CZ4034 Information Retrieval

Project Report

Matric No.	Name
U1320639H	Cao Gaoxu
U1420681G	Chen Ziao
U1321698B	Liu Peng
U1420721K	Pan Jiangdong

School of Computer Science and Engineering

AY 2016-2017 Semester 2

Table of Content

Introduction

News never fails to be the best information which helps people know what is happening currently around the world. With the increasing popularity of Twitter, people like to read the news from tweets posted by famous media or news sources. However, due to their different backgrounds, they may have different preferred areas such as political, business, social and technology, and they would not want to waste time in reading the news they are not interested in.

To address this issue, we developed an integrated and automated Information Retrieval system to capture the news from Twitter and subsequently classify them into 5 categories, which are *Business*, *Political*, *Social*, *Technology* and *Other.* The news is mainly from 5 most popular Twitter news accounts, i.e. *CNN*, *BBC*, *Straits Times*, *New York Times* and *Wall Street Journal*. By using this system, users will not only be able to select their preferred categories, but also to search news by keywords and sort news according to time, popularity as well as retweet count.

In this project, we firstly built up the website and crawl the data on Twitter accounts through API. After that, we use the *Solr* to do the indexing on the dataset. Then, to categorize the news, we use *Python sklearn* packages to perform the different classification methods. Finally, we implement some additional functions on website to fulfill users' requirements.

Crawling

Overview

For this project,we decided to do news information retrieval. Nowadays, many people like using Twitter to read short news from different channels all over the world such as *BBC, New York Times, Wall Street Journals* etc. Since most global newspaper companies have their own account on *Twitter* and Twitter provides a well-established developer API to get all the tweets, user profile and pictures, we decide to crawl tweets from some major news channels on Twitter and the news data will be used for indexing, querying and categorization in later stage.

Question 1

1.1 How you crawled the corpus and stored them

To retrieve raw user information from *Twitter*, we first obtain consumer token and secret key from twitter by registering our application on *Twitter* Developer Website. By passing the token and secret key into *OAuthHandler*, we can establish connection with *Twitter* API. Since we have decided to use *Django* framework for our application, module *tweepy* is used as the connection interface.

To get tweets information from different news account, we need to pass the stream ID of the news source account to Twitter API. The stream ID can be obtained from *Twitter* Developer website. We have decided to get tweets from 5 major news channels which are *CNN, BBC, Straits Times, New York Times* and *Wall Street Journal*. Table 1 shows the stream ID for each news source.

News Source	Stream ID
Straits Times	37874853
BBC World	742143
Wall Street Journal	3108351
CNN	759251
New York Times	807095

Table 1: Stream ID of Each News Source

The news data is retrieved in *JSON* format. An example for a document crawled is shown below:

```
{
  "contributors": null,
  "truncated": false,
  "text": "Keep up with President Trump's address to Congress and our reporters' live analysis
  "is_quote_status": false,
  "in_reply_to_status_id": null,
  "id": 836765910832676864,
  "favorite_count": 93,
  "source": "<a href=\"http://www.socialflow.com\" rel=\"nofollow\">SocialFlow</a>",
  "retweeted": false,
  "coordinates": null,
  "entities": {
    "symbols": [],
    "user_mentions": [],
    "hashtags": [],
    "urls": [
      {
        "url": "https://t.co/OfjNr20lkz",
        "indices": [
          89,
          112
        ],
        "expanded_url": "http://nyti.ms/2mqOxgD",
        "display_url": "nyti.ms/2mqOxgD"
      }
    ]
  },
  "in_reply_to_screen_name": null,
  "in_reply_to_user_id": null,
  "retweet_count": 56,
  "id_str": "836765910832676864",
  "favorited": false,
  "user": {
    "follow_request_sent": false,
    "has_extended_profile": false,
    "profile_use_background_image": true,
    "default_profile_image": false,
    "id": 807095,
    "profile_background_image_url_https": "https://pbs.twimg.com/profile_background_images/736
    "verified": true,
    "translator_type": "none",
    "profile_text_color": "333333",
    "profile_image_url_https": "https://pbs.twimg.com/profile_images/758384037589348352/KB3RFw
    "profile_sidebar_fill_color": "EFEFEF",
```


Figure 1: Crawled Data in JSON Format

However, we did not store the *JSON* data in the format when it is retrieved. Instead, each document will be directly preprocess by our *Django* application because not all fields are needed for our project. After preprocessing, each document we only retain in total 8 fields. The details of the 8 fields are shown below.

Field Name	Description
screen_name	The news source name displayed on their twitter account
id_str	Unique id of a specific tweet
text	Content of tweet
favourite_count	Number of likes of the tweet
retweet_count	Number of retweets
created_at	Time the tweet was created
profile_image_url	The image url of the news source displayed on their twitter account
media_url_https	The image embedded in the tweets (Some tweets may not have it)

Table 2 Selected Field and Their Description

1.2 What kinds of information users might like to retrieve from your crawled corpus (i.e., applications), with example queries

Our project is focused on integration of news from various sources. Therefore, user could retrieve most recent news or news regarding a specific topic from our information database. For example, user might search for "Donald Trump" and get the insights on how each media channel reports about Donald Trump. Some types of information that users might like to retrieve are summarized as below:

1. Most current news from a specific news channel
2. News regarding a certain topic (e.g. Donald Trump, Obama)
3. News of different categories (e.g. Political, Business and Sports)
4. Which or what kind of news is hottest currently (estimated by number of likes or retweets)
5. Compare news from different news channel.

Example Queries:
1. Find me political news from CNN
2. Find me social news from Straits Times
3. Find me Technology news from New York Times
4. Find me news about Donald Trump
5. Find me news about Lee Hsien Loong
6. Find me the most popular news

1.3 The numbers of records, words, and types (i.e., unique words) in the corpus

Overall, there are 12,000 records that are extracted as data base for this project. There are 192,941 words crawled for "*text*" of all posts. However, since our application provides functions for dynamic crawling, which means users can crawl any new tweets posted by news sources at any time, the total number of words will change if users choose to update. The total number of records would remain the same because if any new tweets are added, the old tweets would be removed from Solr correspondingly. Given that normally one tweet would have at least 10 words, the total number of words would always remain above 120,000.

Indexing and Querying

Overview

In our project, *Solr* is used for indexing. Moreover, being an open source enterprise search platform, it has no doubt to perform very well in doing indexing. In the back-end of *Solr*, it provides on-the-fly stemming, tokenization (including transforming to lower case), and stop-words removal.
After getting the raw crawled data, preprocessing was performed before posting the data into *Solr*. The preprocessing includes:

- Renaming the field names to adhere to the schema model defined in *Solr*;
- Convert *Twitter* API data type into *Solr* data type. For example, we split

“*created_at*” which is the time the tweet was created into several fields such as year, month, day and time and then merge them back into date datatype of *Solr*;

- Exception handling on missing values. For example, we will pass a string “*no image*” as field of tweet_image if the tweet does not contain an image.

