proportional to the level of the series. Since Travel domains exhibit the behavior of seasonal variations are changing proportional to the level of the series multiplicative method is used.

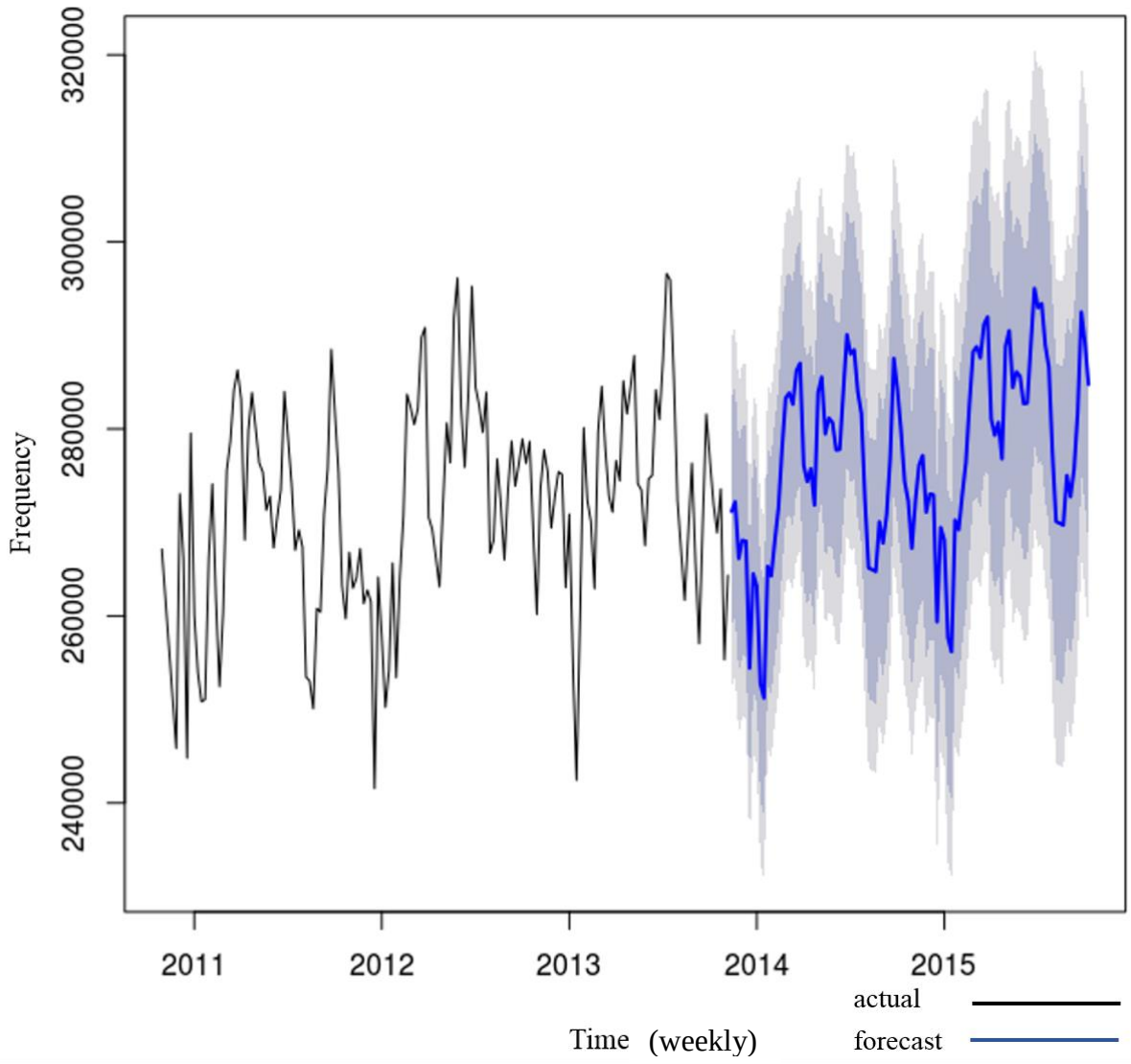


Figure 22 Forecast from Holt-Winters Model

4.5.2.2. Seasonal Autoregressive Integrated Moving Average (SARIMA)

Autoregressive Integrated Moving Average (ARIMA) [40] models provide another approach to time series forecasting. Exponential smoothing and ARIMA models are the two most widely-used approaches to time series forecasting, and provide complementary approaches to the problem. While exponential smoothing models were based on a description of trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data. A Seasonal ARIMA model [41] is formed by including additional seasonal terms in the ARIMA models. This model is used to forecast for monthly data.

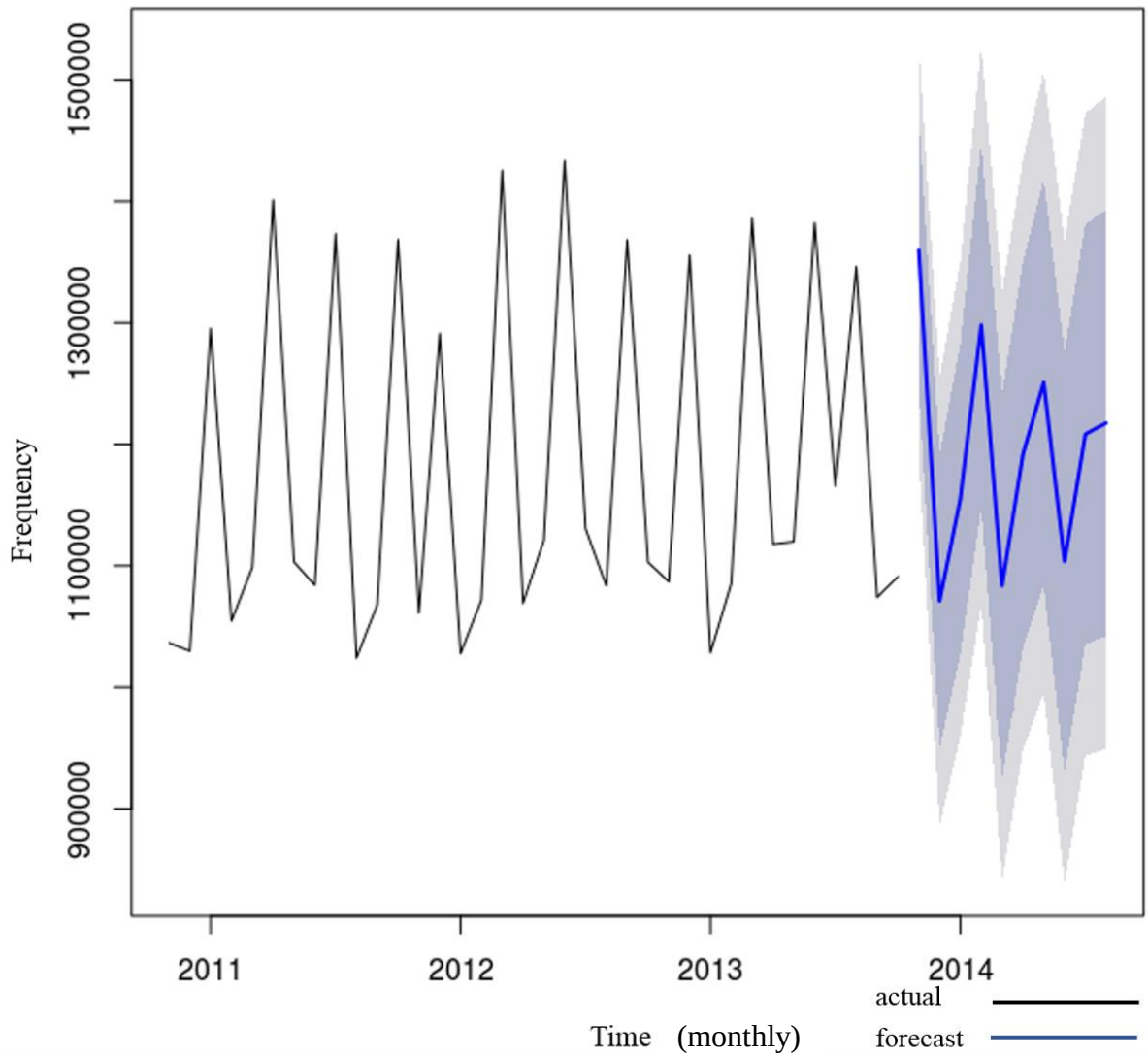


Figure 23 Forecast from ARIMA(2,0,1) with Non-zero Mean

4.5.2.3. Neural Network with Weka time Series Analyser (NN)

Weka's time series framework [42] takes a machine learning/data mining approach to modelling time series by transforming the data into a form that standard propositional learning algorithms can process. It does this by removing the temporal ordering of individual input examples by encoding the time dependency via additional input fields. Various other fields are also computed automatically to allow the algorithms to model trends and seasonality. After the data has been transformed, Weka's neural network "MultilayerPerceptron" algorithm is used to predict both weekly and monthly forecast.

