Cluster and Cloud Computing Assignment 2

Australian Social Media Analytics

CCC2018-35

Melbourne

LingTao Jiang 867583

Yan Jiang 816920

YiJiang Liu 848008

Zhenxiang Wang 879694

Zihua Liu 857673

Table of Content

1. Introduction

With the continuous development of information technology, in recent years, massive amounts of data have become the most valuable wealth. The present society is developing at a high speed. Because of the advanced technology and the circulation of information, people are communicating more and more frequently. Big data is the product of this high-tech product. The number of twitter users in 2017 is already greater than 330 million (statista, 2018). Through the analysis of a large number of tweets, researchers can find many unobvious conclusions. For instance, we can find potential business opportunities and forecast some trends through data mining and analysis. Therefore, the mining and analysis of big data has become an indispensable part of modern science. However, it is not enough to build a big data analysis platform to run on a local computer. Therefore, big data analysis platform is often built on cloud platform.

In this project, the team was required to build a big data analysis platform on Nectar Research Cloud. At the beginning of the project, developers used automated operation and maintenance tools ansible to dynamically deploy instances on Nectar Research Cloud and setup development environment for each instance. Then the team was required to design distributed crawling system to get twitter data based on twitter API and store them in CouchDB. Then We did sentiment analysis and some additional situation analysis of tweets such as drunk analysis and hashtags statistics. Finally, developers did map reduce jobs and displayed the results on a built server. The server was able to display the results through different data visualization pages.

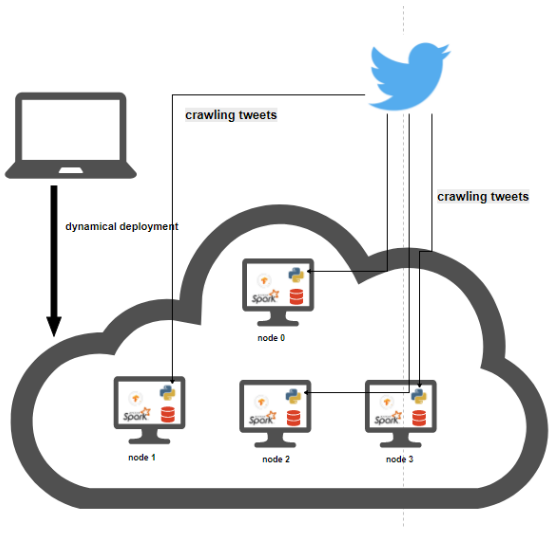


Figure 1- Schematic Diagram of the Project

The paper will introduce the system architecture and design firstly. Then the paper will demonstrate how to dynamically deploy nodes and build development environment on a cloud platform. Next, the paper will provide more details on designing distributed crawling system and sentiment analysis. Additionally, the paper will demonstrate the error handing and the method of removing duplicate tweets. Finally, the paper will combine processed results and data provided by AURIN to explain some actual situations. The source code can be found in https://github.com/lotharJiang/Cluster-Sentiment-Analysis.

Yijiang Liu was arranged to do the work of dynamical deployment. Yan Jiang was arranged to do the work of dynamical deployment. Lingtao Jiang was arranged to do the work of twitter harvest and sentiment analysis. Zhenxiang Wang was arranged to do the work of twitter harvest and sentiment analysis. Zihua Liu was arranged to do the work of data visualization and serer setup.

2. System Architecture and Design

This section will demonstrate the system architecture and design of the project. The system consists of CouchDB cluster, Spark cluster, twitter harvest, tweets analysis and data visualization website. The paper provides details on introducing each component of the system.

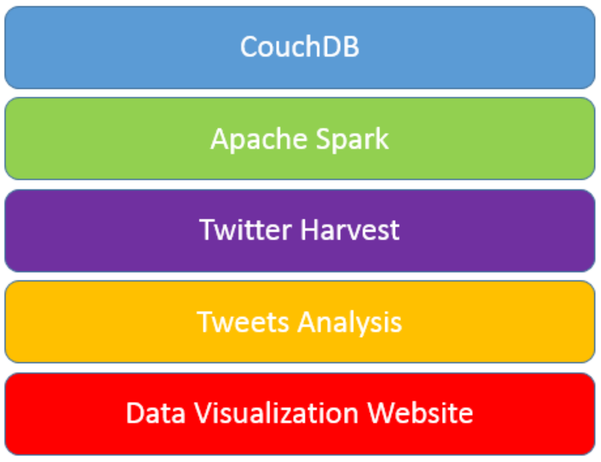


Figure 2- System Architecture

2.1. Instance on Cloud:

This project required the team to dynamically deploy nodes on Nectar Research Cloud. We launched four instances and allocated the same configuration for each instance. Each instance was assigned 2 VCPU, 8 GB of RAM and 60 GB of volume. The operating system installed on instances is Ubuntu 16.04.

2.2. CouchDB:

This project was required to use CouchDB as the database to store the crawled tweets and processed data. Although the relational database management systems have positive effects on ensuring consistency and availability, they have non-linear query execution time and static schema. Therefore, a CouchDB was selected as database for this project because the NoSQL database has better performance than relational database for big data question. In this project, the team built a CouchDB cluster to store and backup documents.

CouchDB is a NoSQL and document-oriented database that stores each piece of data as a JSON document in the database (Lennon, 2009). Unlike a highly structured relational database, the document-oriented database allows users to create different types of unstructured or arbitrarily formatted fields. In addition, CouchDB is able to provide serval powerful functionalities for users. The specific detail will be demonstrated as below:

2.2.1. Graphical User interface

CouchDB provides a native web-based graphical user interface named futon for users. Users are allowed to access the interface and do majority of functionality such as creating, updating, deleting and viewing documents. In addition, Futon is able to design the predefined view to do the map reduce job.

2.2.2. Access Restful API

The CouchDB exposes Restful API for users to access the database. It offers a RESTful HTTP API for reading, adding, editing, and deleting database documents. Python provides a package to support to do CRUD and design the predefined views with CouchDB.

2.2.3. Sharding and Replication

The CouchDB builds a shard for every subset of rows. The main advantage of sharded database is that it improves performance by computing load distribution across nodes. In addition, it makes to move data files easier, such as adding new nodes in clusters. Moreover, CouchDB provides replication functionality that is able to store the same row of document on different nodes in a CouchDB cluster in order to make the database be fault-tolerance. CouchDB will back up two copies of the data by default. Users can also set the number of backups according to their requirements.

2.2.4. Storage File Structure

CouchDB is a document-oriented database whose format of the document is a JSON string. The low-level structure of CouchDB is consisted of a storage and multiple view indexes (Anderson, 2010). Storage is used to store the document. View is used to query documents.

The low-level data structure of CouchDB are two B+ Tree. The first B+ Tree uses the id of the document as key. It's often used to find the location of a document through the id of the document. The second B+ Tree uses the latest reversion number as key. users are able to quickly locate the specified document with B+ Tree.

2.2.5. Map Reduce View

View is a useful function in CouchDB. Users are allowed to design views to filter documents in database and present documents in a specific order according to the specific requirement. In addition, users are able to make sorts of calculations on the data and show final results.

Previously, CouchDB provided two kinds of view for map reduce including temporary view and predefined view. However, the temporary map reduce view was deprecated in latest python package of CouchDB. Therefore, the team decide to design the predefined view of map reduce.

2.2.6. Multi-Version Concurrency Control

CouchDB adopts multi-Version concurrency control strategy to ensure the availability of database. Each node in cluster is able to accept requests and modify the document in a single database. However, the document is not replaced. The database only show a new revision number for modified document. CouchDB returns all update conflicts as a list and submit the list to the application to solve these conflicts. Therefore, CouchDB doesn't have a deadlock problem. In addition, users do not need to wait until the system has processed the last request.

2.3. Apache Spark:

In this project, the team also built spark big data analysis platform to process data. Apache Spark is a fast, general-purpose computing engine designed for large-scale data processing. Spark is a generic parallel framework similar to Hadoop MapReduce. However, Spark is able to store the internal output of Job in memory instead of reading data from HDFS. Therefore, the Spark is better for some iteration algorithms such as data mining and machine learning.

The core components of Spark are cluster manager and worker node. There are also the task control node Driver for each application and Executor for specific tasks on each machine node. The spark context is used to setup the deployment mode and configuration of a spark application.

In the project, we set the node 0 as the master node of spark cluster. The other three nodes are set as slaves of the spark cluster. RDD is an immutable distributed data set in spark. The operation to create, transform, and invoke RDD runs through the entire spark data processing. We put the processed data into RDD. Spark distributes data sets from RDD to the cluster and performs parallel operations.

Spark has two models of installation, one of which is the local installation model, which is simply installing Spark on a single computer. The other cluster model can be divided into three models according to the different cluster manager. The standalone model is the deployment pattern of the simple group cluster manager used by spark. Both the Yarn model and the mesos model adopt cluster managers provided by third parties. This project deployed the spark cluster in standalone model.

