THE UNIVERSITY OF
MELBOURNE

VUE PREEDICTION
WEB APPLICATION

Group11

Chenyang Lu 951933
Echo Gu 520042
Jing Du 775074
Penfcheng Yao 886326
Zhijia Lu 921715

Abstract
The Victoria Unemployment and Crime (VUC) Prediction project is designed to collect and analyze the twitter post located within Victoria, Australia. Sentiment analysis is conducted on the harvested tweets to identify the ratio of wrath posts in each local government area. Further development of regression model is used to identify the relationship between the ratio of wrath tweets and two social factors: crime rate and unemployment rate, for years from 2014 to 2016. This model is used to predict future crime and unemployment rate in each area based on the currently streaming collection of Twitter data.

Table of Contents

Introduction

With an increase use of social media in the modern society, there is great benefit in analyzing Twitter data to inform researchers and policy makers about social and economic issues. In this paper we consider how the sentiment analysis based on twitter texts could be used to construct models to reflect and predict two social factors: the crime rate and unemployment rate for areas in Melbourne. This could be further developed to find which areas are best for living as well as identifying potential threats to cities in Melbourne.

The project designs and builds a cloud based application that runs on virtual machines across the University of Melbourne research cloud. It harvests and categorizes twitter data based on its geometric location, analyze and make comparison with relevant information from Australian Urban Research Infrastructure Network (AURIN) and other source. The application harvests tweets with both stream and search API interfaces, stores and processes the tweets into the CouchDB based database server. Only English language tweets were considered. Basic sentiment analysis is performed on the collected twitter texts, which categorizes into positive, negative or neutral emotions. With the focus of the proportion of negative tweets in each local government area (LGA), the crime rate and unemployment rate of each LGA are investigated to explore the correlation between negative twitter emotion and these social indications. Simple linear regression models are developed using yearly data from 2014 to 2016. These models are further explored to predict future crime and unemployment rate in each area based on the currently streaming collection of Twitter data. Limitations and drawbacks of the system is also discussed.

As a coherent team, each one of the five team member has a clear direction of their responsibilities from the start of the end. Communications are effective to ensure different stages and components of the project join smoothly. Collaborative effort plays an important role in supporting each other when facing difficulties. In particular, the duty accomplished by each team member is:

- Jing Du: database design and administration
- Echo Gu: data analysis and scenario interpretation
- Chenyang Lu: front-end development and data visualization
- Zhijia Lu: twitter harvester and back-end development
- Andy Yao: architecture design and implementation

The YouTube video demonstrating the automatic deployment of the system can be found from:
https://www.youtube.com/watch?v=_nvIL76fBAI&feature=youtu.be
Please note this demonstration was conducted on private instance, in order to not interrupting the data harvesting of the project.

The link to the GitHub repository of this project can be found in the appendix.

System architecture

The system architecture defines the structure of software components on computers, and the relationship between the components (Bass, Clements & Kazman, 2003). It is a coherent design package with certain properties that allows reuse. The two major types of distributed systems are: service-oriented architecture (SOA) and resource-oriented architecture (ROA). The focus of SOA is providing a tool of completing service requested by the client, whilst ROA involves retrieving particular resources. The main relationship between software components in this project is retrieving information: web server requests data from application server to present information at the front-end; application server requests a view of data from database server; Thus ROA is the most suitable solution. Representational state transfer (ReST) defines a uniform connector interface that identifies and navigates resources (Fielding, 2000). It creates an image of the resources' current state while the copy of the data remains on the server. The ROA architecture allows the application to turn into a ReSTful web service.

System design

Distributed system is a system that coordinates multiples processors on a network. It is known to have great performance as tasks were broken into pieces using parallel computing. Among many properties of distributed system, horizontal scalability and reliability are the two major advantages to successfully implement this project. As the processes are running independently from each other irrespective the location of the physical server. It is simple to add nodes and resources to increase computational power, hence ensures scalability. It can recover from component failures and resumes services. The great fault tolerance ensures high availability of the service. The clustering function of Couch DB is the major distributed system in the project, which ensures the great availability of the data. The implementation of crawlers over several instances also provides scalability if more data needs to be harvested.

The 4 tiers system design separates database server, application server, web server and the client (Figure 1). The ability of changing a single component and scaling up or down ensures great flexibility and security.

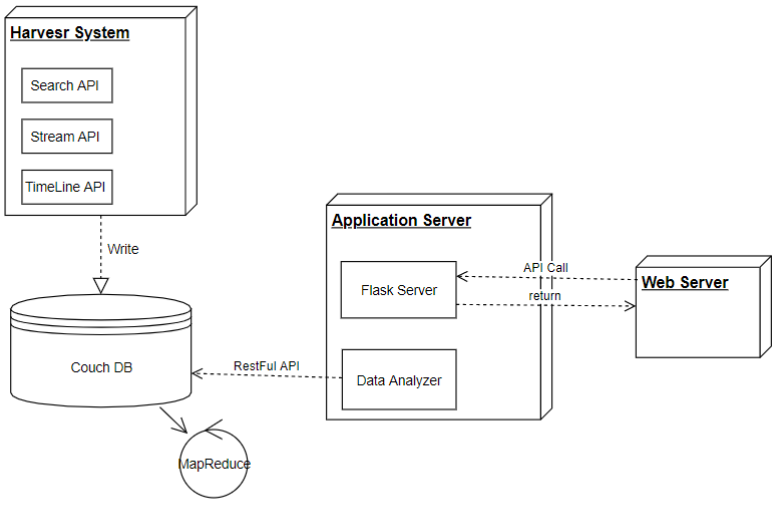

Figure 1

Physical structure

The application is encapsulated into a collection of docker images and distributed over 4 instances (Figure 2). Communication server and web server is constructed on one single instance on the cloud. Database is cluster over the three instances and application server is distributed on these instances as well. The detailed design of the system will be discussed in later sections.