Field Name of Twitter API	Field Name of Solr Schema
screen_name	name
id_str	id
text	content & content_raw
favourite_count	like
retweet_count	retweet
created_at	time
profile_image_url	profile_image
media_url_https	tweet_image

Table 3 Field Name Comparison between Solr and Twitter API

After the data is preprocessed, the *Django* application would direct post them to *Solr* which is running at the back end. A python module *‘solrpy’* is used as the interface of *Solr* and *Django* application. An example of the posted data in *Solr* admin interface is shown as below.


```
{
  "content_raw":"Zouk climbs to No. 4 on DJ Mag top 100, Ce La Vi takes No. 80 spot https://t.co/p2n6Sdb86k https:,
  "like":0,
  "profile_image":"https://pbs.twimg.com/profile_images/630988935720648704/HkmsHBTM_normal.jpg",
  "content":"Zouk climbs to No. 4 on DJ Mag top 100, Ce La Vi takes No. 80 spot https://t.co/p2n6Sdb86k https://t.
  "tweet_image":"no image",
  "time":"2017-03-30T16:42:48Z",
  "retweet_count":[1],
  "id":"847489295585746944",
  "name":"STcom",
  "_version_":1563314086528155648},
```


Figure 2 Crawled Data Posted on Solr

Please note that we store the tweet content in two different fields: “*content”* and “*content_raw”*. This is because “*content”* is used for constructing index for tweet content retrieval while “*content_raw”* is used for constructing index for spell check so that the application would be able to give suggestion on misspelled word input by users.

Below is a screenshot of how we use Solr to index the content field which is of type “*text_en*”.

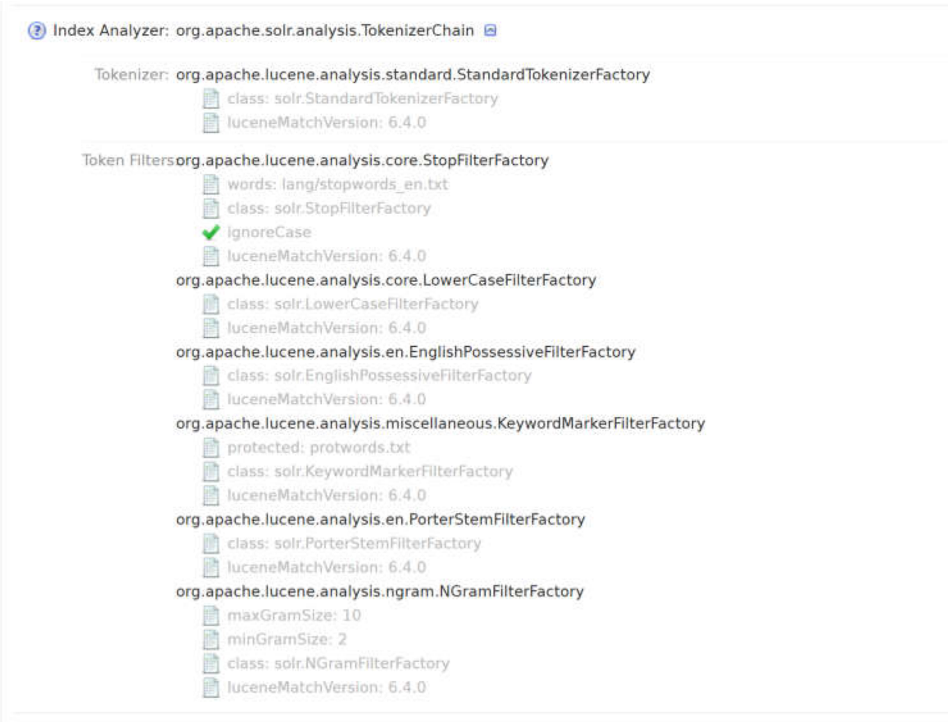


Figure 3 Screenshot of "content" Field Indexing Steps

For field *"content_raw"*, we give it a type *"text_special"* which is defined by us manually. Basically, *"text_special"* is the same as *"text_en"* except that *"text_special"* does not use Ngram Filter and Stem Filter because user do not want words suggestion in Ngram or stemmed form (e.g. User type "*Chine*", system should give suggestion "*Chinese*" instead of "*hines*" from Ngram or "*Chines*" from stemmed form). Below is a screenshot of how we use *Solr* to index *"text_special"*:

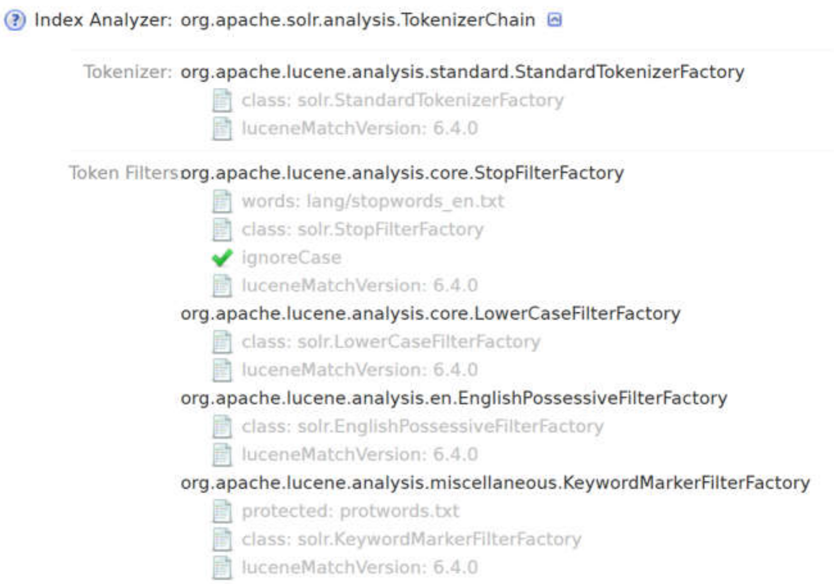


Figure 4 Screenshot of "content_raw" field index steps

Question 2

2.1 Build a simple web interface for the search engine

A simple web interface has been designed to cater to the searching of *Twitter* news posts. We use *HTML*, *Javascript*, *Jquery* and *Django* framework to build this web interface. Below is the design of the web interface.

