COMP90024 Cluster and Cloud Computing
S1 2017

Assignment 2 - Australian City Analytics

Team 33
David Barrell (520704)
Steve Dang (807773)
Bobby Koteski (696567)

Table of Contents

I. Introduction

Melbourne has been proud and praised for six consecutive years by the Economist as “the most liveable city in the world” for its peaceful city surrounded by beautiful beaches, with a well-known coffee culture, restaurants, night-life and love for sports [1]. The city advertises itself as a hub of multicultural communities coming from all over the world. Its high reputation has attracted a wave of hundreds of thousands of foreigners coming in as tourists, students, holiday workers and immigrants. This amazing title naturally sparks our curiosity to know how the people feel when they are in Melbourne: ***does living in the most liveable city make people’s life easier and more satisfying? Do people from around the world, the non-local ones, feel good about the city when they come here to work, study or migrate? And has Melbourne become a diverse international place due to its high reputation?***

Social media offer great channels for us to access people’s thoughts. Twitter, the ubiquitous microblogging service, with 328 millions of total active users and 2.8 millions in Australia alone [2], who constantly share millions of tweets of their opinions, photos and videos on a daily basis, becomes a great tool to help find answers for those exciting questions. In term of technology, Twitter also generously offers a free public API that allows developers to access its vast network of contents by streaming or searching tweets from anywhere, and anyone in the world to build useful social applications.

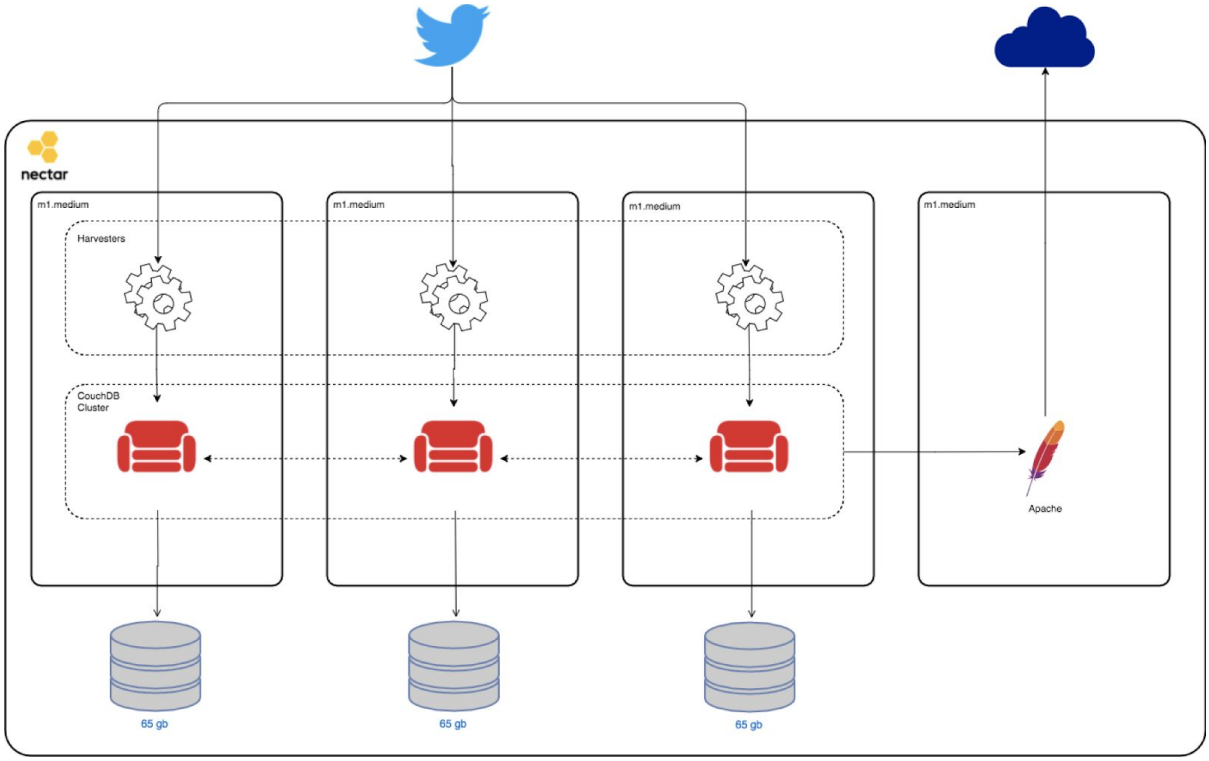
Leveraging a large amount of information from Twitter along with big data and cloud analytics tools such as NeCTAR, CouchDB, MapReduce, Google Cloud API, Python libraries, our team have been able to create a simple software application to analyse how non-English Twitter users in Melbourne have been thinking by analysing the content and languages of their tweets. This paper will firstly describe the technology we used to gather, process, and analyse large datasets, and secondly offer our findings throughout the process. Additionally, the research is conducted with a hope that its findings may help us understand more about the effect of having so many foreigners coming in as well as possibilities to improve dwellers’ lives here.

In a brief summary, this research proves that Melbourne has become a truly multicultural hub with many people speaking many different languages. By analysing the sentiments of tweets’ content, it also reveals that despite its title as the most liveable city, there seem to be as many unhappy moments as happy ones in life for people in Melbourne. Finally, by targeting tweets from university campuses, we also could see how a major group of foreigners – students – feel when they go to schools in Melbourne, and more interestingly how going to different schools may seemingly give you more or less negative and positive moments.

II. System Architecture

1. Architecture

This project's architecture was quite simple, once deployed. The system was deployed on the NeCTAR Research Cloud, with four 'm1.medium' instances running Ubuntu 16.04. Three of these hosted a CouchDB cluster, with the fourth acting as a web server and delivering the front-end using Apache.



System architecture

Each of the three instances running CouchDB had attached a 65 gb volume, used to store the databases, and any large files used in the course of the project.

2. System Design

Many design choices were made in the fabrication of this architecture. The arrangement of the CouchDB databases was the first major decision point, as it acted as the backbone of our data storage and processing. To make use of the resources available, either a cluster could be set up, or a series of replications used. Both had their advantages and disadvantages. Replications are an easy-to-use tool, that has been proven effective and reliable, however does not give any performance enhancements. They must also be set up for any new database. Clusters, being a relatively new feature (as of CouchDB 2.0), are not as well-documented, however they allow for separate instances to be treated as one

installation, as well as providing increased resources for indexing views and other such tasks.

It was decided that a cluster of three nodes would be used, to take advantage of the performance increases, as well as to experiment with the new features. This presented a number of challenges, during deployment, as well as in production. Taking advantage of the cluster's in-built replication, each database was created in 8 shards and 3 replicas. In a three-node cluster, this meant that each database existed entirely on each node, allowing for recovery from a single node if necessary.