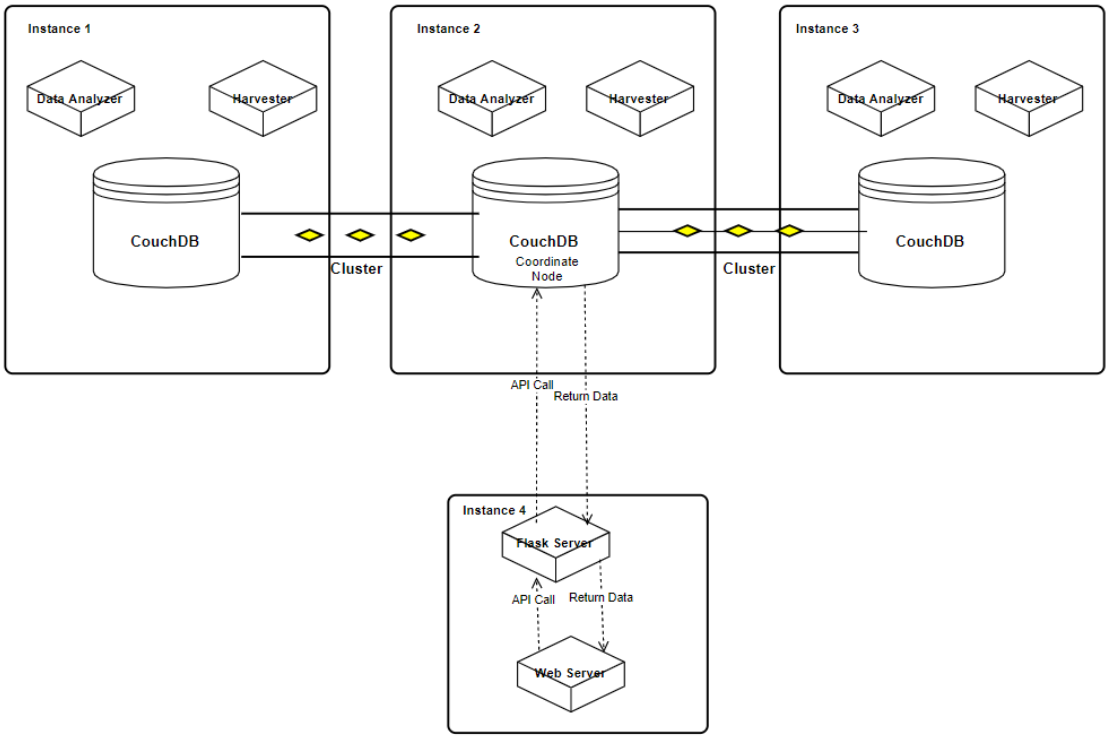

Figure 2

Deployment

Ansible

Ansible, an application-deployment and configuration management tool, is used in this project to do automatic deployment. With ansible, each playbook can deploy customized service/services.

There are two ansible design choice. The first choice is that using one playbook to deploy all the services. The second choice is using several different playbooks and each playbook deploy different services. In this project, the second option is selected, and we have five different playbook and each deploy different service (Figure 3):

- Nectar playbook: automatic deploy and customise instance including security group and volume
- Set-env playbook: automatic deploy instance environment, for example, set-up proxy, install docker and docker configuration
- Website playbook: automatic deploy web server
- couchDB playbook: automatic deploy couchDB cluster
- Application playbook: automatic deploy Flask server and data analyzer server

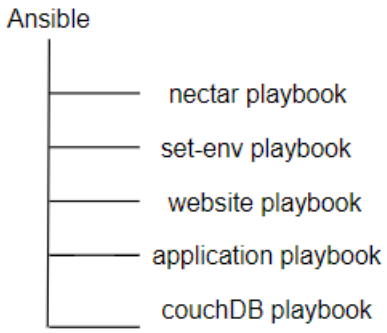

Figure 3

The advantage to use several different playbook to deploy different service is that each one service is down (e.g. Flask server), only that service need to be shut down and debug and only that playbook will used to do re-deployment which avoid the re-deployment of the entire system.

Docker

Docker is used in the implementation of this project in order to achieve good functionality of a distributed system. A Docker container is a standardized unit of software which simplifies the process of development and deployment. It specifies the code, system library, and setting that's required for the program. As an isolated software, it is able to provide same service regardless of its environment infrastructure. The development of a Docker image will become a container at runtime in a portable and efficient manner. It is one of the strategies in this project to maintain smooth running and backup of the software. In the unfortunate event of crushed instances, it would be a relatively simple task to restart the instance and reload the image of the application through Ansible.

Composer is a tool for dependency management which allows us to declare the libraries that our project relies on and it manage those libraries for the project. With the help of docker composer, all the command line can be organized in *'docker-composer.yml'*, thus, the image can be built easily.

Docker structure

Figure 4 displays the design of containers' layout of the dockers. Each container is an individual node running on the instance. The web server and communication server are packed into one Docker, which runs on one instance. The crawler, database server, and application server are distributed over three Dockers. The containers for Couch DB are identical to ensure data redundancy, which is managed over three clusters. The harvesters and application servers are programmed to collect and process data based on the geometric location they are assigned to, thus they have the same functionality but are distributed over three instances. The redundancy can effectively avoids single point of failure, which aims to increase the durability of the system.

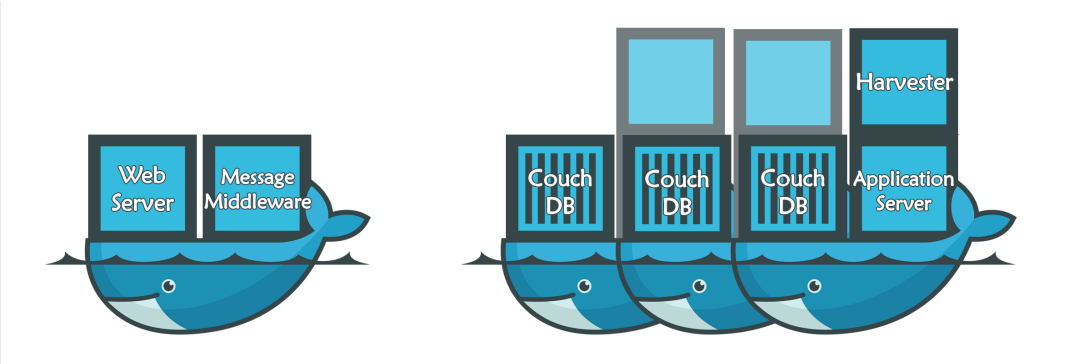

Figure 4

Container structure

For the containers, three different designs of Docker container structure are used for different application components, depends on the need of the service. The container could include: environment only; environment and code; or data storage.

Figure 5a shows the design of the Flask and analysis service. Their images were built with only the system environment required. The implementation code is pulled from remote Git repository at run time. The directory of the specific code stored in the volume is mounted on the directory of the corresponding image in each container. This design facilitates the debug and re-deploy during development stage, as well as amendments for future improvement. In contrast, due to the nature of the project, the role of the web server is simply a data display platform with limited user interaction. Although it could also be structured with environment only, the efficiency of deployment is much higher when both the code and environment are stored in the container. As the communication was ReSTful based, there is no file input or output, the compilation and implementation of the code stays within the container. Thus the design of web server is shown in Figure 5b. The CouchDB database server has great focus on data redundancy to avoid possible information loss. Figure 5c demonstrates that duplicated data is stored in both the container and the volume. CouchDB is able to achieve consistency of data at both locations as well as partition tolerance through its clustering, which protects the integrity of the database.

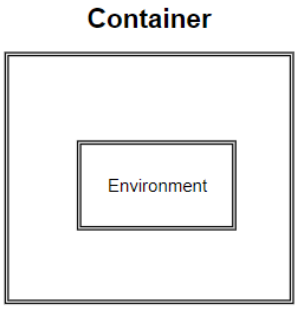

Figure 5a

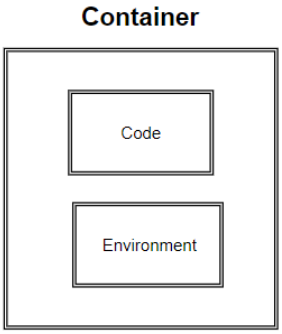

Figure 5b

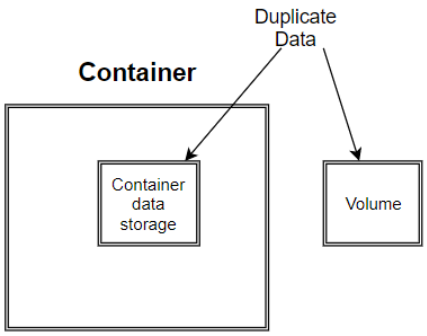

Figure 5c

During runtime, the container would set up the running environment and be able to access the data from the volume. Whilst the implementation of the software is built upon each individual instance, the reading and writing file is directly communicated to volume. If the container is stopped or removed under any circumstances due to uncertainty of the cloud system, all data is still stored in the volume, and the application can be easily restarted without losing any information.