3. Automation Dynamically Deployment

3.1. Overview

The whole work is completed by using playbooks provided by Ansible. In order to deploy the remote server, the work is organized as a tree structure. A root playbook directs different groups of instances to play a series of roles which are actually a set of child playbooks. This means in each round, specific group of nodes will complete the task in the child playbook. The structure is shown as below:

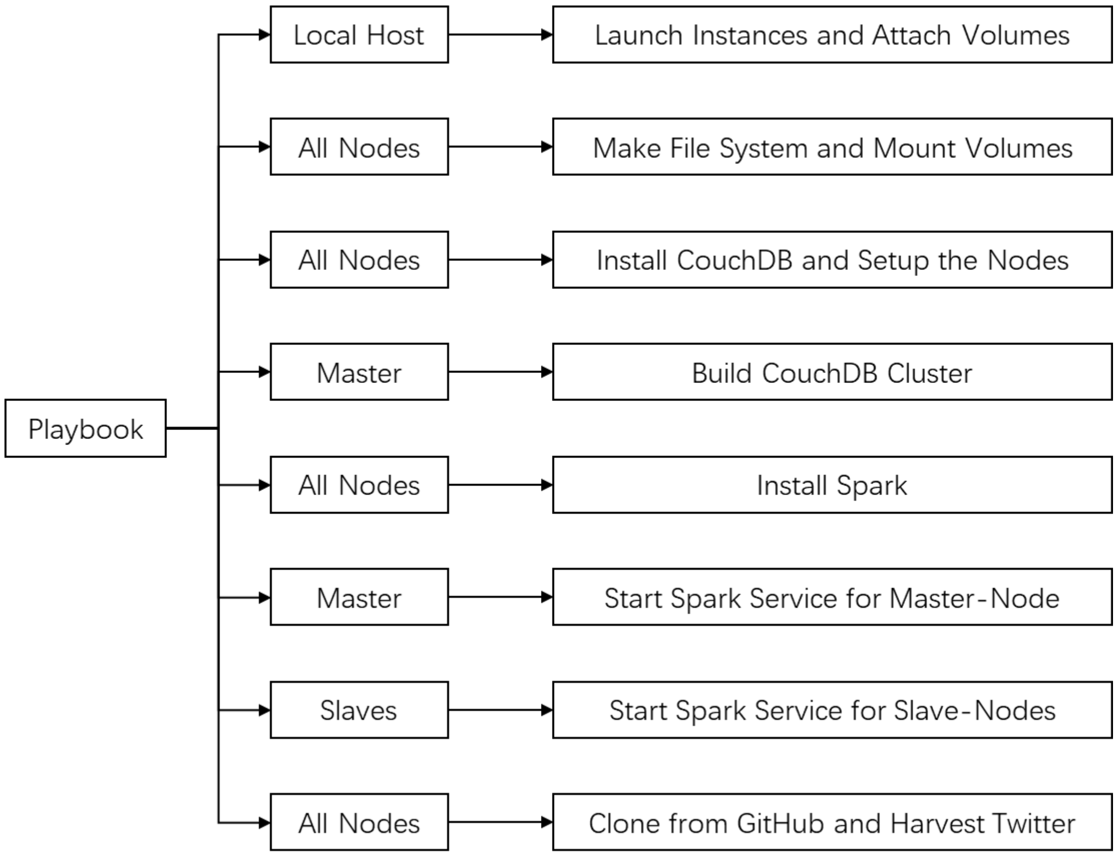

Playbook
Local Host
Launch Instances and Attach Volumes
All Nodes
Make File System and Mount Volumes
All Nodes
Install CouchDB and Setup the Nodes
Master
Build CouchDB Cluster
All Nodes
Install Spark
Master
Start Spark Service for Master-Node
Slaves
Start Spark Service for Slave-Nodes
All Nodes
Clone from GitHub and Harvest Twitter

Figure 3- Structure of the Automation Deployment Work

3.2. Launch Instance and Attach Volume

3.2.1. Comparison Between Boto and Ansible

For the first task, both Boto and Ansible can perfectly do the job.

Boto is an AWS development kit for Python which supports Openstack well. In comparison, Ansible is implemented in Python and can perform well in automates software provisioning, configuration management, and application deployment. The Openstack module of Ansible also has a reliable performance. Both of them can easily achieve our purpose. The reasons why we finally choose Ansible to do this work are as follow:

1) For Boto, after we have launched the instances, we have to keep checking the status to ensure we can successfully attach the volumes to the instances. But for Ansible, we do not need to consider this problem since in Ansible, it will not go to the next step without finishing the current one.

2) Ansible is a tool which is designed to manage the remote hosts. We need to use it to do all automation deployment job after we have the instances launched. However, the Ansible package for Python is not easy to use. So, In order to implement the one-key deployment and keep the consistency of the whole work, we decide only to use one tool to complete our mission.

3.2.2. Introduction of the Modules Used in this Task

os_server: This is an Openstack module which is used to launch servers in the cloud.

os_volume: This is an Openstack module which is used to create volumes in the cloud.

os_server_volume: This is an Openstack module which is used to attach volumes to servers.

add_host: This is a basic module which is used to add specific hosts to a certain group so that we can dynamically manage the hosts.

3.2.3. Task Description

It is local host who does this task. The first thing of all is to launch the first instance which is supposed to be the master node. Its IP address is dynamically added to a group called masterInstance for future use. Meanwhile, its flavor, operating system, security group, key pair and availability zone are selected by using the *os_server* module provided by Ansible. Then. the rest instances are launched the same way but whose IP addresses are registered to the group called slaveInstances. They are grouped differently in order to do the different operations. After all instances launched, volumes are created and attached to them.

3.3. Make File System and Mount Volumes

3.3.1. Introduction of the Modules Used in this Task

filesystem: The module is used here to make the file system.

file: The module is used here to create a directory as the mount point.

mount: This module is used here to mount the volume to the mount point.

3.3.2. Task Description

In this task, all commands are run on remote servers. Each instance we get from last task should use the *ext4* filesystem. Then, a new directory called *database* is created under */mnt* as the mount point and the corresponding volume is mounted here

3.4. Install CouchDB and setup the nodes

3.4.1. Introduction of the Modules Used in this Task

apt: Used here to update the system and build environment for CouchDB.

get_url: The module is used here to download the package of CouchDB 2.1.1.

unarchive: Used here to unpack the CouchDB package.

make: Used here to make release to install CouchDB

user: Used here to create an administer for CouchDB directories and files.

file: Used here to change ownership of the CouchDB directories and files.

replace: The module is used here to edit the config file of CouchDB, including the node name of each instance, the default number of replicas and the cookie the nodes used to communicate with each other.

lineinfile: The module is used here to edit the config file of CouchDB, including communication port, bind address and the username and password of the database administer.

copy: Used to upload the service file for CouchDB.

system: The module is used to enable the service file and start the CouchDB service.

command: This module is mostly the same as we directly type command. it is used here to execute the configure file, change the privilege of specific directories and files for security and use the *curl* command to setup the CouchDB nodes.

3.4.2. Task Description

All the hosts need to do this task. They need to build their own environment and install CouchDB independently. The commands need to be run by the remote user ubuntu with root privilege.

Since the operating system is newly installed, the first thing needs to do is to upgrade the system. we use *dist-upgrade* command to update the system and install several packages to handle the problem of dependencies for CouchDB.

Then, the CouchDB is installed from source. In order to manage the privilege of CouchDB, a user called couchdb is created to own the CouchDB directories. After that, all the nodes are made to bind addresses to 0.0.0.0 both for node-local mode and cluster mode. Their node names and cookie are also reset to prepare for building a cluster. As one of most important things, a service file is uploaded to each node and begin to work whose purpose is to make CouchDB starts when the server boots up and always restarts if it is shut down. CouchDB backs up two copies of the data in the database in a cluster. Finally, a command provided by CouchDB is run to setup the nodes, preparing for building a cluster.

3.5. Build CouchDB Cluster

3.5.1. Introduction of the Modules Used in this Task

command: The module is used here to run the *curl* command to setup cluster, add in slave nodes and finish building cluster.

3.5.2. Task Description

Only the master node needs to do this task. In this task, only *command* module is used since the only thing needs to do is run the *curl* commands provided by CouchDB. The reason why we do not use *uri* module provided by Ansible is that it does not support JSON String well.

3.6. Install Spark

3.6.1. Introduction of the Modules Used in this Task

apt: Used here to build environment for Spark.

get_url: The module is used here to download the package of Spark 2.2.0.

unarchive: Used here to unpack the Spark package.

file: Used here to change ownership of the Spark directories and files and change the privilege of specific directories and files for security.

command: This module is used here to rename the config files

lineinfile: The module is used here to edit the system *hosts* file, *spark-env.sh* file and *slaves* file to update the imformation of master node and worker nodes.

copy: Used to upload the service file for Spark.

3.6.2. Task Description

All the hosts need to complete this task to install their own Spark. Before everything, Java and Scala should be installed as the running environment for Spark. Then, Spark and Hadoop are downloaded from source and unpack into the volume attached to each instance. In this task, version 2.2.0 of spark and version 2.7.0 of Hadoop are selected. After unpacking, The IP and port of master node are appointed and all slaves' IP addresses are added to slave list in master instance. A service file for Spark is also uploaded to each instance for future use.

3.7. Start Spark Service

3.7.1. Introduction of the Modules Used in this Task

lineinfile: The module is used here to edit the *spark.service* file, indicating the start point and stop point with arguments.

system: The module is used to enable the service file and start the Spark service.