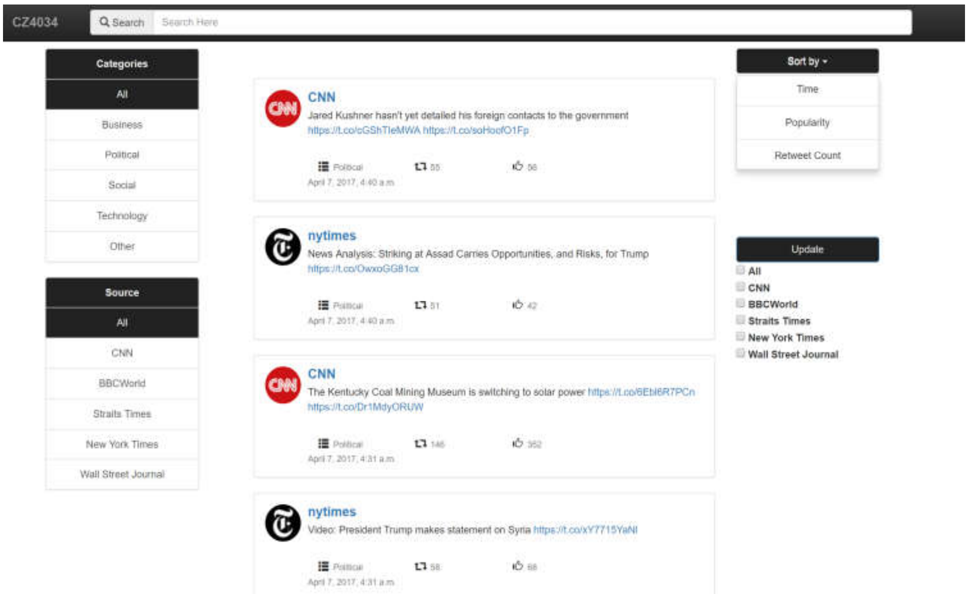


Figure 5: Web Interface

2.2 A simple UI for crawling and incremental indexing of new data would be a bonus

For the convenience of user, we decide not to build a separate UI for incremental crawling and indexing. Instead, we add this function on our current UI. By checking the tick box, user can choose to update tweets from one or more news channel. The update would automatically fetch most current tweets from selected news channels and index them on *Solr*.

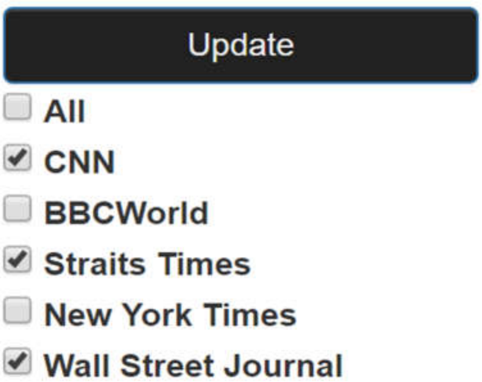


Figure 6: Update data from Twitter

2.3Write five queries, get their results, and measure the speed of the querying

Query	Number of results found	Top result	QTime(ms)	Time required(s)
Donald	225	Why US Tomahawk missiles are the weapon of choice in strikes on #Syria #DonaldTrump https://t.co/oQ5463VhXQ https://t.co/nSfAmQMxAC	2	1.3
Trump	1561	Trump: "Tonight I call on all civilised nations in seeking to end the slaughter and bloodshed in Syria"… https://t.co/BFSjGkr1tc	1	0.91
Trum	1567	Trump: "Tonight I call on all civilised nations in seeking to end the slaughter and bloodshed in Syria"… https://t.co/BFSjGkr1tc	0	0.73
Park Geun Hye	98	From South Korean president to prisoner 503, Park Geun Hye now in jail https://t.co/cSgGW3qc3P https://t.co/pBhtRi4uEK	1	1.06
Chinese President Xi Jinping	111	In Pictures: Chinese President Xi Jinping meets US President Donald Trump https://t.co/KqgTYTKf5N https://t.co/W8xRwFiiRG	6	0.96

Table 4: 5 Sample Query Result

Question 3

Explore some innovations for enhancing the indexing and ranking. Explain why they are important to solve specific problems, illustrated with examples

3.1 Tweet Sorting

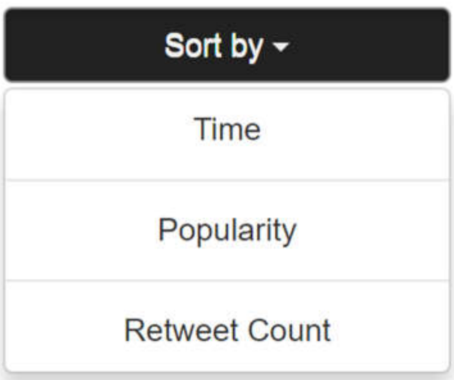


Figure 7: Drop Down Box for Tweet Sorting

Sort by time

This is the default setting when a user first visits this website without doing anything yet. Since news is the information that occurs recently, time is the priority. Example: user wants to know the latest news regarding Donald Trump. When user clicks "*Time*" on the drop-down box, the system will sort the tweets based on its created time.

Sort by popularity

How we measure the popularity is to use the number of likes as a criterion. We believe that a high number of likes means a more popular post so that we can use it to rank all the posts. Example: user wants to know the most popular news regarding Donald Trump. When user clicks "*popularity*" on the drop-down box, the system will sort the tweets based on number of likes.

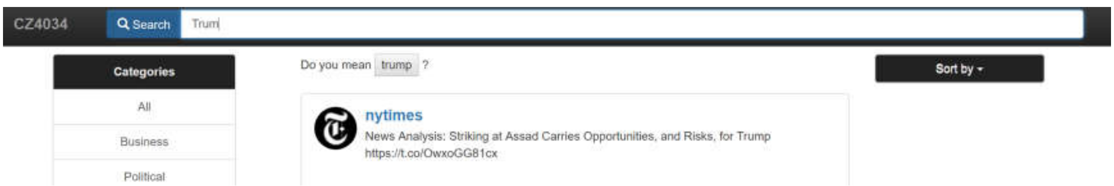
Sort by retweet count

When we use Twitter, we find that besides number of likes, there is a number for retweets, which is another way to measure popularity. Thus, in most of the cases, a post with many likes also has a big number of retweets. However sometimes not, the number of retweets shows how good that post is being propagated among people. In other words, a larger number of retweets means that the post is more relevant to the public and concerned by more people. Example: user wants to know the most influential news regarding Donald Trump.
When user clicks on "*Retweet Count*", the system will sort the tweets based on number of retweets.

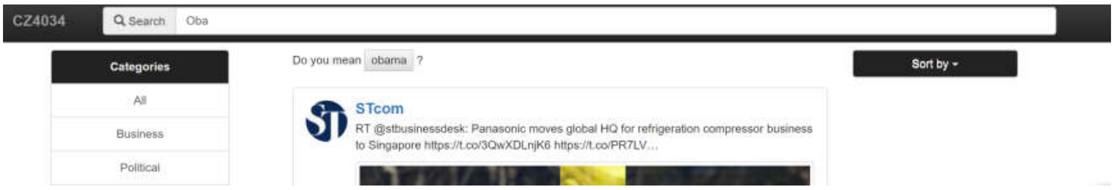
3.2 Search keyword suggestion and spell checking

The system use suggest function provided by *Solr*. To activate the function, we change the Solr solrconfig.xml file to realize it. Our system support both single word correction and multiple words correction. After users see the system's suggestion, they can choose to either use system's suggestion by clicking on it or stick to the original input. One thing to note is that since *Solr* provide suggestions based on index created and words are lowercased when building index, all suggestions all in lowercase form. Here are some examples of spell correction.

Example 1 - User wants "Trump" but input "Trum"



Example 2 - User wants "Obama" but input "Oba"



Example 3 - User wants "Chinese President Xi Jinping" but input "chine preside X Jin"

Classification

Overview

For classification, we decide to perform classification techniques on each tweet based on its content to classify it into a certain category. Due to there are a large variety of news on Twitter, we decide to only classify the retrieve tweets into 5 categories: *Business*, *Political*, *Social*, *Technology* and *Other*. Since *Twitter* does not provide tweets category and we should manually label the data, we decide to only label 1527 tweets with on 215 on *Business*, 655 on *Political*, 238 on *Social*, 199 on *Technology* and 220 on *Other.* This will be our training data set to construct the model.

