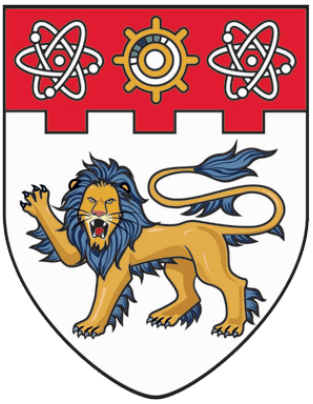

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4034 - INFORMATION RETRIEVAL

Assignment Report
Group 8

Han Xing Jie	U1620807A
Augustus Lee Jian Li	U1621366G
Eng De Wei	U1621256G

April 10, 2019

Report Overview

Overview In this assignment, we crawled a text corpus of 10,000 tweets off Twitter and developed a web application to retrieve tweets that are of relevance to the users. We applied concepts of machine learning to classify and add another dimension of filters for better information retrieval. Our objective is to improve upon the current implementation of search on Twitter, by providing a user-friendly and intuitive way of retrieving information of interest to the users.

Report Structure The report is structured into chapters where each chapter corresponds to a question in the assignment brief, and each subsection further elaborates on the answers:

- **Chapter 1** covers questions on the methodology of crawling, the information extracted and the statistics of the crawled corpus
- **Chapter 2** provides a brief overview of the web application, the features and performance metrics of retrieval using Solr. A full demonstration of the web application can be provided in the YouTube link in chapter 6
- **Chapter 3** explores some improvement over the default indexing provided by Solr, and the innovations we implemented to further enhance the ranking of results for users
- **Chapter 4** provides an overview of the classification models used, a discussion on their performance on the indexed corpus, the performance of the models itself with respects to the speed of classification and the scalability of the system
- **Chapter 5** highlights the improvements we made to the classifiers by implementing different prediction strategies and ensemble classifiers
- **Chapter 6** contains the links to the video and the two compressed files containing the data of the corpus and application

Submitted by Group 8 on the 10^{th} of April, 2019

Contents

1 Crawling

The text corpus was crawled from Twitter using the Twitter developer API. We initialized the crawler on March 7, 2019 at 6.56am to archive 10,000 tweets from 5 accounts - The Verge (@Verge), The Economist (@TheEconomist), CNN (@CNN), BBC Sports (@BBCSport), and The Wall Street Journal (@WSJ).

In this section, we rationalize the reason for choosing these particular Twitter accounts, the process involved for crawling, the information stored (and in which mediums) as well as the statistics of our crawled corpus.

1.1 Crawler and Storage Mediums

How you crawled the corpus (e.g., source, keywords, API, library) and stored them (e.g., whether a record corresponds to a file or a line, meta information like publication date, author name, record ID)

1.1.1 Crawler

A crawler was developed using Python utilizing the Twitter API[1] to crawl Twitter for the text corpus. The crawler was ran on March 7, 2019 at 6.56am to obtain the latest 2000 tweets from each Twitter handler. In total, 10,000 tweets were obtained.

Twitter handler considerations The Twitter accounts which we crawled the tweets from were chosen specifically with the intent of information retrieval, classification and extensibility in mind. For example, ESPN (@ESPN) often have tweets where image forms half the context compared to BBC Sports (@BBCSport), which is not ideal from an information retrieval's perspective. The same can be said to other Twitter accounts - some technology accounts use an excessive amount of emojis to appeal to the younger audience which may take away context if we remove the emojis during pre-propcessing phase.

As the performance of a multi-class classification task can be greatly affected by the distribution of its classifications, we only crawled from Twitter accounts that best provides an evenly distributed text corpus in terms of the classes for which they can be categorized into. For example, CNN (@CNN) and NY Times (@Nytimes) report on roughly the same news - rapid, Americanized news focusing on US politics whereas The Economist (@TheEconomist) covers issues on a more global stage.

[1] `https://developer.twitter.com/en.html`

Figure 1: Comparison of text styles between accounts

In summary, with those considerations in mind, the accounts we crawled the tweets from are:

- CNN (World / US)
- The Economist (World)
- BBCSports (Sports)
- Wall Street Journal (Business)
- Verge (Technology / Entertainment)

1.1.2 Storage Mediums

To ensure extensibility of the applications of the crawled corpus, we stored the crawled data into various formats during various phases of the crawling:

1. Firstly, when the data is crawled from Twitter, the raw response from each Twitter account will be stored in its native JSON format in its own file (WSJ.JSON, CNN.JSON etc)

2. Secondly, we extract the relevant fields for indexing from the JSON files and store it into Comma-Separated Values (CSV) files for ease of processing by the various machine learning libraries.

3. Lastly, for updating Solr with the categorized tweets, we first convert the CSV files to Solr's native XML format, which allows us more granularity in controlling the indexing formats within Solr.

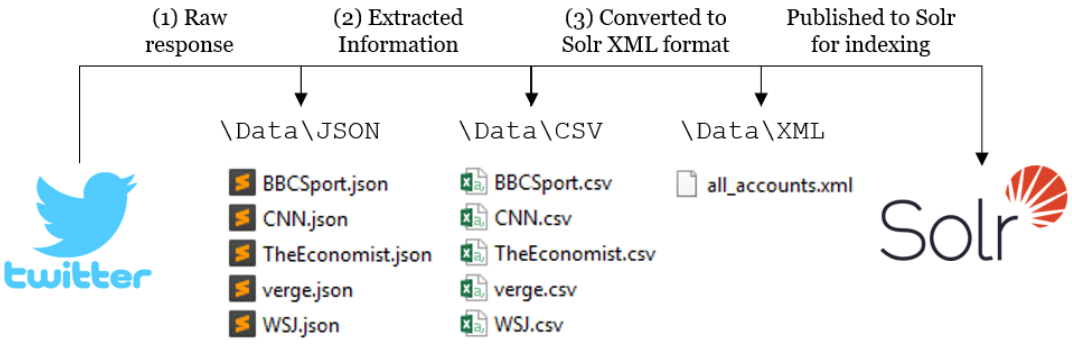

Figure 2: Folders of CSV, JSON and XML in the data pipeline

Meta information We extracted fields that are of interest for this assignment from the raw JSON files to be indexed by Solr. The fields that we extracted are:

Field name	Data type	Description
id	String	Unique ID assigned to every tweet
tweet_text	String	Text of the tweet
username	String	Username of the account
tweet_retweet_count	Integer	Number of retweets
tweet_media_url	String	Image embedded in tweet (if any)
tweet_fav_count	Integer	Number of favorites
screen_name	String	Full name of the account
profile_image_url	String	Image used for the account
profile_description	String	Profile description
tweet_creation	Date time	Time when tweet was created
category	String	Category of the tweet text

Table 1: Information from tweets that will be indexed

These allow more flexibility in the queries. For example, we are able to return political tweets by CNN about India sorted by the when the tweet was made in ascending order.

1.2 Applications of Crawled Corpus and Example Queries

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

1.2.1 Applications of corpus

Firstly, we found that Twitter has a very unintuitive way of customizing search queries. For example, if we would like to search for tweets about Trump from CNN, we will have to:

1. Type 'Trump' in the search bar
2. Click on Advance search text at the left side of the webpage to be redirected to advanced search page
3. Type in 'Trump' as the query and 'CNN' as the account to search from
4. Click on 'Search'

Furthermore, the search does not allow us to order the tweets by certain popularity metrics (number of favorites or retweets) or by chronological order. This takes away the control of information retrieval from users by forcing the use of Twitter's proprietary search indexing and ordering algorithms to display results to the users.

Applications Therefore, for the application of this crawled corpus, we aim to provide users the convenience and control of being able to customize their search with an aggregation of search filters through a *user-friendly* and *intuitive* user interface, backed up by sufficient information from crawled corpus being indexed in Solr.

1.2.2 Example queries

The user will be able to perform the following queries from the web application:

Search for tweets containing a text or phrase...

- From all 5 Twitter accounts
- From only certain Twitter accounts
- Ordered by certain popularity metrics (number of favorites / retweets in ascending or descending order)
- In chronological order (ascending or descending)
- That are categorized as 'Sports'

With these requirements in mind, we found the crawled corpus sufficient in fulfilling these queries.