In order to collect tweets, harvester services were set up on the three database instances, each feeding directly into the same database, though each running a different script. One instance used Twitter's Stream API to harvest tweets as they occurred within a geofence around Australia. A second instance was used variously to either search or stream tweets using a number of 'hashtags' and keywords. The third instance then retrieved historical tweets from each user that appeared in the stream, keeping track of previously searched users in a separate database. To account for potential duplicate tweets, we used each tweet's ID as the ID of the CouchDB document, meaning if the same tweet was inserted more than once, it would be rejected due to conflict.

3. Deployment

A major goal of this project was to automate as much as possible. To that end, scripts were written to spin up instances on NeCTAR, and automatically deploy CouchDB and the harvesters. A task runner, Gulp.js, was also used to automate the building and deployment of the front-end to a fourth instance.

The first step in this process was a Python script, utilising the boto package. This connected to NeCTAR using the OpenStack API, and created a number of instances, based on the arguments passed to it. It then created and attached volume for each instance. Finally, it waited for the instances to be fully operational before executing an Ansible playbook.

Ansible is an open-source automation engine, and in this case, was used for deployment and configuration management. We found it extremely useful in this environment, as its extensibility and modularity allowed for simple iterative development. It also allowed us to reproduce environments from scratch, meaning all environments in the development pipeline (development, staging and production) were virtually identical.

A series of tasks, or a 'playbook', is used by Ansible define the steps that should be undertaken. This allows for tasks, and template files, to be defined locally. It also has the advantage that no software needs to be installed on the host before it can be the subject of Ansible tasks. This project's playbook was extensive, in that it not only handled package installation for CouchDB, it also set up the nodes as a cluster, and deployed tweet harvesters on each instance.

Ansible was invaluable in the setup of the cluster, with the locally-defined configuration files ensuring that each CouchDB instance was always running the same configuration, as well as handling the connection of each instance to the cluster.

Challenges

We encountered a number of challenges through the process of deploying and running this architecture. The first issue was automating deployment of CouchDB. Having decided to utilise a cluster arrangement, version 2.0 or above was required, however the highest version available in Ubuntu's packaging tool (APT) was 1.6. The solution to this was to build CouchDB from its source code. This added another level of complexity, with the installation now taking 17 steps, instead of the single step had we used 1.6. It also increased the deployment time substantially, however as these deployment scripts was not going to be used often, it was deemed that the additional build time would not have an impact.

Post-deployment, we also ran into an issue that was difficult to diagnose. After a period of perhaps 3-4 days, the CouchDB databases would no longer function properly; they would not allow new views to be created, and would often reject document insertions. We were considering moving to a replication-based arrangement, thinking that the new cluster features might have a bug, however we soon discovered the issue. A system database called '_global_changes' had grown rapidly in size, to the point where it filled the entire attached volume.

Some manipulation of shards was required, but eventually a compaction was run, and the size of the data file reduced from 46gb to 240kb. This indicated a situation where the small number of documents in the database were being over-ridden very often, with the previous versions being kept. Due to time constraints, we were unable to determine the root cause of this issue, or at least whether it was an issue specifically with the cluster feature or CouchDB as a whole, or simply a misconfiguration.

III. NeCTAR Research Cloud

NeCTAR, the National eResearch Collaboration Tools and Resources, is a cloud infrastructure to support research in Australia. Students and academics at Australian universities have limited access to resources by default and extra resources can be allocated for projects by request.

We have been allocated 4 medium sized virtual machines for the duration of our project. Specifically, each virtual machine has 2 virtual CPUs, 70GB of disk space and 8GB of RAM. Additionally, access to a volume storage has been made available with a capacity of 250GB. During the project we have made use of all 4 virtual machines and most of the volume capacity, mainly to run a distributed CouchDB database and our twitter harvesters.

While working with NeCTAR we observed a number of advantages and disadvantages. A large proportion of this section is dedicated to discussing those. It is worth mentioning that

we did not use all of the features that NeCTAR offers, such as the object store, orchestration, just to name a few. Having said that, we discuss NeCTAR as a service that runs over a distributed network. We offer a high level overview of the advantages and disadvantages below.

Advantages:

- Slick interface
- Detailed tutorials for all OS users
- Instance location
- Functionality to spawn instances with scripts, API access
- Easy access to popular images, i.e., Ubuntu 16.04
- Ability to set security groups, e.g., open/close ports
- Appears reliable, we did not experience failures
- Back ups are possible

Firstly, NeCTAR has a slick interface. It is comfortable and easy to navigate through the web site, starting from the logon page to the dashboard. We believe that it is right up there with the likes of AWS, Google Cloud, Digital Ocean and MS Azure. This is important so that users of the service don't feel frustrated and saves time by allowing users to navigate without much effort.
Detailed tutorials, with instructions supporting the main operating systems, are published and are easily found with a google search. The existence of these is pivotal to engage a new user of the service. It is often frustrating and time consuming and one is likely not to use a service if it is too difficult and lacks support.

NeCTAR is distributed over a number of locations in Australia. It is possible to generate virtual instances in any one of those locations. The advantage here is that it can help minimise delay and maximise bandwidth when applications are deployed in the one location, provided that the end user of the application is also close to the physical location. We have taken advantage of this and have instantiated our virtual machines in Melbourne. Since we anticipate that we are the only ones who will interact with our application. Extremely beneficial especially transferring large quantities of data over a network. For example, we have collected approximately 11GB of tweets which requires time to transfer over to our website. It is far more efficient to have our webpage and database hosted in the same location.

It is possible to manage NeCTAR services over an API. The ability to auto deploy is an advantage and can reduce deploying a large and complex distributed system down to a click of a button. In the scripts a developer has the ability to completely manage instances, starting from deployment, configuration to editing security groups and terminating instances.

Virtual machines mainly run linux or various linux distributions. NeCTAR makes these easily available as an option when deploying instances. We load the popular Ubuntu 16.04 on our four instances. We view this as an advantage due to the wide array of available linux distributions to fit many application needs. Linux is free and open source and familiar to the

common developer. It can also be argued that linux is less prone to malware due to the fact that the Windows operating system is the main target for attackers, given its market share.

When instances are created it is possible to allocate a security protocols to an instance or a group of instances. Security groups control the opening and closing of operating system ports that are required for communication between applications. NeCTAR employs a one to many architecture in the sense that one security group can be applied to many instances and therefore by changing the security group, it will affect the many instances. This is desirable as it is convenient, reliable and saves time and provides greater control when configuring the virtual machines.