We use Python *json*, *pandas*, *sklearn*, *numpy*, *scipy* and *nltk* packages to do classification. Since our system is written in *Django* Framework, we have also integrated our classification with the system to achieve on-the-fly classification. When system first starts, it will use the locally stored static training data file to train a model (classifier). After that, the model would be stored in the system, which means the model would not change any more. When users access the home page, they will be able to view all the tweets which are classified before. If they choose the update the tweets, our system will start to fetch the most current tweets from the sources specified by users. When tweets are fetched, they would be immediately classified using the trained model and the predicted category will be stored together with tweet itself into Solr. Below is an example of classified tweets on Solr and compare to *Figure 2*, it has an additional field *'category*'.

```
{
  "category":"Political",
  "content_raw":"RT @STsportsdesk: Matthias Wong beats Joshua Cheng 6-1, 7-5 to win the SG qualifying trial for the Longines
  "like":0,
  "profile_image":"https://pbs.twimg.com/profile_images/630988935720648704/HkmsHBTM_normal.jpg",
  "content":"RT @STsportsdesk: Matthias Wong beats Joshua Cheng 6-1, 7-5 to win the SG qualifying trial for the Longines Futu
  "tweet_image":"no image",
  "time":"2017-04-07T03:59:50Z",
  "retweet_count":[2],
  "id":"850196390433009665",
  "name":"STcom",
  "_version_":1563993358071758848},
```

Figure 8: Screenshot of Classified Tweet

Question 4

4.1 Motivate the choice of your classification approach in relation with the state of the art

The models we have chosen to do classification are: *Gaussian Naïve Bayes*, *Random Forest*, *Linear SVC*, *Logistic Regression*, *K-nearest Neighbors*, *Decision Tree* and *Neural Network*.

4.1.1 Gaussian Naive Bayes

Naive Bayes method is a set of supervised learning algorithms based on applying Bayes' theorem with the "Naive" assumption of independence between every pair of features. *Naive Bayes* require a small amount of training data to estimate the necessary parameters. *Naive Bayes* classifier can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one-dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality. On the flip side, the assumption for *Naive Bayes* is overly simplified. *Naive Bayes* is known to be a bad estimator; thus the probability outputs should not be taken too seriously.

4.1.2 Random Forest

A *Random Forest* is a meta estimator that fits several decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacements. *Random Forest* can run efficiently on large dataset. It also offers great accuracy in classifying and estimating. However, the classifications made by random forests are difficult for humans to interpret unlike decisions treas. If the data contain groups of correlated features of similar relevance for the output, smaller groups are favored over large groups.

4.1.3 Linear SVC

Linear SVC is an implementation of Support Vector Classification for the case of a linear kernel. Support Vector classification is the supervised learning method, which is effective in high dimensional spaces and highly versatile. *Linear SVC* implements "One vs the rest" multi-class strategy, thus training n_class models. However, as the name suggested, *Linear SVC* might not work well if the data set contains significant non-linear relationship.

4.1.4 Logistic and Cross validation

Logistic regression is a linear model for classification rather than regression. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function. In our case, we implement *Logistic Regression* with *cross-validation* to find out the optimal C parameters. This reduce the case of overfitting for our sample and modelling, thus improving the estimator performance. However, such methods might not work well in non-linear data as well.

4.1.5 K-Nearest-Neighbors

Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point. *KNN* is highly effective and efficient in most of real-word classification scenarios. However much of the work lie in computing the distance measures in the data and deciding the optimal number of the nearest neighbors.

4.1.6 Decision Tree

Decision Trees is a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. *Decision Tree* is simple to understand and interpret. Tress can be visualized. This also requires little data preparation. Other techniques often require data normalization; dummy variables need to be created and blank values to be removed. *Decision Tree* can handle both numerical and categorical data. The whole algorithm uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic. On the other side, *Decision-tree* learners can create over-complex trees that do not generalize the data well. Decision trees can be unstable because small variations in the data might result in a completely different tree being generated.

4.1.7 Neural Network

In the algorithms of *Neural Network*, we have chosen multi-layer perceptron classifier. Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function f(): R^m -> R^o by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. multi-layer perceptron classifier implements a *multi-layer perceptron (MLP)* algorithm that trains using Backpropagation. This method has the capability to learn non-linear models and learn models in real time. However, *MLP* with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore, different random weight initializations can lead to different validation accuracy.

4.2 Discuss whether you had to pre-process data and why

In our project, data pre-processing is done on both the static data used in model building and the dynamic data got from twitter API to do the classification. The procedure of data pre-processing for the two sets of data is mostly the same.

1. For the static data used in model building, since we only need to use textual data to predict the category, we take out the raw content to do the data pre-processing.
2. After getting all the content we need, we manually labelled 1527 tweets among the sources of *BBCWorld*, *CNN*, *New York Times*, *WSJ* and *Straits Times* based on the content of the tweets. The types of labels include *Business*, *Political*, *Social*, *Technology* and *Other*. After the labeling, we store the labelled content in json type, and pass it into our system for further processing.

Out[1]:

	category	content
0	Social	Zouk climbs to No. 4 on DJ Mag top 100, Ce La ...
1	Other	Football: FIFA reveals proposed slots for 48-t...
2	Political	Aung San Suu Kyi calls for support amid impati...
3	Social	Ten more bodies recovered after Bangladeshi fe...
4	Other	Pilot dies aboard American Airlines flight in ...
5	Other	Gulf airlines Etihad, Qatar work around US cab...
6	Other	RT @STsportsdesk: Football: Self-taught sculpt...
7	Political	China says "no such thing" as man-made islands...
8	Social	Rumours that 2 children were kidnapped at Juro...
9	Political	Minister of State Sam Tan attends Internationa...
10	Political	South Korean judge to deliberate into the nigh...
11	Political	#Britain downplays #security row as #Brexit wr...
12	Political	#Britain targets legal certainty with plan to ...
13	Business	NEA rubbishes #fake message claiming used tiss...

Figure 9: Training Data in Python Data Frame Format

3. Remove urls

   URLs are removed because the random combination of characters in a URL, e.g. https://t.con/P3IY0Gix3p, has no contribution in category prediction.

4. Remove non-alphabetic characters

   Numbers, punctuations, non-English words are removed because they are less meaningful compared to normal English words.

5. Convert to lowercase

   All the words are converted into lowercase to simplify the process and remove the redundancy caused by lower/upper case differences.

6. Remove stop words

   Stop words, such as ‘the’ and ‘is’, are removed as the reason that stop words are always have a huge term frequency but are less meaningful.

7. Stemming

   PorterStemmer imported from nltk.stem.porter is used to stem the words into their root form to improve the efficiency by reducing size of word list.