1.3 Corpus Statistics

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

The following table shows the statistics of the crawled corpus.

Metric	Values
Number of records	10,000
Number of words	201,196
Number of words after preprocessing	191,730
Number of words after preprocessing (*Unique*)	17,079
Number of words after removing stopwords	123,885
Number of words after removing stopwords (*Unique*)	16,936

Table 2: Statistics of the crawled corpus

The processing phase mentioned in the table consists of:

- Removing Twitter mentions (e.g. @DonaldTrump)
- Removing HTTP links (e.g. https://t.co/Jr6snSf2eS)
- Removing special characters and numbers

Stopwords was obtained using NLTK version 3.3.

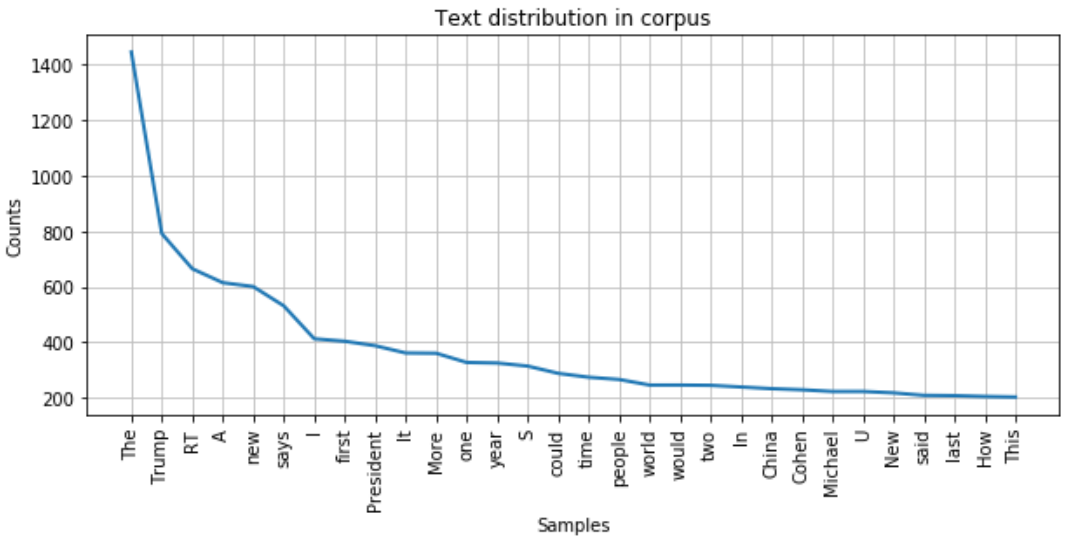

Figure 3: Distribution of tokens after preprocessing and removing stopwords

2 Indexing and Querying

We used the recommended Apache Solr for indexing and searching of the crawled corpus. For the web application, we used VueJS to utilize the Flux state management pattern to handle search requests from the user and responses from Solr. To support incremental indexing of data, we developed a script which automates the entire data pipeline - from crawling of tweets, saving of the files, categorization via a trained classifier model, and pushing the processed corpus to Solr for indexing.

In this section, we will showcase some aspects of the web application, explain the implementation of the automated incremental indexing of data, and report on the performance metrics of queries. Lastly, we will explain the modifications we made to enhance indexing and filtering of search queries.

2.1 Web Interface

Build a simple Web interface for the search engine (e.g., Google)

Users will first be greeted with a landing page of the website which states the purpose and objective of the web application - allowing users to retrieve tweets by order or relevancy through indexing, filtering and sorting capabilities.

Figure 4: Landing page of the web application

The tweets will be rendered using Twitter's native embedding[2].

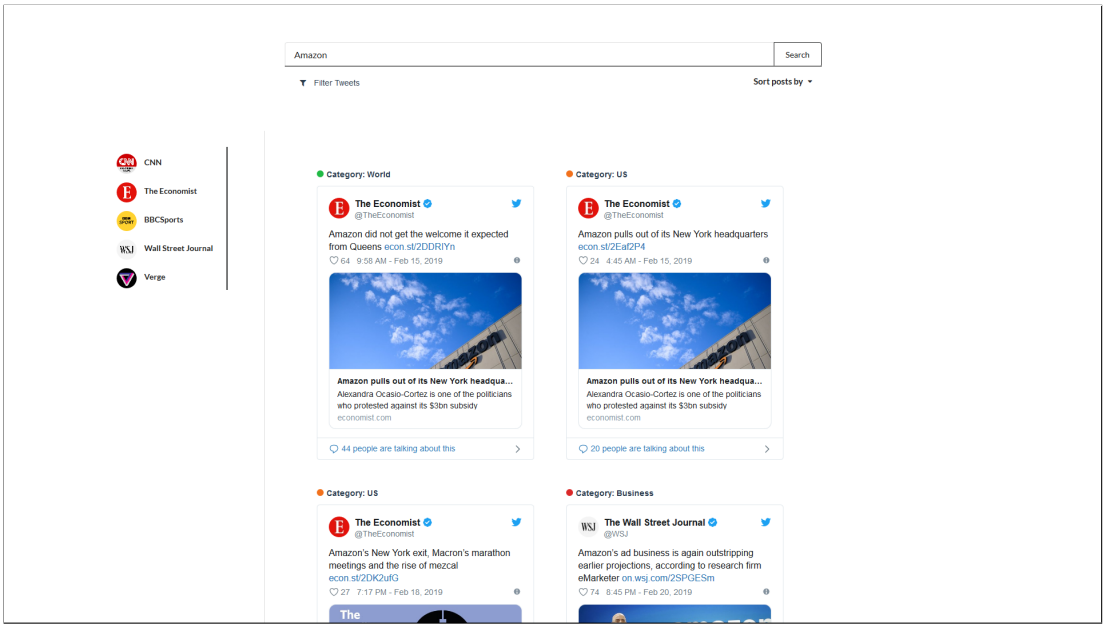

Figure 5: Displaying results for query 'Amazon'

Twitter account selection, category filtering and results sorting are presented in an intuitive and user-friendly manner, allowing the users to focus on the content of retrieval.

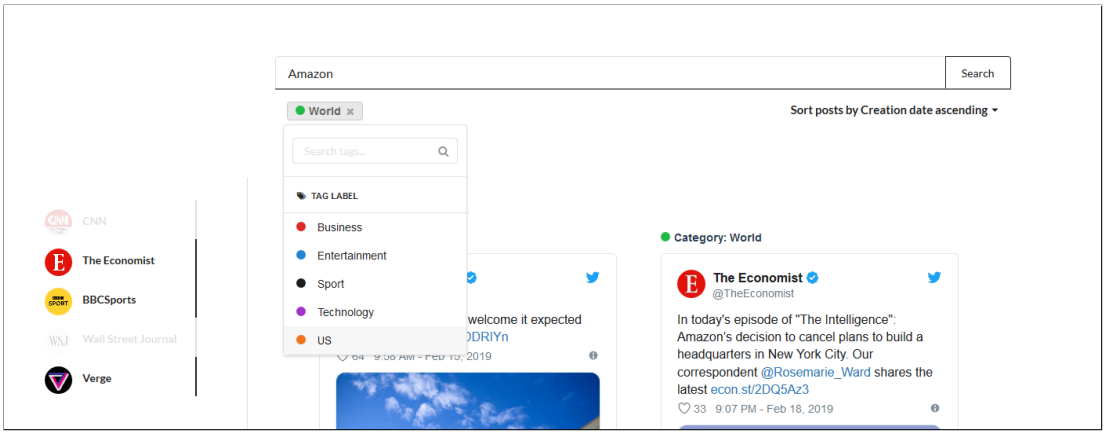

Figure 6: Applying filters for Twitter accounts, sorting, and filtering by categorization in one page

[2]`https://developer.twitter.com/en/docs/twitter-for-websites/embedded-tweets/overview.html`

Spell checks will be performed if insufficient results are returned. Users can search for the suggestions directly by clicking on it.