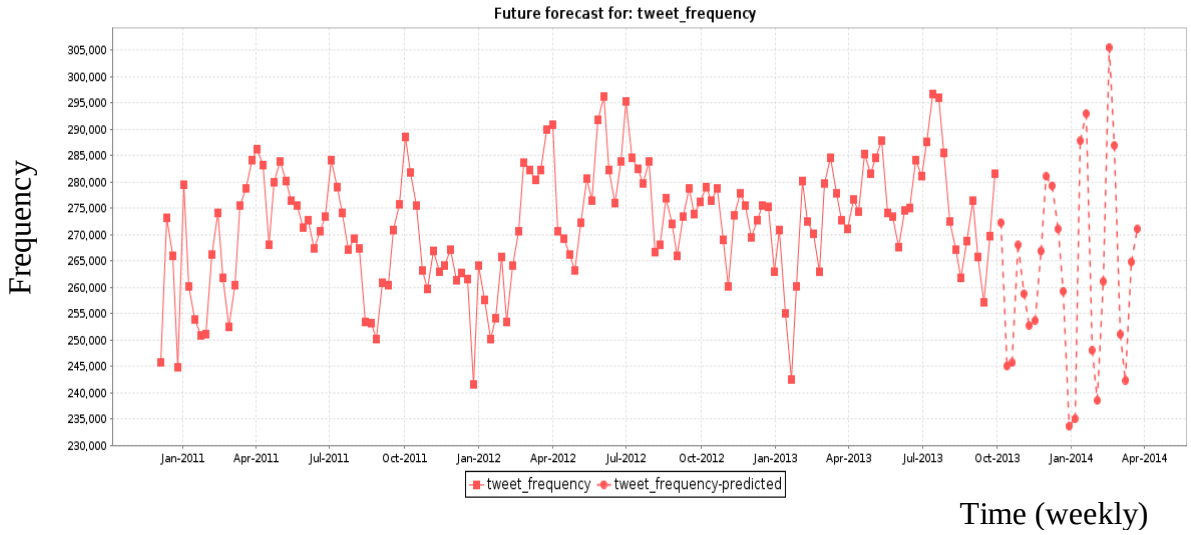


Figure 24 Forecast from Weka's Neural Network

For each place, activity pair datasets models from each method is created and predict using combining all three models. For Implementation of Holt-Winters seasonal method and SARIMA method, R [43] which is an environment and language for statistical computing is used. R interface for python Rpy2 [44] is used to access R from Python. For Neural Network with Weka time Series Analyzer, weka's time series forecast plugin is used. Boosting method is used to aggregate the three models. Python module py4j [45] is used to access Java objects from Python

4.5.3. Improve Performance

Since fetching data from database and creating models takes time, this process takes time. Therefore performance improvement is mandatory as handling big data. Multiprocessing parallel system is implemented to overcome performance issues. Whole process is broken in to three major layers where as

4.5.3.1. Data Fetching Layer (DFL)

Handles data fetching from database. DFL layer contains two parallel processes which fetch data from frequency databases and add them into input queue. If the input queue is full these processes will sleep until a space is available.

4.5.3.2. Data Analyze layer (DAL)

DAL layer contains with two parallel processes. The fetched data in the input queue are analyzed by those processes using time series analyzing models which described above and after analyzing them, analyzed data is fed into output queue.

4.5.3.3. Data Output Layer (DOL)

Update the ontology according to the analyzed data by calling the Travel profiling web service.

Each layer consists with multiple processes to handle the data and using shared FIFO queue for communication between adjacent layers. Python multiprocessing module [46] which is a rich library for handle multi process system is used for implantation of the parallel system

4.6 Ontology

In our travel profiling application we had a requirement to represent extracted information from social media in a domain knowledge base. Because of the research already had done by Code gen, we stick to representing domain knowledge in an OWL Ontology. OWL was the most sophisticated domain knowledge representation technology available at the time of our project implementation. It supports relationship reasoning where it does not require to state those relationships explicitly.

OWL can identify and reason on various properties such as Functional, Inverse Functional, Transitive, Symmetric, Asymmetric, Reflexive and Irreflexive. For example, it can indicate that IF "A isMarriedTo B" then this implies "B isMarriedTo A", here the property "isMarriedTo" should be defined as a symmetric property. Or that if "C isAncestorOf D" and "D isAncestorOf E" then "C isAncestorOf B", here the "isAncestorOf" property should be defined as a transitive property.

The ontology that we created in this project is a domain knowledge base which represents location information such as Continents, Countries, what countries are in each Continent, what are the regions in each country, what are the sub regions inside country regions and what are the places and activities available inside regions. After representing data in an Ontology, later we have to query the knowledge as well. Therefore it requires some additional reasoning to be done. For an example identifying all the places inside a region which requires browsing through all the sub regions and identifying their regions as well. Considering all these requirements, we started to create an OWL ontology using the given Ontology by codegen.

The ontology given by codegen had lot of unwanted classes and some of their properties were also irrelevant. Therefore we removed those Classes, Properties and made a simplified Ontology structure.

4.6.1 Ontology Classes

For the travel profiling application ontology we wanted to represent 5 different types of objects such as Continents, Countries, Regions, Places and Qualifiers. The relationship of those objects is as follows.

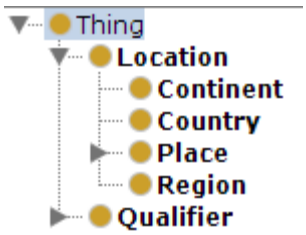


Figure 25 Ontology Class Hierarchy

Inside continents there can be one of more Countries. Inside Countries there can be one or more Regions. In side Regions also there can be Regions. Regions has Places such as Hotels, Beaches and Mountains. These places can have Qualifiers such as Mountain biking, Surfing, Rock climbing and so on. The object type Place has set of subclasses to specialize each available different types of places. Following is the complete hierarchy.