   Word list after pre-processing is shown below:

```
[u'abandon', u'aboard', u'abort', u'abound', u'abram', u'abroad', u'abrup
t', u'absolut', u'abus', u'ac', u'aca', u'academi', u'accept', u'access',
 u'accident', u'accomplish', u'accord', u'account', u'accur', u'accuraci',
 u'accus', u'acquir', u'acquit', u'acquitt', u'act', u'action', u'activ',
 u'activist', u'actual', u'ad', u'adam', u'add', u'addict', u'address', u'a
delaid', u'admin', u'administr', u'administratio', u'adoptaneld', u'ador',
 u'adrian', u'adrift', u'advanc', u'advic', u'advis', u'advoc', u'aerospa
c', u'affair', u'affect', u'afford', u'afghanistan', u'afraid', u'africa',
 u'afternoon', u'ag', u'age', u'agenc', u'agenda', u'agent', u'aggress',
 u'agit', u'ago', u'agre', u'agreement', u'ahca', u'ahead', u'ahm', u'ai',
 u'aid', u'ail', u'aim', u'air', u'airbrush', u'aircraft', u'airlin', u'air
port', u'airstrik', u'ajao', u'al', u'alabpilipina', u'albuquerqu', u'aler
t', u'alexey', u'alfa', u'alga', u'algal', u'algorithm', u'aliv', u'alleg',
u'allegedli', u'allig', u'allow', u'alon', u'alongsid', u'alreadi', u'ama',
u'amateur', u'amaz', u'amazon', u'ambiti', u'ambush', u'amer', u'america',
 u'american', u'americana', u'americorp', u'amid', u'amnesti', u'amp', u'am
pute', u'analog', u'analysi', u'analyst', u'anchor', u'andersoncoop', u'an
g', u'angel', u'angela', u'anger', u'ani', u'anim', u'announc', u'anoth',
 u'answer', u'anthem', u'anti', u'antoinett', u'anybodi', u'anymor', u'anyt
```

Figure 10: Word List from Training Data

8. Document-term matrix

   Next, we use Python *CountVectorizer()* function to convert all documents into a document term matrix of dimension (1527,3096). The document term matrix output by Python is shown below.

```
matrix([[0, 0, 0, ..., 0, 0, 1],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]]
```

Figure 11: Document Term Matrix

9. *tf-idf* Matrix

   After getting the document term matrix, we apply *tf-idf* conversion to convert it into a *tf-idf* matrix which would be final output of data preprocessing step. The *td-idf* matrix output by Python is shown below.

```
matrix([[ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.3629008528],
        [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ],
        [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ],
        ...,
        [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ],
        [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ],
        [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ]])
```

Figure 12: tf-idf Matrix

4.3 Build an evaluation dataset by manually labelling 10% of the collected data (at least 1,000 records) with an inter-annotator agreement of at least 80%

In total, we have manually labelled 1,527 records. Two members of our group has performed the labelling. Below is the labelling result.

	Business	Political	Social	Technology	Other	Total
Business	199	0	1	3	5	208
Political	1	644	7	1	6	659
Social	2	2	218	5	0	227
Technology	9	0	8	193	1	211
Other	8	1	5	6	202	222
Total	219	647	239	208	214	1527

Table 5: Labelling Result Comparison

Based on the definition of Cohen's Kappa Measure in the lecture slides.

$$Kappa = [P(A) - P(E)] / [1 - P(E)]$$

$$P(A) = (199 + 644 + 218 + 193 + 202) / 1527 = 0.9535$$

However, we find that the definition of P(E) on lecture slides is different from the definition on Wikipedia. Hence, we just show two P(E) measures.

$$P(E) = \frac{1}{(1527+1527)^2}[(208+219)^2 + (659+647)^2 + (227+239)^2 + (211+208)^2 + (222+214)^2]$$
$$= 0.2649 \ (\textit{Definition From Lecture Slides})$$

$$P(E) = \frac{1}{1527^2}(208*219 + 659*647 + 227*239 + 211*208 + 222*214)$$
$$= 0.2648 \ (\textit{Definition From Wikipedia})$$

It can be observed that two definitions of probability that two team members agree by chance is almost the same. Hence we decide to use 0.26485, the average of two P(E) above as our final P(E).

$$Kappa = \frac{0.9535 - 0.2648}{1 - 0.2648} \approx 0.9368$$

Since the Kappa Measure is above 0.8, it is considered as a good inter-annotator agreement.

4.4 Provide evaluation metrics such as precision, recall, and F-measure and discuss results

We split our labelled tweets data into training data set and testing data set. Training data is *70%* of total labelled records and it is used to train the classifier while testing data is *30%* of total labelled records and it is used to evaluate the model performance. Below is the model evaluation of different models we have used based on Question 4.1

Gaussian Naïve Bayes				
Class	**Precision**	**Recall**	**F_Measure**	**Accuracy**
Business	0.69	0.62	0.65	0.7187
Political	0.77	0.90	0.83	
Social	0.63	0.49	0.55	
Technology	0.86	0.76	0.43	
Other	0.47	0.40	0.43	
Weighted Average	0.71	0.72	0.71	

Table 6: Evaluation Results for Naïve Bayes

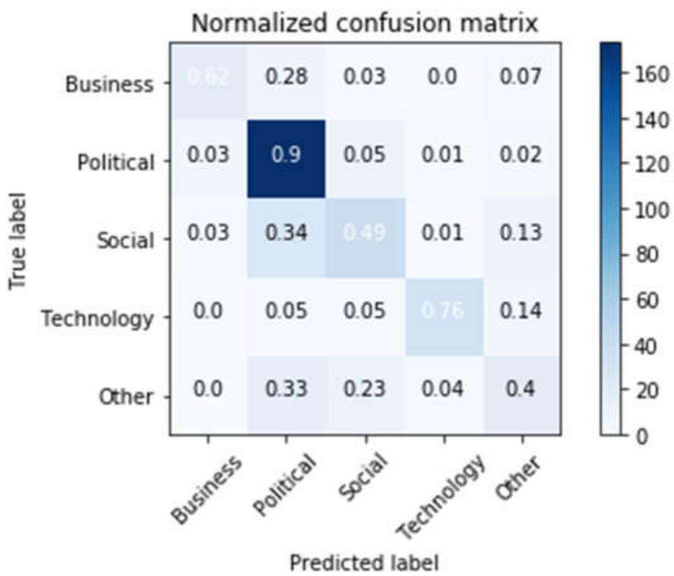


Figure 13: Confusion Matrix for Naïve Bayes

Random Forest				
Class	**Precision**	**Recall**	**F_Measure**	**Accuracy**
Business	0.89	0.55	0.68	0.7366
Political	0.78	0.90	0.84	
Social	0.76	0.43	0.55	
Technology	0.87	0.81	0.84	
Other	0.45	0.62	0.52	
Weighted Average	0.75	0.74	0.73	