3.7.2. Task Description

For this task, the master node and the slave nodes should register their services respectively. For master node, it should use *start-master.sh* as the point to start the service while it should be start-slave.sh for slave nodes. Then, all of them should enable the service files and start working. In the service file, all the nodes are set to always restart the Spark service when the system detects any exception

3.8. Build Python Environment

3.8.1. Introduction of the Modules Used in this Task

apt: Used here to build environment for the application we have developed.

command: The module is used here to upgrade the setup tools of pip3.

pip: Used here to import the libraries we need for our applications.

git: Used here to clone the application we have developed from GitHub.

shell: This module is used here to apply a new process to execute our application in the background.

3.8.2. Task Description

All the hosts we have should do this task to harvest twitter distributedly. Firstly, Python 3.5 and python3-pip is installed to setup the environment and then the required python packages are imported by using pip3. After that, the applications developed are cloned from repository of GitHub and then being executed to harvest twitters.

4. User Guide

4.1. Specification

To use this application to do the Deployment Automation, you need to do the following in advance:
- You need to install Ansible on your PC properly with no dependency problem
- You need to have a key pair and use key-agent to add in the key pair:
  ```
  // Suppose you have a key pair id_rsa and id_rsa.pub at /root/.ssh/
  sudo ssh-agent bash
  ssh-add /root/.ssh/id_rsa
  ```

4.2. Functionality

- Launch instances in Nectar Cloud, create volumes and attach them to the instances
- Make file system and mount the volumes at /mnt/database/
  Install CouchDB and build a CouchDB cluster (Register as service so that it will start when system boots up and always restart when being shut down)
- Install Spark and build it as standalone mode (Master-slaves) (Register as service so that it will always restart when crashing down)
- Install Python 3.5 and clone our python code from GitHub and run the code
- Add new nodes to the existing cluster

4.3. Usage

- Unpack the package to a directory you like
- Change the privilege of the directory which is called cloud recursely to let you have the right to run the sh file, you can simply do like this:
  ```
  chmod -R 777 could/
  ```
- Simply use sh command to execute the play.sh file without doing any change:
  ```
  // Make sure you run this command inside the directory could/
  ```

```
sh play.sh
```

- Follow the instruction and do the things you want to do.

5. Tweets Harvest

5.1. Get data from Twitter API

In the implementation of this system, a python library called ‘Tweepy’ is adopted. It provides access to the entire twitter RESTful API and Streaming API.

5.2. Authentication

As Twitter require all requests for data must be authenticated, only those who is authenticated via OAuth service can harvest Twitter data via Twitter API. To get authentication, the developer has to create Twitter application in Twitter apps website and get consumer key, consumer secret, access token and access token secret. In view of the rate limit for single application, several applications were created to improve the efficiency of harvesting and the access tokens of them are stored as JSON file. When the twitter harvesting program starts, this information is specified in the parameters of command line.

5.3. RESTful API

RESTful API is a common application program interface that uses HTTP requests to GET, PUT, POST and DELETE data from the API provider. The Twitter RESTful API accepts parameters like query, geocode, language. In this scenario, the geocode is made fully use of because the geographic information of the twitter is the main concern. The API returns tweets by users located within the particular geographic area, which is specified with given latitude, longitude and radius. However, the tweets returned are not always valuable. The preference of API is the location geotagged by the users, but it will fall back to the location information in users’ profile. Obviously, it is not direct relationship between users’ location and the position they post their tweets. Thus, the tweets that matter are only those provide coordinates position.

In utilization of Twitter RESTful API, pagination is exploited a lot to iterate through twitter messages, users’ timelines, etc. Performing pagination requires a page parameter with each requests, which leads a problem that a mass of code is needed to handle the pagination loop. Nevertheless, as Tweepy is adopted, the Cursor class enables pagination to be implemented easier with less code.

Yet, the RESTful API has its own restriction, as the area accepted geocode represents a circle area using latitude, longitude and radius. In a distributed Twitter harvesting program, it is challenging and counterintuitive to make arrangement so that there is no overlapping or omission among the searching areas of all nodes. Therefore, Streaming API is also crucial to develop an efficient and accurate Twitter harvesting system.

5.4. Streaming API

Twitter Streaming API is provided to monitor tweets of particular users, tweets of particular topics or tweets in particular areas in real time. While RESTful API is used to pull data from twitter, the Streaming API pushes twitter message to a persistent session. It is supportive when obtaining a high volume of tweets or real-time feedback is needed.

The Tweepy library provides a stream listener class to receive messages from streaming session. In this system, a subclass inheriting from Tweepy StreamListener is created. The class override the on-data method, which enables listener to call functions to deal with received data. In addition, filter is available in Streaming API. In this case, the listener is set to listen the tweets posted in particular area. The parameters is four numbers(latitude and longitude) indicating the vertexes of a rectangle. To ensure efficiency and accuracy of the system, the monitoring areas of four nodes are designed carefully so there are no overlapping or omission.

5.5. Parallel Tweets Harvesting

In this system, the multiple instances on Nectar Cloud work run the tweets harvesting program simultaneously. As shown in the figure below, each instance maintains a RESTful API program and a Streaming API program harvesting tweets posted on different areas at the same time. Like aforementioned, the harvesting areas are elaborately designed since any omission is not endurable. Afterwards, twitter data will be handled in several processes. These processes, which will be discussed in following sections, includes sentiment analysis, drunk detection, hashtag extraction. Subsequently, the processed data and raw twitter will be stored separately in the Couchdb.

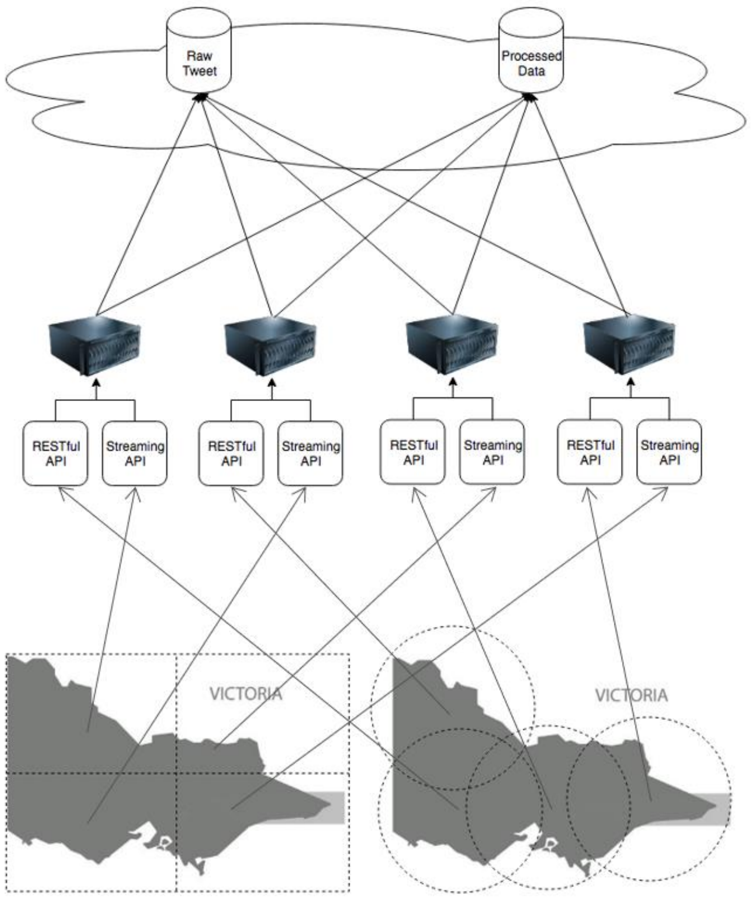


Figure 4- API Twitter Harvesting System

5.6. Get data from Other sources

To enhance the extensibility of the system, the ability of getting data from various sources are developed as well. In some scenarios, it is necessary to acquire data from other database using Curl command. To make this process more effortless, we implement a python program which request data using Curl command and process the returned JSON data, be followed by storing them into the database.

Also, the system allows data to be acquired from JSON file. The implementation references on the High Performance Computing Analysis from Jiang (2018). As the Twitter JSON data is commonly a large file, it is assumed that the most time-consuming task is to load the JSON object from file considering the cost of reading file. To solve this issue, a rational solution is Single Instruction, Multiple Data Stream (SIMD) architecture. To obtain results more rapidly, the tasks are divided and run parallelly on multiple processes with different data. The adopted architecture is illustrated below.

