THE UNIVERSITY OF MELBOURNE

COMP90024

Cloud-based Product Category Analysis of Tweets in Melbourne

May 26, 2021

Team 2

Vihanga Jayalath (1088212)

Ying Mao (1224693)

Shumin Liu (1063288)

Jiexin Liu (1044804)

Robert Sloan (1142998)

Contents

1 Introduction

In this project, we develop an application that aims to analyse what topics people are interested in based on Twitter data in Greater Melbourne. We classify the dataset into ten categories. They are entertainment, business, technology, gaming, music, sports, politics, fashion, health and food respectively. Users can use our application to find popular topics that people often talk about in different regions to meet various business needs.

We first crawl the Twitter data from Twitter v2 APIs and then save them into CouchDB that is distributed into three nodes. After that, we use MapReduce to match the geo-location of each tweet with corresponding polygons of the suburbs in Melbourne. Then we aggregate the number of tweets in each topic and display the result in the frontend.

In this report, we analyse supported scenarios of our application and also point out some limitations. Some graphical results are analysed based on our findings in particular three suburbs including Altona, North Melbourne and Melbourne. Then we briefly introduce how to use our systems and illustrate the process with some figures. In addition, we discuss the architecture of our system and also mention some reasons why we design our system with that architecture. Then we discuss how to implement the Twitter harvester based on parallel computing and how to store data in the CouchDB cluster. More importantly, we discuss the algorithms that are used to classify tweets into different topics and also highlight the limitations of our classifier. After that, we discuss how to design the frontend and what tools we use to develop the frontend in detail. Then we discuss the implementation of connecting frontend, backend and CouchDB. Particularly, we also list some key commands in the backend and describe their functionalities. Then we briefly describe the mechanism of applying MapReduce to match data. We also analyse some pros and cons of Melbourne Research Cloud and Docker in the following section based on our practical experience. Finally, we summarise our project and discuss the limitations of it.

2 System Functionalities

2.1 Supported Scenarios

This application analyzed what kind of topics people are interested in different regions of Great Melbourne Area. Twitter data are harvested and categorized into 10 categories, which are entertainment, business, technology, gaming, music, sports, politics, fashion, health and food. Five suburbs

are selected to demonstrate the findings in section 2.2.
User can use this application to visualize the distribution of people's interest in different topics. All the analysis are based on live twitter data, which can give a good indication of the trend at the current time. These findings could potentially be used by advertising companies to effectively target different suburbs with different products.
One limitation of the analysis is that the result might be biased due to the number of tweets harvested and how data are categorized into different topics, which are something that can be improved in the future.

2.2 Graphical Results

According to the analysis (see graph below), tweets in Altona region are mostly about food (40%), sports (30%) and music (20%). This might be because Altona Beach Festival and musicals performed at Altona Civic Theatre providing people with opportunities to chat about food and music. Altona is also home to many sporting clubs including Australian rules football, soccer, cricket, etc, which may explain why there are more people tweet about sports.
Above 75% of the people tweet about sports in North Melbourne. The reason might be that North Melbourne Football Club, which plays in the national Australian football competition, is based in this region.
In terms of Melbourne CBD region, there is a great diversity in the topics that people have tweeted, which is as expected as there is a diverse range of businesses and people working, living and travelling in this region. The majority of the tweets fall in the buckets of sports (34%), entertainment (22.7%) and politics (20.32%).

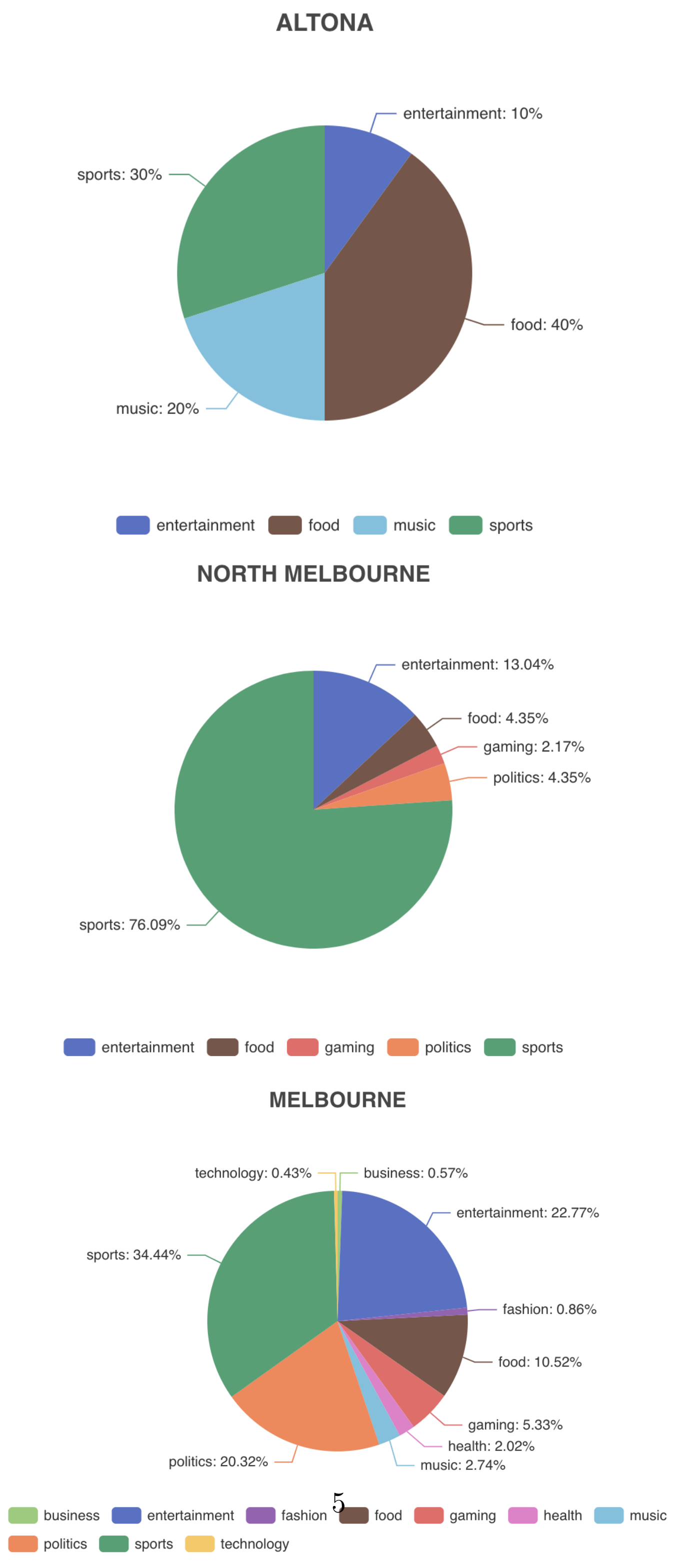


Figure 1: Distribution of people's interest in different topics for different regions in Great Melbourne Area

3 User Guide for Testing

3.1 System Deployment

We use Ansible as well as Docker to deploy the project. In order to re-deploy it, follow the steps described below.