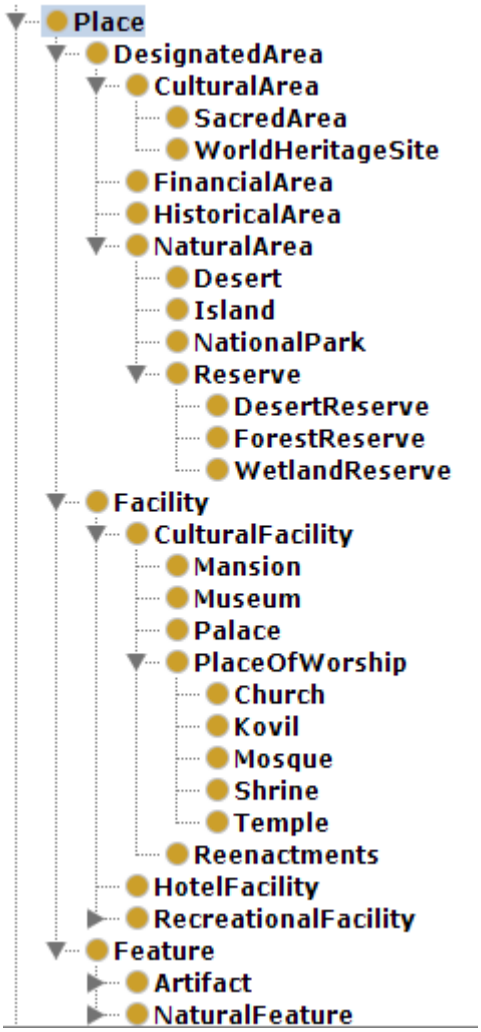


Figure 26 Ontology Class Hierarchy of type Place

4.6.2 Ontology Properties

In the construction of the Ontology we have used two types of properties, Data type properties and Object Properties. Object properties are to describe relationships between individuals. Following are the Object properties that we have used.

4.6.2.1 Object Properties

Following is the complete list of Object Properties,

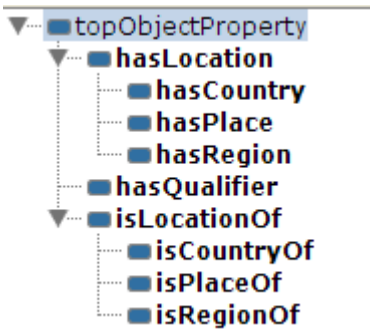


Figure 27 Ontology Object Properties

hasLocation property has child properties hasCountry, hasPlace and hasRegion. Here the hasLocation is used as the parent property because when we want to find places in side a Continent or a Region we can use this property instead of using separate properties hasCountry, hasRegion and hasPlace. We can get all the relationships using the single property hasLocation. All of the inverse properties of the child relationships of hasLocation is listed under isLocationOf property and the inverse property of hasLocation itself is the isLocationOf property.

“hasCountry” relationship defines relationships between continents and Countries. It has the domain Continent and the range is Country.
The domain of hasRegion property is Region or Country and the range is Region. Therefore it indicates relationships between Countries, Regions and Regions, Regions. For an example Consider Country Sri Lanka and regions Kandy and Wattegama. Using the hasRegion property we can write down relationships as follows,
Sri Lanka hasRegion Kandy, Kandy hasRegion Wattegama

“hasPlace” property indicates the static places inside a region. Here static places means places such as Museums, Airports, Cinema Halls, and Stadiums and so on. The domain of “hasPlace” property is Region and its range is class type Place and it's child types.

The property “hasQualifier” indicates types of activities available inside a particular region. It indicates whether a particular place has what types of activities to do. If a place has Adventure activities to do, we indicate it as that place has Adventure qualifier. Likewise places can have qualifiers such as Diving, Rafting, Racing, Swimming, Bird Watching, Golf, and Picnic and So On. The domain of “hasQualifier” property is domain of type region and the range of it is Qualifier types.

4.6.2.2 Data Properties

In the TravelProfiling ontology we had to use different Data type Properties to store different trend values. As for the current status we used mainly 2 social media sources (Twitter, Foursquare) to update the trend values in the ontology. Therefore for we used different data type properties to hold trend values derived from different sources. Following is the list of data type properties.

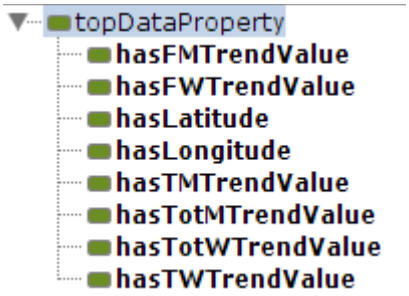


Figure 28 Ontology Data Properties

Here “hasTWTrendValue” and “hasTMTrendValue” are used to store the trend values respectively weekly trends and monthly trends derived from Twitter. “hasFWTrendValue” and “hasFMTrendValue” are used to store the trend values respectively weekly trends and monthly trends derived from Foursquare. “hasTotWTrendValue” and “hasTotMTrendValue” are used to store the combined weekly trend values and monthly trend values derived from twitter and Foursquare. Whenever Twitter or Foursquare, monthly or weekly trend value is changed, application updates the Total weekly and monthly trend values using the application business logic. The logic is basically to get the average values from both Twitter and Foursquare. If one values is 0 or missing we will consider the available weekly or monthly value as the total value.

All these properties which are used to store the trend values have the Domain Qualifier (Activity) and Place, and have the range integer.

Furthermore we have used two properties namely “hasLatitude” and “hasLongitude” to store the longitude and latitude values of a region. Those two properties have the domain Region and range double.

4.6.3 Changes in the Ontology Structure

Even though we designed the Ontology as described above, we had to face for problem when using it. Initially we wanted to populate the ontology to create a static base so that we can update the existing data using real time social data feeds. To do this we used Wikitravel data dumps. Using WikiTraveler data dumps we extracted the existing locations and the places, activities inside those locations. The problem we faced here is the matching of activity (qualifier) to a place. We were only able to identify the places and Activities separately not the relationships between them. Therefore we had to change the ontology so that there is not relationship between a place and a qualifier. In the newly created ontology this relationship exists, a region has places and also qualifier, Place type individual does not have any relationship with Qualifier types.

4.7 Querying the ontology

Once we have populated the ontology we have to use a query engine to get the data from the ontology. Query engines uses reasoning before output the results. That means, reasoning will give the largest set of individuals matching the query. For an example, let's consider we want to find Adventure type activities available inside a Country. When we query this information, a reasoner will look in to all the regions available inside that particular country and look in to the available activities of type Adventure. If there are more regions available inside a direct region of the country, the reasoner will look into those as well. This is even possible because we have created the “hasRegion” Object property as a Transitive property. This shows how important is to have a good reasoner.

To query the OWL ontology we used the OWL API java library. The text processing part in TravelProfiling project was done using python language, for the querying task also we wanted to use a python library but there were no matured python libraries available. Therefore we decided to stick to Java OWL API [30] library. In the Java OWL API library we can use third party reasoning libraries. OWL API itself has a reasoner factory called Structural Reasoner Factory but it is only useful for simple queries. The mostly used third party reasoning libraries are HermiT [47], FaCT++ [48], Pellet [49] and ELK reasoners [50]. These third party libraries each has been optimized for a particular task.

4.7.1 Reasoners and their performances

To perform a POC on querying results from Ontology we used the Ontology which was populated using WikiTravel Data dumps. It was about 30MB of size and there were around 20,000 individuals. Following are the specifications of the computer we used to test the reasons,

CPU - Intel Core i5-2410M CPU 2.30GHz
RAM – 4GB

HermiT reasoner - HermiT is the first publicly-available OWL reasoner based on a novel “hypertableau” calculus which provides much more efficient reasoning than any previously-known algorithm. Ontologies which previously required minutes or hours to classify can often by classified in seconds by HermiT, and HermiT is the first reasoner able to classify a number of ontologies which had previously proven too complex for any available system to handle. But even though HermiT is optimized for efficiency, it could not initialize property for our populated ontology. There are 3 steps to create an OWL reasoner. Following is the code sample of that procedure.

```
OWLReasonerFactory reasonerFactory = new Reasoner.ReasonerFactory();
OWLReasoner reasoner = reasonerFactory.createReasoner(ontology);
reasoner.precomputeInferences();
```

The code didn't execute below the third line, because there was an Exception in thread "main" java.lang.OutOfMemoryError: Java heap space. This is due to the memory shortage in Java heap space. We tried increasing the Java heap size up to 3GB but still we got the error and couldn't move beyond that due to hardware limitations. Apart from that memory limitation error, the reasoner took lot of time until it gives an error, this shows how much the reasoner is not compatible with the Ontology that we have created.