Table 7: Evaluation Results for Random Forest

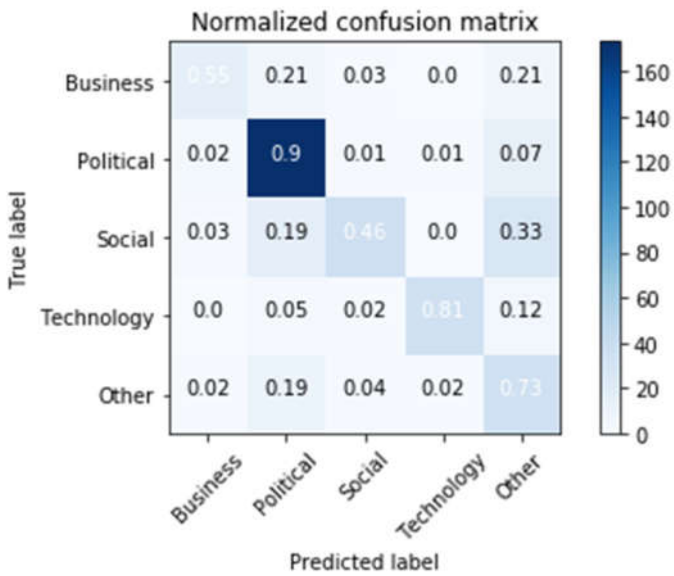


Figure 14: Confusion Matrix for Random Forest

Linear Support Vector Classification				
Class	**Precision**	**Recall**	**F_Measure**	**Accuracy**
Business	0.81	0.72	0.76	0.7852
Political	0.80	0.94	0.86	
Social	0.73	0.61	0.66	
Technology	0.85	0.83	0.84	
Other	0.71	0.46	0.56	
Weighted Average	0.78	0.79	0.77	

Table 8: Evaluation Results for Linear Support Vector Classification

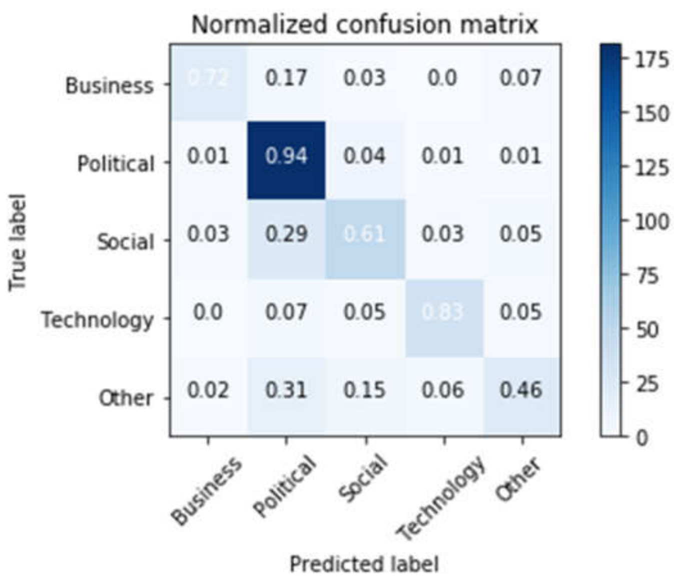


Figure 15: Confusion Matrix for Linear Support Vector Classification

Logistic Regression with Cross Validation				
Class	Precision	Recall	F_Measure	Accuracy
Business	0.86	0.62	0.72	0.7442
Political	0.72	0.95	0.83	
Social	0.90	0.46	0.61	
Technology	0.91	0.74	0.82	
Other	0.55	0.46	0.50	
Weighted Average	0.77	0.74	0.73	

Table 9: Evaluation Results for Logistic Regression with Cross Validation

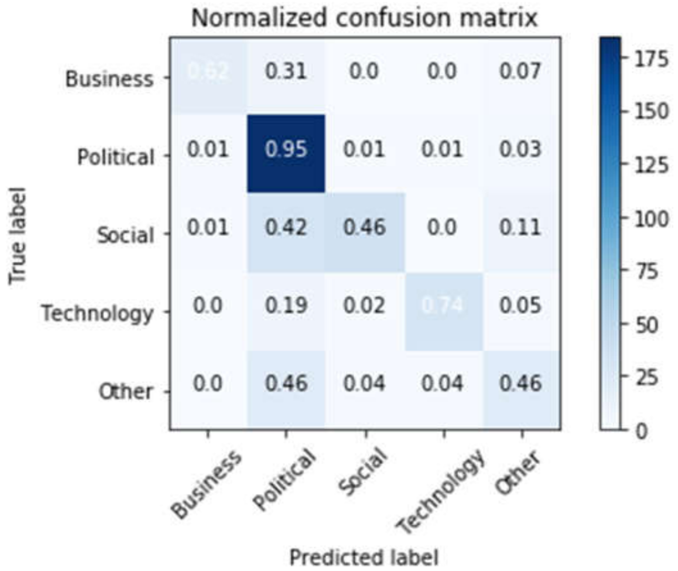


Figure 16: Confusion Matrix for Logistic Regression with Cross Validation

K-Nearest Neighbors				
Class	Precision	Recall	F_Measure	Accuracy
Business	0.72	0.45	0.55	0.5601
Political	0.90	0.69	0.78	
Social	0.89	0.10	0.18	
Technology	0.96	0.57	0.72	
Other	0.21	0.85	0.34	
Weighted Average	0.81	0.56	0.58	

Table 10: Evaluation Results for K-Nearest Neighbors

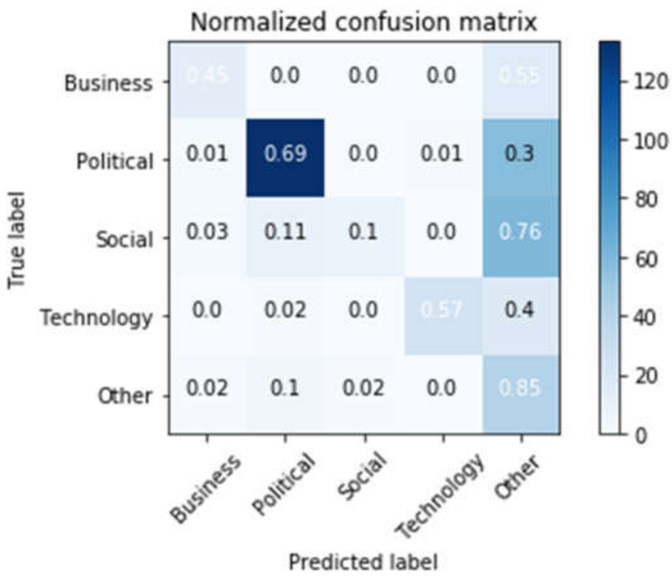


Figure 17: Confusion Matrix for K-Nearest Neighbors

Decision Tree				
Class	Precision	Recall	F_Measure	Accuracy
Business	0.81	0.59	0.68	0.7340
Political	0.87	0.85	0.86	
Social	0.67	0.43	0.52	
Technology	0.82	0.86	0.84	
Other	0.42	0.75	0.54	
Weighted Average	0.76	0.73	0.74	