1. Checkout the source code at `https://github.com/Liu233w/ccc-assignment2`

2. Build and push docker images in folders *flaskapp*, *twitter-analyser*, and *twitter-harvester*. You may refer to the example commands stored in *docker-build.sh* file in each folder

3. Navigate to *deploy*, following the description below to create instances

   (a) Install python, pip and ansible

   (b) Install ansible dependencies by `ansible-galaxy install -r requirements.yml`

   (c) Create a key-pair in MRC and set your key name in `host_vars/openstack.yaml`. The field name is `instance_key_name`

   (d) Use command `. openrc-downloaded-from-mrc.sh` and input your password to import correct environment variable

   (e) Run `ansible-playbook openstack.yaml` to create instances and volumes

4. Add IP addresses created from the last step into `hosts.ini`. Put the instance with name `webservice` into `webservices`. Put one of the instances whose name started with `db` into `db_head`, and the rest into `db_workers`.

5. Edit `host_vars/app.yaml`. Change the image names to the image created in Step 2.

6. Run the command to deploy the whole project: `ansible-playbook --private-key=PATH-TO-YOUR-PRIVATE-KEY -i=hosts.ini app.yaml`

7. Use following steps to setup the twitter-harvester.

   (a) Create academic accounts and projects in twitter.

   (b) Use AnyConnect to connect to the University of Melbourne VPN

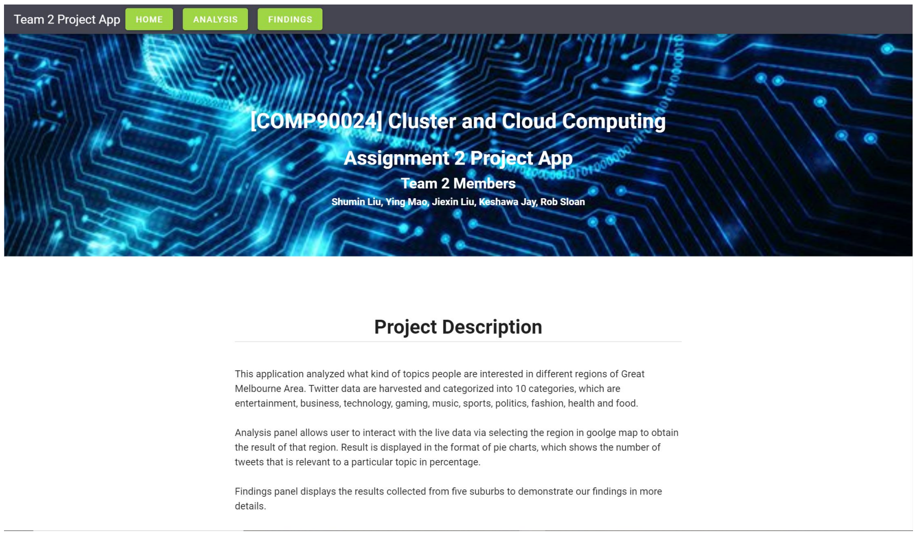
   (c) Go to `http://172.26.129.48:5984/_utils/#database/tokens/_all_docs` and create a document with `"token": (your token)` for each of your tokens.

Note: We decided to manually insert tokens as non-administrative users should not have access to them.

(d) The twitter harvester will automatically find and use these tokens and start harvesting.

3.2 End User Invocation

By visiting `http://172.26.134.58/`, users can enter the **Home** page. On this page, we have a section that shows our project title and also lists the names of our team members. In addition, we briefly describe the purpose of our project in the following section. Figure 2 shows the start page of our app.



[COMP90024] Cluster and Cloud Computing

Assignment 2 Project App

Team 2 Members

Shumin Liu, Ying Mao, Jiexin Liu, Keshawa Jay, Rob Sloan

Project Description

This application analyzed what kind of topics people are interested in different regions of Great Melbourne Area. Twitter data are harvested and categorized into 10 categories, which are entertainment, business, technology, gaming, music, sports, politics, fashion, health and food.

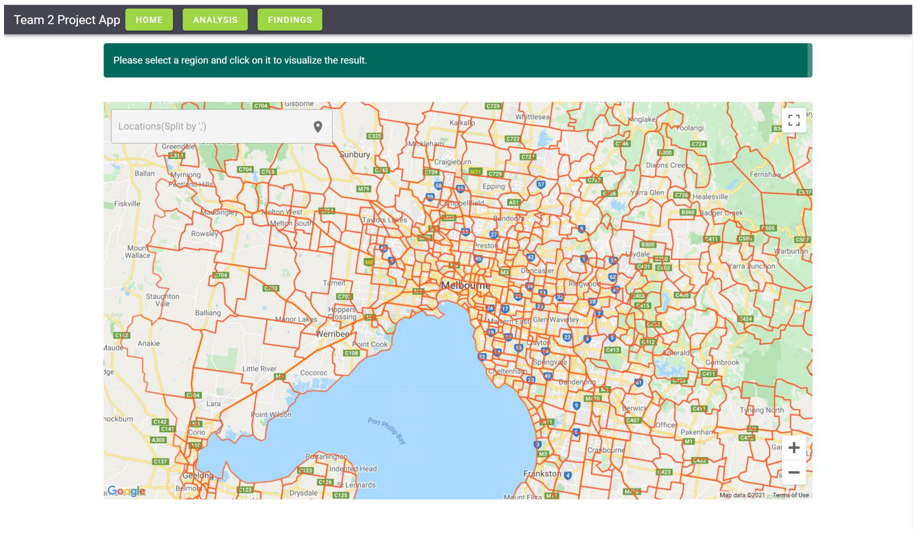
Analysis panel allows user to interact with the live data via selecting the region in goolge map to obtain the result of that region. Result is displayed in the format of pie charts, which shows the number of tweets that is relevant to a particular topic in percentage.

Findings panel displays the results collected from five suburbs to demonstrate our findings in more details.