FaCT++ reasoner - FaCT++ is a new version of implementation of well-known FaCT OWL-DL reasoner. It is implemented using C++ language for fast execution and using the established FaCT algorithm but with a different internal architecture. Even with FaCT++ reasoner Java Heap size was not enough and it was too slow, even to raise the exception.

Pallet reasoner – Pallet reasoner is another OWL reasoner. It supports OWL 2.0 and also OWL DL reasoning in addition to HermiT and FaCT++. Pallet reasoner was also failed to initialize raising an out of memory exception. We couldn't resolve that with the limitation of Hardware we have.

ELK reasoner – Because of the inability to support large Ontologies, we tried some other reasoners which has native support for large ontologies. ELK reasoner is designed to perform very fast reasoning on OWL EL. OWL EL profile is a subset of OWL 2, and it particularly suitable for applications employing ontologies that define very large numbers of classes and/or properties. But even with the ELK reasoner we couldn't get much luck because it also required a high heap size than 3GB. Apart from the heap size it was also pretty slow. Therefore we couldn't use that as well.

4.7.2 Optimizing the Ontology reasoning

After the results of available reasoners and their performance, we identified another problem with OWL reasoners. That is, in TravelProfiling application we have an Ontology which updates periodically with the extracted data from social media. Therefore even if we use a reasoner with a higher heap size (more than 3GB), it will take some time to initialize the reasoner each time we update the Ontology. Initializing the reasoner is important because otherwise it will not consider newly update data when querying from the Ontology.
Because of these limitations we moved to another querying option which is to use RDF (Resource Description Framework) and then use SPARQL queries to retrieve domain knowledge. We use Apache Jena Java library to query using SPARQL. Since we had initially created our Ontology in OWL format we had to change it to RDF format using Protégé Ontology editor.
We tested SPARQL querying using apache Jena library with the populated Ontology. Following are the some of the results,

* Query a set of same type individuals: Get all the region type individuals
The query engine returned a 37,293 size of list of individuals in 440 milliseconds
* Query an available set of size 10 activities of a type in a region: Get 10 Adventure type activities in side England
The query engine returned results in 89485 milliseconds.

Therefore we can identify that, SPARQL engine is also not optimized for querying large Ontologies. The time taken to query a set of results vary according to the number of individuals available in the particular set and the subjective properties associated with them. But according to our requirement it's is the best option available and we decided to use that.

4.8 TravelProfiling Web Service

Following diagram show the overview of the TravelProfiling web service implementation.

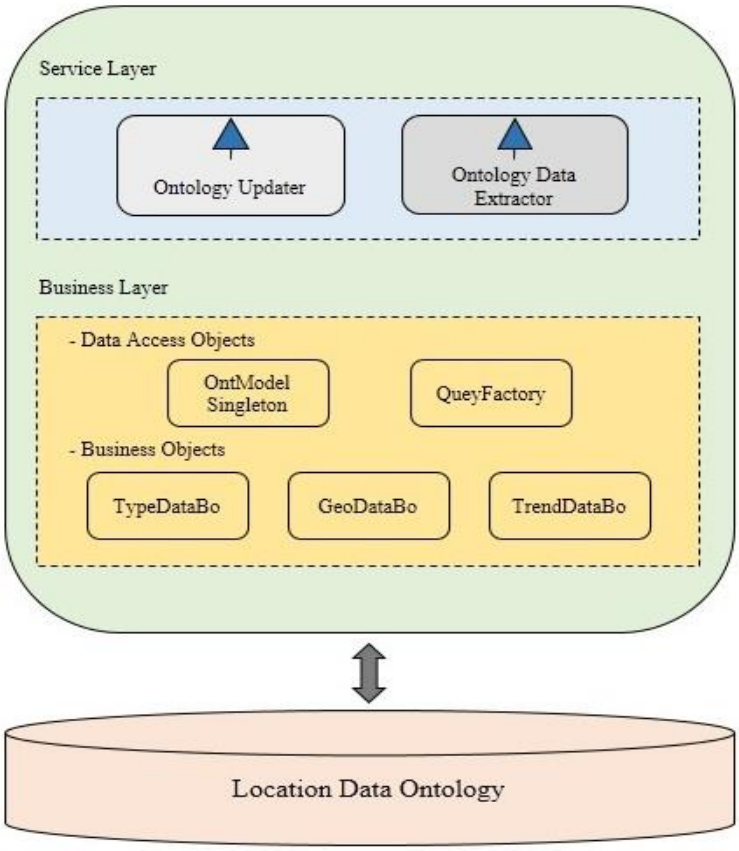


Figure 29 Travel Profiling Web Service Overview

The application was implemented as a RESTFUL web service using Jersey Java Library. The application was hosted on Apache Tomcat 6. Mainly the TravelProfiling web service has two components, which Ontology Updater and Ontology Data Extractor. All the service end points supports XML and JSON outputs.

4.8.1 Ontology Data Extractor

This is the web interface that is used to query data from Ontology. The query interface supplies 5 services.

I. Geo Coordinates

URL: localhost:8080/TravelDataWebService/rest/ontodata/geocodes/{region}

This service will provides Longitude and Latitude values for a given “region”
Sample XML output:

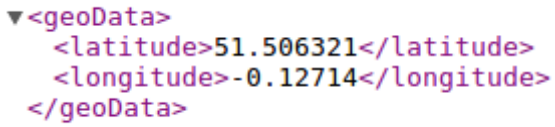


Figure 30 XML response for Geo Coordinates

II. Geo List

URL: localhost:8080/TravelDataWebService/rest/ontodata/geolist

This service will provide all the regions and their Longitudes and Latitudes as a list.
Sample XML output:

Figure 31 XML Response for Geo List

III. Type Data

URL: localhost:8080/TravelDataWebService/rest/ontodata/typedata/{type}

This service will output information about a particular Place type or an Activity type. The information are immediate subclasses of the particular type and its top 7 instances with highest trending values.
Sample XML output:

Figure 32 XML Response for Type Data

IV. Trends around

URL:
localhost:8080/TravelDataWebService/rest/ontodata/trendsaround/{region}/{type}/{isActivity}

This service is used to get the trending Activity places and Locations of a type inside a region.
For an example if we want to find the trending Beaches inside Colombo area, we can query
TravelDataWebService/rest/ontodata/trendsaround/Colombo/Beach/false

Sample XML output:

Figure 33 XML Response for Trending Places/Activities in a region

V. Trending regions for a particular activity

URL:
localhost:8080/TravelDataWebService/rest/ontodata/trendingregionsfor/{type}/{isActivity}

This service outputs the top regions which has top most trends for a particular activity or place type.
For an example consider if someone wants to find the top regions for Adventure activities, then we can query.
TravelDataWebService/rest/ontodata/trendingregionsfor/Adventure/true

Sample XML output:

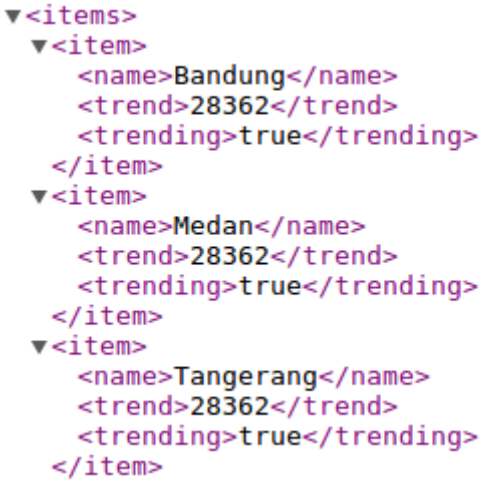


Figure 34 XML Response for regions of particular Activity/Place type

4.8.2 Ontology Updater

This is a web interface provided to update the ontology trend values from the data gathered from social media. The Ontology Updater interface has two services, Update trends and finalize.