Table 11. Evaluation Results for Decision Tree

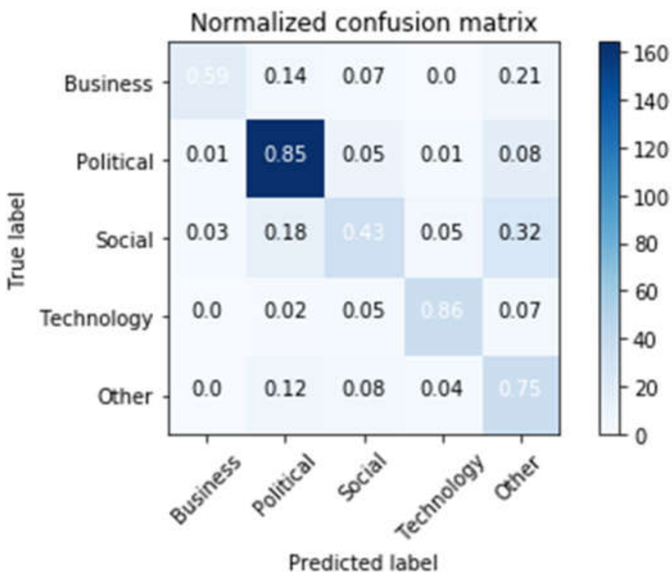


Figure 18: Confusion Matrix for Decision Tree

Neural Network - Multi-Layer Perceptron Classifier				
Class	Precision	Recall	F_Measure	Accuracy
Business	0.80	0.69	0.74	0.7570
Political	0.80	0.92	0.86	
Social	0.74	0.51	0.60	
Technology	0.87	0.79	0.82	
Other	0.49	0.76	0.75	
Weighted Average	0.76	0.76	0.75	

Table 12: Evaluation Results for Neural Network

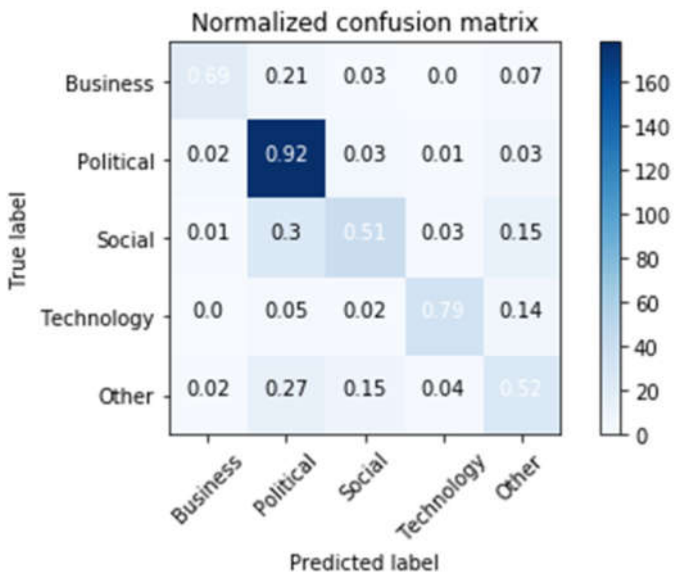


Figure 19: Confusion Matrix for Neural Network

Before discussing the result, we first do a summary of the metrics data from Question 4.4. Please note that all metrics in the following table are weighted average from tables in Question 4.4

	Precision	Recall	F_Measure	Accuracy
Gaussian Naïve Bayes	**0.71**	**0.72**	**0.71**	**0.7187**
Random Forest	**0.75**	**0.74**	**0.73**	**0.7366**
Linear Support Vector Classification	**0.78**	**0.79**	**0.77**	**0.7852**
Logistic Regression with Cross Validation	**0.77**	**0.74**	**0.73**	**0.7442**
K-Nearest Neighbors	**0.81**	**0.56**	**0.58**	**0.5601**

Decision Tree	**0.76**	**0.73**	**0.74**	**0.7340**
Neural Network - Multi-layer Perceptron Classifier	**0.76**	**0.76**	**0.75**	**0.7570**

Table 13: Summary of Evaluation Results for All Classifiers

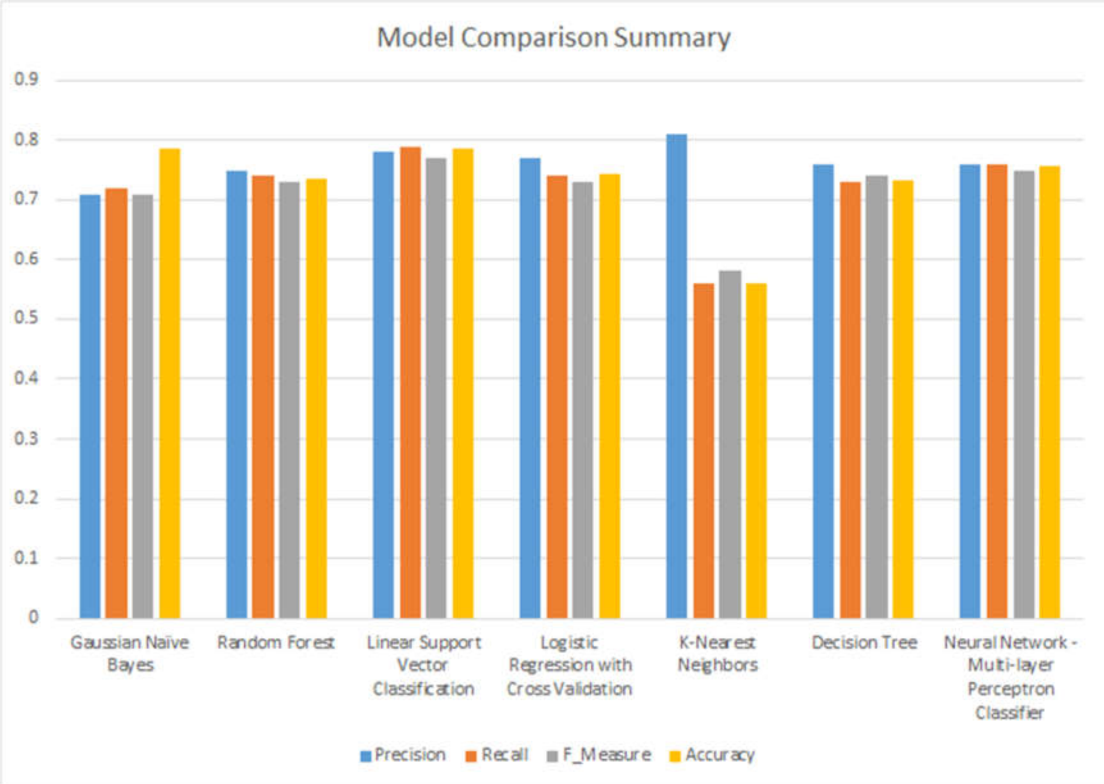


Figure 20: Summary of Model Comparison

Observations and discussion:

1. K-Nearest Neighbors model has the highest precision.
2. Linear Support Vector Classification model has the highest recall.
3. Linear Support Vector Classification model has the highest F-measure.
4. Linear Support Vector Classification model has the highest accuracy.
5. For all models, the precision, recall and F-measure are always high in "*Political*", while "Social" and "Other" are relatively low. These may be because in our training samples, there are in total 655 records of "*Political*" in training data set,238 records of "Social" and 220 of "*Other*". Hence, "*Political*" has far more records than other categories. We will try to address this issue by undersampling "*Political*" tweets to achieve a relative balanced data set.
6. Although K-Nearest Neighbors had the highest precision, it performs worst in terms of Recall, F-Measure and accuracy. This is because the KNN model has 89% precision but only 10% recall in Social category. This means only about 10% news are irrelevant news in social category, but about 90% social news are wrongly classified.