Figure 2: Home page

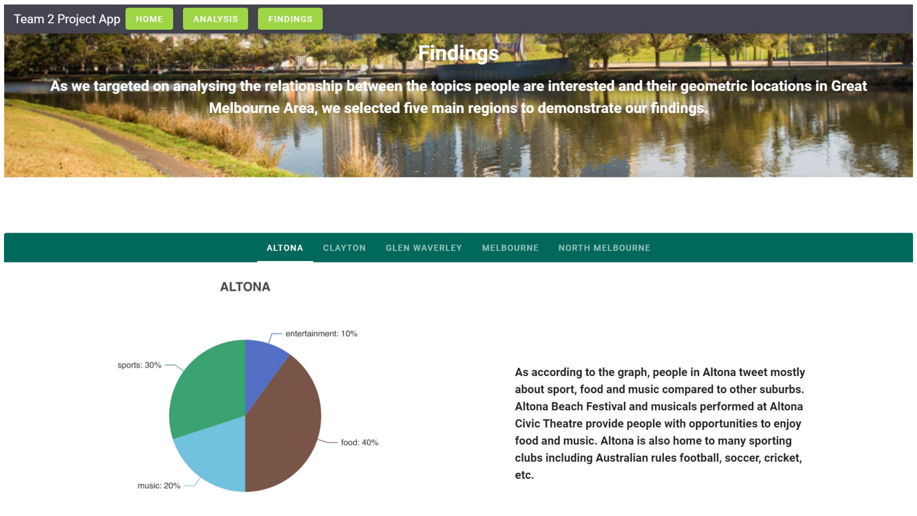
Users can click the button on the navigation bar to switch to other pages. On our **Analysis** page as Figure 3 shows, users can interact with Google Map and discover some interesting facts related to the semantics of tweets. Users can search for particular suburbs in the searching box and corresponding polygons will show on the map.

Our findings in a suburb are presented in the pie chart. Users can click on the polygons of interest to explore the result. Generally, a pie chart contains the relevant topics in that region and the number of tweets that represent each topic. Users can also find popular topics in each region intuitively and decide which topics they should focus on in the business.



Please select a region and click on it to visualize the result.

Figure 3: Analysis page



Findings

As we targeted on analysing the relationship between the topics people are interested and their geometric locations in Great Melbourne Area, we selected five main regions to demonstrate our findings.

As according to the graph, people in Altona tweet mostly about sport, food and music compared to other suburbs. Altona Beach Festival and musicals performed at Altona Civic Theatre provide people with opportunities to enjoy food and music. Altona is also home to many sporting clubs including Australian rules football, soccer, cricket, etc.

Figure 4: Findings page

On the **Findings** page as Figure 4 shows, we highlight some suburbs that have more obvious characteristics based on our knowledge. Users can refer to these examples and apply them to their own business case. The link of our demonstration video is `https://www.youtube.com/watch?v=vHLvIn_jtGA`

4 System Design and Architecture

4.1 Architecture

4.1.1 Overview

This project is deployed on 4 servers, whose architecture is shown in Figure 5. The servers can be divided into 2 parts, namely *Web Service* and *Database Cluster*. *Web Service* contains the front-end, back-end and a Redis instance used by Twitter harvester, while each server in *Database Cluster* have a node of CouchDB and a process that holds both Twitter harvester and text analyser (classifier).

Since the backend is only a wrapper of the CouchDB, it is less source-intensive. So it is deployed to one server only, while there are three servers in *Database Cluster*. Because both Map-Reduce and Classifier are CPU intensive.

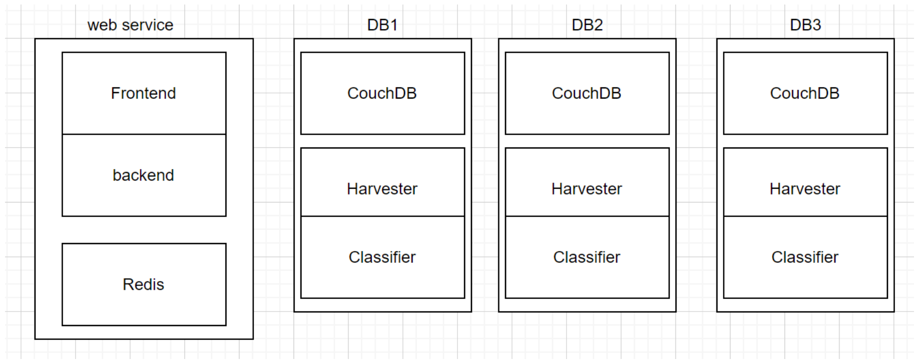


Figure 5: System Architecture

4.1.2 Fault Tolerance

The server of *webservices* does not have fault tolerance. However, CouchDB is running in cluster mode, and the worker nodes of CouchDB are fault-tolerable. The Python tweet harvester instances are fully independent, meaning that if one instance crashes or a new instance is created, it will not affect the effectiveness of other instances. (See: 4.2.7)

4.1.3 Dynamically Scaling-Out

Since only the Map-Reduce functionality of CouchDB, Twitter harvester and Twitter classifier are resource-intensive, the ability to dynamically scale-out is only applied to these services. It is possible to scale out them by simply adding more server IPs into *db_workers* section of *hosts.ini*, and use Ansible to deploy again. However, you still need to manually use the HTTP request [1] to add a CouchDB node to an existing cluster.

The Twitter harvester services work together via document metadata stored in CouchDB. Each instance of the harvester attempts to acquire and collect the most out-of-date location's tweets and will acquire the next best location's tweets if a parallel service acquires it first. Given enough accessible Twitter tokens, additional servers will scale the processing speed in a linear fashion.

4.2 Tweet Harvesting

4.2.1 Overview

The Twitter harvester service is a Python service which makes requests to the Twitter API, classifies them, and stores tweets as documents on the CouchDB cluster.

4.2.2 Data Storage

A CouchDB connection is made in the Python service via IBM's Cloudant Python library. Cloudant was chosen due to its simpler use of partitioned databases.
The documents are bulk inserted using the /db/_bulk_docs POST request. We found in preliminary testing that bulk inserting 500 tweets would take under 1s on average, in contrast to 40s when inserting individual tweets.

4.2.3 Twitter v2 API

The *GET /2/tweets/search/all* endpoint was chosen to efficiently collect the relevant tweets. The Twitter v2 API accepts several more inputs 500 tweets per request. One month per request. This endpoint is only accessible for

[1] `https://docs.couchdb.org/en/stable/cluster/nodes.html#adding-a-node`

academic tokens, so each member of the team applied for and received an academic token for the application.