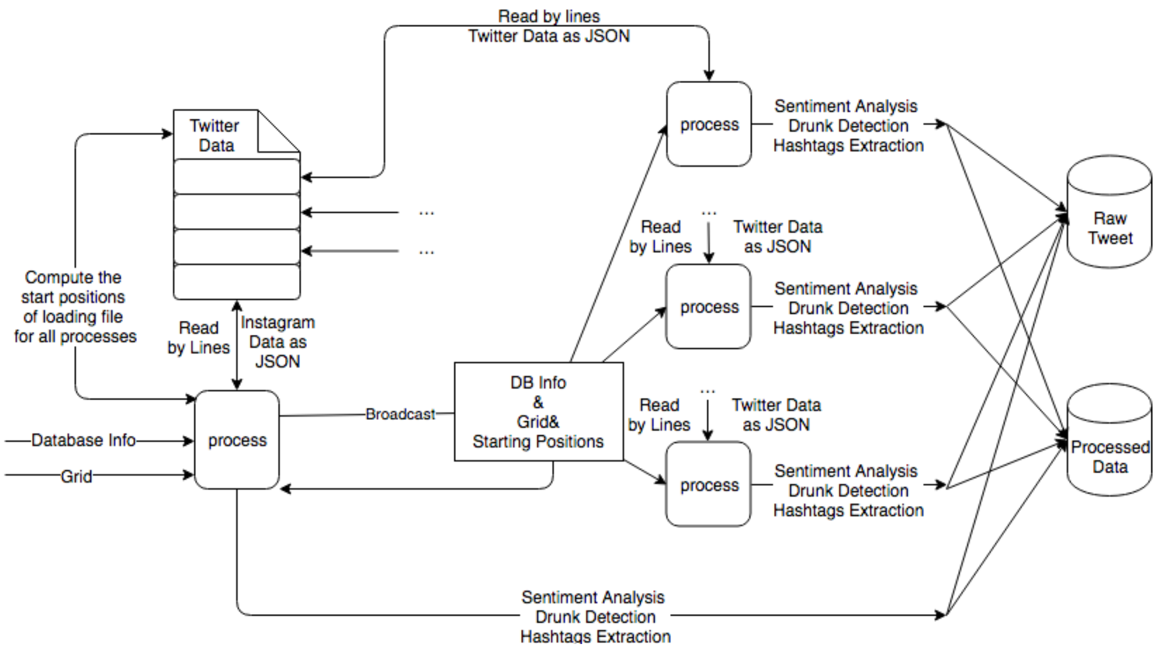


Figure 5- The Architecture of Twitter Harvesting System

In this architecture, the main process calculates the starting position of processes' reading. In this case, starting position refers to the number of bytes from the beginning to the position where the process should start reading the file. Afterward, it uses MPI broadcast function to broadcast this information to all processes, including itself. Before reading the file, all processes invoke file.seek() function to locate the start position. While reading the lines from the JSON file, all process simultaneously matchs the Twitter data to the blocks in the grid. Once the position of the file reading pointer goes over the start position of other processes, the processes stop reading.

In JSON file reading program, since the data will be received more frequently than RESTful and Streaming API, updating database every time it gets JSON will introduce significant delay, leading considerable reducing of performance. To deal with this challenge, batch updating is adopted as well.

5.7. Error Handling and Removal of Duplicate Tweets

Because geographic information is represented as circular in RESTful API, the overlapping of areas is inevitable, which means different nodes on the cluster will probably get the same tweet data. In order to solve the problem of data duplication problem in database, we exploit the ID of raw tweet as the ID in CouchDB. When data is updated to CouchDB, the tweets with existing ID will be handled as a new 'rev' version of the original data.

In order to ensure the stability of service, Twitter have officially restricted the rate speed of harvesting tweets for single Twitter application. To cope with this restriction, we have set up waiting time for API invoking in addition to apply multiple applications for different nodes in the cluster. When the streaming listener receives 420 errors types (rate speed limit error), the program will automatically wait until the limit is lifted.

Another possible error is that the file format may not be well organized when data source is JSON file. As the JSON file is processed line by line, when the program encounters a file line that cannot be turned into JSON, it only abandons the line having wrong format and continue to read subsequent lines. So the error is handled with the minimum volume data lost.

6. Twitter Analysis

For the tweets harvested, we first analyze whether it contains coordinate information and discard all the tweets without coordinate information. For the remaining tweets, we maintain two databases. One database is used to store the raw twitter data, which contains all the information of the tweets. Another database is used to store the data that analyzed and filtered, which contains only the suburb the tweet belongs to, tweet sentiment, whether it is sent under drunk and its hastags (if any). Retaining the raw data can provide us with possibilities to analysis more interesting scenarios in the future, while keeping the processed data can make map reduce more convenient and faster.

6.1. Sentiment Analysis

Tweets sentiment analysis is to determine whether the sentiment of a tweet is positive, negative, or neutral. The method used for tweets sentiment analysis is Sentiment Analysis with Long Short Term Memory Units (LSTMs). This method is from “Perform sentiment analysis with LSTMs, using TensorFlow" (Deshpande, A., 2017). We do this task following O'Reilly tutorial and use his teaching sample code (O'Reilly, 2018).

6.1.1. Design Choice

LSTMs is a deep-learning-based method. We choose deep learning for several reasons: In the past, sentiment analysis requires a lot of domain knowledge, such as linguistic and psychological knowledge, which may need several years’ study. However, in recent years, deep learning has greatly reduced the difficulty of sentiment analysis. Instead of requiring a lot of domain knowledge, deep learning methods use general and understandable mathematical and statistical methods to process text, and then use the processed data to train the model. It greatly reduces the threshold of entry into this field and often performs well.

6.1.2. Data Set

The data set we use is the Imdb movie review dataset. It has 25,000 labelled movie reviews, half of which have positive labels and half have negative labels. People tend to express emotions in movie reviews, so this dataset should be suitable for sentiment analysis. We divided this data set into 90% training data and 10% testing data.

6.1.3. Steps

The task then can be divided into 4 steps:

1)Build the word vector model and create id matrix for training data

2) Build RNN (with LSTMs)

3) Train the model

4) Compare performance with Textbolb and VADER Sentiment Analysis tools

6.1.4. Build the word vector model

The input for neural network can not be raw string, because some basic operations such as backpropagation or dot products cannot be performed on string. The input should be some scalar numbers or vectors or matrices of scalar numbers. In order to turn the raw text into the input for neural network, we need to create word embeddings, mapping words from the vocabulary to vectors of real numbers. "Word2Vec" model is useful is to this task. It creates word vectors by taking as its input a large corpus of text and produces a vector space. Word vectors are positioned in the vector space such that words that share similar contexts in the corpus are placed in close proximity to one another in the space (Mikolov, T., Chen, K., Corrado, G., & Dean, J., 2013). After process data though Word2Vec model, it will output an embedding matrix, which contains word vectors for every word in the training dataset.

For simplicity, we use GloVe pre-trained word vectors to generate a 400,000*50 dimensional embedding matrix (Pennington, J., Socher, R., & Manning, C., 2014). Each row of the matrix is a word vector.

For the training set, we remove punctuation, parentheses, question marks, etc., and leaves only alphanumeric characters for each sentence. Then we use Tensorflow's embedding lookup function to generate the vector representation for each sentence in the training set and create an id matrix containing these vectors as training input.

6.1.5. Build RNN (with LSTMs)

6.1.5.1. Recurrent Neural Networks (RNNs)

The temporal information of the text is important when it comes to natural language processing, because each word in a text is very dependent on its context. In order to extract and use context information, we use RNNs instead of traditional feedforward neural network.

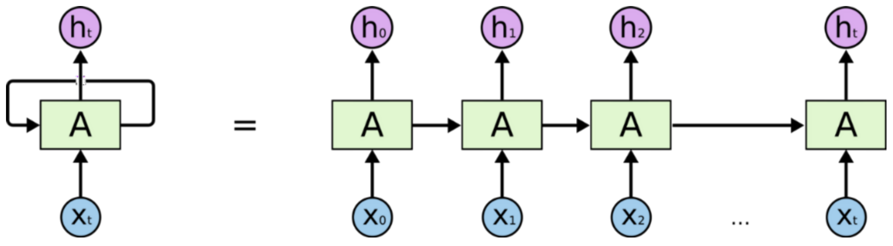


Figure 6-Sequential processing in RNN, from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Figure-7 is the sequential processing in RNN. x_t represents for input word. Each x_t is related to a time step t and each time step t is also corresponds to a hidden state h_t. The hidden state h_t contains the information from previous time steps. Hidden state h_t is calculated by the following equation:

$$h_t = \sigma(W^H h_{t-1} + W^X x_t)$$

In the above equation, σ is the activation function. W^H and W^X represents the weight matrices. For all time steps, W^H is the same, while W^X varies for each input, so that the hidden state is affected for both current input and previous hidden state. These matrices are updated through backpropagation as time goes. Finally, a binary softmax classifier is used for the final hidden state h_t and output values between 0 and 1, which represents the probabilities of positive and negative sentiment.

6.1.5.2. Long Short Term Memory Units (LSTMs)

There is a problem in the traditional RNNs: when the gap between the relevant information become very large, RNNs are unable to learn to connect the information. In other words, the traditional RNNs performs bad in the long-term dependencies. To solve this problem, we add a long short term memory units into the previous RNNs.

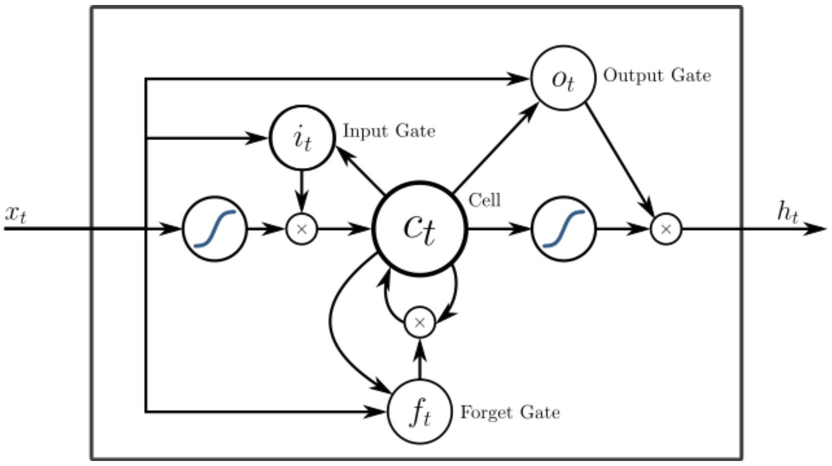


Figure 7- A peephole LSTM unit with input (i.e. i), output (i.e. o), and forget (i.e. f) gates, from https://www.wikiwand.com/en/Long_short-term_memory

As shown in Figure-8, instead of using a simple function discussed above to calculate hidden state vector h_t, LSTM units use a more complex function to calculate h_t. It introduces an input gate to decide how much should the model care about each input, a forget gate to throw away some information the model don't needed, and an output gate to get input from the intermediate state and output the final hidden state.