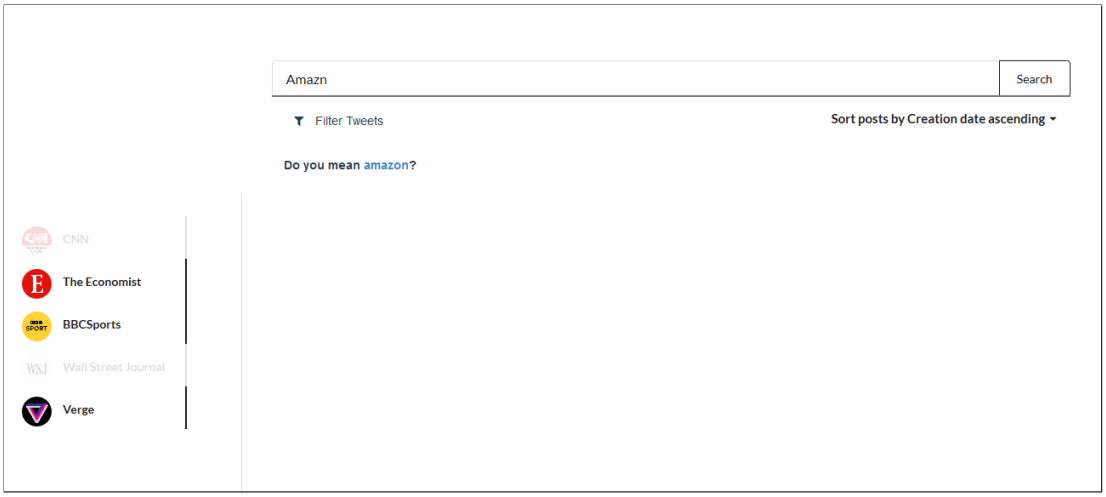

Figure 7: Spelling suggestion for query 'Amazn' returned 'Amazon'

The full demonstration of the web application can be found in the video presentation. Kindly note that the web application was designed for screen resolutions of size 1920x1080.

2.2 Incremental Indexing

A simple UI for crawling and incremental indexing of new data would be a bonus (but not compulsory)

To reduce visual clutter, we decided to implement incremental indexing of new data by the use of a Python script from the back-end directly. The user only interacts with the web application for information retrieval, and the maintenance of the indexed documents will be handled separately, out of sight of the user.

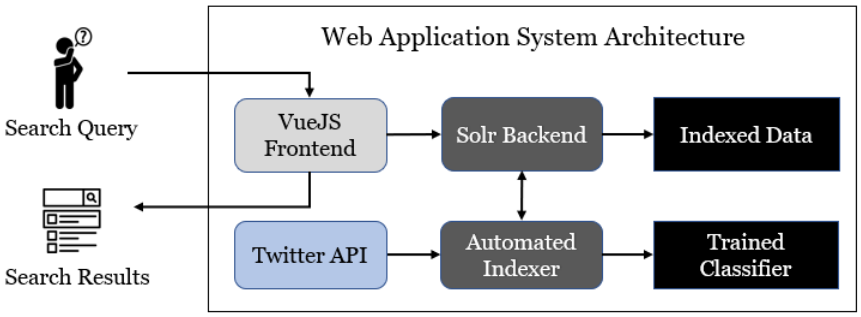

Figure 8: System architecture of the web application.

Automated Indexer The script is capable of retrieving new tweets from the 5 Twitter accounts, preprocess them, save the data in the various formats previously illustrated in Figure 2, classify using a trained classifier and commit the new data to Solr for indexing - all automatic, without any user actions. This allows the system to incrementally index the new tweets from the 5 accounts of interest for retrieval by users, while allowing the users to focus on the task of querying tweets.

2.3 Query Performance

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

To measure the performance of the queries, we used the 'QTime' parameter of a Solr search response. The QTime measures in milliseconds the time it took for Solr to execute a search. As most queries return multiple results, we only note down the top result of the query in the following table, where the query is highlighted in bold and the top result of the query in italics.

Query	QTime	Results
'North Korea' from all 5 Twitter accounts *Top result: North Korea's economy is proving surprisingly resilient to sanctions. That makes things tricky for Donald Trump as he heads to this weeks summit with North Korea's leader https://t.co/yIuJpp7p6X*	0	203
'North Korea' from only CNN *Top result: Satellite images appear to show that North Korea has begun rebuilding a part of a long-range missile test facility, raising potential questions about the future of US-North Korea negotiations https://t.co/TtNiTAbowS*	1	96
'Noth Korea' from only CNN ordered by number of retweets (ascending) *Top result: While the failure to reach an agreement in Hanoi reflects poorly on both Trump and Kim, it looms larger for the North Korean leader, writes S. Nathan Park for @CNNOpinion https://t.co/lVjxNfmz5O*	0	96
'North Korea' from only CNN ordered by creation date (ascending) *Top result: He promised hed denuclearize. We hope hell make a big step towards that in the week ahead, Secretary of State Mike Pompeo says about Kim Jong Un ahead of President Trumps second summit with the North Korean leader in Hanoi this week. #CNNSOTU https://t.co/qVaN1DEMT8 https://t.co/eX2gduY1xV*	0	96
'North Korea' from only CNN that are categorized as 'World' *Top result: Vietnamese Prime Minister Nguyen Xuan Phuc said his country will "spare no effort" to host a successful summit between North Korea and the United States because in the end, it would be less costly than a war https://t.co/NTUXKYrLj6*	1	26

Table 3: Performance and results of querying with Solr

3 Indexing and Ranking Enhancements

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

Indexing We made a few changes to the way tweets are indexed in Solr to improve the robustness of the search. The configuration can be made to the XML file in directory `\solr-7.7.1\server\solr\ir\conf\managed-schema.xml`. The changes and the problems it solves are illustrated in the following table:

Indexing changes	Problems and Justifications
Use WhitespaceTokenizerFactory instead of StandardTokenizerFactory to preserve links	If we use StandardTokenizerFactory, any URL present within a particular tweet such as: https://site.com/analysis/, will be split into 3 tokens ['https', 'site.com', 'analysis']. If the user searched for the query 'Analysis', that particular record will be retrieved even though the word was only found in the URL and *not in the main text*. Using WhitespaceTokenizerFactory solves that issue by only tokenizing the text on white spaces, preserving the link as a whole.
Added EnglishPossessiveFilterFactory for indexer	EnglishPossessiveFilterFactory normalizes possessive nouns, allowing us to query for 'Amazon's' by retrieving tweets containing the word 'amazon'.
Added PorterStemFilterFactory for indexer	Allows us to search for queries with different variations of the word. E.g. searching for 'directs' returns queries such as: "... having a direct effect on the arts...", or "...Sarri represented a new direction for Chelsea..."

Table 4: Modifications to the default text indexer for Solr

We created a new field type called TWEET_TEXT which allows us to apply the modifications in table 4 to the tweets. The final configuration used to index tweets found in MANAGED-SCHEMA.XML are as shown below:

```
<fieldType name=''tweet_text'' class=''solr.TextField''positionIncrementGap=''100''>
  <analyzer type=''index''>
      <tokenizer class=''solr.StandardTokenizerFactory''/>
      <filter class=''solr.StopFilterFactory''words=''lang/stopwords_en.txt''
        ignoreCase=''true''/>
      <filter class=''solr.LowerCaseFilterFactory''/>
      <filter class=''solr.EnglishPossessiveFilterFactory''/>
      <filter class=''solr.PorterStemFilterFactory''/>
    </analyzer>

    <analyzer type=''query''>
      <tokenizer class=''solr.StandardTokenizerFactory''/>
      <filter class=''solr.SynonymFilterFactory''expand=''true''ignoreCase=''true''
        synonyms=''synonyms.txt''/>
      <filter class=''solr.StopFilterFactory''words=''lang/stopwords_en.txt''
        ignoreCase=''true''/>
      <filter class=''solr.LowerCaseFilterFactory''/>
      <filter class=''solr.EnglishPossessiveFilterFactory''/>
      <filter class=''solr.PorterStemFilterFactory''/>
    </analyzer>
</fieldType>
```

Listing 1: Code listing for indexing and querying filters for tweet texts

Ranking To allow users to sort tweets by date, we modified the format of the datetime representation in the crawled corpus and the schema file to read in the tweet creation field during indexing as PDATE format, which is Solr's native way of representing datetime. We also allowed the user to rank their results by popularity metrics such as number of retweets or favorites. This solves the problem of users not being able to sort tweets by chronological order or by popularity metrics through Twitter's default interface.