The virtual machines we deployed for the project appear reliable. Fortunately, we did not experience any failures to do with NeCTAR, although discussed elsewhere in this report, we had challenges with CouchDB. In the unlikely event, we prepared for an instance failure by mounting volumes to our instances. In such an event, we would retain the most important CouchDB data.

A snapshot of a virtual machine can be backed up and retrieved at any time. Particularly useful when working with distributed systems that run over a network, where many components can fail and cause the system to stop operating. Having the ability to back up on a regular basis is critical to a recovery procedure and NeCTAR makes this available.
Disadvantages:

- Possible loss of data, virtual instance failures and downtime
- Requires expert knowledge to make use of features, i.e., opening ports
- Delay associated with processing large quantities of data
- Difficulty in moving large files
- Physical access to data not possible
- Security vulnerabilities

On the other hand, there are disadvantages associated with NeCTAR services. One that is not so obvious is that components can fail, whether it's hardware or software, virtual machines can fail. In such events, important data may be lost. To protect against failures, it is vital that developers ensure constant backups offsite. Again this poses another challenge, if the data that is required to back up is large. Moreover, there is potential for downtime of the NeCTAR services and one should prepare for that. Possible causes are maintenance or failures as discussed.

Expert knowledge is required to use NeCTAR safely, that is without exposing the application running on the virtual machines to outside threads. In particular, it is a good idea that one knows about internet technologies and operating systems, i.e., when opening up ports.

Another challenge exists with the use of NeCTAR and services alike to do with processing large quantities of data. It is even more challenging moving data from and to virtual machines. A large json file can take a couple of hours before it can be uploaded to an instance. On the other hand, it is also very time consuming when downloading large files from a virtual machine for backup. It is important for one to consider all these aspects before

committing to use of the cloud as it is often a better idea to keep data locally. Also worth mentioning is that one can access their data on the virtual machine only through the internet.

Lastly, the question of how safe is the data and application on NeCTAR is difficult to answer. Security is a great concern for cloud computing especially since all communication occurs over the internet. It is possible for data to be leaked or compromised and ensuring that the application is secure is vital.

V. Tweet Scenario Analysis

1. Data targeting methodologies

Twitter's streaming API

The main goal of the research started with the idea of finding out how non-local people feel when they are in Melbourne based on their tweets. Twitter offers a free public API that allows developers to stream tweets in real time based on different customized parameters. Using the popular Python library Tweepy, we firstly started by streaming all tweets that come from Melbourne and storing them in a database in our CouchDB on Nectar. For each tweet, Twitter API will return a JSON message that consists of abundant metadata about details of the tweet and its creator. After getting around three million tweets, we proceeded to filter the tweets using MapReduce on CouchDB and solved our first puzzle*: which tweets are from non-local people?*

Hypotheses about non-local users

Since English is the language of Australians, we want to target people for whom English is not their mother tongue. The metadata of each tweet has a field called "lang" that indicates the language of the tweet with language identifiers using BCP 47 codes [3], so the first type of tweets we look for have "lang" that is not "en" which is the identifier of English. Yet, besides people who tweets in their own languages, there are people who come from a different language background but tweet in English. For this type, Twitter's metadata has a field called "lang" within details of the tweet's creator to indicate the language of the account that the user has set up in, so we also target tweets that have user's "lang" is not "en". Obviously, the limitation of this strategy is that we have to neglect people who, for instance, are Vietnamese yet both set up their accounts and tweet in English as the metadata has no indication for such group of people. We are also aware that we might miss people who come from many countries that uses English as their first language and therefore always tweet in English.

For each tweet, we create a new field in its CouchDB document called "community_lang" to assign the language community that the tweet belongs to based on the its "lang" or user's "lang". The algorithm is that if the tweet's "lang" is in English, then its "community_lang" is the same with the user's "lang", which must be not English. If the tweet's "lang" is not English, then its "community_lang" is the same with the tweet's language. There might be

some rare cases when both “lang” fields are not English and have different values, meaning that a non-English user tweets in a non-English language. Again, there is no way to know exactly the user’s language just based on the metadata, so the algorithm makes decision based on mentioned priority.

Throughout this paper, we identify all non-local tweets as non-English tweets which mean they are either written in non-English languages or the user’s account is set up in non-English languages.

Filter retweets, translate the text and assign sentiment scores

	Quantity
Total tweets downloaded	3,811,298
Post-filtered tweets	592,788
Translated tweets	13,237
Unique non-English users	5,438

Statistics of collected/processed tweets

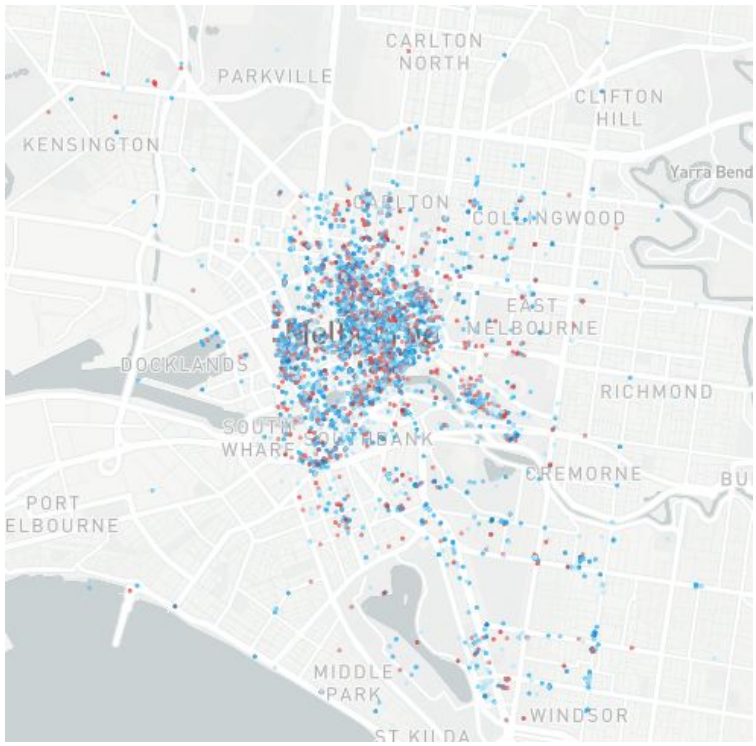
The filtering process also avoids all retweets, which users just repost tweets from others, to reduce the noise that many of them can create from meaningless non-original duplicates. Next, we translate the tweet’s text to English by using Google Translation API. This turns out to be a quite challenging process as the API is not free, hence limits the number of tweets that could be translated.