We firstly construct a LSTM cell with 64 units using tensorflow's nn.rnn_cell.BasicLSTMCell, then use a dropout wrapper to the LSTM cell to prevent overfitting. After that, we put both input data and the LSTM cell into the dynamic RNN then go through a dense layer to get the final output. The output contains 2 classes, positive or negative. We use standard cross entropy loss with a softmax layer for the final prediction then use Adam optimizer to update the neural network.

6.1.5.3. Train the model

We use Tensorboard to monitor the loss and accuracy. The following charts show the change of accuracy and loss over time. The model is run for 90,000 iterations and finally converged. However, there is possibility that the model overfits the training data.

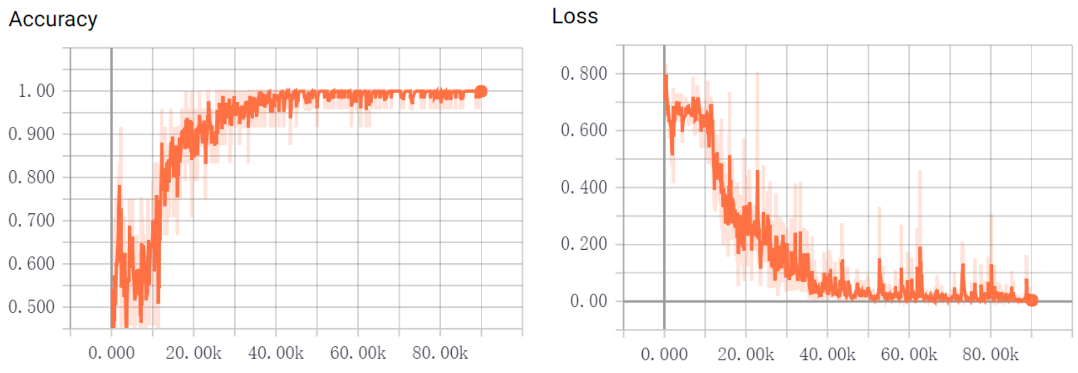


Figure 8- The Accuracy and Loss of LSTMs model in different iterations

6.1.5.4. Compare performance with Textbolb and VADER Sentiment Analysis tools

After training the model, we first test the performance of this model on the testing data. The accuracy is 87.5%. The result seems general acceptable. However, when it is applied to the twitter data, compared to other sentiment analysis APIs such as Textbolb and VADER Sentiment Analysis, our model performs not as well as expected.

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a sentiment analysis tool. It is a lexicon and rule-based tool. It is specifically suitable for sentiments expressed in social media (Gilbert, C. H. E., 2014). It has 4 outputs for each sentiment analysis, positive value, negative value, neutral value and compound value. Compound value is a float in the range [-1.0, 1.0], where -1.0 is negative and 1.0 is positive. It combines the first three values and can be used as a total sentiment index.

TextBlob is a Python library for processing textual data. The text processed by TextBlob has a sentiment property, which can be used to sentiment analysis. The sentiment property returns a (polarity, subjectivity) tuple. The polarity is a float within [-1.0, 1.0] where -1.0 is negative and 1.0 is positive. The subjectivity is a float within [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective (Loria, S., Keen, P., Honnibal, M., Yankovsky, R., Karesh, D., & Dempsey, E., 2014).

The following examples reflects performance of our LSTMs model over the other two methods.

For the first two examples, three methods all performs well as expected. However, the remaining three examples show the problems of the LSTMs model and TextBlob. Firstly, in the third example, the LSTMs model cannot tell neutral sentiment because itself is a binary classifier. In addition, as shown in the fourth and fifth examples, LSTMs model cannot tell the sentiment of the emoji in tweets. The reason is that when we process the training data, we only leave the tokens of words or numbers as training data. In this case, the model hadn't been trained with any emoji data, so it cannot classify correctly the tweets containing a large number of emojis but only a small number of words. The Textbolb sentiment analysis method has the same problem. As shown in the fourth and fifth examples, Textbolb incorrectly classify the tweets to neutral sentiment. These tweets are special, in which words has neutral sentiment but there are still some emoji reflecting strong sentiments. This shows that Textbolb cannot deal with emoji well.

ID	Tweet	LSTMs	Textblob	VADER
1	"Happy birthday! @Josh"	Positive	1.000	0.6114
2	"Beautiful Friday #smile"	Positive	0.575	0.5994
3	"It's just 10 days 'til #AllStarLanes #ShepherdsBush opens."	Negative	0.000	0.0000
4	"😊😊😊 #Saturday"	Negative	0.000	0.5614
5	"Photoshop? 🙄😒"	Negative	0.000	-0.5423

Figure 9- The example of sentiment analysis on tweets using LSTMs, Textblob and VADER sentiment analysis

In order to ensure the accuracy of the sentiment analysis part of this task, VADER Sentiment Analysis model is finally chosen. The reason why VADER can deal with emoji well is that the VADER model incorporate numerous lexical features common to sentiment expression such as a full list of Western-style emoticons, so that it can extract sentiment information from emoji.

6.1.5.5. Future work

In the future, there are some possible improvement in the LSTMs motel to make it more practical:

1. Change the binary classifier to classifier with 3 outputs, respectively positive, negative and neutral.

2. Use larger data set. Google created 3 million word vectors. Each word vector has a dimensionality of 300. This larger word vector can produce a more general model.

3. Select emojis as features of the network to make the classifier sensitive to emoji information.

6.2. Drunk Detection

Since analysis of Twitter has become a widespread approach for geo-spatial studies of human behavior, we have been inspired by many previous studies. In 2016, Nabil Hossain introduced a 3-SVM model to detect whether tweets are sent under the condition that the poster is drunk (Hossain, 2016). In addition, he provides a set of alcohol-related keywords, which is adopted in this system. In the data processing, tweets are filtered depending on if they included a mention of alcohol, defined by the inclusion of any one of several drinking-related keywords (e.g., “drunk”, “beer”, “party”) and their variants. After drunk detection, one value, 0 or 1, is stored for each tweet, where 0 represents this tweet has no relationship with drinking or alcohol and 1 represents that this tweet is sent under the condition that the poster is drunk.

6.3. Hashtags Extraction and Week sequence data

Hashtag of each tweet (if any) is collected and used to analyze the hot topics of each suburb. Week sequence data is also collected to analyze the sentiment change and drunk rate change in one week. These data can easily get from raw tweets.

7. MapReduce

7.1. Compare map reduce in spark and couchDB.

The map reduce job in this project was implemented by CouchDB and Spark. The paragraph will demonstrate and compare the differences between using couchDB and spark to do map reduce job.

7.1.1. Map reduce in couchDB

Couchdb traverses the by_seqnum B+ tree of the stored file and gets the document through seqnum (Lennon, 2009). Then CouchDB delivers each document back to the View Server for the map operation. The View server calls the map (doc) function to produce the intermediate key value pairs result. Finally, the result set of the map(doc) function is returned to CouchDB. View Server calls Reduce(key, value) function and returns the result to CouchDB. CouchDb will update the B + tree leaf node and point it to the value of reduce. After that, CouchDb traverses the parent of each B + tree and sends data from the corresponding reduce child node to the View Server. The View Server calls reduce(key, value) function to do reduce again. Finally, the results calculated by rereduce are returned to CouchDB. CouchDB will update the parent B + tree node and point it to the value of rereduce. CouchDB repeatedly perform rereduce until the result of rereduce of the root node is updated.

7.1.2. Data processing in Spark

Spark uses the concept of RDD to implement fast and efficient MapReduce operations. This means that it can store the memory state as an object across different jobs that can be Shared between the jobs. The following diagram shows the iterative operation on Spark RDD. In this scenario, distributed memory is used to store intermediate results, rather than using stable storage (usually disk), which speeds up the system.

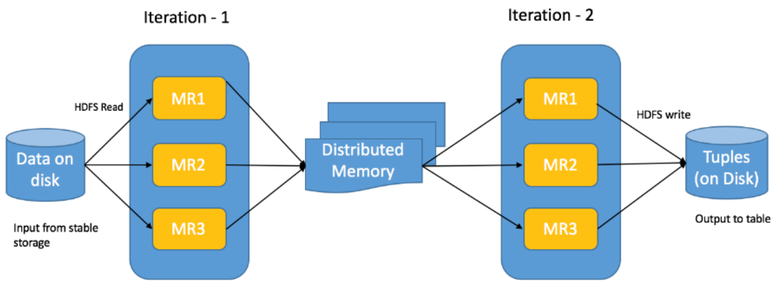


Figure 10- The Principle of Running Spark RDD in Iteration Job

However, we don't have to do the work of iterative operations in this project. Therefore, the consumption of IO operations on the hard disk is the same as doing map reduce. Moreover, couchDB is a document-oriented database and it does not provide an interface wrapped by JDBC. Therefore, if the developer wants to get the data from couchdb to Spark, Apache Bahir will be used to support to write the instructor which is similar to SQL to query the data. However, this interface doesn't work well for a lot of data to be accessed in CouchDB. Therefore, we processed the data and store it in a temporary txt file. Then, we submitted this temporary file and job to the spark shell. It will assign job to workers according to cluster manager.