4.2.4 Token Limiting

The Twitter API limits the number of requests that can be made over different specified intervals. The /2/tweets/search/all requires that no more than one request per minute, and 300 requests per 15 minutes be made. To handle this, we record the usage of tokens in the token documents. Each token document contains:

```
{
  "token": (twitter academic bearer token),
  "last_used": (most recent access),
  "2/tweets/search/all": {
    "since": (first use in 15-minute window),
    "total": (total accesses made in the window)
  }
}
```

Initially, the document only contains the “token” field, and the service adds and modifies the metadata during runtime. The 15-minute window information is stored as a start timestamp, and a total request count; If more than 15 minutes has passed since the “since” value, the endpoint usage fields will be reset. Only tokens which have passed at least one second since the “last_used” field and have under 300 accesses in the “total” field will be selected; ensuring Twitter’s enforced token limit will not be violated.

4.2.5 Location Filtering

Due to the limitations of the requests, the tweet location area and time intervals were carefully selected. Each polygon area of interest has a corresponding location document found in the “features” database. Each location is defined by a geographic bounding box, contains some metadata about the location and stores the time interval for which tweets have been collected.

Each location document contains:

```
{
  "box": [
    (smaller longitude),
    (smaller latitude),
    (larger longitude),
    (larger latitude)
  ],
  "name": (suburb name),
  "loc_pid": (location id),
  "newest": (newest tweet timestamp),
  "oldest": (oldest tweet timestamp)
}
```

The most recent tweets were prioritised, so the location select query finds locations which have not been updated in over 1 day and fine sorts by "newest" (ascending) to find the least recent updated location. If all locations have been updated within a day, the historical tweets are then accessed; The least recently updated history location is then selected sorted by "oldest" (descending).

4.2.6 Tweet Augmentation

Once Tweets have been returned by the Twitter API, the text is classified by our PyTorch BERT classifier (detailed in the following section 4.3) and appended to the tweet data before being stored in CouchDB. The location metadata and Twitter user metadata are also included in the Tweet document before storage. This additional data is intended for later iterations of the application which may include additional statistics such as total users tweeting about a specified category.

4.2.7 Conflict Handling

CouchDB uses multiversion concurrency control instead of transactions, so to handle conflicting updates, we restart the query if the document has a conflicting revision MVCC token. This was necessary to avoid overusing tokens, and also to select unprocessed locations. Because all servers try to access the same least-recently-modified location document, a race condition is created where multiple servers will try to process the same document. This was simple to handle with the token documents, because their select query was based on a value that would be immediately updated (*last_used*). For location documents, we needed to mark a document's status as 'in use' without updating the *newest* or *oldest* fields until completion. To do this we simply added a status field to the location documents.

We considered dividing the workload by allocating different locations or time intervals to different services, but due to the speed of acquiring vs. processing and the simplicity of spinning up additional servers independently, we decided that this approach was appropriate.

To avoid duplicate records, we also uniquely identified each document. Tweet documents are uniquely identified by their unique Tweet IDs, location documents are identified by the SHA1 hash of their bounding box, and token documents are randomly allocated IDs, because they will not overlap during execution.

4.3 Tweet Classification

4.3.1 Classification Task

The overall task for the classification was to identify the topics that twitter users were talking about in order to categorise which product categories these tweets belonged to. For this purpose we built a classifier that predicts what product category each harvested tweet belongs to. We had 10 product classes including: *entertainment, business, technology, gaming, music, sports, politics, fashion, health* and *food*.

For the task of predicting these product categories that each tweet is talking about, we built a text classifier using the BERT (transformer-based) pre-trained language model (Devlin et al., 2019). Unlike traditional Recurrent Neural Network models, BERT is a transformer-based neural model which takes the word context into consideration. Where recurrent neural models use traditional word embeddings which applies the same word embeddings to words in different contexts (e.g. *duck* in the sentences “I had *duck* for dinner” and “*Duck* or you’ll get hit” have two different meanings), BERT assigns unique word embeddings for each word, generating contextualised word embeddings. Hence we can utilise word context for better classifications when using BERT.

4.3.2 Model Architecture and Training

For the task of *training* the model for classification, we used a dataset from the ***average-joe-tweet-classification***[2] dataset from github, which had datasets relating to 10 different topics on product categories. The tweets in these datasets were collected using key-word matching, using keywords relating to the specific classes. Each class consisted of around 30k tweets. Hence the entire dataset consisted of over 300k tweets.

[2]`https://github.com/JasonzxZhang/average-joe-tweet-classification/tree/master/data/tweets`

For the purpose of training and evaluation of our classification model, we spit this dataset of 300k tweets into training and development datasets where 80% of the data would be in the training set, and 20% in the development set.

Using this dataset, we trained a BERT model (Devlin et al., 2019) on the 10 classes using the following hyper-parameters:

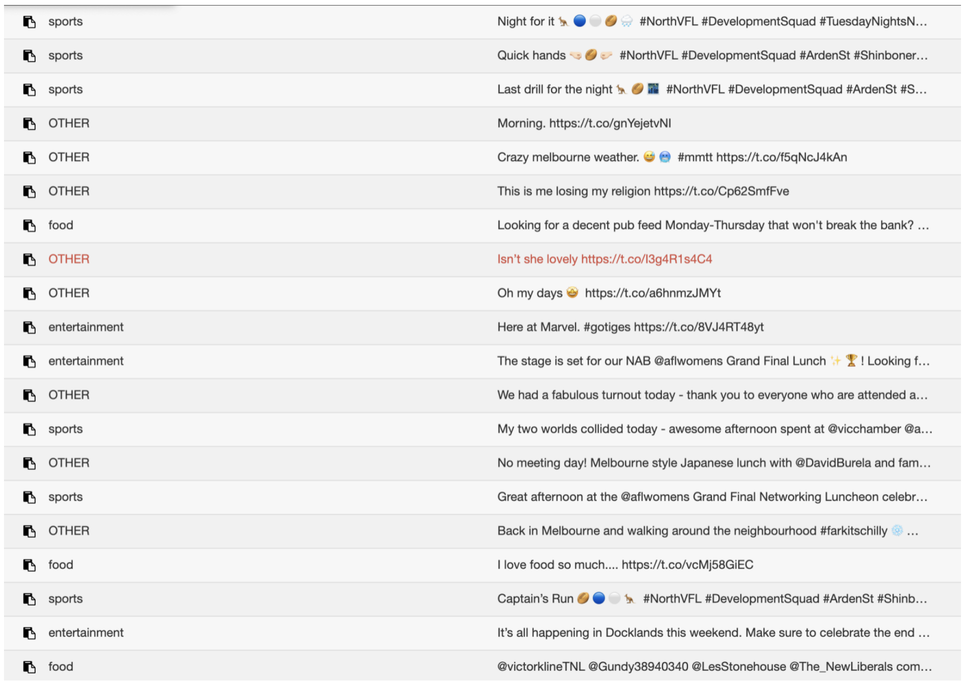
- Batch size = 30
- Max Length = 128
- Learning Rate = 2e-5
- Epochs = 1

The BERT model with these hyper-parameters was trained on the dataset consisting of the 10 classes discussed above using Google Colab using a Tesla T4 GPU. The reason we used Google Colab was was because we required GPU processing to train the model it as little time as possible. Since the training dataset consisted of over 240k instances, the training of the model took around four hours. Since Google Colab only allows a limited time of continuous training at a time, we were only able to use one epoch to train the model. However the evaluation metrics obtained using the development dataset showed an F1 score of over 90% (Table 1).