Next, for each translated tweet, we use the Python library TextBlob to analyse its meaning and assign a sentiment score for it. TextBlob returns two values, polarity for sentiment and subjectivity for the writer’ subjectivity, with a range from negative 1 to positive 1 indicating the sentiment level from negative to positive. TextBlob is a very suitable solution for this task, as its algorithm analyse a text’s sentiment based on occurrences of biased words [4]. For example, a tweet that has the word “excellent” would have a high positive score while one with “terrible” would receive a low score. This works well for this research since each non-English tweet via a translation machine might only become separate unconnected terms in English instead of a complete meaningful phrase or sentence.

Due to the limitation of translation expenses, we also narrow down the number of tweets by avoiding all tweets whose “coordinates” field, which indicate the GeoJSON coordinates, is not “null”. This means that we can visualise every relevant tweet on a map.

2. Visualisation and Sentiment Analysis

2.a. First visualisation: all sentiments from non-English tweets



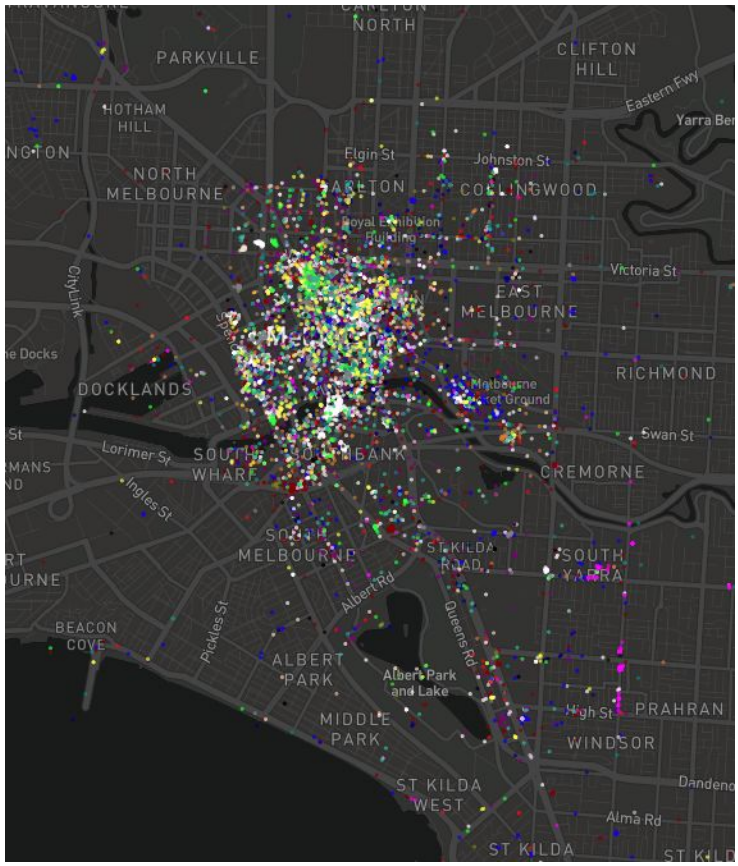
Visualisation of all non-English tweets with sentiment colors

All tweets with GeoJSON coordinates are visualised on maps using Mapbox's API [5] written in JavaScript and JQuery. For the first visualisation, we plot all tweets whose 'lang' or user's 'lang' is not English and that we have assigned sentiment scores. Any tweet with a sentiment score that is greater than zero indicating positivity is coloured in blue, and in red representing negativity for ones with less than zero.

From a high-level bird-eye view, we can see that almost all non-English tweets happen to be tweeted densely within and around Melbourne's CBD area which may clearly have two implications. Firstly, most of non-English users are tourists or students who only come to the city temporally, meaning that this targeted group of Twitter users may allow us to correctly understand non-local people. Secondly, many red dots appear along with the blue ones supposedly indicate that non-local people certainly have negative moments when they are the most liveable city.

2.b. Second visualisation

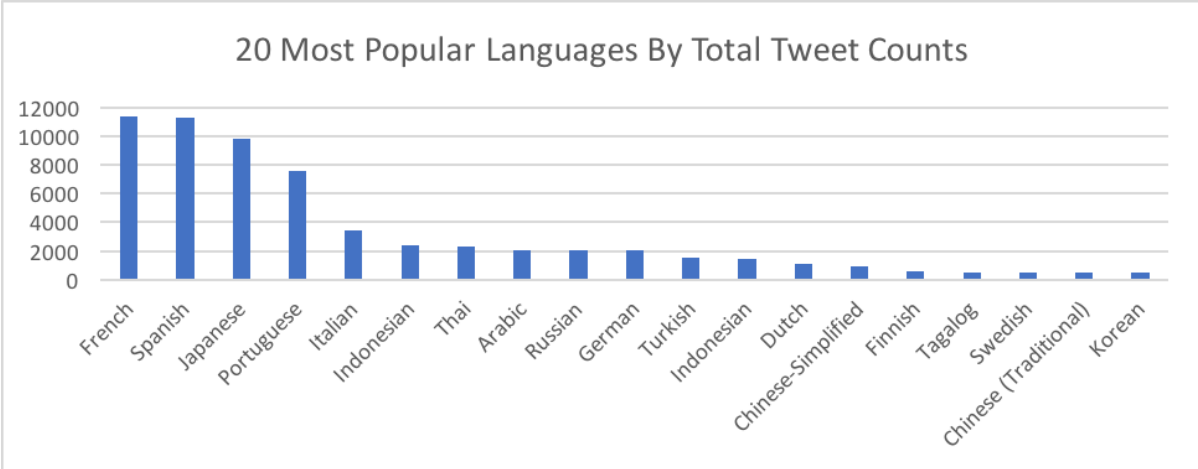
Only positive and negative tweets could only tell so much about the sentiment of non local people. The next question we want to know is: how many non-English languages we may find in the city, thus we visualise the same number of tweets but instead of coloring the dots based on sentiment score, different colors indicate different languages.