Overall, using spark and couchdb to do map reduce job doesn't have much of a performance difference in this project. However, the environment configuration process for spark is a lot more complicated than couchdb. In addition, it is more complicated to query the data in spark.

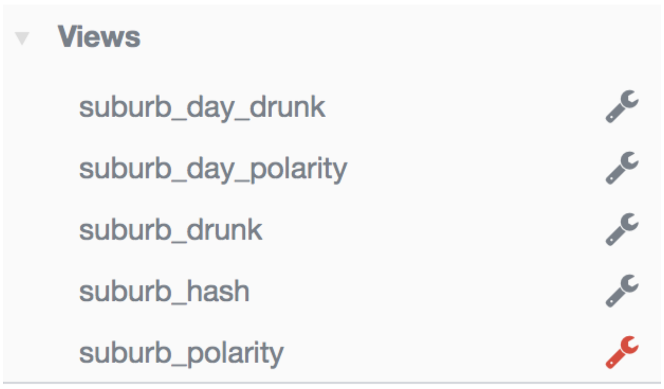
After tweets being processed, the information we store in the database only includes tweet ID, suburb, sentiment polarity and drunk classification we get from previous process, hashtags, day of week. However, these massive and poorly organized information cannot be directly used for data visualization. In CouchDB, we need to predefine several views to enable the front-end server to access data for various scenarios. The information of these views is listed below.

Key	Value	Reduce
Suburb	Sentiment polarity	Average value of the sentiment polarity in the suburb
Suburb	Drunk classification	Average value of drunk classification in the suburb
Suburb, Day of Week	Sentiment polarity	Average value of the sentiment polarity in the suburb in this particular day of week.
Suburb, Day of Week	Drunk classification	Average value of the drunk classification in the suburb in this particular day of week.
Suburb, Single Hashtag	1	The overall count of the particular hashtag posted on the suburb.

Figure 11- Predefined View on Processed Tweets CouchDB

8. Data visualization

In the data visualization part, we should design a web front-end to visualize data we got both from tweeter and AURIN. After processing, the tweeter data were map-reduced to a format mapping to each suburb. Views that used to process data are as follow:



Views

suburb_day_drunk

suburb_day_polarity

suburb_drunk

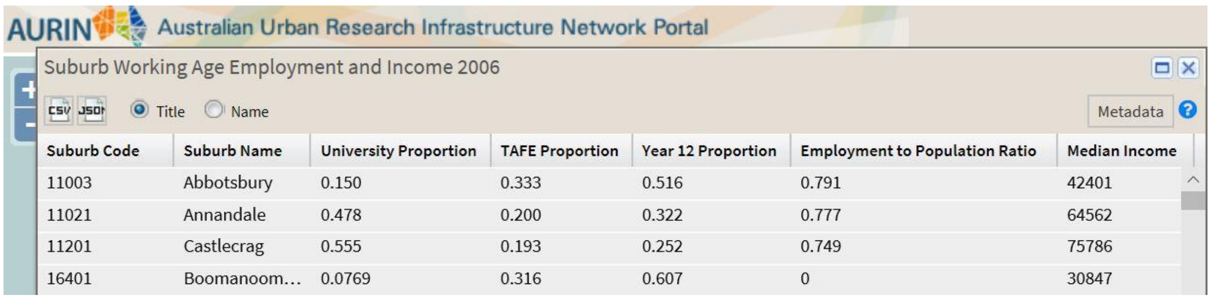
suburb_hash

suburb_polarity

Figure 12- CouchDB Views

- Suburb-polarity: Data are from -1 to 1 and bigger represents positive sentiment.
- Suburb-day-polarity: These data represent positive and negative sentiment for each day in a week.
- Suburb-drunk: Data are from one to more and they represent the number of tweets involving with alcohol.
- Suburb-day-drunk: Similar to data 'Suburb-day-polarity', these data represent tweets involving with alcohol in a week.
- Suburb-hash: As we found, some tweets have hashtag, which represents the theme or topic of the tweet. This data is a statistic of hashtag for each suburb.

As required, we use AURIN data ("AURIN. Australian Urban Research Infrastructure Network", 2018) which have the statistics of suburb incoming and education level (beyond bachelor) to make comparation with our results of sentiment.



AURIN Australian Urban Research Infrastructure Network Portal

Suburb Working Age Employment and Income 2006

CSV JSON [✓ Title] Name Metadata

Suburb Code	Suburb Name	University Proportion	TAFE Proportion	Year 12 Proportion	Employment to Population Ratio	Median Income
11003	Abbotsbury	0.150	0.333	0.516	0.791	42401
11021	Annandale	0.478	0.200	0.322	0.777	64562
11201	Castlecrag	0.555	0.193	0.252	0.749	75786
16401	Boomanoom…	0.0769	0.316	0.607	0	30847

Figure 13- AURIN DaTa

To accomplish our expectations, we used google map API ("Google Maps API | Google Developers", 2018) and Echarts API ("ECharts", 2018).

8.1. Design Structure

To visualize the results, we need to design a visualization interface. In this project, we divide our system into three parts, which are responsive web, a back-end and a database.

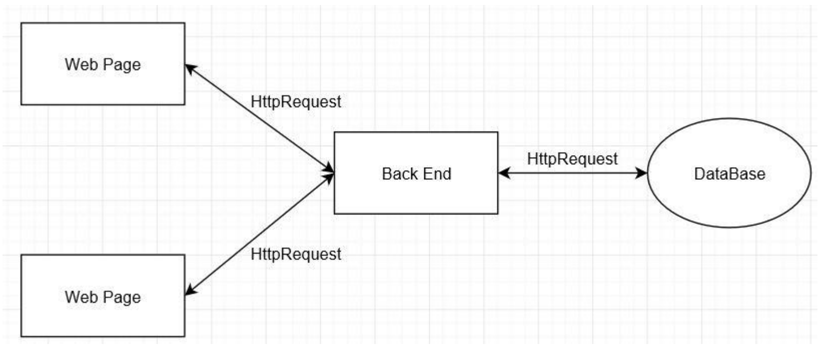


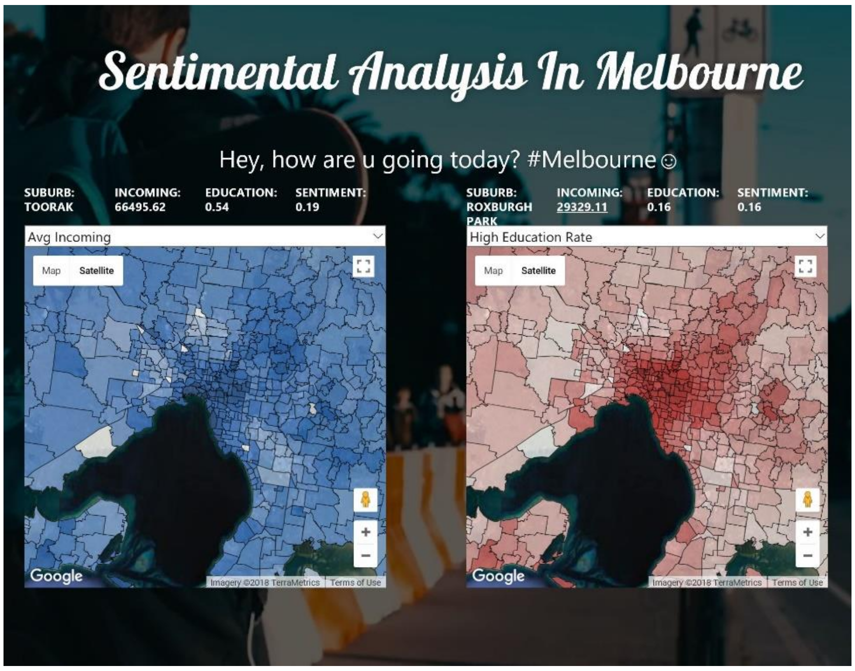
Figure 14- Visualization System Structure

As we can see from figure above. Responsive Web part will show the map and the correspondent input data, using Ajax to communicate with server to acquire data needed.

Back-End is designed in Flask Frame. It handles the request from font web and acquire and store required data from and to database we use.

We use CouchDB as our database, which is a document-based database. We use map-reduce to process our harvested data and restore them into new CouchDB.

8.2. Responsive Web Design



Sentimental Analysis In Melbourne
Hey, how are u going today? #Melbourne☺
Avg Incoming
Map
Satellite
Google
Imagery ©2018 TerraMetrics
Terms of Use
High Education Rate
Map
Satellite
Google
Imagery ©2018 TerraMetrics
Terms of Use

SUBURB: TOORAK INCOMING: 66495.62 EDUCATION: 0.54 SENTIMENT: 0.19

SUBURB: ROXBURGH PARK INCOMING: 29329.11 EDUCATION: 0.16 SENTIMENT: 0.16

Figure 15- Homepage of Data Visualization Website

In this project, we have two main pages to process visualization. The homepage is a map page, in which the boundaries of all Vic suburbs are drew as polygon.