Metrics	**Value**
Training Loss	0.32
Validation Loss	0.29
F1 Score (Macro)	0.905
F1 Score (Micro)	0.902

Table 1: BERT Model Evaluation Metrics

According to these evaluation metrics, in was evident that the model performed quite well even though it was trained only on one epoch. In this case, we observed that the model predicted the tweets with the word contexts quite well, as evident in Figure 6.



sports	Night for it 🦘 🔵 ⚪ 🏉 🌨️ #NorthVFL #DevelopmentSquad #TuesdayNightsN…
sports	Quick hands 🤜 🏉 🤛 #NorthVFL #DevelopmentSquad #ArdenSt #Shinboner…
sports	Last drill for the night 🦘 🏉 🌃 #NorthVFL #DevelopmentSquad #ArdenSt #S…
OTHER	Morning. https://t.co/gnYejetvNI
OTHER	Crazy melbourne weather. 😅 🥶 #mmtt https://t.co/f5qNcJ4kAn
OTHER	This is me losing my religion https://t.co/Cp62SmfFve
food	Looking for a decent pub feed Monday-Thursday that won't break the bank? …
OTHER	Isn’t she lovely https://t.co/l3g4R1s4C4
OTHER	Oh my days 🤩 https://t.co/a6hnmzJMYt
entertainment	Here at Marvel. #gotiges https://t.co/8VJ4RT48yt
entertainment	The stage is set for our NAB @aflwomens Grand Final Lunch ✨ 🏆 ! Looking f…
OTHER	We had a fabulous turnout today - thank you to everyone who are attended a…
sports	My two worlds collided today - awesome afternoon spent at @vicchamber @a…
OTHER	No meeting day! Melbourne style Japanese lunch with @DavidBurela and fam…
sports	Great afternoon at the @aflwomens Grand Final Networking Luncheon celebr…
OTHER	Back in Melbourne and walking around the neighbourhood #farkitschilly ❄️ …
food	I love food so much.... https://t.co/vcMj58GiEC
sports	Captain’s Run 🏉 🔵 ⚪ 🦘 #NorthVFL #DevelopmentSquad #ArdenSt #Shinb…
entertainment	It’s all happening in Docklands this weekend. Make sure to celebrate the end …
food	@victorklineTNL @Gundy38940340 @LesStonehouse @The_NewLiberals com…

Figure 6: Predictions of the harvested tweets.

4.3.3 Fine-tuning the classifier.

Although this classification model performed quite well according to the evaluation metrics, in order to minimise ambiguity in the predictions, another class was added. This class was called `OTHER`.

This `OTHER` class included low confidence predictions (comparing the prediction probabilities). In this case, we defined a threshold value (0.55 in this case). If the prediction probability (confidence) was less than the defined threshold value, the prediction was *rejected* and put into the `OTHER` class.

This process is also known as ***Rejection-Based Decision Making***. Hence using *rejection-based decision making*, we can reduce the ambiguity in the predictions by *rejecting* uncertain predictions from our analysis.

4.3.4 Predictor Script

Once the BERT classification model was trained and evaluated, it was stored on Google Drive to be used for the predictor script when harvesting each tweet.

Using this trained BERT model, we designed a script which can be used as a *prediction library* that can be imported in the tweet harvester script to predict each tweet as they are harvested.

One important aspect in using this predictor script was the time it took to predict each tweet as they were harvested. Since we were predicting each tweet as they were harvested, the prediction time of the harvested tweet needed to be as low as possible in order to harvest as much tweets as possible in the given time period. When testing out the predictor script locally, the average prediction time for each tweet was around 0.6 seconds. This meant that around 6k tweets could be classified in one hour. However this number will significantly decrease when we harvest each tweet before predicting each of them since we need to consider the time to harvest the tweets as well.

For this reason, we tested out several other classifiers including using the DistilBERT model (Sanh et al., 2019), which is a simplified version of the BERT model (Devlin et al., 2019) . We also built a completely rule-based classifier (no machine learning) which assigned a score for the words in each class based on the word frequencies that occur in our defined 10 classes. However we observed that both these alternative methods did not perform better than the original BERT model in prediction time. Hence we stuck with the original classifier in this case.

When testing out in the MRC, we observed that each tweet was harvested and classified in around 0.75 seconds, which although significant, gave us the ability to harvest, predict and store around 4500 tweets per hour on average in couchDB.

4.3.5 Limitations of the classifier

One of the main limitations of the classifier is that the predictions made on the tweets are highly dependant towards the dataset we used to train the classifier.

Since we trained and evaluated our model on the dataset we obtained from *average-joe-tweet-classification*[3] dataset, our predictions will be biased towards these data instances we have for each class. This is because although this dataset was formed by extracting tweets based on the keywords pertaining to each class, the tweets may not actually be talking about the products in a positive manner. They may even have a completely different context when talking about these *keywords* on the respective classes.

[3] `https://github.com/JasonzxZhang/average-joe-tweet-classification/tree/master/data/tweets`

E.g. tweets that were talking *negatively* about the food served for them in a restaurant, may have been classified as actually *talking about food.* Hence although the model performed quite well in terms of the evaluation metrics, these predictions will not be helpful and may also provide conflicting information to the users.

A better approach to improve upon our current existing model would be to use the sentiments of each tweet as features where the model will use multitask learning to predict the product category as well as the sentiment towards the product category. This will give users the ability to ascertain what the twitter users are talking about along with the overall sentiment towards the product, giving users more information to work with.

4.4 Front End

4.4.1 Background

Nowadays, the majority of web applications use JavaScript to shape forms of web pages. The traditional methods can achieve functionality based on multi-page HTML documents. However, the whole HTML document may be loaded whenever new contents are added which is not efficient. Therefore, modern frameworks such as Vue.js and React.js use the single-page strategy to develop applications. That means only the parts that are altered by users would be updated, therefore the loading time can be reduced significantly in a large project to improve user experience. Considering the technology stack of the team, we decide to use Vue.js to build our frontend pages.