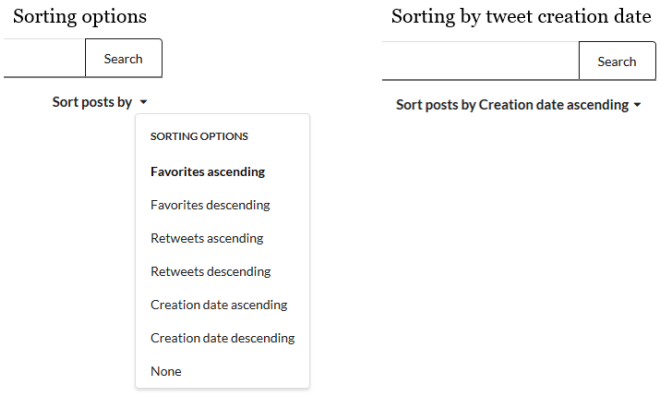

Figure 9: Users are able to sort by number of favorites or retweets, and even by creation date of the tweets

Furthermore, we utilized a trained classification model to categorize the contents of the tweets into categories. This solves the issue of users not being able to search for tweets by categories within various Twitter accounts (e.g. Searching for technology tweets by CNN (@CNN) might yield more factual information compared to the entertainment nature of The Verve (@Verve)).

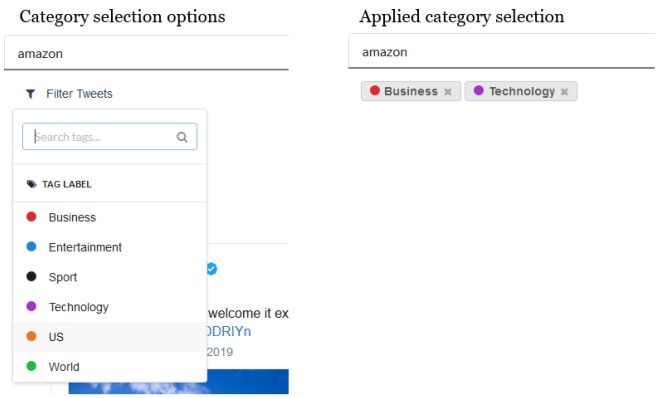

Figure 10: Category filters available to the user through a clear and intuitive interface

4 Classification

Define a set of categories (minimum three) the collected data could belong to and perform automatic classification on them. For example, you could crawl news articles and classify them into specific topics, e.g., sport, finance, and politics (auto-categorization). Another option is to classify data into positive, negative or neutral (sentiment analysis)

For classification, we crawled news articles from 5 different news channels in Twitter in order to diversify the results. These articles are classified based on the tweet content into 6 different categories: US, Sports, World, Technology, Entertainment and Business. The crawled data from Twitter does not contain any category label, thus we manually labelled 1000 tweets to use it as our training dataset to construct our classification model.

We construct our model by using the training dataset with classification models from sklearn package. After the model is trained, it is stored in the system and used to classify the initial 10,000 tweets. The model will be loaded during incremental indexing by a script and used to classify new tweets obtained.

4.1 Classification Motivation

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

The models that are used for the classifications are Naïve Bayes, Decision Tree, Random Forest, K-Nearest Neighbour, Logistic Regression and Linear SVC. The following section will discuss the strengths and weaknesses of each model:

Naïve Bayes Naïve Bayes methods are a set of supervised learning algorithms based on the Bayes theorem along with the naïve assumption of independence between each pair of features. The model is able to withstand irrelevant attributes and does not require a lot of training data. It is easy to use and fast to predict the classes of the test data. However, Naïve Bayes requires assumption of independent predictors which is almost impossible. It is also known to be a bad estimator, thus the predicted probability is not to be taken seriously.

Decision Tree Decision Tree is a non-parametric supervised learning method for classification and regression. It aims to build a model that can predict the target value based on simple decision rules inferred from the data features. The

model is easy to interpret and understand which is great for visual representation. It can work with numerical and categorical features and requires a small amount of data pre-processing. However, Decision Tree tends to overfit.

K-Nearest Neighbour K-Nearest Neighbor classification is based on the computation of a simple majority vote of the nearest neighbors of each query point. K-Nearest Neighbor classification is easy to understand and implement. It does not assume any probability distribution of input data and is able to respond quickly to changes. However, K-Nearest Neighbor classification is sensitive to localized data and it falls short on high dimensional dataset.

Logistic Regression Logistic Regression is a linear model used for classification. A logistic function is used to model the probabilities which describe the possible outcome of a single trial. It has low variance and is able to provide probabilities for outcomes. To reduce overfitting, inverse of regularization strength is used. However, Logistic Regression is highly bias and does not work well on non-linear data.

Support Vector Machine Support Vector Machines (SVM) are a set of supervised learning methods used for classification, regression and outliers detection. SVM classifies the data into different classes based on the line which separates the training data into different classes. The points that are close to the line are called support vectors. It is versatile with different kernel functions which can be used to specified decision function and it is also very effective in high dimensional data. However, SVM do not provide probability estimates. It is difficult when choosing the best kernel functions and it becomes less effective with nosier dataset that has overlapping classes.

Stochastic Gradient Descent Stochastic Gradient Descent (SGD) is a variation of Gradient Descent (GD). GD is an iterative method that requires the weight and biases of the model and a cost function. The aim is to reduce the value of the cost function to as low as possible. To achieve a higher accuracy for the model, the value of the cost function will need to be low. To do that, the algorithm will need to tweak the weight and biases of the model while going through the training data repeatedly.

In GD, the algorithm has to run through all the samples in the training data to perform an update on the parameter of a specific iteration. However, this is not a problem in SGD as the algorithm only has to run through one sample to

perform an update on the parameter of a specific iteration. SGD is also preferred when the dataset is large as the algorithm will take a shorter time to update the parameters as compared to GD.

4.2 Data Preprocessing

Discuss whether you had to preprocess data and why

In this project, we are trying to build different models for classifying the tweets into different categories and in order to do so, we have to ensure the text within each tweet can be used for classifying. The following steps are what we had done to preprocess the tweets.

1. Removing any Twitter handles.
2. Removing any links.
3. Removing any special characters and numbers.
4. Removing any words that has less 3 characters.
5. Performing stemming on the text using PorterStemmer
6. Converting text to word count vectors using CountVectorizer
7. Converting text to word frequency vectors using TfidfTransformer

Steps 1-4 We wrote a few functions that iterate through every tweet to remove the text that were unnecessary for classifying.

Step 5 We used PorterStemmer imported from the nltk.stem.porter package. This stemmer helps to reduce each word within a tweet to the base form and by using this, we are able to reduce the size of our vocabulary. This also helps to speed up the time for classifying the tweets.

Step 6 We used CountVectorizer imported from the sklearn.feature_extraction.text package. With this, we can convert the text from the above steps into a matrix of token counts for each word of the text.

Step 7 We used TfidfTransformer from the sklearn.feature_extraction.text package. With this, we can convert the matrix from CountVectorizer into a matrix represented by the normalized tf-idf values of each word.

4.3 Labeling and Inter-annotator Agreement

Out of 10,000 tweets crawled, we assigned 2 members to manually label 1000 tweets to obtain the categories. The tweets were according to the 6 categories we had come up with. The table below shows results of the comparison of the labelling between the 2 members.