I. Update trends

URL:
localhost:8080/TravelDataWebService/rest/updatetrends/{isTwitter}/{region}/{isActivity}/{type}/{MonthlyTrend}/{WeeklyTrend}

Using this Link we can update monthly and weekly trend values derived from Twitter and Foursquare. This will update the Ontology model in the memory.

Output of a successful update will be a "Done" String and if anything goes wrong "Error" String will be returned.

II. Finalize

URL: localhost:8080/TravelDataWebService/rest/updatetrends/finalize

This service is used to save the Ontology model to local file system. After repeatedly calling the Update trends web service one can save the Ontology model to local file system by calling the finalize web service.

Output of a successful update will be a "Done" String and if anything goes wrong "Error" String will be returned.

The reason to use two web services is to speed up the update process.

4.8.3 OntModelSingleton

This Object give the singleton access to other objects which requires Ontology Model object. Ontology model object is used to query the Ontology (using query factory) and also to add new changes to the ontology. When the application is running, there can be situation where third parties query from the ontology and also the trend analyzer updates the ontology. If there are two ontologies used, there can be situation where updates are not present instantly in the ontology. To avoid such situations we use this approach.

4.8.4 Query Factory

This object is used from the Jena Library and it was used to query the Ontology. This model is created on demand.

4.8.5 Business Objects

Business Objects are used to transfer data within the application and to transfer return values to clients. Jersey REST framework provides very useful tools to implement this, where by just using a single line of code, we can convert the return object to a JSON or XML.

Following is the code sample:

```
3  import javax.xml.bind.annotation.XmlRootElement;
4
5  @XmlRootElement
6  public class GeoDataBo {
7
8      private String region;
9
10     public String getRegion() {
11         return region;
```

Figure 35 Syntax for output as XML/JSON object

4.8.6 Location Data Ontology

We have used Protégé Editor to design the Ontology in RDF/XML syntax. We use Apache Jena [37]Library to query the ontology and update new values.

4.9 Dashboard

After implementing our system, our next step is to provide interaction for users toward our system. Our main outcome for users is facilitating querying through our API. But sometimes querying becomes complex for users and we decided to breakdown querying to several api calls. We decided on breaking these to several API calls depending on data that we provide. But we can provide more insights from our data and we implemented a Dashboard web application. Functionality of each page of the Dashboard is described as follows.

4.9.1 Travel Data Statistics

Following diagram shows the stat page of our Dashboard. Task of each dashboard indicator is as follows.

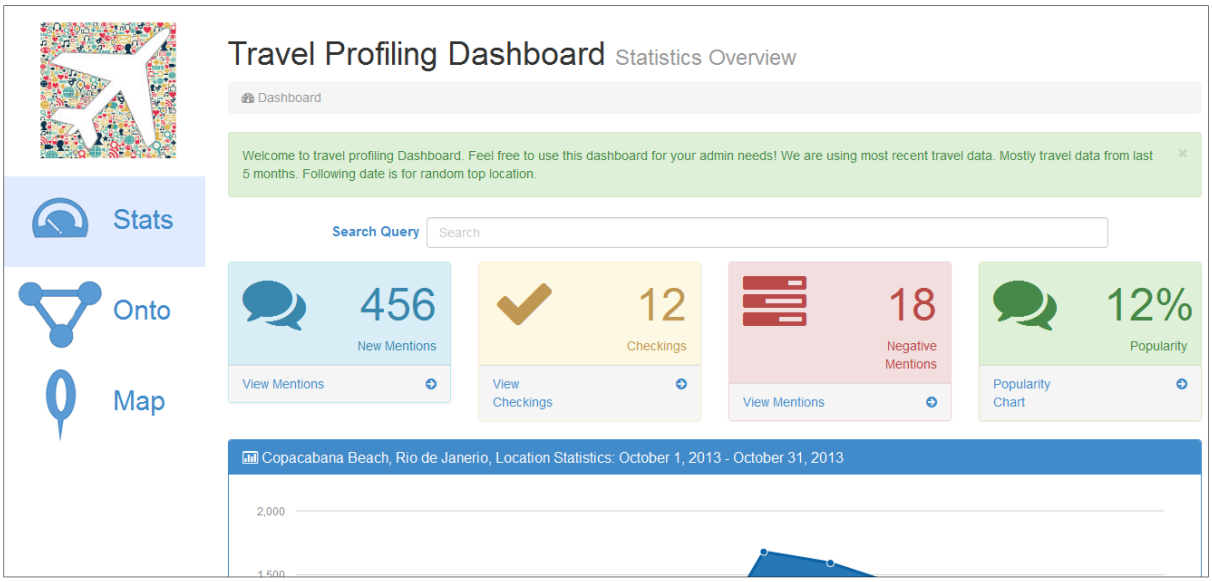
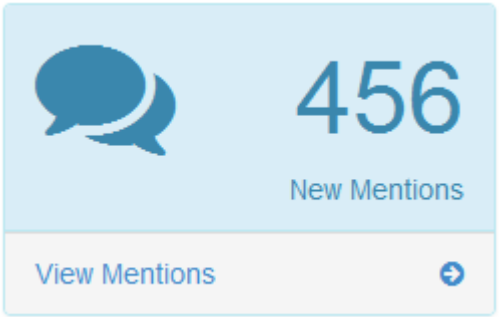


Figure 36 Statistics Page

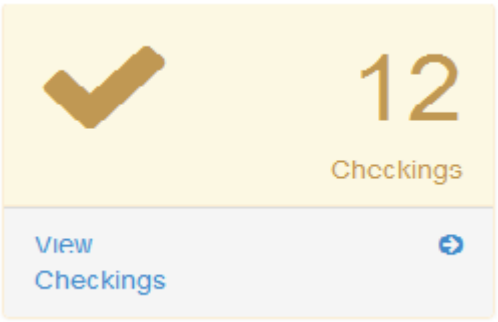
4.9.1.1 Mini Indicators

Following are some data indicators.



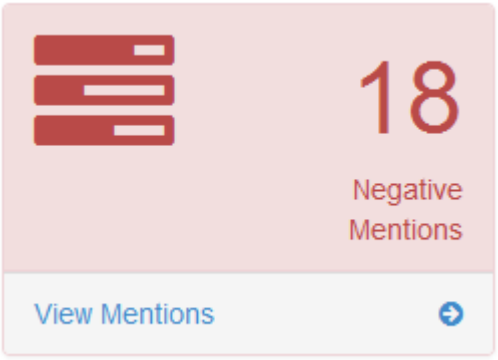
This indicator indicates new mentions compared to Previous week.

Figure 37 New Mentions



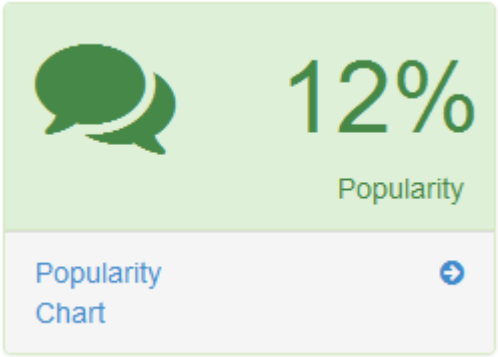
This indicator shows how many new geo tagging for the current location.

Figure 38 New Checkings



This indicator shows how many new negative mentions for this week. But unfortunately we do not have enough negative mentions or we are not detecting negative mentions thoroughly.

Figure 39 Negative Mentions



This indicates up to what extend people given positive sentiment about this activity or place compared to previous week as a percentage.

Figure 40 Popularity Chart

4.9.1.2 Time Series Graph

This graph indicates time series data for a history of 15 weeks.