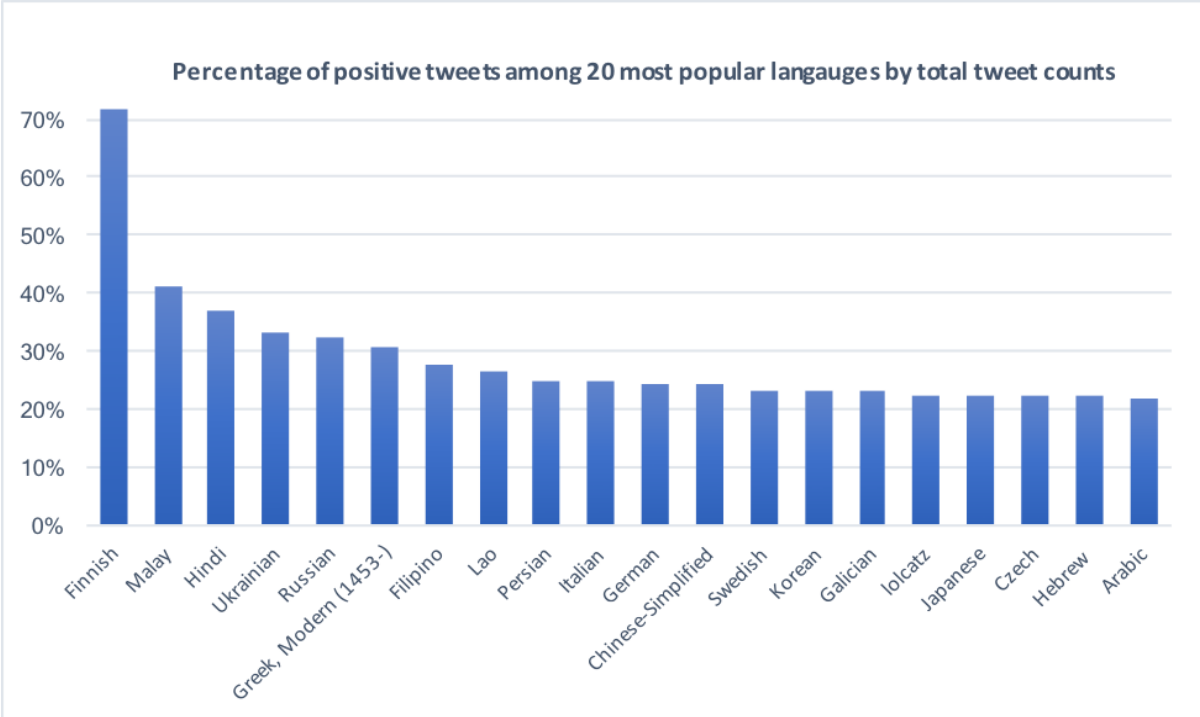
Visualisation of all non-English tweets with different language colors

From the same view, Melbourne's CBD turns out to be beautifully colourful with many different colors. This map validates an important hypothesis for this research and proves that: ***Melbourne is truly a multicultural city where you may find so many different people speaking 61 different languages just within its CBD area.***

The top five most popular languages based on tweet counts are French, Spanish, Japanese, Portuguese and Italian. Based solely on the number of tweets, we find that Finnish has the 70% of the tweets are considered positive. Additionally, the colourful visualisation shows many clusters of dots that have the same colours makes us wonder if they come from different users, meaning people of the same culture background might often hang out in the sample places, or all are from the same users who might happen to tweet from where they live or work at the particular locations.



Number of tweets for common languages



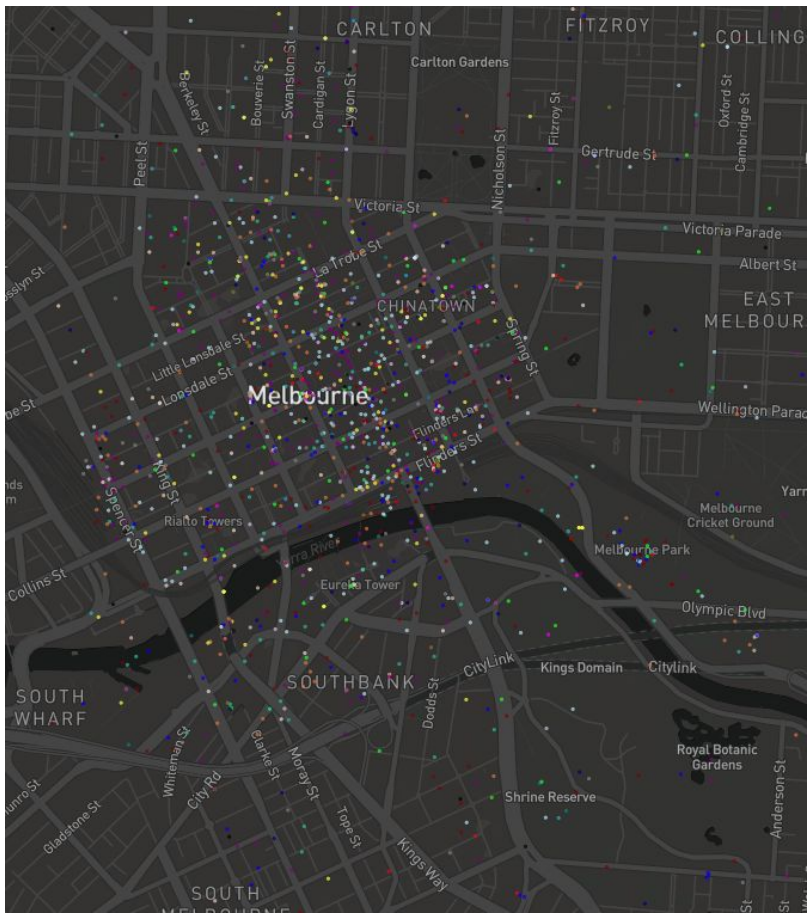
Percentage of positive tweets amongst most common languages