	Business	Entertainment	Sport	Technology	US	World	Total
Business	69	1	1	2	5	4	82
Entertainment	0	92	3	0	1	1	97
Sport	3	2	205	1	1	2	214
Technology	4	1	2	137	3	1	148
US	10	2	3	2	246	9	272
World	3	0	1	2	5	176	187
Total	89	98	215	144	261	193	1000

To calculate the inter-annotator agreement, we had to use Cohens Kappa coefficient. The formula for this is:

$$Kappa = \frac{P(A) - P(E)}{[1 - P(E)]} \tag{1}$$

First, we had to calculate $P(A)$, which is the observed agreement of labels from the 2 members.

$$P(A) = \frac{69 + 92 + 205 + 137 + 246 + 176}{1000} = 0.925 \tag{2}$$

Next, we had to calculate $P(E)$, which is the probability of chance the 2 members agree on derived using equation 3:

$$P(E) = \frac{1}{n^2} \sum_k n_{k1} n_{k2} \tag{3}$$

Where

$$\begin{aligned} \sum_k n_{k1} n_{k2} &= (89 + 82)^2 + (98 + 97)^2 + (215 + 214)^2 \\ &+ (144 + 148)^2 + (261 + 272)^2 + (193 + 187)^2 \quad = 765,060 \end{aligned} \tag{4}$$

And

$$\frac{1}{n^2} = \frac{1}{(1000 + 1000)^2} \tag{5}$$

Therefore,

$$P(E) = \frac{1}{(1000+1000)^2} \times 765,060 \approx 0.191 \tag{6}$$

Lastly, we had to calculate the Kappa coefficient with the calculated $P(A)$ and $P(E)$ with equation 1:

$$Kappa = \frac{[0.925 - 0.191]}{[1 - 0.191]} \approx 0.907 \tag{7}$$

We obtained a Kappa coefficient of **90.7%** which is higher than the **80%** needed.

4.4 Evaluation and Discussion of Results

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

The evaluation metrics of the models we used are shown in the table below.

	Precision	Recall	F1-Score	Accuracy
Naïve Bayes	0.65	0.60	0.55	0.60
Logistic Regression	**0.71**	**0.71**	**0.70**	**0.71**
K-Nearest Neighbors	0.65	0.63	0.63	0.63
Decision Tree	0.55	0.55	0.54	0.55
Support Vector Machine	0.76	0.62	0.58	0.62
Stochastic Gradient Descent	0.68	0.69	0.68	0.69

Table 5: Performance overview of the models

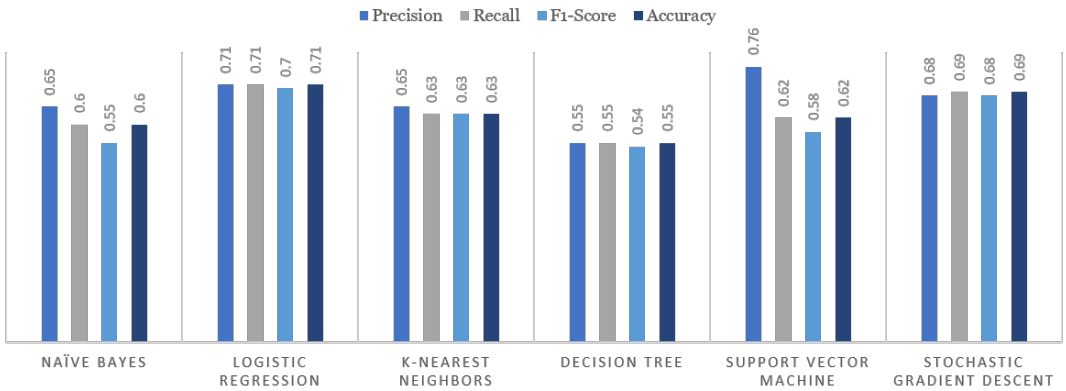

Figure 11: Comparison of classification results

4.4.1 Detailed Results for Naïve Bayes

	Precision	Recall	F1 Score	Accuracy
Business	0.00	0.00	0.00	
Entertainment	1.00	0.15	0.27	
Sport	0.86	0.87	0.86	
Technology	0.79	0.47	0.59	0.60
US	0.44	0.96	0.61	
World	0.79	0.35	0.49	
Weighted Average	0.65	0.60	0.55	

Table 6: Performance Metrics for Naïve Bayes

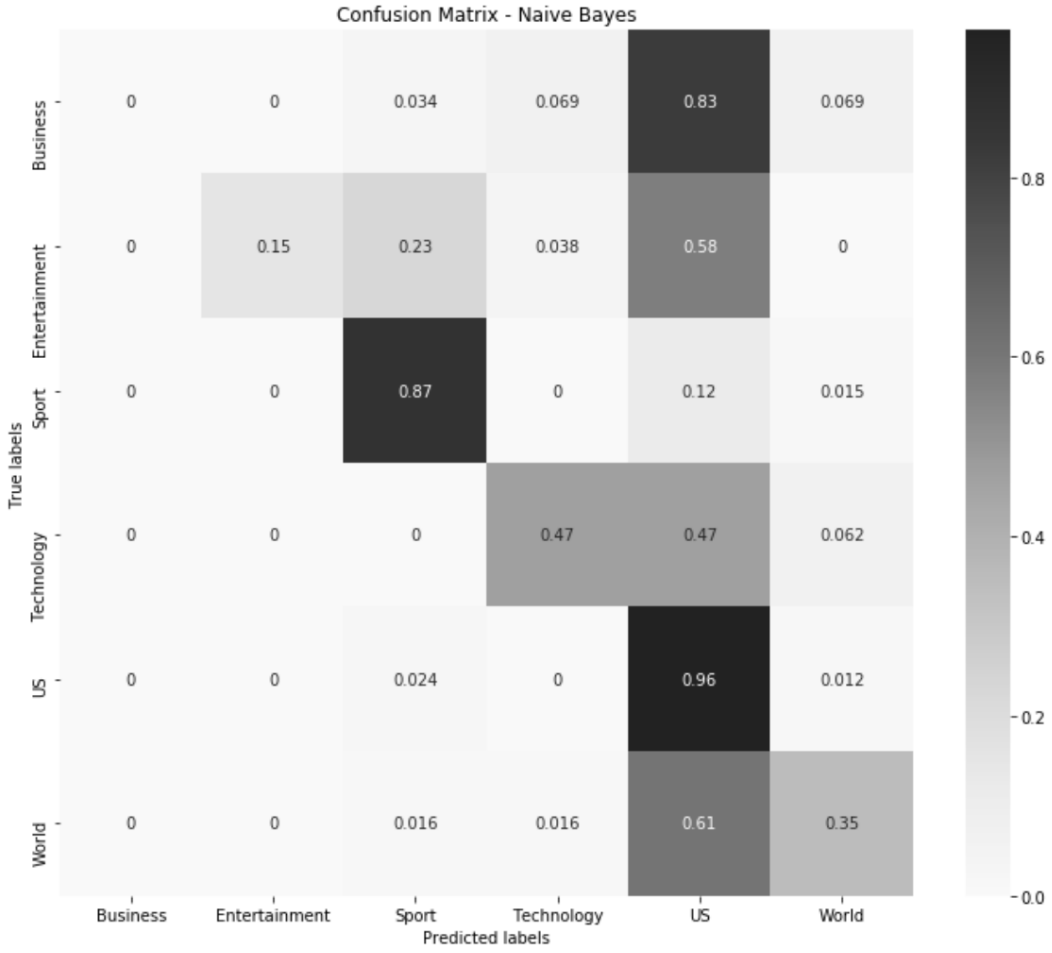

Figure 12: Naïve Bayes Confusion Matrix

4.4.2 Detailed Results for Logistic Regression

	Precision	Recall	F1 Score	Accuracy
Business	0.59	0.45	0.51	
Entertainment	0.76	0.62	0.68	
Sport	0.69	0.87	0.77	
Technology	0.84	0.50	0.63	0.71
US	0.72	0.84	0.78	
World	0.71	0.63	0.67	
Weighted Average	0.71	0.71	0.70	