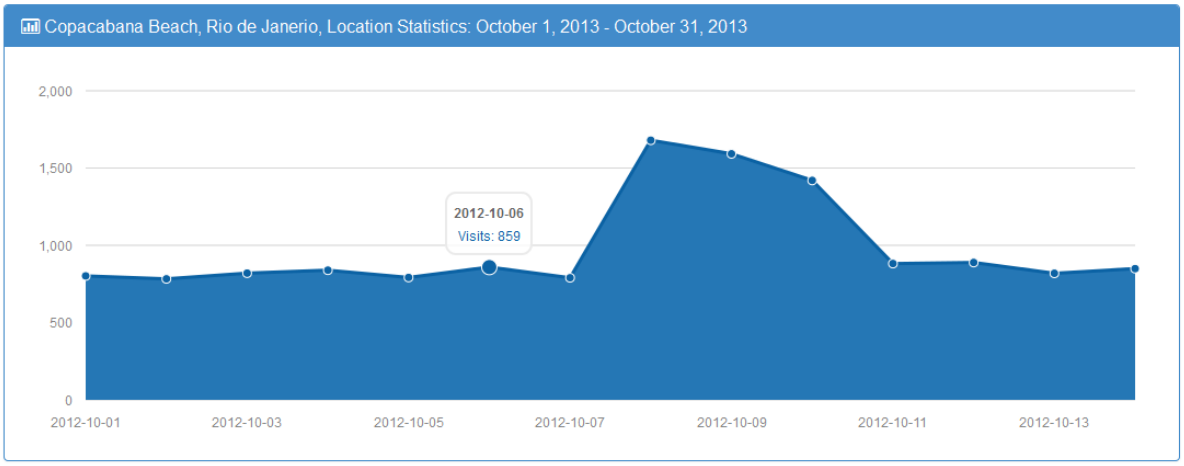


Figure 41 Time Series Graph

4.9.1.3 Comparison Chart

Following pie chart represents a comparison for a given activity or a place for a given region.

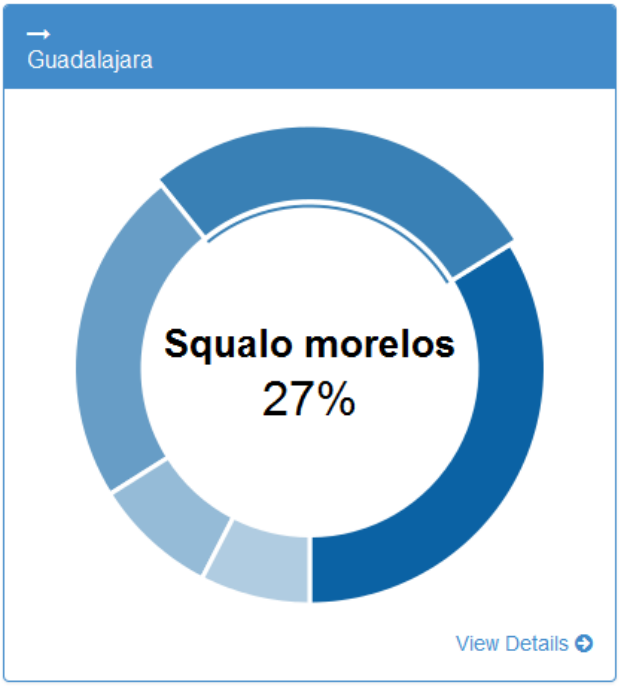


Figure 42 Comparison Pie Chart

4.9.1.4 Top trending destinations

Next we have a table which shows top trending travel locations for current week.

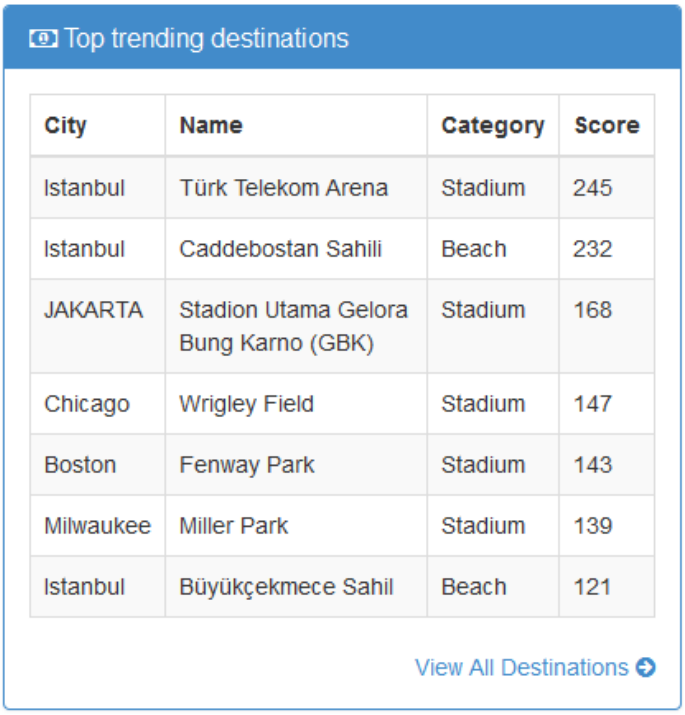


Figure 43 Top Trending Table

4.9.2 Ontology View

This is a highly interactive animated view to show our ontology which is our main resource. This will facilitate user to expand and collapse ontology classes and explorer its instances. Each edge in this graph represents a relationship and node represents a class or an instance.

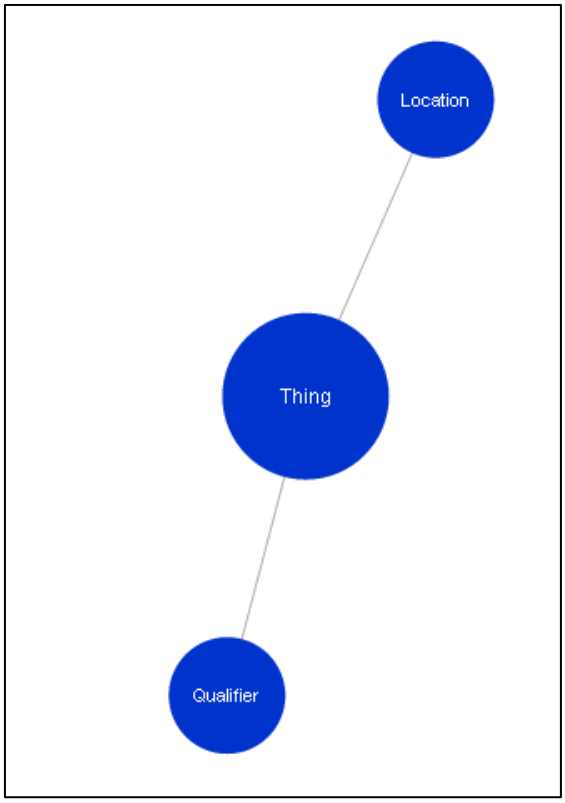


Figure 44 Ontology View

4.9.3 Map View

Next we have map view for our project. This will geographically shows trending locations in the map.

Figure 45 Map

5. Conclusion

In our research project we tried several methods to extract travel related data from social media and represent it in a structured way using an ontology. We were able to accomplish the aforesaid task for certain extent using twitter API and Foursquare API and Travel wiki as data sources. But facebook was not taken as a data source since facebook API doesn't support public data streaming due to privacy issues. Apart from that Twitter even provide limited number of random stream for particular user API key.

Natural language processing in unstructured social media text made our life hard when we try to find sentiment when people use sarcasm and ironies. Existing training corpora doesn't match with our domain. Nevertheless annotating a new corpus is time consuming and need more cost.

While processing data, we need more processing power and more disk space when the collection of data grows continuously. Representing mass amount of data in an ontology is also problematic by means of resource problems like heap size limitations when querying with available reasoners.

Furthermore for trend analysis we need at least data for one year in order to get good accuracy. But we had little amount of data which cannot give better forecasting.