Couch DB

CouchDB Cluster Setup

CouchDB is designed as a distributed database, which can perform better under distributed systems. Generally, CouchDB cluster are usually set up between multiple instances. For many scenarios, only one database is required in a distributed system, while many services are required to lower the burden on handling requests. In this project, 4 instances are created to establish a cloud-based system, and 3 of them are assigned to support services of CouchDB. In fact, only one data storage is used for such a small system. The cluster of 3 CouchDB instances is achieved. It not only assigned tasks on three nodes, but also maintain a shared data storage between three instances.

Pre-request

To set up the cluster, three individual CouchDB services are required in advance. When the status of each service is running, the setup process can be launched. In addition, each request toward configuring cluster can be achieved by *_cluster_setup api*.

Join node to cluster

Two requests to coordinator node can achieve this behavior:

```
-X POST -H "Content-Type: application/json" http://admin:password@<setup-
coordination-node>:5984/_cluster_setup -d '{"action": "enable_cluster",
"bind_address":"0.0.0.0", "username": "admin", "password":"password",
"port": 5984, "node_count": "3", "remote_node": "<remote-node-ip>",
"remote_current_user": "<remote-node-username>",
"remote_current_password": "<remote-node-password>" }'
```

```
curl -X POST -H "Content-Type: application/json"
http://admin:password@<setup-coordination-node>:5984/_cluster_setup -d
'{"action": "add_node", "host":"<remote-node-ip>", "port": <remote-node-
port>, "username": "admin", "password":"password"}'
```

This will join the current remote node to coordinator node. Keep running the above commands for each node you want to add to the cluster.

Complete the setup

Once join behaviors are done, the following command can be execute to complete the cluster setup and add the system databases:

```
curl -X POST -H "Content-Type: application/json"
http://admin:password@<setup-coordination-node>:5984/_cluster_setup -d
'{"action": "finish_cluster"}'
```

Verify cluster

After finishing the cluster setup, there are two operations to check whether the cluster is available now.

```
curl http://admin:password@<setup-coordination-node>:5984/_cluster_setup
```

This command here is to test the status of cluster setup.

```
curl http://admin:password@<setup-coordination-node>:5984/_membership
```

This command is to request the cluster membership of coordinator node, Normally, all the nodes attempted to join the cluster should appear in “all_nodes” and “cluster_nodes”, and the response of the command is like:

```
{
    "all_nodes": [
        "couchdb@couch1.test.com",
        "couchdb@couch2.test.com",
        "couchdb@couch3.test.com",
    ],
    "cluster_nodes": [
        "couchdb@couch1.test.com",
        "couchdb@couch2.test.com",
        "couchdb@couch3.test.com",
    ]
}
```

Once the response is the same form like the template below, the process of cluster setup is complete and successful.

Flask Server

The Flask framework is a Python based microframework that provides ReSTful API for communication with the front-end. The Flask server component in the system is in charge of extracting statistics from CouchDB and sending them to web page. It also connects with database through ReSTful API which is achieved by CouchDB package in Python. It simplifies the establishment of network routing.

Since the server only plays the role of displaying data, it is only issued with two *HTTP GET* interfaces, *'/vucprediction/api/tasks'* and *'/ vucprediction/api/tasks/<string:task_id>'*, where the first one gives all the information to the web server and the second one offers the specific data according to *'<string:task_id>'*. It can retrieve the data required by the web server in *json* format.

The server keeps running at the cloud instance with 172.26.37.33, port 8080 to continuously get request from web server.. When it starts, it will extract all the demanded statistics by *'refresh()'* method and then waits to be called. The three crawlers keep running all the time, there are always new-collected data injected into database and therefore the analyzers will also keep working and repetitively calling Mapreduce's view every hour. Consequently, the server should refresh the data in case the analyzers update the statistics. When correctly formatted request comes in, it will load the specific task or refresh all data again if the call is only 'task'. When it does not receive the correct request, it will simply return a 404 error message:

```
"@app.errorhandler(404)
      def not_found(error):
      return make_response(jsonify({'error': 'Not found'}), 404)"
```

Meanwhile, the method below ensures the system provides reply if bad request error occurs.

```
"@app.errorhandler(400)
      def not_found(error):
      return make_response(jsonify({'error': 'Bad request'}), 400)"
```

Twitter harvesting

Based on the geographical information of a tweet, Twitter provides APIs that can be used to harvest posts within a predefined geospatial location. It is one of the distributed system implementations in this project, in order to achieve both scalability and availability.

The Harvester Component

The harvester component of this software consists of two parts, the crawler and the writer. The crawler is responsible for collecting data and the writer takes charge of injecting the collected data into CouchDB database. Figure 6 shows the flowchart of the harvesting process and relationship between different components.