Table 7: Performance Metrics for Logistic Regression

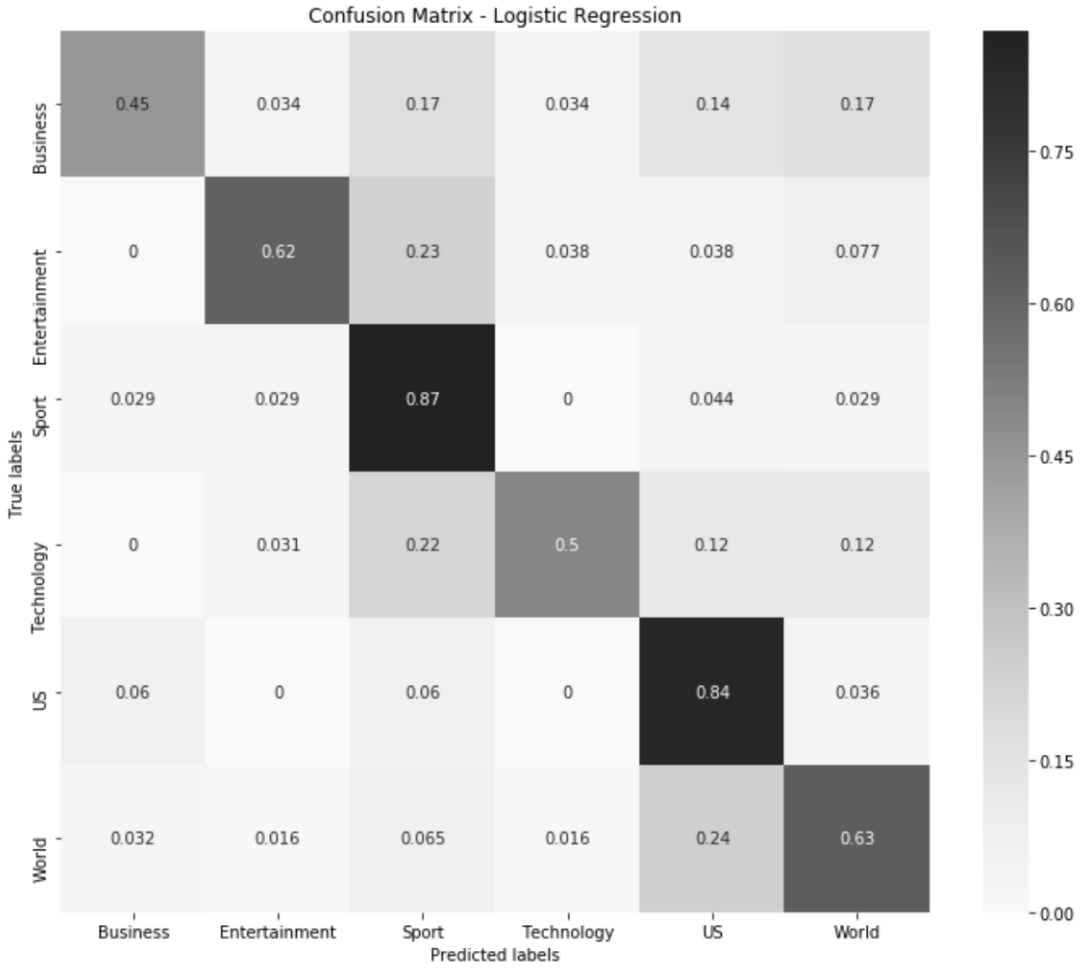

Figure 13: Logistic Regression Confusion Matrix

4.4.3 Detailed Results for K-Nearest Neighbors

	Precision	Recall	F1 Score	Accuracy
Business	0.50	0.52	0.51	
Entertainment	0.67	0.69	0.68	
Sport	0.59	0.87	0.70	
Technology	0.50	0.53	0.52	0.63
US	0.78	0.69	0.73	
World	0.67	0.39	0.49	
Weighted Average	0.65	0.63	0.63	

Table 8: Performance Metrics for K-Nearest Neighbors

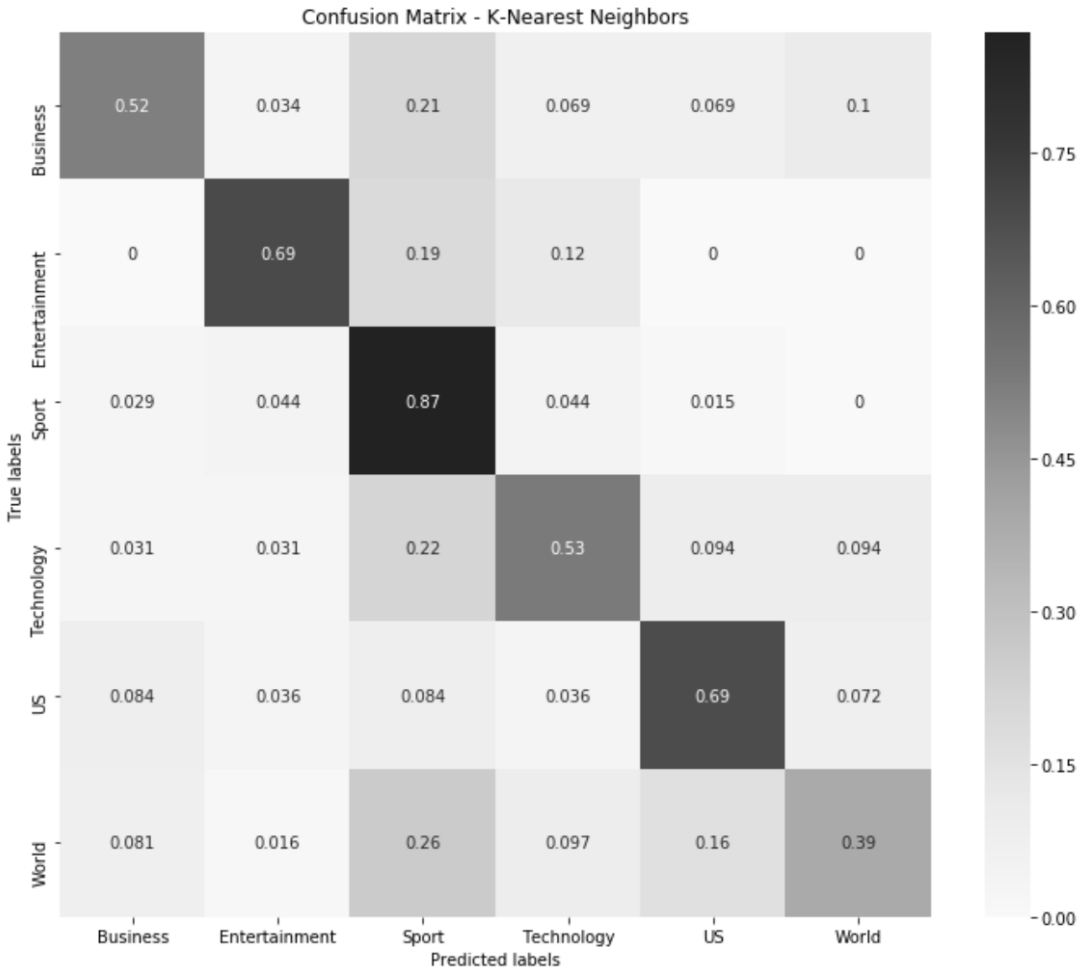

Figure 14: K-Nearest Neighbors Confusion Matrix

4.4.4 Detailed Results for Decision Tree

	Precision	Recall	F1 Score	Accuracy
Business	0.56	0.34	0.43	
Entertainment	0.40	0.38	0.39	
Sport	0.55	0.71	0.62	
Technology	0.33	0.41	0.37	0.55
US	0.69	0.71	0.70	
World	0.56	0.40	0.47	
Weighted Average	0.55	0.55	0.54	