4.4.2 Vue.js

Vue.js is a lightweight library for building user interfaces created by Evan You. Model–view–controller is a software design pattern that is commonly used for developing user interfaces. Vue.js focuses on the view layer to provide functional components for users. Generally, a Vue.js application is based on numerous components which are relatively independent and reusable in many use cases. Vue uses virtual DOM to detect DOM elements. When the nodes in the DOM tree are altered, new values would be compared with the old ones and Vue only updates the parts that have been changed. This property ensures the effectiveness of frontend applications and also accelerates the speed of loading elements in the both development and production environment.

Many third-party packages from the Vue ecosystem are also used to enhance the performance of the frontend. Vuex is responsible for state management and store components using a centralized pattern. It can mutate states in both synchronous and asynchronous ways. Vue-router is quite helpful when dealing with routing issues. It can provide high readability for URLs and manage components hierarchically. All the components can be registered globally in the *main.js* file.

On top of Vue framework, Vuetify is used as the main Vue UI library for creating and managing UI components. It is quite flexible for arrangement in layouts whatever the size of screen or types of devices are applied. Vuetify also applies the principles of web accessibility to eliminate the barriers to prevent effective interaction between users and websites.

4.4.3 Echarts

Echarts is an open-source JavaScript library supported by Apache providing numerous interactive charts and graphs for data display. It is also integrated with Vue framework well. In *vue-echarts* library, all available charts are exposed as components and they can be easily invocated through <*v-chart*> tag. The chart can be configured in option including types, titles and other properties.

4.4.4 Google Map

Our analysis results are mainly displayed on Google Map. In order to make the best use of Vue framework, we use *gamp-vue* library to design Google Map and also integrates it with *vue-echarts* to display results. Google Map can be registered as follows:

```
Vue.use(GmapVue, {
  load: {
    key: KEY
  },
  installComponents: true
})
```

After that, Google Map can be used in the template with <*gmap*> tag. All the available components including polygons and info-windows can be used as child components under.

4.4.5 Polygon Extraction

We use the dataset from Aurin called “PSMA - Administrative Boundaries - Suburbs/Localities (Polygon) August 2020” to extract the polygons of

Greater Melbourne with several properties. The file is wrapped in GeoJson and the general structure for the file is as follows:

```
{
    type:...
    bbox:...
    crs:...
    features: [
        {
            type: "Feature"
            geometry: {
                type: "MultiPolygon"
                coordinates: []
            },
            properties: {
                prim_pcode: ""
                loc_pid:...
                name: "WEST MELBOURNE"
                state_pid: "2"
            },
            id: ...
        },...
    ]
}
```

We mainly use the information in the geometry to extract coordinates and region names in the properties. Then we aggregate corresponding regions and coordinates in an array that is ready to draw polygons. Two functions are working on this part. The first one is *getAllPolygons* which can return all polygons of Greater Melbourne and show the polygons on Google Map. The other one is called *getPolygonsByNames* and it provides an interface to get particular polygons when users seek suburbs through the searching box. We use the *GmapPolygon* component that wraps Google Map APIs to draw polygons. The property *paths* define coordinates of regions and we create all polygons iteratively.

```
<GmapPolygon
 v-for="(r, i) in regions"
    :key="i"
    :paths="r.path"
    :options="{ ...polygon_options, ...extra_options[r.name] }"
    @mouseover="togglePolygon(r, true)"
    @mouseout="togglePolygon(r, false)"
    @click="toggleInfoWindow(r)"
/>
```

The default frontend page is shown as the following:

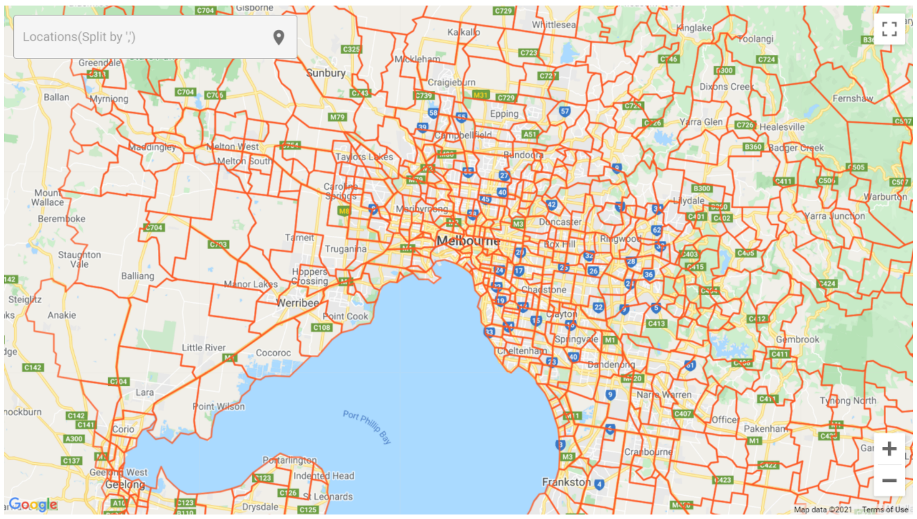


Figure 7: Google Map polygons

When users enter the analysis page, a default map with all polygons of Greater Melbourne is shown in Figure 1. Generally, all polygons are separate components on the top of Google Map. Each of them listens to three main events. They are *mouseover*, *mouseout* and *click* respectively. When the mouse focuses on a particular region, the *mouseover* event is triggered and then the colour on that region becomes darker. The polygon would go back to transparent when the mouse moves out of that region. If users click on a particular region, the analysis result shows in the info window. An *InfoWindow* can display contents in a popup window above the map. The window location is defined when users click on a particular polygon. The coordinate of an *InfoWindow* is calculated as intermediate values of corresponding coordinates of the polygons in that region. A pie chart is integrated with the *InfoWindow*.