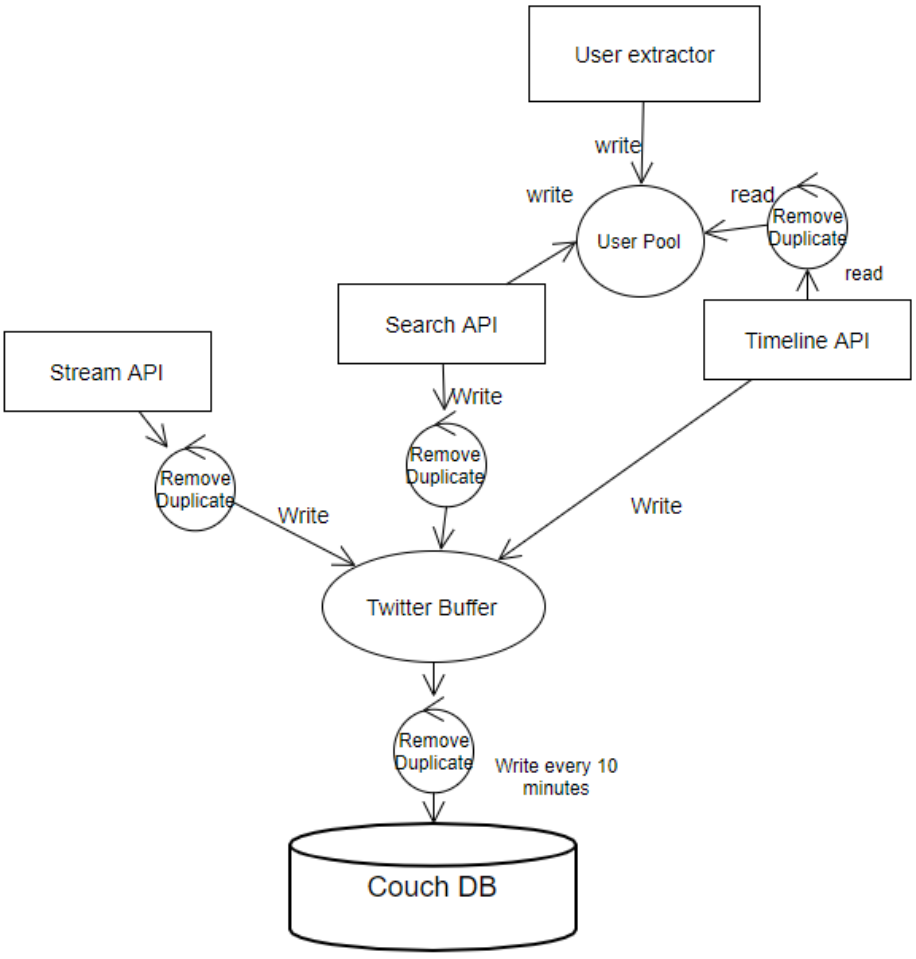

Figure 6

The crawlers

Three crawlers are implemented in the project, namely *crawler1*, *crawler2* and *crawler3*. According to the numerical order of Local Government Area ID, cities in Victoria are divided into three subsets. Each crawler collects data with geographic coordinates located in one of the areas respectively. The implementation logic behind the crawlers are the same except the area-classification mechanism as showing in below:

```
crawler_area = {}
for key in boundaries:
      if count < 30:
            crawler_area[key] = boundaries[key]
      count += 1
```

Each crawler harvests tweets from all three kinds of Twitter API: search, stream and timeline. All APIs are accessed from Python's *tweepy* and *twitter* package (Tweepy, 2019). At the beginning of the harvest stage, the crawler will firstly search APIs provided by *tweepy*. Cursor parameter is used with each request to perform pagination with less codes. It will collect 10,000 tweets from recent 7 days with coordinates restriction inside Victoria, as Twitter limits the timeframe for tweets access through standard search API. The LGA ID is used as a search filter to allocate the crawler's destination area, for example:

```
search_query = 'place:0ec0c4fcacbd0083'
```

Afterwards, the crawler will start a thread to implement stream API. It continuously monitors and captures freshly generated tweets in a certain geographical polygon standing for Victoria provided by Australian Government (2019), extracts user ID and store them into file waiting to be processed. Meanwhile, the main thread will enter the timeline harvesting for the particular user, from which most of the data will be harvested. It constantly digest 200 historical posts from a single user as tweepy's timeline method only return 20 tweets per request. It checks the existence of unprocessed user at every query stage and will sleep for 15 minutes if the unprocessed user file is empty. All the tweets obtained from methods mentioned above will be stored into a json file *'tweet_recent.json'*.

The writers

One writer is allocated to each crawler, who injects collected data into CouchDB. It is achieved by CouchDB package in Python which is established through ReST API. The writer transfers every line it reads to the databased and deletes it afterwards, to ensure no duplicates arise. It works continuously until the harvested tweet file becomes empty, then it is programmed to sleep for 10 minutes before attempting to write again.

Challenge

Two major challenges are revealed in this process: handling duplicate twitter and Twitter API rate limit.

Duplicate twitters

Duplicate tweets generally come from search API and user timeline as streaming tweets would only be captured once. For every 10,000 tweets obtained from search API, they are temporarily stored into a buffer pool named *'duplicate_tweet_recent_json'*. Once finished, a program logic removes duplicate data in the file and stores them in to *'tweet_recent.json'*. Since each crawler collects data from different region, they would not harvest identical tweets. Hence duplicates from search API are eliminated.

On the other hand, the duplicates that may come from user timeline are resolved in two steps. Like tweet buffer pool, users from standard API will also be put into a temporary file and wait to be processed. Initially, all the users from twitter data provided by the university and streaming data will be combined to store into *'unprocessed_user_list.json'* file. Such file potentially have duplicate users since the one in the university's databased may post new tweets and be captured by stream API. Consequently, *'processed_user_list.json'* is created to store processed user ID. The timeline crawler will check whether an user ID exists in this document at the beginning of harvesting. User who was not captured before will have their ID stored into the file after processing.

In addition, although new data from timeline search is free of duplicates, they may conflict with static data stored in CouchDB retrieved from the university. Such condition can be handled by error catch. Documents in CouchDB all have a key named 'rev' which stands for its revision number, all the updating operations should obey such number. Hence, when attempting to store an existed twitter without revision number into database (twitters ID are set as docs ID in CouchDB), it will return a conflict error showing below. As long as such exception is caught, the database will be free of duplicate tweets.

```
File "C:\Users\62463\Anaconda3\lib\site-packages\couchdb\http.py", line
    574, in_request credentials=self.credentials)
File "C:\Users\62463\Anaconda3\lib\site_packages\couchdb\http.py", line
    426, in request raise ResourceConflict(error)
couchdb.http.ResourceConflict: (' conflict', 'Document update conflict.')

Process finished with exit code 1
```

Twitter API rate limit

Twitter set a restrictive time rate on its APIs, which returns errors and crashes the process if rate is exceeded. In this project, all rate limits are handled by putting the crawler into sleep until the start of the next window. Timeline and search API has default method called *'sleep_on_rate_limit'* in both *tweepy* and *twitter* package showing below:

```
api = tweepy.API(auth, wait_on_rate_limit=True,
     wait_on_rate_limit_notify=True)

aut2 = twitter.Api("aMlAt9xTsFOxaD11InOfeo8NV",
     "37pfRhPJKSyZcCNPPJUfaoJxVhWobaqkSkR8oQobnkCQ5g3Et8",
        "851934201506971648-H5d9b02jl5AdCG0eOm2EoZbXNq0xY4K",
        "yKlL4K5PB4g9YDqWDzDAcu2AOzSDQNs3gZAYpdOUs3b6A")
```

Stream API will return a *HTTP 420* response and crawler is programmed to go to *'on_timeout method'* showing below, which forces it to sleep 10 minutes before resuming the harvesting process.

```
def on_error(self, status_code):
     if status_code == 420:
        self.on_timeout()

def on_timeout(self):
     time.sleep(600)
```

Error Handling

Main errors captured in this component are *'not authorized error'* and *'page does not exist'* from Twitter and *'document update conflict'* error from CouchDB. The first Twitter error happens because of protected user ID and the second one occurs when user can no longer be found. Both of them can be simply captured when running timeline crawler and jump to next user. The CouchDB error has been briefly mentioned above. Other potential exceptions such as thread creating error and authentication error can also be caught.

CouchDB

Apache CouchDB is a document-oriented database management system which has high scalability, availability and reliability. It is a kind of NoSQL database system (DBS), and it is based on distributed architecture, whose functions tightly meet the requirements of this assignment tasks. Based on this concept, CouchDB is implemented in this assignment and in charge of controlling the database for the cloud-based system.

Characteristic

As a distributed database, CouchDB can distribute storage systems to many physical nodes and it can also coordinate and synchronize data between these nodes. For web-based large-scale applications, its distributed storage structure makes CouchDB does not have to split tables like the traditional relational database and make a lot of changes in the application code layer.