Table 9: Performance Metrics for Decision Tree

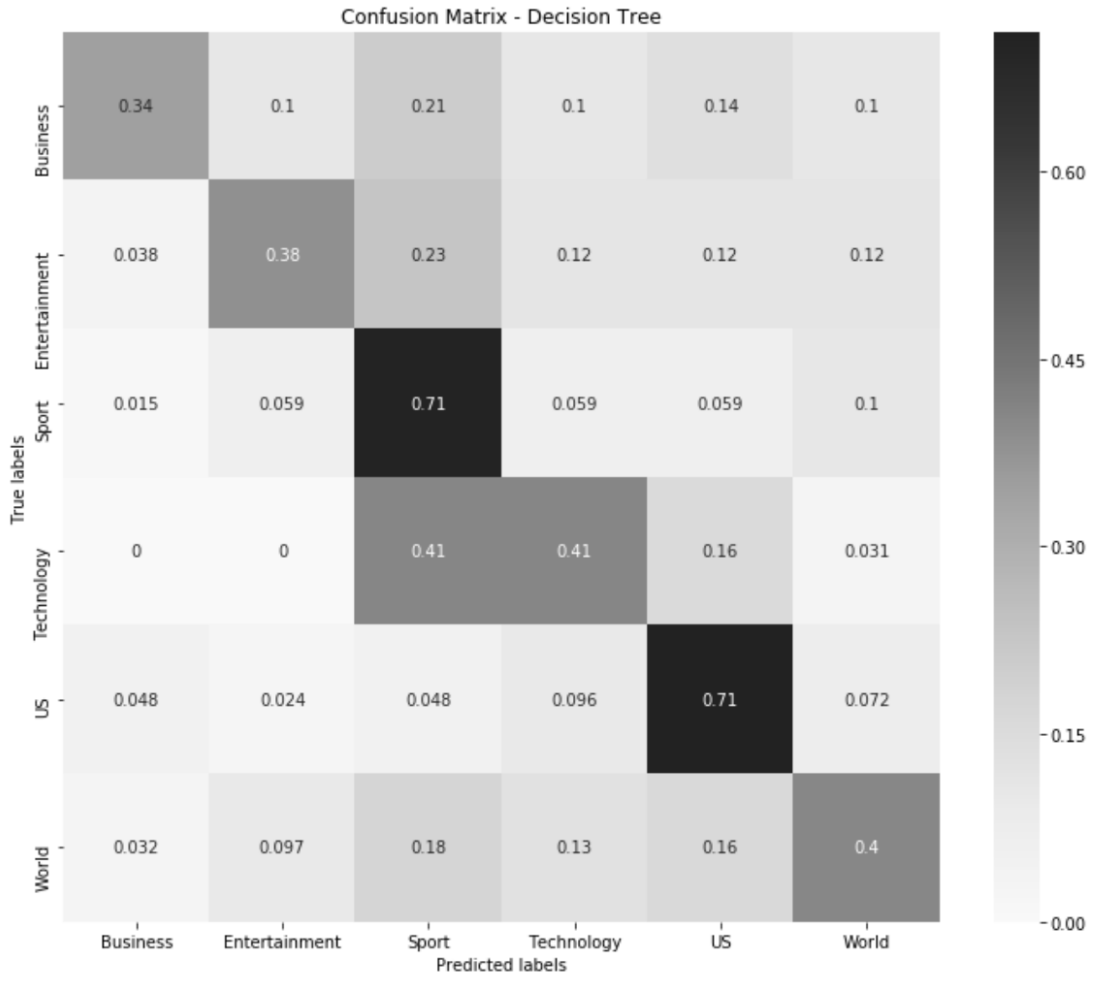

Figure 15: Decision Tree Confusion Matrix

4.4.5 Detailed Results for Support Vector Machine

	Precision	Recall	F1 Score	Accuracy
Business	1.00	0.03	0.07	
Entertainment	1.00	0.19	0.32	
Sport	0.91	0.78	0.84	
Technology	0.88	0.44	0.58	0.62
US	0.47	0.99	0.64	
World	0.69	0.50	0.58	
Weighted Average	0.76	0.62	0.58	

Table 10: Performance Metrics for Support Vector Machine

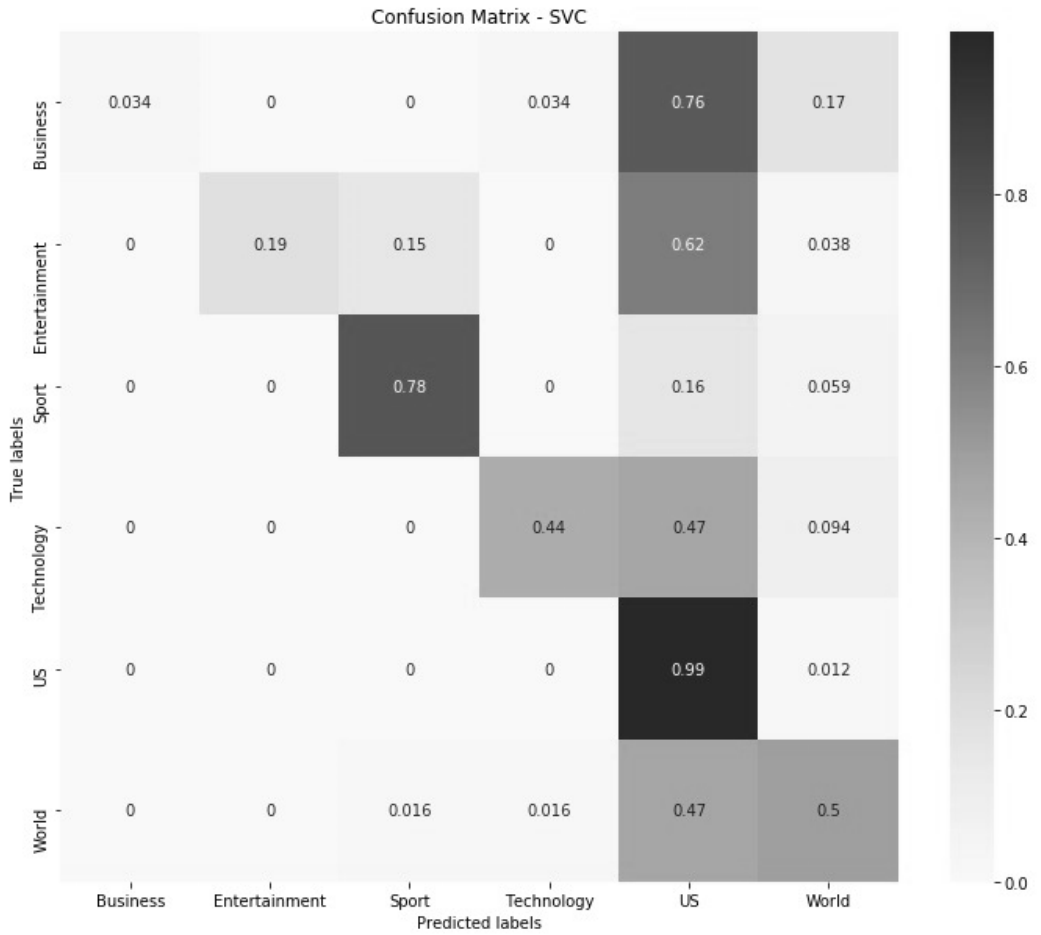

Figure 16: Support Vector Machine Confusion Matrix

4.4.6 Detailed Results for Stochastic Gradient Descent

	Precision	Recall	F1 Score	Accuracy
Business	0.61	0.38	0.47	
Entertainment	0.62	0.62	0.62	
Sport	0.78	0.84	0.81	
Technology	0.66	0.59	0.62	0.69
US	0.72	0.77	0.74	
World	0.60	0.63	0.61	
Weighted Average	0.68	0.69	0.68	