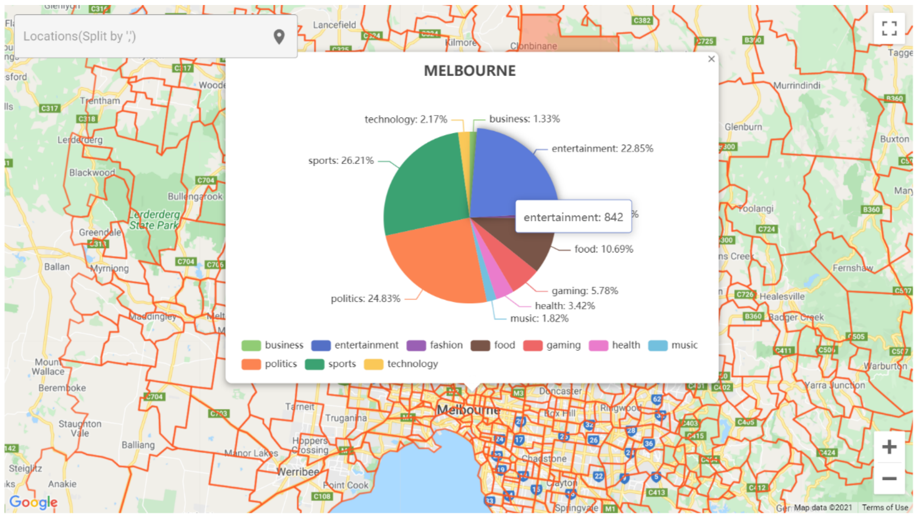


Figure 8: Result display

Figure 8 shows one analysis result in the Melbourne suburb. The pie chart is embedded in *InfoWindow* and it's also interactive when users move the mouse to different sections. Around the pie chart, the percentage of each category is shown. The number of tweets that are classified into that category is also displayed in the toolbox above the chart.

4.4.6 Backend connection

Vue.js is used to build frontend user interface. The application is a single-page application, with backend and frontend separated. On top of Vue framework, Vuetify is used as the main Vue UI library for creating and managing UI components. To make HTTP request to backend, Axios, a promised based HTTP client, is used.
Axios.get is used to send asynchronous GET request to backend REST endpoints to retrieve analyzed twitter results. Analyzed twitter results are sent as a response back to frontend and frontend awaits for this result.

```
res = await this._vm.
$axios.get('/api/analysis/suburb_category/record')
```

4.5 Back End

The backend of web application is implemented with Python Flask, which is a micro framework for building RESTful APIs. Flask supports various

HTTP protocols, including GET, POST, PUT, etc. from specified URL to send and retrieve data.
The design of web application is a single page application, with frontend and back end separated from each other. The pros of this design is that the change made in frontend will have minimum impact on the backend. Frontend built content is deployed into a dist folder. Backend serves index.html located in this dist folder.
The serve port for web application is configured to be 9797. All view functions that are used to handle HTTP requests are managed by Blueprint, which is registered with flask app. When a request is made through a URL, associated view function will be called and its value returns as a response to the frontend. A simplified code snippet is attached below to demonstrate the basic implementation of flask app.

```
from flask import Flask, render_template
def _main():
    app = Flask(__name__, static_folder=dist_folder,
    template_folder=dist_folder)
    register_blueprints(app, 'blueprints', ['./blueprints'])

    @app.route('/')
    def index():
        return render_template("index.html")

if __name__ == '__main__':
    _main()
```

4.6 Communication Between Back End and CouchDB

Python couchdb is installed to work with CouchDB database. The following command is used to connect to remote CouchDB server served in Melbourne Research Cloud. This connection enables a direct connection with CouchDB to get data, make updates to a document or delete data.

```
couchdb = couchdb.Server(Config.COUCHDB_URL)
twitter = couchdb['twitter']
```

4.7 Communication Between Front End and Back End

As mentioned above, HTTP requests is managed by blueprint. In the application, a GET request will be made to backend to request for category data (topics) and calculated tweets in percentage for all regions. Upon receiving this request, backend will get the data from CouchDB and return the retrieved data in json format to frontend.

```
bp.route('/<id>/record', methods=['GET'])
def get_records(id):
  # Get all records of a view by view id
  res = list(twitter.view(id+'/all', group=True))
  return jsonify({
    'result': res
  })
```