As shown in the figure above, the whole Victoria are divided via suburb boundaries and the selected suburb id “Licola”, which has average incoming 12948 Dollar per year. We use colour gradient to represents the change of data. In this topic, the darker colours represent that the corresponding suburb has higher income than suburb with shallower colour.

In this page, we use Google Map API to load the 'GeoJson' format file, which contains all boundaries of the VIC, and we set the StateName as the ID to identify each state.

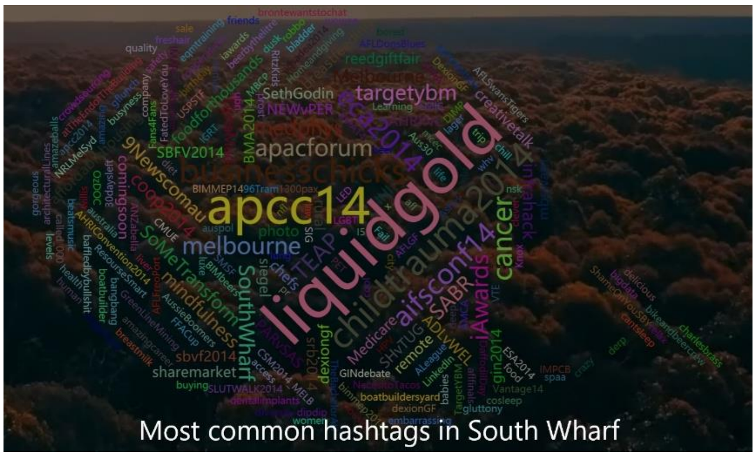


Figure 16-Second Visualization Page

The second Page is more details on each suburb, once we click a suburb, it will redirect the user to suburb details.

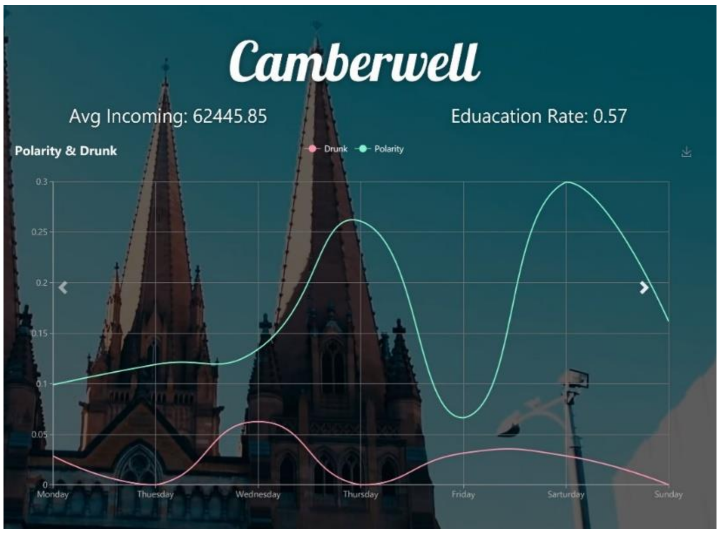


Figure 17-Second Visualization page

As shown in the Figure-19, there are two areas in which the first is a line table for the comparison between the Sentiment and Drunk, and the second the 'wordCloud' for this suburb. All data will be acquired from the CouchDB in real time via Ajax Http request.

8.3. Back-End Design

In this project, we use python framework 'flask' to develop our back-end server. Indeed, the efficient the processing rate of python framework is lower than J2EE framework, like Spring. Considering that we do not need to handle complex interactions in this project, light-weight is the best choice for us. Our restAPI is as follow:

- **GET /**: Empty path represents the home page of our visualization page, which is the map page.
- **GET /table/<name>**: This API would happen when user click each suburb in the map, it will redirect to the second page, which is the data analysis page.
- **POST /drunk**: This API would return the data involving with sentiment and with alcohol labelling with day, such as Monday, Friday. The returned format of data is {{"key": {"suburb","Mon"},"value"}, … }
- **GET /sentiment**: This API would return the data involving with sentiment, labelling with suburb. The returned format of data is {{"suburb","value"}, … }
- **POST /wordCloud:** This API would return the data used to create wordCloud, return format is {{"hashtag","value"}, … }.

To connect server with CouchDB, python is used to implement curl function with **requests** package, it will directly access the view we predefined and acquire required data.

8.4. Data Analysis with AURIN Data

8.4.1. Comparison between Incoming Rate, Education Rate and Sentiment Rate (beyond bachelor)

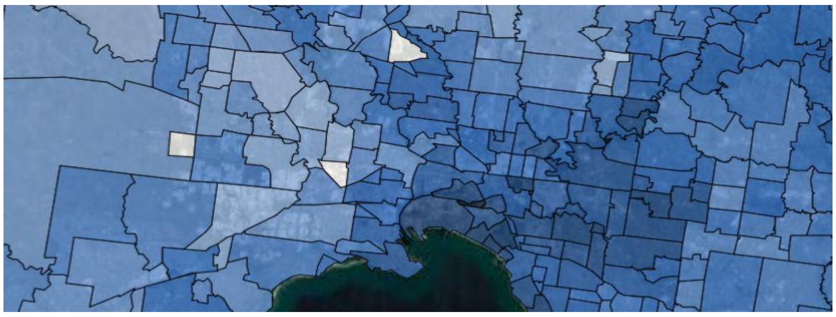


Figure 18 - Incoming Distribution of Victoria

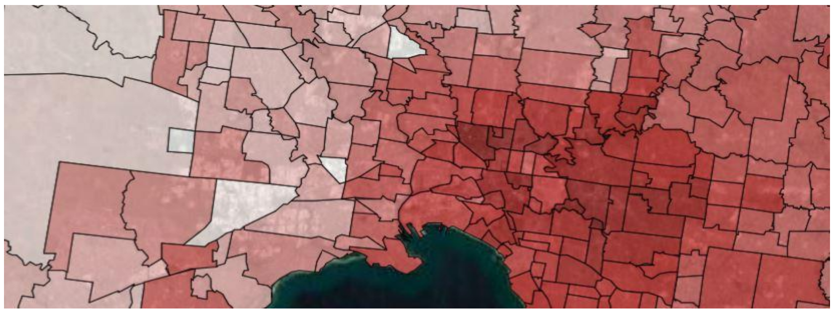


Figure 19- Education Rate Distribution of Victoria

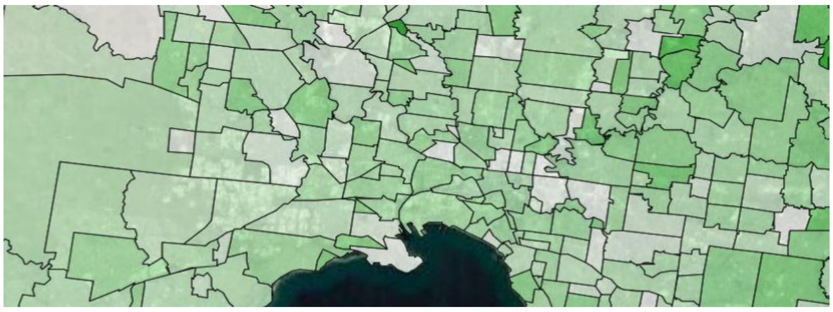


Figure 20- Sentiment Rate Distribution of Victoria

The blue figure represents the incoming distribution, the red one represents the education rate (beyond bachelor) distribution and the green one represents the sentiment rate of each suburb. The deeper the blue, the higher the income. The deeper the red, the higher the level of education. The deeper the green, the happier the people in this area are. The distribution of the income graphs and education figure is roughly similar, which represents that the income situation is directly proportional to the degree of education. Though there seems to be no obvious relationship between sentiment and income or sentiment and education level, the sentiment of citizens in eastern Melbourne is more positive than that of the western people. At the same time, the income and education level in the East is also higher than that in the West.

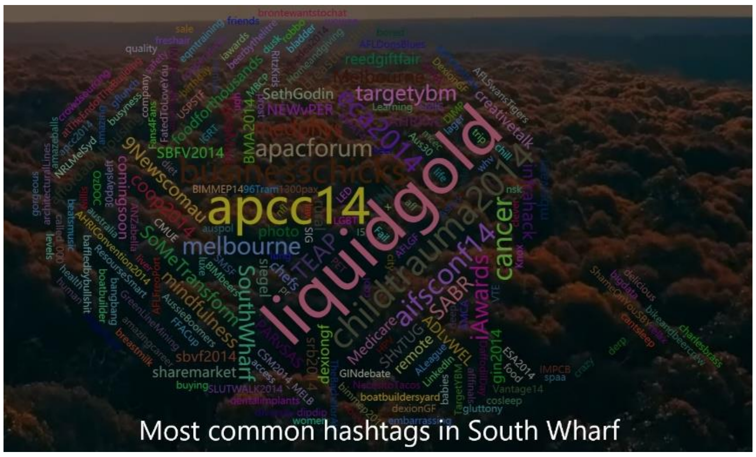


Figure 21- Word Cloud in South Wharf

Hashtags can also show some interesting relationships. As shown in word cloud in South Wharf, the more frequent words in South Wharf are ‘Liquidgold’ and some geographical position. South Wharf suburb has various clubs and bars, so it is reasonable that the most common hashtags relates to alcohol.