As a document-oriented database, CouchDB stores semi-structured data, which is like Lucene's index structure. Because of this index structure, it also suitable for CMS, phone book, address book and other applications. In these applications, the document-oriented database is more convenient and better than the relational-oriented database.

CouchDB supports the ReST API. It allows users to manipulate CouchDB databases using JavaScript, as well as write queries in JavaScript. CouchDB can also manipulate JSON as a data format and manipulate the organization and presentation of documents through views.

Accessing CouchDB

According to the REST approach, the **CouchDB** request is issued by HTTP methods like GET, POST, PUT, and DELETE. For example, by executing "curl -X Get http:// localhost:5984 / _all_dbs" on the command line, all the database in CouchDB can be displayed.

Another way to get database information is to visit the website supported by CouchDB service such as http://localhost:5984/_utils/. This displays the Fauxton web page, which is a native web-based interface built into CouchDB. All database information can be retrieved through it.

```
ubuntu@demo:~$ curl -X Get http://admin:admin@172.26.37.254:5984/_all_dbs
["boundary","crime_2012","crime_2013","crime_2014","crime_2015","crime_2016","negative_tweets_2014","negati
ve_tweets_2015","negative_tweets_2016","negative_tweets_2017","negative_tweets_2018","negative_tweets_2019"
,"neutral_tweets_2014","neutral_tweets_2015","neutral_tweets_2016","neutral_tweets_2017","neutral_tweets_20
18","neutral_tweets_2019","positive_tweets_2014","positive_tweets_2015","positive_tweets_2016","positive_tw
eets_2017","positive_tweets_2018","positive_tweets_2019","t_2014","t_2015","t_2016","t_2017","t_2018","t_20
19","unemployment"]
```

Figure 7: A command to get all of DBS in the CouchDB

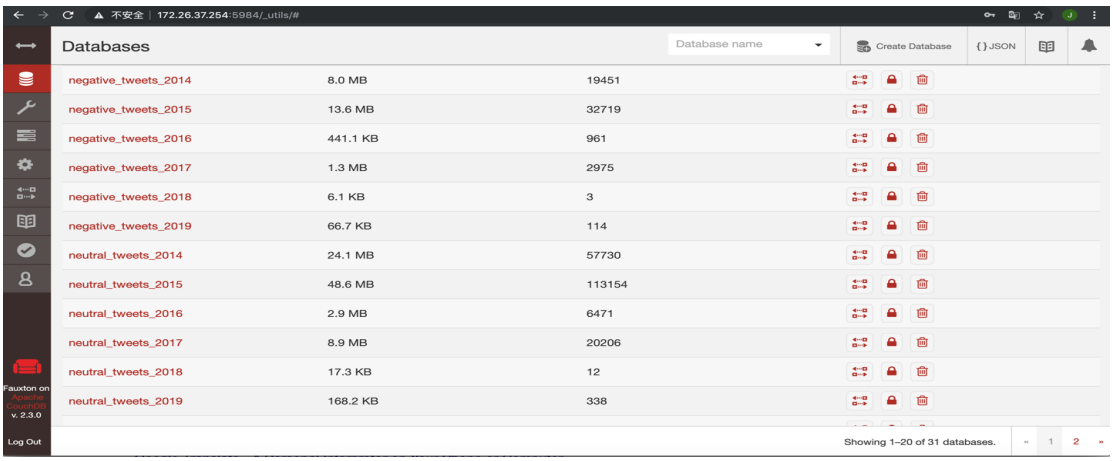

Figure 8: Fauxton webpage of Databases in the CouchDB

Database initializations

In this assignment, 34 databases are created in CouchDB, and these databases can be divided into the following four categories.

Twitter

There are 18 databases are used to store the tweets based on year and sentiment type (Figure 9). The tweets are created during 2014-2019, and "positive", "negative" or "neutral" will be tagged on each tweet by sentiment analysis.

Figure 9: Fauxton webpage of Databases store tweets

Statistical data of tweets

There are 7 databases used to store the statistical data of tweets for each year during 2014-2017 and an extra database which stores statistical data for all the years. Furthermore, each database has 88 documents (Figure 10). 87 of these documents are used to store the statistical information for 87 different areas, and the other one document stores a map/reduce method. The data structure of each document is showed in Figure 11.

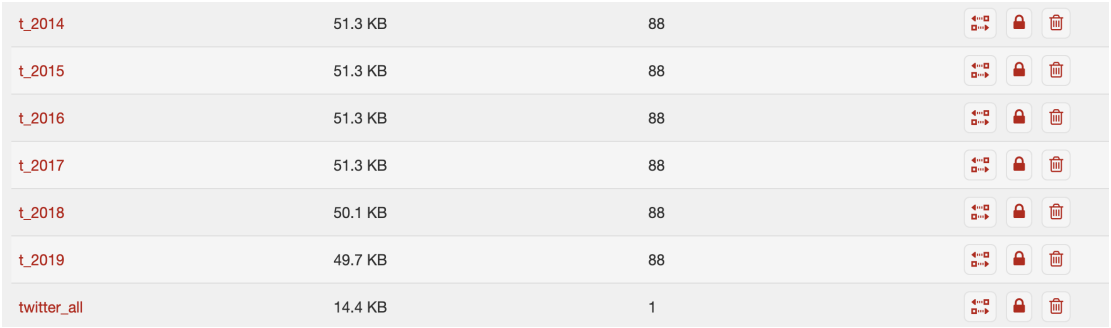

Figure 10: Fauxton webpage of Databases store statistical data of tweets

Figure 11: The data structure of each document

Crime

There are 5 databases used to store the data of crime for each year during 2012-2016, and an extra database named “crime_all” stores the data for the five years (Figure 12).

Figure 12: Fauxton webpage of Databases store crime information

Unemployment

There are 2 databases used to store the data of unemployment in five years (Figure 13). The *'unemployment'* database stores all the unemployment counts tagged by years (2012-2016) for each area. The *'unemployment_all'* database stores the unemployment count within five years for each area.

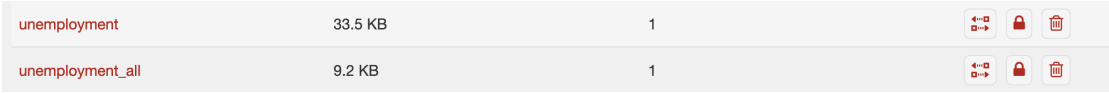

Figure 13: Fauxton webpage of Databases store unemployment information

Data statistics

Map/Reduce is a programming paradigm and related implementation for processing and generating large data sets using parallel distributed algorithms on a cluster. In CouchDB, the MapReduce algorithm contains two important tasks: Map and Reduce. Map accepts a set of data and converts it into another set of data, where each element is broken down into tuples (key/value pairs). Then, the task of Reduce is to take the output from the map as inputs and group the data metadata into a smaller set of tuples. The result of Map/Reduce is a storage B+ tree called view.

In this project, Map/Reduce is used to generate views of pre-defined Twitter statistics like counting number, calculating average score and filtering hashtags. To analyze, the first step is to use a Map/Reduce function to aggregate the tweets for same locations or years. Some Map/Reduce functions can simply calculate the number of Tweets in a particular mood from a particular location in a particular year (for example, the number of angry tweets in 2014).