#	language	language id	mean score	positive	negative	total	positive proportion	negative proportion
1	French	fr	0.088247178	2229	549	11358	19.62%	4.83%
2	Spanish	es	0.104697123	2299	389	11282	20.38%	3.45%
3	Japanese	ja	0.111499883	2208	371	9846	22.43%	3.77%
4	Portuguese	pt	0.056923914	866	178	7568	11.44%	2.35%
5	Italian	it	0.127728045	847	101	3430	24.69%	2.94%
6	Indonesian	id	0.097518504	507	106	2431	20.86%	4.36%
7	Thai	th	0.071703781	412	129	2299	17.92%	5.61%
8	lolcatz	xx-lc	0.053705023	478	280	2128	22.46%	13.16%
9	Arabic	ar	0.106044478	453	77	2062	21.97%	3.73%
10	Russian	ru	0.187305968	660	38	2039	32.37%	1.86%
11	German	de	0.130318716	500	56	2030	24.63%	2.76%
12	Turkish	tr	0.040794212	133	38	1529	8.70%	2.49%
13	Indonesian	in	0.053576882	226	92	1484	15.23%	6.20%
14	Dutch	nl	0.096611773	233	48	1098	21.22%	4.37%
15	Chinese-Simplified	zh-cn	0.11256142	238	39	976	24.39%	4.00%
16	Finnish	fi	0.269818786	439	8	611	71.85%	1.31%
17	Tagalog	tl	0.108693724	85	8	522	16.28%	1.53%
18	Swedish	sv	0.099616648	117	34	503	23.26%	6.76%
19	Chinese (Traditional)	zh-tw	0.083574244	98	26	487	20.12%	5.34%
20	Korean	ko	0.113928378	112	26	485	23.09%	5.36%
21	Danish	da	0.031655245	21	4	319	6.58%	1.25%
22	Chinese	zh	0.098957249	70	18	273	25.64%	6.59%
23	Chinese	zh-Hans	0.181506221	78	11	231	33.77%	4.76%
24	Norwegian	no	0.030811487	16	2	221	7.24%	0.90%
25	Polish	pl	0.085233064	33	4	204	16.18%	1.96%
26	Estonian	et	0.025188621	9	2	154	5.84%	1.30%
27	Haitian; Haitian Creole	ht	0.059634387	19	8	138	13.77%	5.80%
28	Greek, Modern (1453-)	el	0.164583846	27	3	88	30.68%	3.41%
29	Hungarian	hu	0.11940634	15	0	84	17.86%	0.00%
30	Catalan; Valencian	ca	0.036179798	11	7	75	14.67%	9.33%
31	Hindi	hi	0.17444417	17	1	46	36.96%	2.17%
32	Czech	cs	0.121021324	10	1	45	22.22%	2.22%
33	Slovenian	sl	-0.008941156	4	4	43	9.30%	9.30%
34	Chinese	zh-CN	0.123520085	10	1	43	23.26%	2.33%
35	Vietnamese	vi	0.093395493	7	0	39	17.95%	0.00%
36	Latvian	lv	0.001233766	2	2	35	5.71%	5.71%
37	Chinese	zh-HK	0.1472047	13	3	35	37.14%	8.57%
38	Romanian; Moldavian; Mold	ro	0.0390625	4	1	32	12.50%	3.13%
39	Chinese	zh-TW	0.081435185	6	2	24	25.00%	8.33%
40	Filipino	fil	0.129530423	5	1	18	27.78%	5.56%
41	Slovak	sk	0.015151515	0	0	18	0.00%	0.00%
42	Malay	msa	0.188502674	7	0	17	41.18%	0.00%
43	Lao	lo	0.087916667	4	1	15	26.67%	6.67%
44	Galician	gl	0.146163559	3	1	13	23.08%	7.69%
45	Chinese	zh-Hant	0.205681818	5	0	12	41.67%	0.00%
46	Welsh	cy	-0.063636364	0	1	11	0.00%	9.09%
47	Icelandic	is	0	0	0	11	0.00%	0.00%
48	Hebrew	iw	0.086574074	2	1	9	22.22%	11.11%
49	Persian	fa	0.076041667	2	1	8	25.00%	12.50%
50	Nepali	ne	0.047916667	1	0	8	12.50%	0.00%
51	Malayalam	ml	-0.185714286	0	2	7	0.00%	28.57%
52	Lithuanian	lt	0	0	0	6	0.00%	0.00%
53	Serbian	sr	0.041666667	0	0	5	0.00%	0.00%
54	Ukrainian	uk	0.111111111	1	0	3	33.33%	0.00%
55	Bokmí«l, Norwegian; Norwe	nb	0.1	0	0	2	0.00%	0.00%
56	Panjabi; Punjabi	pa	0	0	0	2	0.00%	0.00%
57	Bulgarian	bg	0.2	0	0	1	0.00%	0.00%
58	Basque	eu	0	0	0	1	0.00%	0.00%
59	Maori	mi	0	0	0	1	0.00%	0.00%
60	Portuguese	pt-PT	-0.5	0	1	1	0.00%	100.00%
61	Sinhala; Sinhalese	si	0	0	0	1	0.00%	0.00%

Sentiment analysis of 61 non-English languages

2.c. Third visualisation: visualisation of unique-users in each language community

To better understand the distribution of non-English tweets in Melbourne, we visualise only unique users on the map, meaning that each dot will represent only one user in the color of her language. We create MapReduce functions in CouchDB to filter these users. For users who have multiple tweets, we calculate the average longitude and latitude as well as the average sentiment score of all tweets. The location of the user on the map would be at her average GeoJSON coordinates.

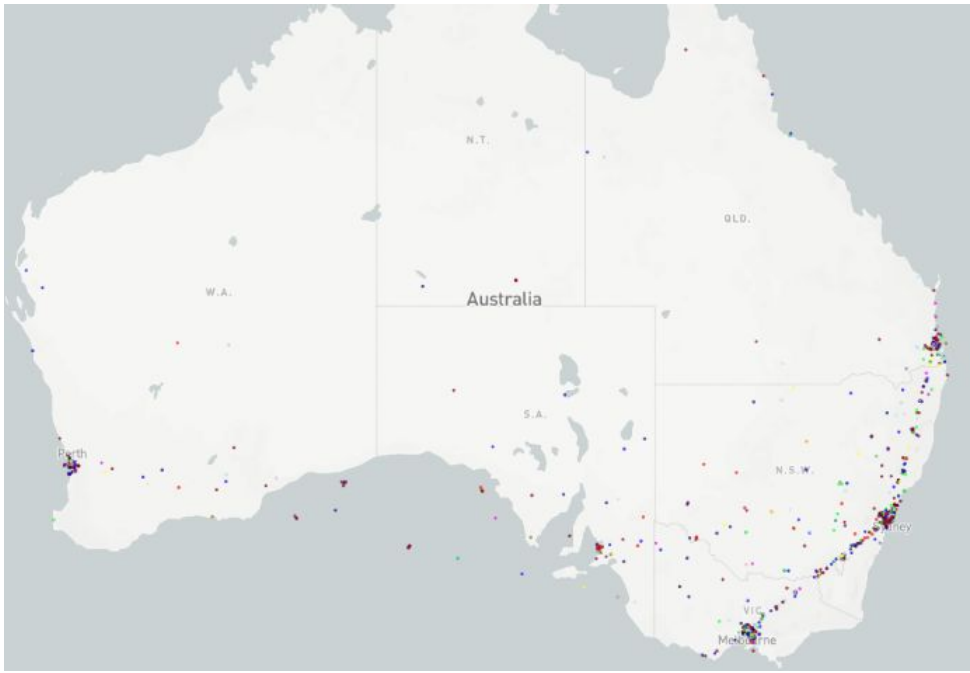


Unique users in every non english language

The new visualisation shows much fewer number of dots on the map and no clear clusters of similar colored dots come together. This means that there are many tweets of the same languages coming from the same users. Despite being a bit disappointed that there are no clear clusters, it is still intriguing to see that many non English users continue to use their own languages even when they work or live in Melbourne. *This again confirms the cultural diversity of the city.*