8.4.2. Comparison between alcohol and mood

There is no significant positive correlation between emotion and alcohol during the analysis of the data, but the two attributes were found to be related to the two factors of time and region, respectively.

8.4.3. The relationship between emotion, alcohol and time

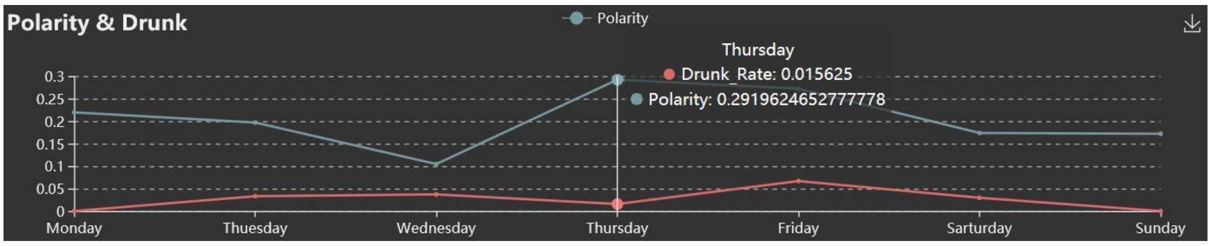


Figure 22-Polarity and drunk data of South Wharf

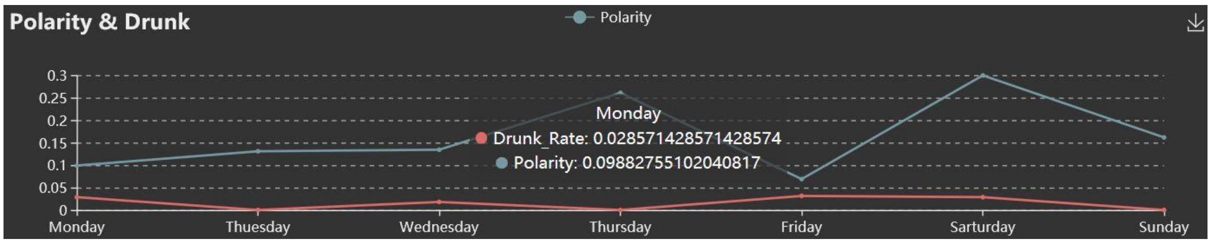


Figure 23- Polarity and drunk data for Camberwell

As shown in the above picture, after Thursday, the value of the sentiment increased significantly, behind the possible reasons for the majority of Companies in Australia paid on Thursday, and Friday afternoon off, making a week's tense mood relieved. During the relaxation process, there may be alcohol related activities, and Australian law stipulates that party can be opened at home on Friday and Saturday, so the alcohol index can be seen significantly at the end of the week.

8.4.4. The relationship between alcohol and geographical location

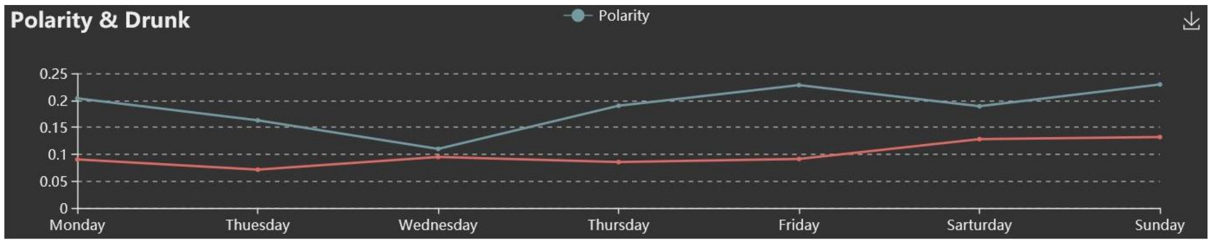


Figure 24- Polarity and drunk data for South Bank

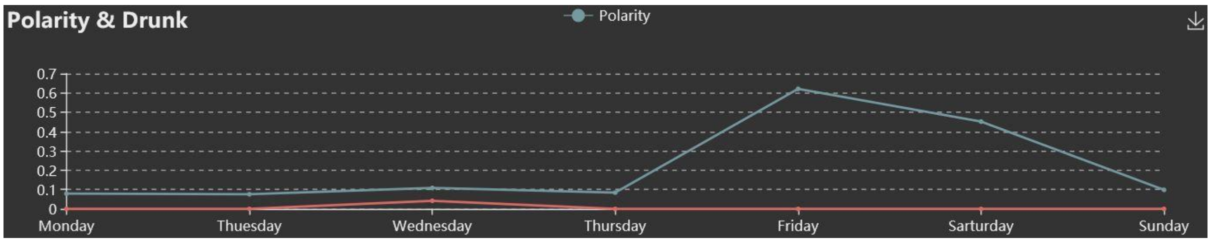


Figure 25-Polarity and drunk data for Box Hill South

The number of alcohol related tweets in a bar like Southbank and a lot of nightclubs is significantly higher than a living - based residential area like Box hill. Data visualization provides an intuitive data display for the analysis of our data, and allows us to acquire knowledge more conveniently and deliver the information expressed more directly. Many clues cannot be intuitively obtained. Only through the support of big data can we find out some invisible clues. Our analysis, if used later in a specific field of business, can play the role of trend prediction. For example, helping wine sellers establish the best time and place to sell wine. Data can provide foresight to market, so we should use the data to guide the market.

9. Pros and cons of Nectar cloud research:

Nectar Research cloud is an integral part of the National eResearch Collaboration Tools and resources project that is developed on the basis of University of Melbourne. Numerous researches use Nectar Research Cloud to do their research in different fields such as biological sciences including computer science, mathematical science and engineering. This section will list the advantages and disadvantages of using Nectar Research Cloud.

9.1. Pros of using Nectar Research Cloud:

1. Nectar Research Cloud provides virtual machines for users to deploy the system environment and run their program on instances. It has positive effects on saving the cost of building the physical machines.

2. Nectar Research Cloud could provide more than one instances at the same time. Researchers are able to build clusters to do large-scale and high performance computing on Nectar Research Cloud.

3. The Nectar Research Cloud provides large volume for users to store their data and software.

4. The Nectar Research Cloud provides images and snapshot for users to deploy required development environment quickly.

5. Nectar Research Cloud provides security group and keys to secure the service.

9.2. Cons of using Nectar Research Cloud:

1. There may be problems with batch operations on the instance on the nectar. For example, it might terminate one instance when a user terminates serval instances at the same time.

2. The time of dealing with the mission of instance and volume on Nectar Research Cloud is too long. In addition, users will be able to see a response to the task after a long time. After terminating an instance, the user might see the change on dashboard after a while.

3. The processing speed of server is not fast. It might show timeout error.

10. Conclusion

This paper has demonstrated the process of building a big data analysis platform on Nectar Research Cloud to analyse data from social network twitter, which is in order to get the undetectable connection in real life. The paper has introduced the architecture of the system and provide specific procedures for dynamically deploying nodes on cloud. In addition, the paper has illustrated the method of tweets harvest and sentiment analysis. Finally, a front-end web application is built to visualize data and some interesting scenarios are identified from analysed data.

Automation deployment is capable to deal with large scale cluster deployment without being manually deployed, which decrease the repeated works. Also, with setting scalability and extendibility, it is unnecessary to re-deploy as the system can be adaptive to the growing data. Data harvesting and visualization can be used to find potential links in real world so as to know the market.

11. Reference

Anderson, J. C., Lehnardt, J., & Slater, N. (2010). *CouchDB: The Definitive Guide: Time to Relax*. " O'Reilly Media, Inc.".

Deshpande, A. (2017). Perform sentiment analysis with LSTMs, using TensorFlow.

Deshpande, A. (2018). *Perform sentiment analysis with LSTMs, using TensorFlow*. [online] O'Reilly Media. Available at: https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow.

ECharts. (2018). Retrieved from http://echarts.baidu.com/

Google Maps API | Google Developers. (2018). Retrieved from https://developers.google.com/maps/?hl=zh-cn

Gilbert, C. H. E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf*.

GitHub. (2018). *lotharJiang/High-Performance-Computing-Analysis*. [online] Available at: https://github.com/lotharJiang/High-Performance-Computing-Analysis [Accessed 9 May 2018].
AURIN. Australian Urban Research Infrastructure Network. (2018). Retrieved from https://aurin.org.au/

Hossain, N., Hu, T., Feizi, R., White, A. M., Luo, J., & Kautz, H. (2016). Inferring fine-grained details on user activities and home location from social media: Detecting drinking-while-tweeting patterns in communities. *arXiv preprint arXiv:1603.03181*.

Loria, S., Keen, P., Honnibal, M., Yankovsky, R., Karesh, D., & Dempsey, E. (2014). Textblob: simplified text processing. *Secondary TextBlob: Simplified Text Processing*.

Lennon, J. (2009). Introduction to CouchDB Views. In *Beginning CouchDB* (pp. 107-123). Apress.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Number of monthly active Twitter users worldwide (2018, May 9). Retrieved from

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... & Ghodsi, A. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM*, *59*(11), 56-65.