Take the most popular angry topic as an example. The Map/Reduce function _design/tags he can count the top 5 hashtags in the angry twitter. If so, the function will issue a list of key/value pairs in the form of *<hashtag_name, 1>* for every hashtag. The result list then passed to the Reduce function. Then, the count of hashtags with the same name will be summarized.

Web application

In this project, a front end web application is developed to achieve a better data visualization. The Web Application is developed based ReactJS (Reactjs.org, 2019) alone with the styling solution named Material-UI (Material-ui.com, 2019) which implements Google Material design principles. In addition, another packages canvasJS (CanvasJS, 2019) is used to visualize our data. The root component is the only one that is responsible for requesting data from the server, where ReSTFUL API facilitates the communications between them. It manages and passes the data to all child components, which ensures the data integrity of the entire web.

Brief Introduction of the technology stack

React is a javaScript library for building user interface with component based structure. Each component manages its own state, which makes it easy to pass rich data between components. This characteristic suits the project perfectly as we need to pass rich data through the entire website frequently and dynamically in order to visualize the information delivered from server.

Material-UI is implemented as the styling solution in the website. Material-ui is react components that implements google material design which standardized our web design. CanvasJs and Material-Table are javaScript data visualization packages that helps to build scatter chart and Table.

Moreover, Google Map API is one of the major implementations on the website to present data as heatmap over the entire Victoria. The GeoJson data of Victoria Local Government Area map is retrieved from Australia government (2019).

Web Architecture

The web Architecture is indicated in figure16. The react application is organized by components and each component has its own state. Thus, the key aspect of this project is passing data based between each component and keep the data consistent. In order to achieve this through a well-structured design, only *HomePage* component is used to communicate with FLASK server. It then passes the data down to each child component, which ensures the data integrity. Any change or updates of data on the FLASK server, all child components will re-render to match the new data.

In addition, CurrentMap component and predicatedMap component will communicate with Google map api individually since the hold different map and won't cause any problem of data integrity.

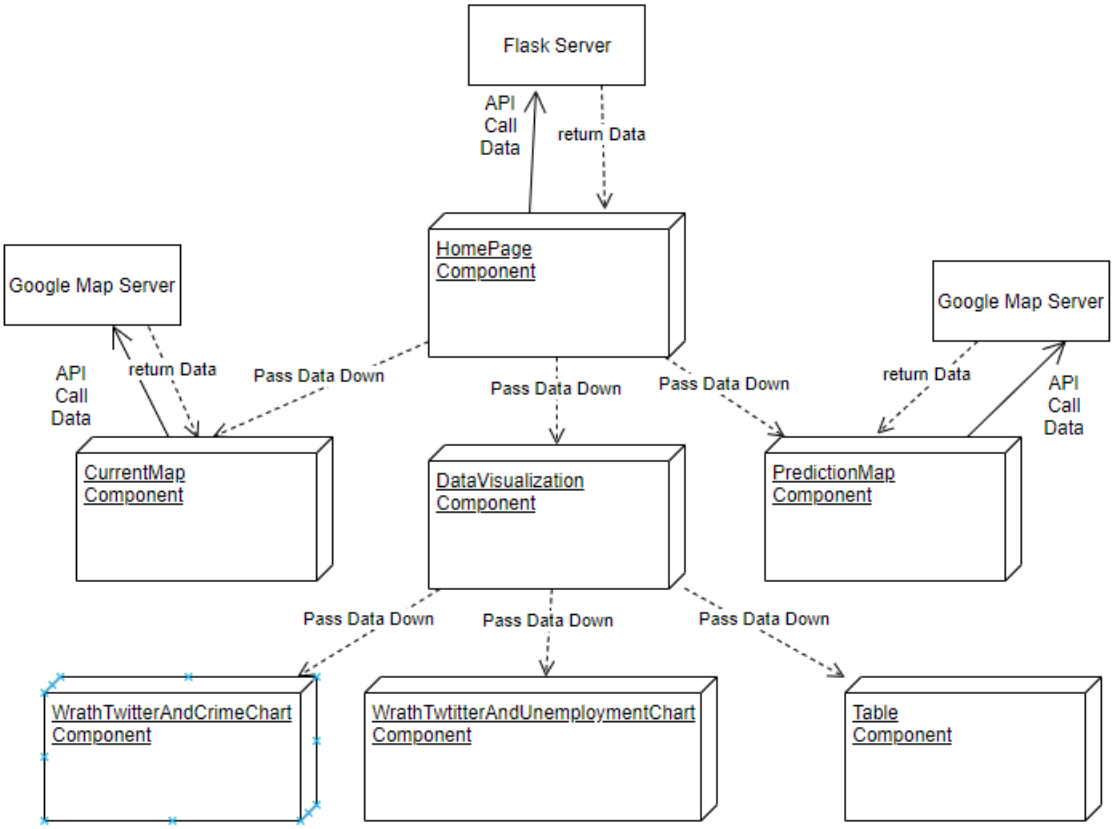

Figure 16: The web application

Web page Functionality

The webpage is divided into three pages: current map, historical analysis and prediction.

Current Map Page

Current Map Page shows the heat map for crime data, unemployment data, wrath twitter data in 2014, 2015, 2016, 2017 and 2018. Detail information is displayed in the right side of the page by clicking the area in the map (Figure 17).

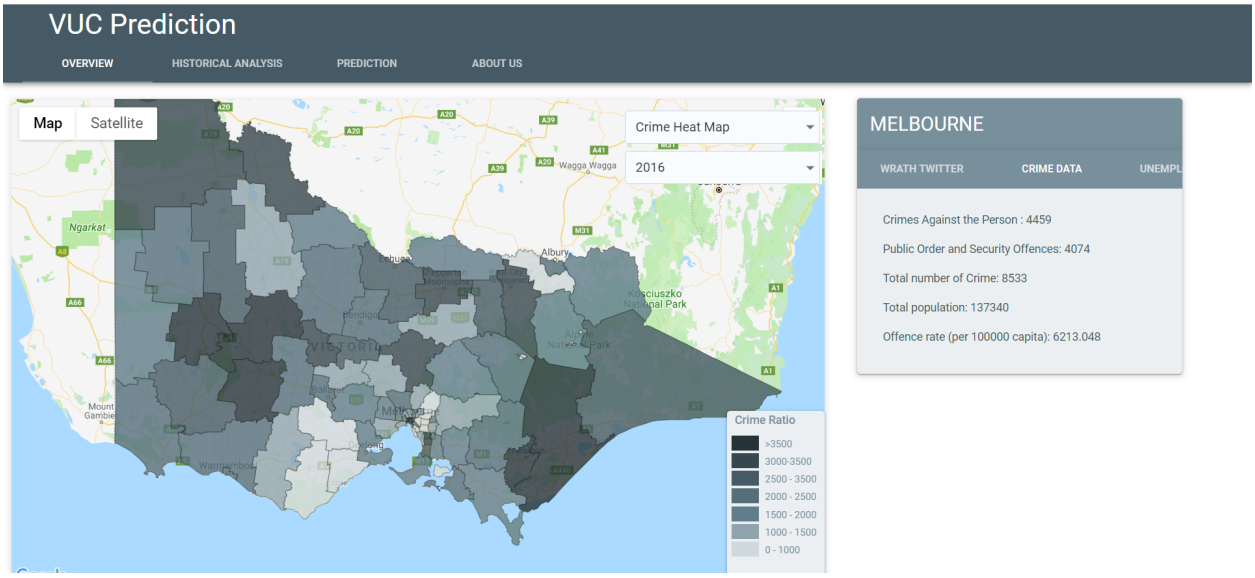

Figure 17

Data visualization Page

This page has three sub-pages, each sub-page displays chart or table (Figure 18).