Despite above problems we were able to provide a web application where users can query certain location good for certain activity types. Then it provides trending locations as results. As a research project, we could observe both negative and positive results as above while addressing our research problem.

6. Reference

[1] H. Tsukayama, March 2013. [Online]. Available: http://articles.washingtonpost.com/2013-03-21/business/37889387_1_tweets-jack-dorsey-twitter.

[2] "The Stanford Parser: A statistical parser," [Online]. Available: http://nlp.stanford.edu/software/lex-parser.shtml.

[3] "RelEx Dependency Relationship Extractor.," [Online]. Available: http://wiki.opencog.org/w/RelEx_Dependency_Relationship_Extractor.

[4] "The Department of Linguistics at the University of Pennsylvania.," [Online]. Available: http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

[5] "Natural Language Toolkit.," 2012. [Online]. Available: http://nltk.org. [Accessed 23 04 2013].

[6] "Lorg. Algorithm.co.il.," [Online]. Available: http://www.algorithm.co.il/blogs/programming/python/cheap-language-detection-nltk/.

[7] "ENGLISH STOPWORDS," [Online]. Available: http://www.ranks.nl/resources/stopwords.html.

[8] "Stemming Algorithms," [Online]. Available: http://xapian.org/docs/stemming.html.

[9] "JavaRAP," [Online]. Available: http://aye.comp.nus.edu.sg/~qiu/NLPTools/JavaRAP.html.

[10] M. P. M. A. Kabadjov, "A General-Purpose, off-the-shelf Anaphora Resolution Module," LREC, 2004.

[11] "protégé," [Online]. Available: http://protege.stanford.edu/.

[12] "Mining Social Media: A Brief Introduction," [Online]. Available: http://www.public.asu.edu/~pgundech/book_chapter/smm.pdf.

[13] "100 more social media statistics for 2012," [Online]. Available: http://thesocialskinny.com/100-more-social-media-statistics-for-2012/.

[14] B. Liu, "OPINION MINING," [Online]. Available: http://www.cs.uic.edu/~liub/FBS/opinion-mining.pdf.

[15] "Open Shift," [Online]. Available: https://www.openshift.com/products/online.

[16] "Foursquare Venues API," [Online]. Available: https://developer.foursquare.com/overview/venues.html.

[17] "Open Shift Cron Jobs," [Online]. Available: https://www.openshift.com/blogs/getting-started-with-cron-jobs-on-openshift.

[18] A. Rodriguez, "RESTful Web services: The basics," IBM, 06 November 2008. [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-restful/.

[19] Deborah L. McGuinness, Frank van Harmelen, "OWL Web Ontology Language Overview," 10 February 2004. [Online]. Available: http://www.w3.org/TR/owl-features/.

[20] "Resource Description Framework (RDF)," 10 February 2004. [Online]. Available: http://www.w3.org/RDF/.

[21] Eric Prud'hommeaux, Andy Seaborne, "SPARQL Query Language for RDF," 15 January 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/.

[22] "PyEnchant," [Online]. Available: http://pythonhosted.org/pyenchant/.

[23] "WikiTravel," [Online]. Available: http://wikitravel.org/en/Main_Page.

[24] "Virtual Tourist," [Online]. Available: http://www.virtualtourist.com/.

[25] "Trip Advisor," [Online]. Available: http://www.tripadvisor.com/.

[26] "Gate," [Online]. Available: http://gate.ac.uk/.

[27] "JAPE," [Online]. Available: http://gate.ac.uk/sale/tao/splitch8.html#chap:jape.

[28] "ANNIE," [Online]. Available: http://gate.ac.uk/sale/tao/splitch6.html#chap:annie.

[29] "Sesame," [Online]. Available: http://www.openrdf.org/.

[30] "The OWL API," [Online]. Available: http://owlapi.sourceforge.net/.

[31] "nltk classify package," NLTK, [Online]. Available: http://nltk.org/api/nltk.classify.html.

[32] "NLTK featstruct module," NLTK, [Online]. Available: http://nltk.googlecode.com/svn/trunk/doc/api/toc-nltk.featstruct-module.html.

[33] "NLTK English stopwords," NLTK, [Online]. Available: http://nltk.googlecode.com/svn/trunk/doc/api/nltk.corpus-module.html#stopwords.

[34] "Stanford CoreNLP home page," The Stanford Natural Language Processing Group, [Online]. Available: http://nlp.stanford.edu/software/corenlp.shtml.

[35] "Stanford Sentiment Analysis," [Online]. Available: http://nlp.stanford.edu/sentiment/.

[36] "SentiWordNet," [Online]. Available: http://sentiwordnet.isti.cnr.it/.

[37] "Time series analysis," [Online]. Available: http://en.wikipedia.org/wiki/Time_series. [Accessed 01 12 2013].

[38] "Pandas Home Page," Pandas, [Online]. Available: http://pandas.pydata.org/.

[39] "Holt-Winters seasonal method," [Online]. Available: https://www.otexts.org/fpp/7/5.

[40] "Introduction to ARIMA," [Online]. Available: http://people.duke.edu/~rnau/411arim.htm.

[41] "Seasonal ARIMA models," [Online]. Available: http://people.duke.edu/~rnau/seasarim.htm.

[42] "Time Series Analysis and Forecasting with Weka," [Online]. Available: http://wiki.pentaho.com/display/DATAMINING/Time+Series+Analysis+and+Forecasting+with+Weka.

[43] "The R Project for Statistical Computing," [Online]. Available: http://www.r-project.org/.

[44] "Introduction to rpy2," [Online]. Available: http://rpy.sourceforge.net/rpy2/doc-2.1/html/introduction.html.

[45] "Py4J - A Bridge between Python and Java," [Online]. Available: http://py4j.sourceforge.net/.

[46] "multiprocessing - Process-based “threading” interface," [Online]. Available: http://docs.python.org/2/library/multiprocessing.html.

[47] "Hermit OWL Reasoner," [Online]. Available: http://hermit-reasoner.com/.

[48] D. Tsarkov, "FaCT++," [Online]. Available: http://owl.man.ac.uk/factplusplus/.

[49] "Pallet : OWL 2 Reasoner for Java," [Online]. Available: http://clarkparsia.com/pellet/. [Accessed 10 March 2013].

[50] "ELK Reasoner," [Online]. Available: https://code.google.com/p/elk-reasoner/.

[51] O. Streibel, "Trend Mining with Semantic-Based Learning".

[52] C. Pring, 2 2012. [Online]. Available: http://thesocialskinny.com/100-more-social-media-statistics-for-2012/.

[53] . M. Poesio and . M. A. Kabadjov, "A general-purpose, off-the-shelf anaphora resolution module: Implementation and preliminary evaluation.," in *Conference on Language Resources and Evaluation*, Lisbon, 2004.

[54] Lorg. [Online]. Available: http://www.algorithm.co.il/blogs/programming/python/cheap-language-detection-nltk/.

[55] M.-Y. K. T.-S. C. Long Qiu, "A Public Reference Implementation of the RAP," in *Fourth International Conference on Language* , 2004.

[56] B. Liu, 2008. [Online]. Available: http://www.cs.uic.edu/~liub/FBS/opinion-mining.pdf.

[57] A. M. Kaplan and M. Haenlein, "Users of the world, unite! The challenges and," *Business Horizons,* pp. 59-68, 2010.

[58] M. P. M. A. Kabadjov, "A General-Purpose, off-the-shelf Anaphora Resolution Module," in *LREC*, 2004.

[59] P. Gundecha and H. Liu, 2012. [Online]. Available: http://www.public.asu.edu/~pgundech/book_chapter/smm.pdf.

[60] M. E. Eugene Charniak, "EM works for pronoun anaphora resolution," in *EACL*, 2009.

[61] S. Bird, E. Klein and E. Loper, " Named Entity Recognition," 2010.