7. Linear Support Vector is better than other classifiers in terms of the overall values in different matrices.

4.5 Discuss performance metrics, e.g., records classified per second, and scalability of the system

Metrics (average)	Value	Description
Crawl Time per Tweet	17.94 milliseconds	Time spent to crawl one tweet from Twitter and store it in Solr
Classification Time per Tweet	0.12 milliseconds	Time spent to classify one tweet
Model Training Time	0.91 seconds	Time spent to train a model

Table 14: Performance Metrics

From the table, we can see that the classification time is very small compared to other two. Even if we need to classify 100,000 tweets, the system only need around 12 seconds (8333 tweets per second). Therefore, system is highly scalable in terms of classifying high volume of tweets.

For training classification model, although it needs around 1 second to train the model, the model only needs to be trained once when the system first start, which means if system is running without terminating, the model would be stored in the system without any need to retrain it.

However, the crawling time is quite substantial compared to other two. To crawl 100,000 tweets, the system needs 17.94 seconds (5574 tweets per second). Although it seems time-consuming to crawl new data, data crawling would not be performed all the time by users unless users keep updating the tweets. Even if users keeps updating tweets, the news sources would not update their tweets very frequently. Hence, on average, when user choose to update tweets, it will normally take 3 to 4 seconds, which are quite reasonable. Currently, we crawl data from 5 major news sources and even if we increase our system to incorporate 50 news sources, it will only take less than a minute to crawl and store all tweets data to Solr.

4.6 A simple UI for visualizing classified data would be a bonus (but not compulsory)

For this part, we did not create a separate UI. Instead, we display the classification result in our UI for Question 2. Each tweet will have its own category which will be displayed together with number of likes and number of retweets. Some examples are shown below.

Figure 21: Social News From Straits Times

Figure 22: Political News From BBC World News

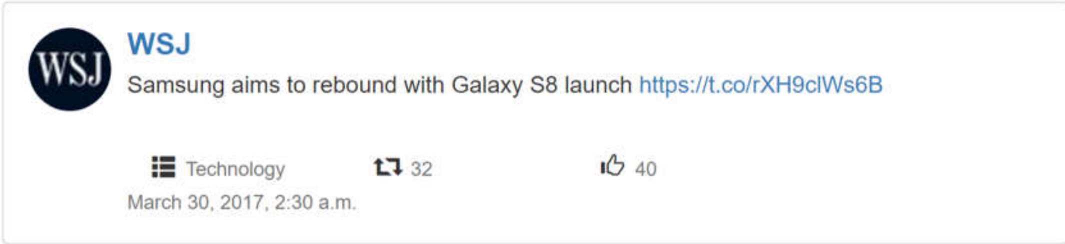


Figure 23: Technology News From Wall street Journal

Question 5

5.1 Explore some innovations for enhancing classification. Explain why they are important to solve specific problems, illustrated with examples

Classification On-the-fly

Although the model we used to do the classification is based on static data we initially crawled from Twitter API, the process of classification is on-the-fly. When user update the tweets, tweets would be firstly classified into one of the five categories and then store in Solr.

Undersampling on "Political" News

To investigate the problem of undersampling on Political data, we randomly removed 300 Political data in the original data set for training and rebuild the model to predict the label of each tweets again. The results are shown as the following. Please note that the cell highlighted in yellow are number that increases after undersampling.

	Precision		**Recall**		**F_Measure**		**Accuracy**	
	Before	**After**	**Before**	**After**	**Before**	**After**	**Before**	**After**
Gaussian Naïve Bayes	0.71	0.71	0.72	0.7	0.71	0.7	0.718	0.704
Random Forest	0.75	0.78	0.74	0.69	0.73	0.71	0.736	0.691
Linear Support Vector Classification	0.78	0.81	0.79	0.79	0.77	0.78	0.785	0.787
Logistic Regression with Cross Validation	0.77	0.81	0.74	0.77	0.73	0.78	0.744	0.774
K-Nearest Neighbors	0.81	0.76	0.56	0.51	0.58	0.54	0.560	0.511

Decision Tree	0.76	0.76	0.73	0.73	0.74	0.74	0.734	0.734
Neural Network - Multi-layer Perceptron Classifier	0.76	0.72	0.76	0.7	0.75	0.71	0.757	0.704

Table 15: Summary of Evaluation Results Comparison after Undersampling

Linear Support Vector Classification								
	Precision		Recall		F_Measure		Accuracy	
	before	after	before	after	before	after	before	after
Business	0.81	0.95	0.72	0.68	0.76	0.79	0.7852	0.7874
Political	0.8	0.88	0.94	0.96	0.86	0.92		
Social	0.73	0.62	0.61	0.89	0.66	0.73		
Technology	0.85	1.00	0.83	0.69	0.84	0.81		
Other	0.71	0.62	0.46	0.41	0.56	0.49		
Weighted Average	0.78	0.81	0.79	0.79	0.77	0.78		

Table 16: Linear Support Vector Classification Result Comparison after Undersampling

Based on the result, we can see the performances of most of the models we tried are dropped after we randomly remove about 300 political data. The reason is because the most significant contributor to the performance - political data are less and pull down the quality of prediction. However, Linear Support Vector Classification, which model we implemented in our system has an obvious improvement after data pruning. In the detailed table, we can figure out that the prediction of Business, Political, Social and Technology achieves improvement significantly. Thus, there is indeed a problem of undersampling in our original data set.

Overfitting Issue

To avoid possible overfitting issue, we use Cross Validation to investigate whether the model is only good for the training data set or will also perform well on the incoming real data from Twitter API.
Method *cross_val_score* from *sklearn.model_selection* is used to implement overfitting checking. We set the parameter *cv* as 10, which means we are using 10-fold cross validation. Cross validation scores for all the models are shown in the following table.

Model	10-fold Cross Validation Score
Gaussian Naïve Bayes	0.73318
Random Forest	0.74511
Linear Support Vector Classification	**0.79020**

Logistic Regression with Cross Validation	0.77162
K-Nearest Neighbors	0.58908
Decision Tree	0.72872
Neural Network - Multi-layer Perceptron Classifier	0.77257

Table 17: 10-fold Cross Validation Result

As we can see, the model we finally used in our system, Linear Support Vector Classification, achieves the highest 10-fold cross validation score among all the models we tried. As such, we can conclude that Linear Support Vector Classification model can avoid overfitting issue and will perform well on the classification of incoming real data.

Conclusion

Throughout this project, a Django web application is built to perform news retrieval, search and classification. Crawling up-to-date data from 5 popular Twitter news sources is realized via Twitter API. Necessary data manipulation, selection and indexing are performed by integrating the functionalities of the indexing system, Solr. On-the-fly data pre-processing including text cleaning, stop words removal, stemming is done whenever new updates for news are required by user. All the functions including requiring updates, inputting a search query, sorting the news displayed, change the source or category of the news displayed, are performed through a simple UI implemented with Bootstrap. Classification to 5 categories is implemented with Python *sklearn* package. The classification model is built by a static training data set with 1527 manually labelled categories and evaluated according to Precision, Recall, F-Measure, Accuracy, Confusion Matrix, and Cross Validation.