a. Wrath Twitter/Crime Chart displays the scatter chart (2014-2016), detail information of each data point can be displayed when move over each data point
b. Wrath Twitter/Unemployment Chart has the similar function as wrath Twitter/Crime page.
c. Table page. This table has all the data collection (i.e. city name, wrath twitter ratio, crime rate, unemployment ratio, population) . Filter and sort functionality is embedded inside table

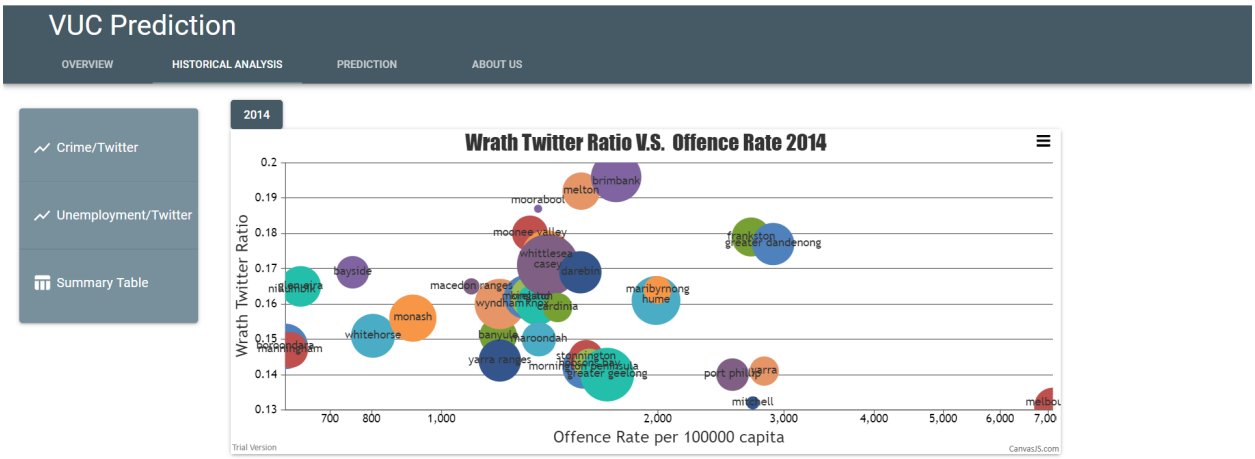

Figure 18

Data prediction Page

This page shows the 2019 crime rate prediction and 2019 employment prediction based on the twitter posted in 2019 (Figure 19).

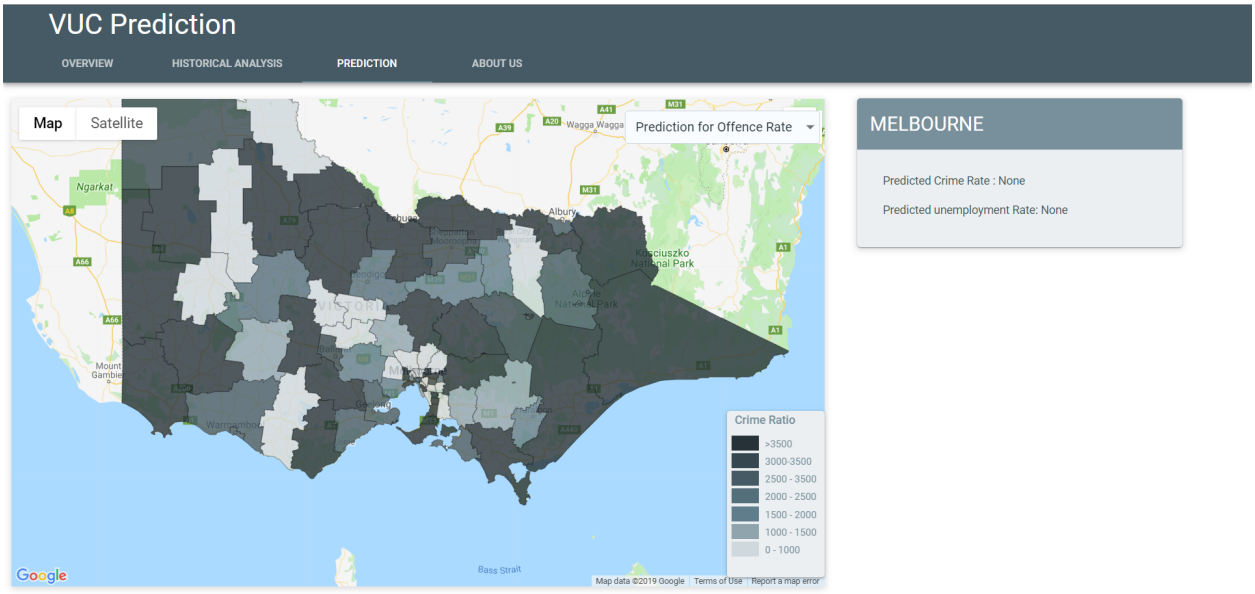

Figure 19

Ngix

Ngix was used in the web server to do reverse proxy, however, since only 4 instances are allocated in this project and the web server only hosted in one instance, therefore, ngix is not implemented for web server. However, if more instances are allocated, we can host the web server in several instanced and do the load balance.

Data analysis

Sentiment analysis

Sentiment analysis is a subfield of Natural Language Processing that intends to identify the opinion of a text. It helps to reveal the emotions and feelings embedded in the language and extracts the writer's attitude. It provides valuable information to marketing, sociology and political science fields. There are many challenges in the process as the tokenized text could have different meaning, and emoticons are commonly used in social media.

Valence Aware Dictionary and sEntiment Reasoner (VADER) is a lexicon based analysis tool focusing on emotions expressed through social media (Hutto, 2018). It caters for a variety of text format, such as standard text, acronyms, and emojis (Hutto & Gilbert, 2014). The sentiment score of the post is based on rating of the semantic tongue, ranging between -1 (most negative) and +1 (most positive). The compound score metric classifies text into three categories:

- Positive sentiment: compound score >= 0.05
- Neutral sentiment: -0.05 < compound score < 0.05
- Negative sentiment: compound score <= -0.05

In reference to measure one of the seven deadly sins (Seven deadly sins, 2019), this projects investigates the negative sentiment expressed in people's twitter posts as 'Wrath twitter'.

Data handeling

Australian Urban Research Infrastructure Network (AURIN) provides researchers and developers access to data from an array of different areas that focuses on the population and demographic features based on the geographical classification.

In the study of negative emotions reflected from the twitter text in any area, it is considered to be possible related to the crime rate and unemployment rate, the two important factors indicating social and economic activity and quality of life.