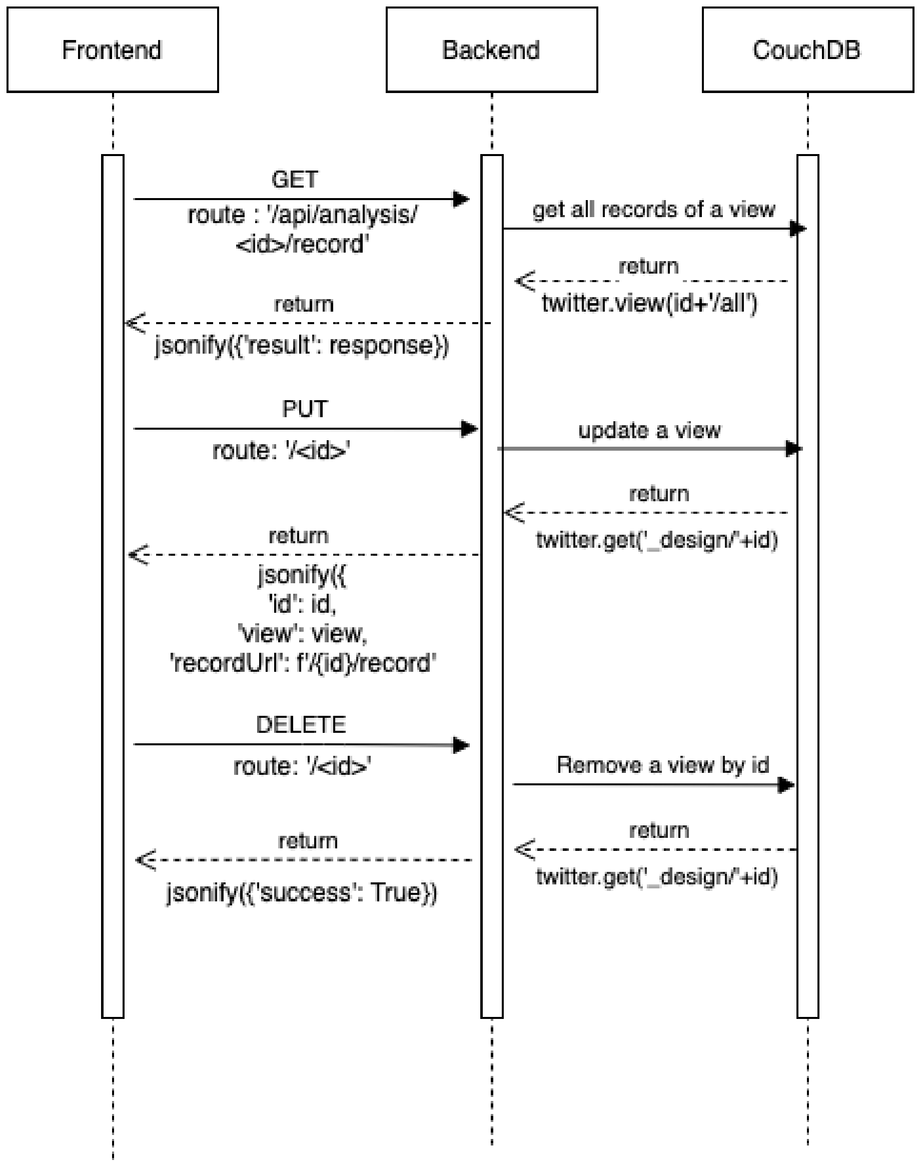

Frontend
Backend
CouchDB
GET
route : '/api/analysis/<id>/record'
get all records of a view
return
twitter.view(id+'/all')
return
jsonify({'result': response})
PUT
route: '/<id>'
update a view
return
twitter.get('_design/'+id)
return
jsonify({
'id': id,
'view': view,
'recordUrl': f'/{id}/record'
DELETE
route: '/<id>'
Remove a view by id
return
twitter.get('_design/'+id)
return
jsonify({'success': True})

Figure 9: Backend RESTful Design

4.8 The API of the Back End

The back end provides APIs to modify the analysis (CouchDB view) as well as returning the result of an analysis.

There is a swagger document that shows the API of the back-end, which is deployed in `http://172.26.134.58/apidocs/`. Following are the 4 APIs provided by the backend.

4.8.1 GET /api/analysis/

It returns all supported analysis. Following is an example of the returned value.

```
[
  {
    "id": "suburb_category",
    "recordUrl": "/suburb_category/record",
    "view": {
      "map": "// the map function in javascript",
      "reduce": "// the reduce function in javascript"
    }
  }
]
```

4.8.2 DELETE /api/analysis/{id}

It removes a CouchDB view by id provided in the url and returns {*"success": true*}.

4.8.3 PUT /api/analysis/{id}

It updates a view by the id provided in the url. The input is an *application/json* and the output has the same format as the return value in Section 4.8.1.

4.8.4 GET /api/analysis/{id}/record

It returns the result of a view (all records produced by map-reduce). An example of that is shown below.

```
{
  "result": [
    {
      "key": [
        "MELBOURNE",
        "business"
      ],
      "value": 257
    }
  ]
}
```

4.9 Map-Reduce in CouchDB

We use Map-Reduce to associate the geographic position with suburbs, and count the number of tweets of each category in suburbs in Melbourne. The source codes are located in *twitter-analyzer*, which include the code of the JavaScript function of the view named *suburb_category* and the code to upload them into the CouchDB.

The *map* function of the view uses a package called *geojson-geometries-lookup* to match the geolocation of each tweet into polygons of the suburbs in Melbourne. Both package and a JSON that contains the polygons are packed into one JavaScript file by Rollup.js [4]. The function emits the name of the suburb from which the tweet is sent and the category of the tweet.

The *reduce* function of the view is just a built-in function in CouchDB called *_count*, which counts the number of the existence of each key. So we can get the number of tweets of each category in suburbs by aggregate the result.

The view is uploaded by *index.js* in the *twitter-analyser* using the HTTP interface of CouchDB. The docker image that contains the uploader and the JavaScript Function is run in the CouchDB head node once by Ansible. In addition, other configurations such as the user name and password of the cluster can be read from environment variables.

5 Pro and Cons of the Tools and Processes

5.1 UniMelb Research Cloud

The pros of the MRC are that it is free to use and supports OpenStack, which means that all infrastructures that can be deployed by OpenStack can be migrated to MRC easily. However, there are more cons of MRC compared to the pros, which can be listed below.

To begin with, it is just Infrastructure as a Service (IaaS). It has few supports of the functionality such as PaaS and SaaS compared with other

[4] `https://github.com/rollup/rollup`

cloud products such as AWS. For example, it has limited features in visualization, querying and logging, and unlike the CloudWatch in AWS, it does support setting up alarms to notify us when certain resources are over a limit.

Last but not the least, we do not have permission to assign a public IP to a server, which means that we and the user of our project have to use a VPN to access the server.

5.2 Docker

The advantage of Docker can be listed below.

Firstly, the docker build script is easy to deploy because it contains all the necessary information for an application to run such as the operating system (Linux distribution) and required software, and deploying a Docker container is just simply running a docker command.

Secondly, the docker technology can be easy to test compared to Ansible because a well-formed Dockerfile can be quick to built thanks to the layered structure of the docker image.

However, there are still some disadvantages of docker.

To start with, it has some difficulties when it is used with Ansible. One of them is that we need to build the docker image in our local machine, push it into a docker registry such as docker hub and use another script to let Ansible download it into the target server, which is time-consuming and introducing extra unnecessary steps. Besides, the private repository in the docker hub is a paid service. In my point of view, things can be better if docker supports pushing an image directly into another docker host.

Secondly, each layer of docker image strictly relies on the previous one, and it can lead to the waste of the disk space, which means that, if two applications are based on different images such as *node:16* and *python:3.9*, separately, and the two images are based on different operating systems such as *Ubuntu* and *Debian*, redundant files of the two systems will be downloaded, although the two interpreters can work on the identical system.

6 Conclusion

Overall, our system provides users with information about the product topics the twitter users talk about most (according to our definition) in different suburbs in Greater Melbourne. Currently the system shows the analysis and breakdown for each suburb for the total harvested tweets in couchDB. One way to improve upon this performance is to categorise these tweets in couchDB based on the time they were created. In other words we could obtain a *monthly* product category analysis based on the created time of the harvested tweets in the suburbs. This could provide the users with a more granular analysis, leading to a temporal analysis of the product categories talked about in the suburbs in Greater Melbourne.

One aspect where our system improves upon AURIN is that users can obtain almost *real time* analysis on the product categories most tweeted about. This analysis can be used further down the line for targeted marketing for each suburb in greater Melbourne.

There are limitations in our system, mainly the use of Twitter API limits our system to *twitter users*. However compared to Americans, not many Australians use twitter on a daily basis, which limits the data collection. Also, we harvest tweets in the suburbs in Greater Melbourne using the geo-location of the tweets. One observation we made was that the tweets that include geo-location was a small fraction of the tweets provided by the Twitter API, which limits our analysis to a rather small number of tweets. Also, the Twitter API only provides a subsection of the total tweets posted on Twitter, limiting the data collection process. One way to improve upon this system is to use other social media platforms with geo-location (such as facebook) to maximise the data collection process to obtain a better, more granular monthly analysis for each suburb.

Another limitation is that the username and password of CouchDB is not encrypted in the Ansible vault, which may cause security issues.

References

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.