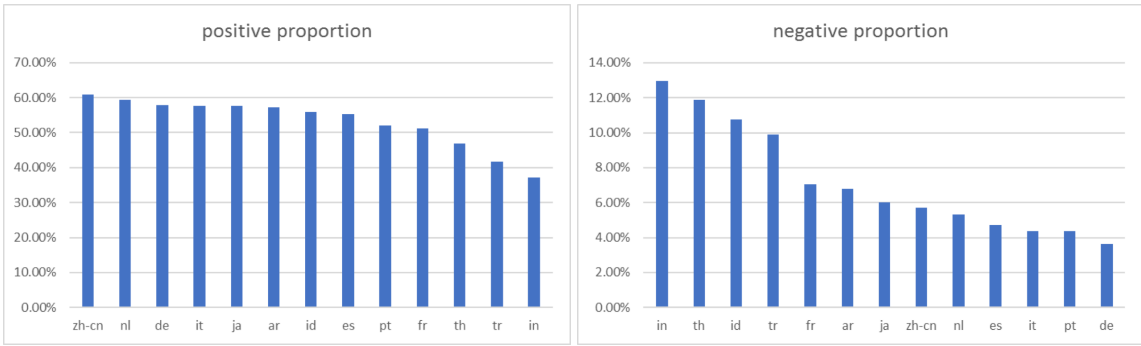
Besides, when we zoom out to see the entire map of Australia, we notice another interesting scene: *there are clear straight lines of dots connecting major cities of the country, especially between Melbourne, Sydney and Brisbane.* We can safely conclude that many non English

Twitter users in Australia move from city to city and a large number of them could be tourists who come to visit these capital cities.



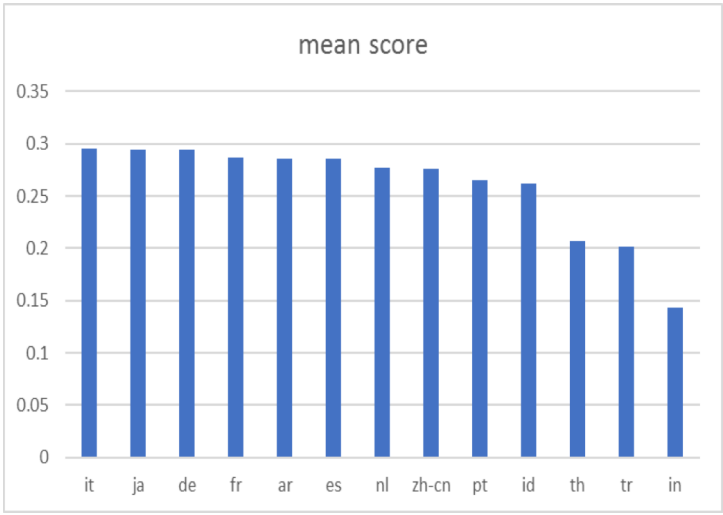
Unique users in every non english language in australia

Among unique users, it is interesting to find out that around 60% of Chinese tweet positively, which is the highest percentage followed by Dutch, German, Italian, Japanese, Arabic with slightly less than 60%. Meanwhile Indian seems to be a little bit more negative with only more than 35% unique users tweet positively while making the top number one for the negative proportion with 13% of its unique users.



Percentage of unique users with positive / negative average sentiment score

When we dig a bit deeper by calculating the average sentiment score of all unique users for every language, we find that on average Italian, Japanese, and German communities have the highest sentiment scores at around positive 0.3 implying that their tweets tend to be mostly positive. On the other hand, the Indian community tends to tweet much less positively with only 0.14 for the sentiment score.



Average sentiment scores of language communities

2.d. Melbourne Universities' Tweets Analysis

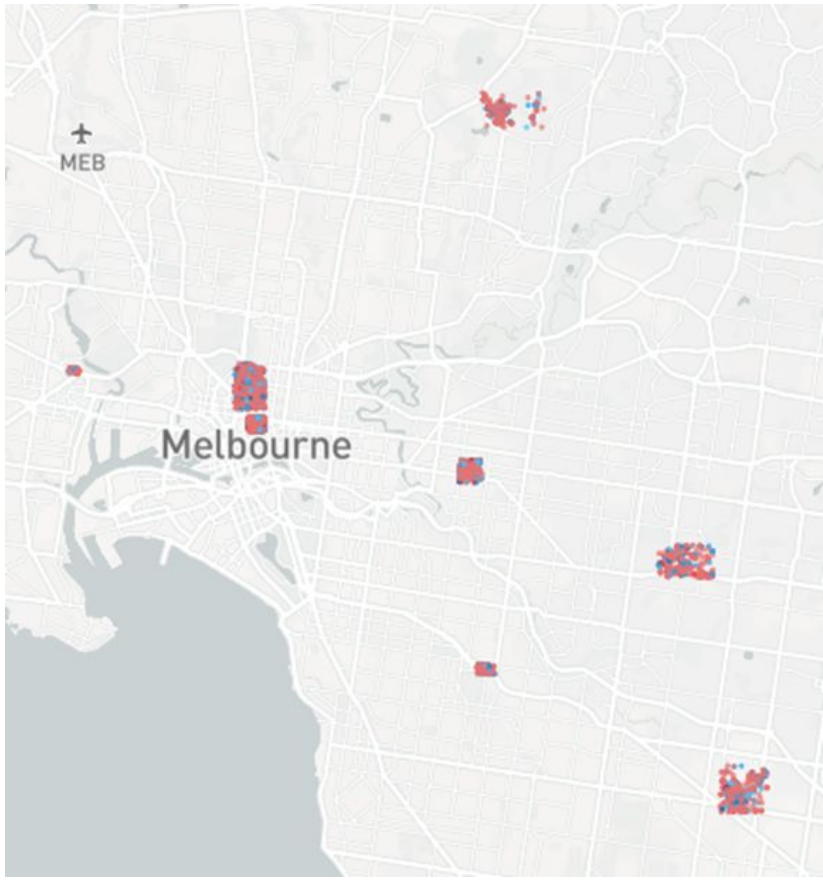
The motivation here is to see if tweets concentrated at Victorian universities show a difference in sentiment when compared against each other. In the collection of tweets we have gathered, we grouped a number of tweets that were within the boundaries of major Universities in Victoria and computed some simple statistics. The tweets are visualised below.

The results show that the University of Melbourne has the largest proportion of both positive, surprisingly, and negative tweets. Of the tweets located at The University of Melbourne, 22.12% were positive while 6.13% were negative. The remaining tweets were neutral. This suggests that The University of Melbourne has the smallest proportion of neutral tweets among the universities in Victoria. Possible explanation is that twitter at the location are able to express themselves in such a way that appeals to the sentiment scoring algorithm.