[62] L. Backstrom, D. Huttenlocher, J. Kleinberg and X. Lan, "Group formation in large social networks: membership, growth, and evolution," in *12th ACM SIGKDD international conference on Knowledge discovery and data mining* , Philadelphia, 2006.

[63] R. Alan, C. Sam, Mausam and E. Oren, "Named Entity Recognition in Tweets:An Experimental Study," in *Empirical Methods in Natural Language Processing* , Edinburgh, 2011.

[64] [Online]. Available: http://xapian.org/docs/stemming.html.

[65] [Online]. Available: http://nlp.stanford.edu/software/lex-parser.shtml.

[66] [Online]. Available: http://nlp.stanford.edu/software/lex-parser.shtml.

[67] [Online]. Available: http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

[68] "Social Media Growth".

[69] April 2012. [Online]. Available: http://wiki.opencog.org/w/RelEx_Dependency_Relationship_Extractor.

[70] [Online]. Available: http://www.ranks.nl/resources/stopwords.html.

[71] [Online]. Available: http://protege.stanford.edu/.

[72] "Users of the world, unite! The challenges and".

[73] "Apache Jena," [Online]. Available: http://jena.apache.org/.

7. Appendix

Appendix 1 – Twitter public feed response.


```
{
'contributors': None,
       'truncated': False,
       'text': u"Washed up on a beach in Brooklyn NY, where there really aren't very
many horses\n http://t.co/D2JOpK3mKC",
       'in_reply_to_status_id': None,
       'id': 408142508989091840,
       'favorite_count': 0,
       'source': '<ahref="http://www.apple.com"rel="nofollow">iOS</a>',
       'retweeted': False,
       'coordinates': None,
       'entities': {
       'symbols': [

       ],
       'user_mentions': [

       ],
       'hashtags': [

       ],
       'urls': [
              {
              'url': 'http: //t.co/D2JOpK3mKC',
              'indices': [
                     81,
                     103
              ],
              'expanded_url': 'http: //imgur.com/qqMhrYY',
              'display_url': 'imgur.com/qqMhrYY'
              }
       ]
       },
       'in_reply_to_screen_name': None,
       'id_str': '408142508989091840',
       'retweet_count': 0,
       'in_reply_to_user_id': None,
       'favorited': False,
       'user': {
       'follow_request_sent': None,
       'profile_use_background_image': True,
       'default_profile_image': False,
       'id': 2201424232,
       'verified': False,
       'profile_image_url_https': 'https:
//pbs.twimg.com/profile_images/378800000809648072/b8b67ea48368419eb54ca2ae2e557299_nor
mal.jpeg',
       'profile_sidebar_fill_color': 'DDEEF6',
       'profile_text_color': '333333',
       'followers_count': 12,
       'profile_sidebar_border_color': 'C0DEED',
       'id_str': '2201424232',
       'profile_background_color': 'C0DEED',
       'listed_count': 0,
       'profile_background_image_url_https': 'https:
//abs.twimg.com/images/themes/theme1/bg.png',
       'utc_offset': None,
       'statuses_count': 15,
       'description': 'Posting...whatwedobest.',
```

```
        'friends_count': 102,
        'location': '',
        'profile_link_color': '0084B4',
        'profile_image_url': 'http:
//pbs.twimg.com/profile_images/378800000809648072/b8b67ea48368419eb54ca2ae2e557299_nor
mal.jpeg',
        'following': None,
        'geo_enabled': True,
        'profile_banner_url': 'https:
//pbs.twimg.com/profile_banners/2201424232/1385798454',
        'profile_background_image_url': 'http:
//abs.twimg.com/images/themes/theme1/bg.png',
        'name': 'MindlessBanter',
        'lang': 'en',
        'profile_background_tile': False,
        'favourites_count': 0,
        'screen_name': 'MinderBantless',
        'notifications': None,
        'url': None,
        'created_at': 'SatNov3007: 28: 32+00002013',
        'contributors_enabled': False,
        'time_zone': None,
        'protected': False,
        'default_profile': True,
        'is_translator': False
        },
        'geo': None,
        'in_reply_to_user_id_str': None,
        'possibly_sensitive': False,
        'lang': 'en',
        'created_at': 'WedDec0407: 55: 31+00002013',
        'filter_level': 'medium',
        'in_reply_to_status_id_str': None,
        'place': None
}
```

```
{
'contributors': None,
       'truncated': False,
       'text': u"Washed up on a beach in Brooklyn NY, where there really aren't very many
horses\n http://t.co/D2JOpK3mKC",
       'in_reply_to_status_id': None,
       'id': 408142508989091840,
       'favorite_count': 0,
       'source': '<ahref="http://www.apple.com"rel="nofollow">iOS</a>',
       'retweeted': False,
       'coordinates': None,
       'entities': {
       'symbols': [

       ],
       'user_mentions': [

       ],
       'hashtags': [

       ],
       'urls': [
              {
              'url': 'http: //t.co/D2JOpK3mKC',
              'indices': [
                     81,
                     103
              ],
              'expanded_url': 'http: //imgur.com/qqMhrYY',
              'display_url': 'imgur.com/qqMhrYY'
              }
       ]
       },
       'in_reply_to_screen_name': None,
       'id_str': '408142508989091840',
       'retweet_count': 0,
       'in_reply_to_user_id': None,
       'favorited': False,
       'user': {
       'follow_request_sent': None,
       'profile_use_background_image': True,
       'default_profile_image': False,
       'id': 2201424232,
       'verified': False,
       'profile_image_url_https': 'https:
//pbs.twimg.com/profile_images/378800000809648072/b8b67ea48368419eb54ca2ae2e557299_normal
.jpeg',
       'profile_sidebar_fill_color': 'DDEEF6',
       'profile_text_color': '333333',
       'followers_count': 12,
       'profile_sidebar_border_color': 'C0DEED',
       'id_str': '2201424232',
       'profile_background_color': 'C0DEED',
       'listed_count': 0,
       'profile_background_image_url_https': 'https:
//abs.twimg.com/images/themes/theme1/bg.png',
       'utc_offset': None,
       'statuses_count': 15,
```

```
{
'contributors': None,
       'truncated': False,
       'text': u"Washed up on a beach in Brooklyn NY, where there really aren't very many
horses\n http://t.co/D2JOpK3mKC",
       'in_reply_to_status_id': None,
       'id': 408142508989091840,
       'favorite_count': 0,
       'source': '<ahref="http://www.apple.com"rel="nofollow">iOS</a>',
       'retweeted': False,
       'coordinates': None,
       'entities': {
       'symbols': [

       ],
       'user_mentions': [

       ],
       'hashtags': [

       ],
       'urls': [
              {
              'url': 'http: //t.co/D2JOpK3mKC',
              'indices': [
                     81,
                     103
              ],
              'expanded_url': 'http: //imgur.com/qqMhrYY',
              'display_url': 'imgur.com/qqMhrYY'
              }
       ]
       },
       'in_reply_to_screen_name': None,
       'id_str': '408142508989091840',
       'retweet_count': 0,
       'in_reply_to_user_id': None,
       'favorited': False,
       'user': {
       'follow_request_sent': None,
       'profile_use_background_image': True,
       'default_profile_image': False,
       'id': 2201424232,
       'verified': False,
       'profile_image_url_https': 'https:
//pbs.twimg.com/profile_images/378800000809648072/b8b67ea48368419eb54ca2ae2e557299_normal
.jpeg',
       'profile_sidebar_fill_color': 'DDEEF6',
       'profile_text_color': '333333',
       'followers_count': 12,
       'profile_sidebar_border_color': 'C0DEED',
       'id_str': '2201424232',
       'profile_background_color': 'C0DEED',
       'listed_count': 0,
       'profile_background_image_url_https': 'https:
//abs.twimg.com/images/themes/theme1/bg.png',
       'utc_offset': None,
       'statuses_count': 15,
```