The aim of the project is to implement sentiment analysis of twitter text hence make connection with the crime rate and unemployment rate in each local government area. Whilst the twitter information is harvested and processed using the software, the crime rate is fetched from AURIN and the unemployment rate is obtained from Department of Jobs and Small Business (2019). The three key numerical data in each LGA city to be analyzed and investigated are:

- Wrath twitter ratio $= \frac{number\ of\ Wrath\ tweets}{total\ number\ of\ tweet}$
- Offence rate per 100,000 capita $= \frac{number\ of\ crimes}{total\ population}$
- Unemployment rate, provided as a percentage value

Crime offences can be classified into five categories. Amongst these classifications, the focus of this project is: *type A crimes against the person*, including 'homicide offences, assault, sex offences, stalking, robbery'; and and *type D public order and security offences*, including 'disorderly and offensive conduct, public nuisance offences, weapons and explosive offences and public nuisance offences' (Crime Statistics Agency, 2019). These two categories are more closely related to each individual in the society, where they are more likely to express one's concerns or grudge against the crime matters, hence being reflected through negative twitter posts.

Unemployment rate is another key indicator for the society's wellbeing as the unstable situations in the society are often related to lack of employment.

Scenario interpretation

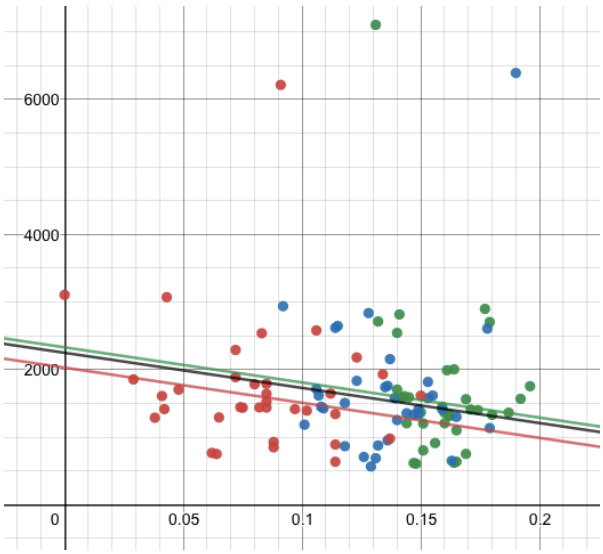

Figure 14

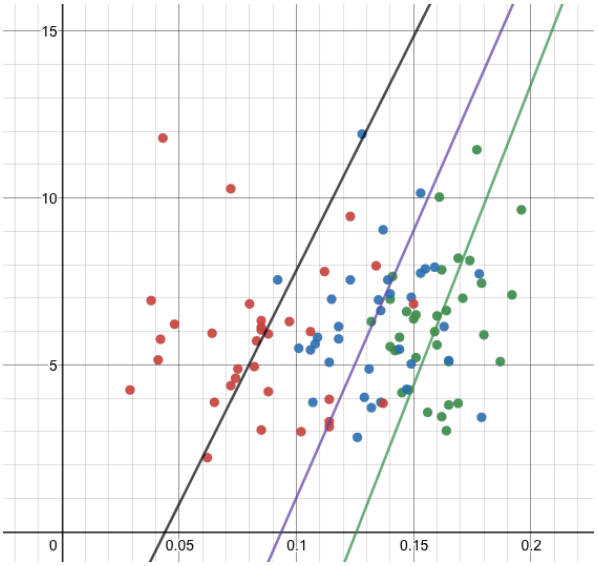

Figure 15

Based on the wrath twitter ratio, scatter plots are produced to demonstrate a negative correlation with the crime rate (Figure 14), yet a positive relationship with unemployment rate (Figure 15). This shows areas with larger wrath Twitter proportion could be caused by high levels of unemployment rate. Yet th decline of crime rate with the increase of wrath twitter proportion may suggest that the population who express one's anger through online platform are less likely to conduct real crime. Simple linear regression we conducted to show the trend of the data over the time period 2014 to 2016. It is noticed that the crime rate at Melbourne is significantly higher that other areas, hence are classified as outliers when performing analysis. These regression were further analysed to deduce the prediction of 2019 crime rate and unemployment rate. However, due to limited number of twitter data harvested, and the availability of AURIN data only up to 2016, the noise in the model is rather large. It should only be used as a brief indicator of the real society.

Advantage and limitations of the research platform

In the development of database for the application, it would be ideal to have data retrieved from the data source automatically through programmatic API calls, hence updated data and recently added data can be easily reflected in the database. In our project, the crime rate obtained from AURIN was manually downloaded and processed before stored into the online database server. This would increase the maintenance cost of the application in long run, and any modification of the current data would not be updated in timely manner. As the documentation of AURIN API is still in progress and does not yet support Python language, it is one of the limitations of the current implementation. It comes to our attention that NeCTAR cloud platform processes user request rather slowly. Creation and deletion of instances is rather

The NeCTAR cloud is not very stable either. Since the first deploy of our system, the platform has experienced several issues during the project period. This emphasized the importance of having data stored in the volume rather than on the instance. It also shows the advantage of using Docker. When the cloud system is crashed or flawed, the application can be easily reconstructed without losing any data.

Reference

Australian Government. (2019). Retrieved from: https://data.gov.au/

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley Professional.

CanvasJS. (2019). *Beautiful HTML5 JavaScript Charts. CanvasJS*. Retrieved from: https://canvasjs.com/

Crime Statistics Agency (2019). *Offence classification.* Retrieved from: https://www.crimestatistics.vic.gov.au/about-the-data/classifications-and-victorian-map-boundaries/offence-classification

Department of Jobs and Small Business (2019). *Small Area Labour Markets publication.* Australia Government. Retrieved from: https://www.jobs.gov.au/small-area-labour-markets-publication

Fielding, R. T., & Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures* (Vol. 7). Doctoral dissertation: University of California, Irvine.

Tweepy. (2019). GitHub repository. Retrieved from: https://github.com/tweepy/tweepy

Hutton, C. J., (2018). *vaderSentiment.* GitHub repository. Retrieved from: https://github.com/cjhutto/vaderSentiment

Hutto, C. J., & Gilbert, E. (2014, May). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*.

Material-ui.com. (2019). *The world's most popular React UI framework - Material-UI*. Retrieved from: https://material-ui.com/

Reactjs.org. (2019). *React – A JavaScript library for building user interfaces*. Retrieved from: https://reactjs.org/

Seven deadly sins. (2019). Retrieved from: https://en.wikipedia.org/wiki/Seven_deadly_sins

Appendix

https://bitbucket.org/zhijial/vuc-prediction-dbsetup_final/src/master/

https://bitbucket.org/zhijial/vuc-prediction-crawler3_final/src/master/

https://bitbucket.org/zhijial/vuc-prediction-crawler2_final/src/master/

https://bitbucket.org/zhijial/vuc-prediction-crawler1_final/src/master/

https://bitbucket.org/zhijial/vuc-prediction-analyzer3_final/src/master/

https://bitbucket.org/zhijial/vuc-prediction-analyzer2_final/src/master/

https://bitbucket.org/zhijial/vuc-prediction-analyzer1_final/src/master/

https://bitbucket.org/zhijial/vuc-prediction-flaskserver_final/src/master/