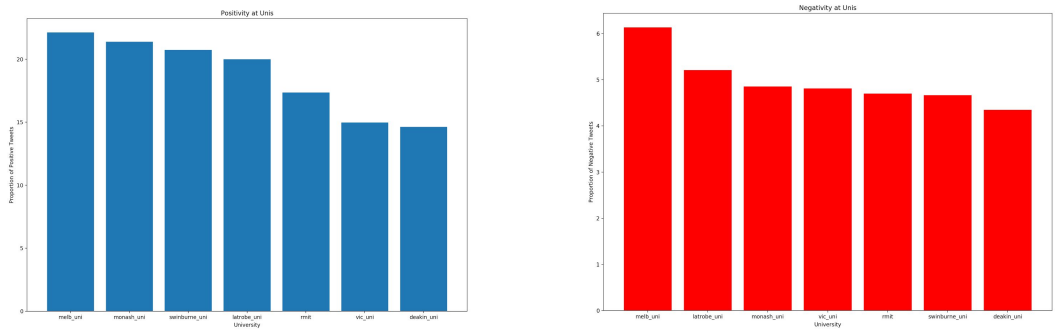
Another notable outcome is that tweets at Deakin University contain the least positive and least negative tweets. RMIT, closest to the University of Melbourne encapsulates the largest collection of tweets but is susceptible to noise, given the location of RMIT. RMIT has the third least positive and negative tweets.

University	Positive Tweets	Negative Tweets	Total Number of Tweets	Proportion of Positive Tweets (%)	Proportion of Negative Tweets (%)
Latrobe Uni	142	37	710	20	5.21
Victoria Uni	28	9	187	14.97	4.81
Deakin Uni	185	55	1265	14.62	4.35
Uni of Melbourne	707	196	3196	22.12	6.13
RMIT	709	192	4084	17.36	4.7
Monash Uni	326	74	1524	21.39	4.86
Swinburne Uni	271	61	1307	20.73	4.67

Sentiment analysis of non-English tweets from university campuses



Tweets with sentiment scores in universities in Melbourne



Comparisons of positive and negative tweets in universities in Melbourne

3. Conclusion

Life is colourful in Melbourne, the most liveable city in the world. By utilising the rich data of non-English tweets, we have been able to validate the idea that Melbourne is a true cultural hub that has become the destination for people from all over the world. As for all other places in the world, people in Melbourne express themselves in both negative and positive moments. Perhaps being in the most liveable city would not make one's life always happy as the title may slightly suggest.

In addition to understand the sentiment of non-local people, this research also provides a good glimpse on how we may utilise social media platforms to help the city be better off by tracking the moods of tourists, students or foreign workers in real-time. Foreigner-dependent businesses such as universities, retail shops or tourism could take advantage of such a tool to understand their customers who come from so many different cultural backgrounds.

Finally, the research project has helped us learn and enabled us to experiment in building a small scalable cloud system utilising NeCTAR, CouchDB as well as automation tools like Boto, Ansible, and Gulp. We have been able to gain hands-on experience in how such a system works while also grasping its advantages and disadvantages.

The combination of an interesting social idea and prominent computing technologies makes this research fascinating and truly valuable.

Appendix A - Deployment Guide

CouchDB Deployment

Deployment requires Python 3.5+ to be installed on your computer, and prerequisites to be installed – these can be found in *requirements.txt*. Before deployment, set the following two environmental variables:

export NECTAR_ACCESS_KEY = <nectar access key id>
export NECTAR_SECRET_KEY = <nectar secret access key>

Before deployment, some variables need to be configured. In *deployment/playbook/roles/cluster/vars/main.yml*, set the username and password of the *CouchDB* cluster's admin account. Similarly, insert a set of Twitter API keys into *deployment/playbook/roles/harvester/vars/main.yml* for each instance you will be running. Note that these all need to be different, as Twitter will not allow the same keys to be used from two different IP addresses.

To deploy, call

./deployment/deploy.py --nodes <num of nodes> --size <vol size> --type <inst type>

Note: the SSH private key can either be placed at *~/.ssh/ccc-project* or the path can be supplied in the *CCC_PRIVATE_KEY* environmental variable.

For example, the configuration used in this project was 3 `m1.medium` instances, each with a volume of size 65 gb. This was the call used to deploy this configuration:

./deployment/deploy.py --nodes 3 –size 65 --type m1.medium

The deployment script will take some time to run, as it is spinning up instances, installing packages, and compiling and configuring *CouchDB* from source. In our example above, the deployment took 20 minutes.

The final result will be instances set up in a *CouchDB* cluster. Each instance will have a volume attached at */mnt (/dev/vdb/*) that contains the *CouchDB* databases. This volume will persist if the instance is terminated, and can be reattached to a new instance if required.

The script will output the admin links for each node, and they can be accessed using the admin username and password set in *deployment/playbook/roles/cluster/vars/main.yml*.

Each instance will also have a tweet harvester running as a service. If all is successful, you should be able to access the *Fauxton* interface for the *CouchDB* cluster, and see tweets streaming in immediately.

The harvester service can be controlled with the following commands:

sudo sv stop harvester
sudo sv start harvester

Website Deployment

The website is compiled and deployed using *Gulp.js*, a task runner. With *NodeJS* installed, the prerequisites can be installed by calling the following command in the *websites/* folder:

npm install

Building the website can then be done with the following command:

gulp

Finally, it is easy to deploy with Gulp as well. With the relevant server details set in *gulpfile.js*, simply call:

gulp deploy --prod

Further details can be found in *README.md*

Appendix B - Links

Github repository:
https://github.com/dabarrell/ccc-project

Deployment demo:
https://youtu.be/BRe6DCLoXEo

Website demo:
https://youtu.be/YUtPjBfd4uk

Appendix C - Team Contributions

David Barrell	CouchDB, Architecture, Deployment, Front-end
Steve Dang	Analysis, Visualisation, CouchDB
Bobby Koteski	Harvesting, Analysis, Visualisation

Appendix D - Citations

1. Tv, A. (2011, April 29).Retrieved May 18, 2017, from http://www.abc.net.au/news/2016-08-18/melbourne-ranked-worlds-most-liveable-city-for-sixth-year/776164
2. Social Media Statistics Australia – January 2017. (n.d.). Retrieved May 18, 2017, from https://www.socialmedianews.com.au/social-media-statistics-australia-january-2017/
3. Tweets — Twitter Developers. (n.d.). Retrieved May 18, 2017, from https://dev.twitter.com/overview/api/tweets
4. (n.d.). Retrieved May 18, 2017, from http://planspace.org/20150607-textblob_sentiment/
5. Mapbox API Documentation. (n.d.). Retrieved May 18, 2017, from https://www.mapbox.com/api-documentation/