Table 11: Performance Metrics for Stochastic Gradient Descent

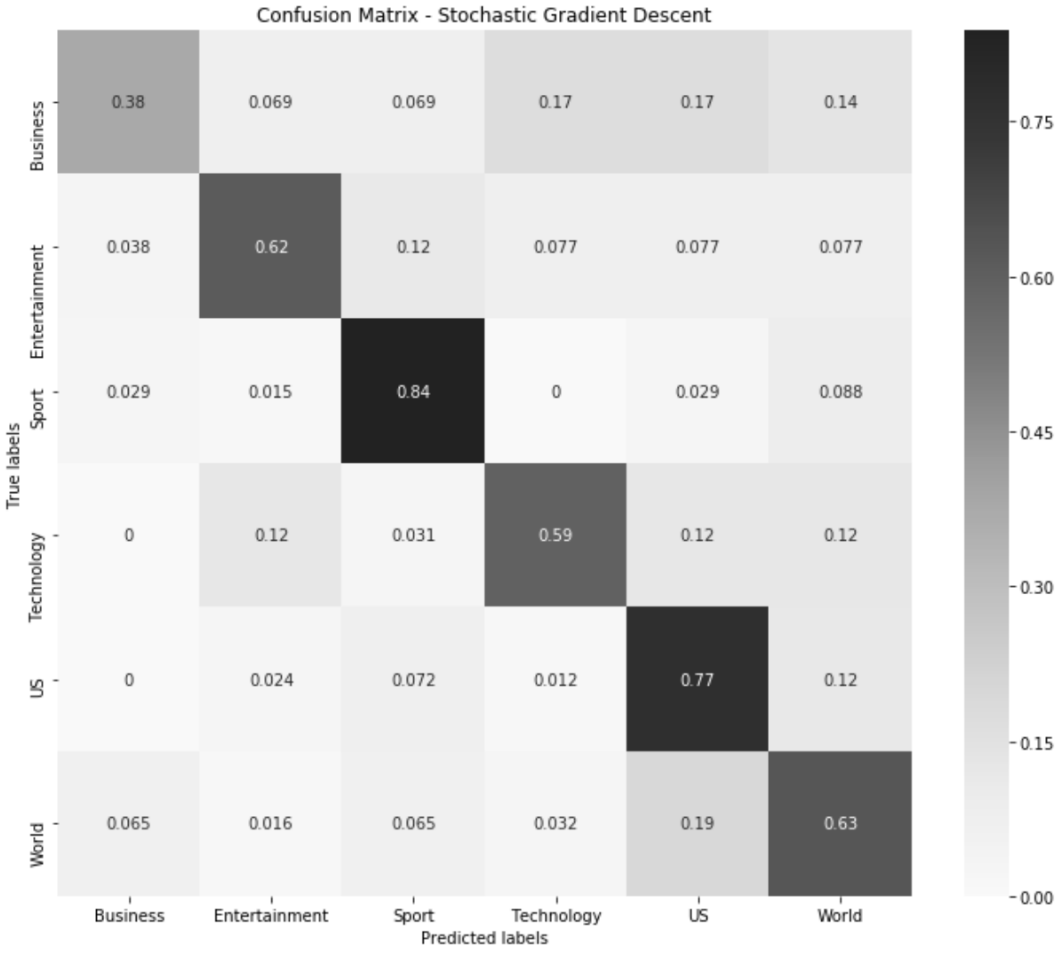

Figure 17: Stochastic Gradient Descent Confusion Matrix

4.4.7 Results Discussion

The following are the observations we made for the classification:

- The classification model with the highest precision is Support Vector Machine.
- The classification model with the highest recall, highest F1-score and highest accuracy is Logistic Regression.
- Among the 6 categories, Business category on average has a lower precision, recall and F1-score when compared among the 6 classification models. This might be due to the category being labelled the least amount of time in our training data. In our training data of 1000 tweets, only 84 tweets were labelled under the Business category.
- Among the 6 categories, Sport and US category on average have a higher precision, recall and F1-score when compared among the 6 classification models. This might be due to the two categories being labelled the most in our training data. In our training data of 1000 tweets, 265 tweets were labelled under the US category and 211 tweets were labelled under the Sport category.
- In the results for Support Vector Machine classification model, both Business category and Entertainment category has a precision score of 1. However, the recall score was very low at 0.03 and 0.19 respectively. This means that 100% of the retrieved tweets for Business and Entertainment categories were relevant, but only 3% of the Business labelled tweets and 19% of the Entertainment labelled tweets were correctly classified while the rest were missed out by the classification model.

4.5 Discussion on performance Metrics

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

Metric	Description	Time
Crawl time per tweet	Time spent to crawl from Twitter	3.00
Classification time per tweet	Time spent to classify one record	0.12
Total processing time per tweet	Time spent to crawl a tweet from Twitter, classify the tweet and store it in Solr	9.00
Time to build classification model	Time spent to build training model	80.00

Table 12: Performance Metrics for classification with time in milliseconds

The time taken to train the classification model is the longest. Despite having to take around 0.8 second to construct the model, it only needs to be train once. After the model is trained, it will be loaded and stored in the system to help classify new tweets. The model will not be retained by the system once it is loaded. The time taken to crawl tweets from Twitter is significantly higher. Our system can approximately crawl one tweet every 3 milliseconds (333 tweets per second). Our dataset of 10,000 tweets will take approximately 30 seconds to crawl. Thus, if the user needs to crawl a huge amount of data, a substantial amount of time will be needed to do so.

The time taken for classifying tweets is generally shorter compared to the time taken for crawling tweets. Our system can approximately classify one tweet every 0.12 milliseconds (8333 tweets per second). Therefore, the scalability of our system can be high as the system is able to classify a huge number of tweets within a short period of time.

Our system takes approximately 9 milliseconds to crawl one tweet from Twitter, classify the tweet and stored it in Solr (111 tweets per second). With our current dataset of 10,000 tweets from 5 different news channels, it will only take approximately 90 seconds to complete the whole process. Despite having 1/3 of the time spent crawling the tweets, it is still substantially fast as data crawling only happens when the users are updating for new tweets.

4.6 Visualization for Classified Data

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

The categories of the tweets will be displayed to the user above the tweets. This allows the users to quickly identify the context of the tweets and apply filters accordingly.

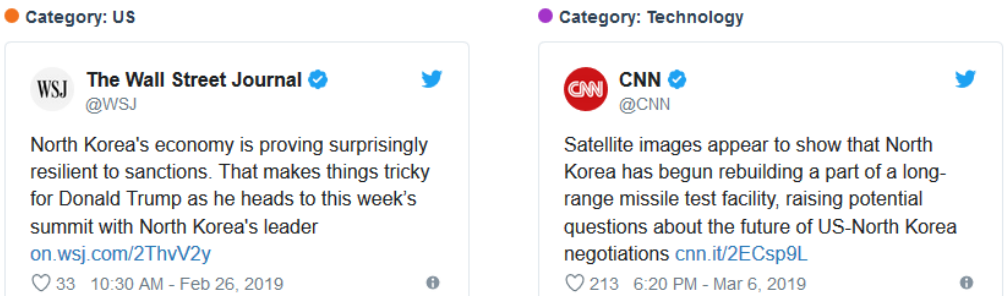

Figure 18: Categories of classified data will be displayed on top of the retrieved tweets

5 Enhancing Classification

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

After experimenting with the 6 different classification models, we wanted to improve on the classification models to get a higher accuracy. In the end, we can improve on the classification model for SVM. While doing so, we also tried out two different ensemble classification models, Random Forest and Gradient Boosting. The following section will explain the result of the newly tested classification models.

5.1 Ensemble (Random Forest)

One of the ensemble classifications used is Random Forest. Random Forest is a meta estimator that fits a handful of decision tree classifiers on different subsamples of the dataset and improves its accuracy with the use of averaging. For our multiclass problem, the accuracy score for Random Forest did improved when compared with the accuracy score for Decision Tree. However, it did not perform that well when compared to the other classification models.

	Precision	Recall	F1 Score	Accuracy
Business	0.80	0.28	0.41	
Entertainment	1.00	0.35	0.51	
Sport	0.53	0.90	0.67	
Technology	0.42	0.44	0.43	0.60
US	0.74	0.75	0.74	
World	0.53	0.42	0.47	
Weighted Average	0.64	0.60	0.58	

Table 13: Performance Metrics for Random Forest

	Precision	Recall	F1-Score	Accuracy
Decision Tree	0.55	0.55	0.54	0.55
Random Forest	**0.64**	**0.60**	**0.58**	**0.60**

Table 14: Comparison between Decision Tree and Random Forest

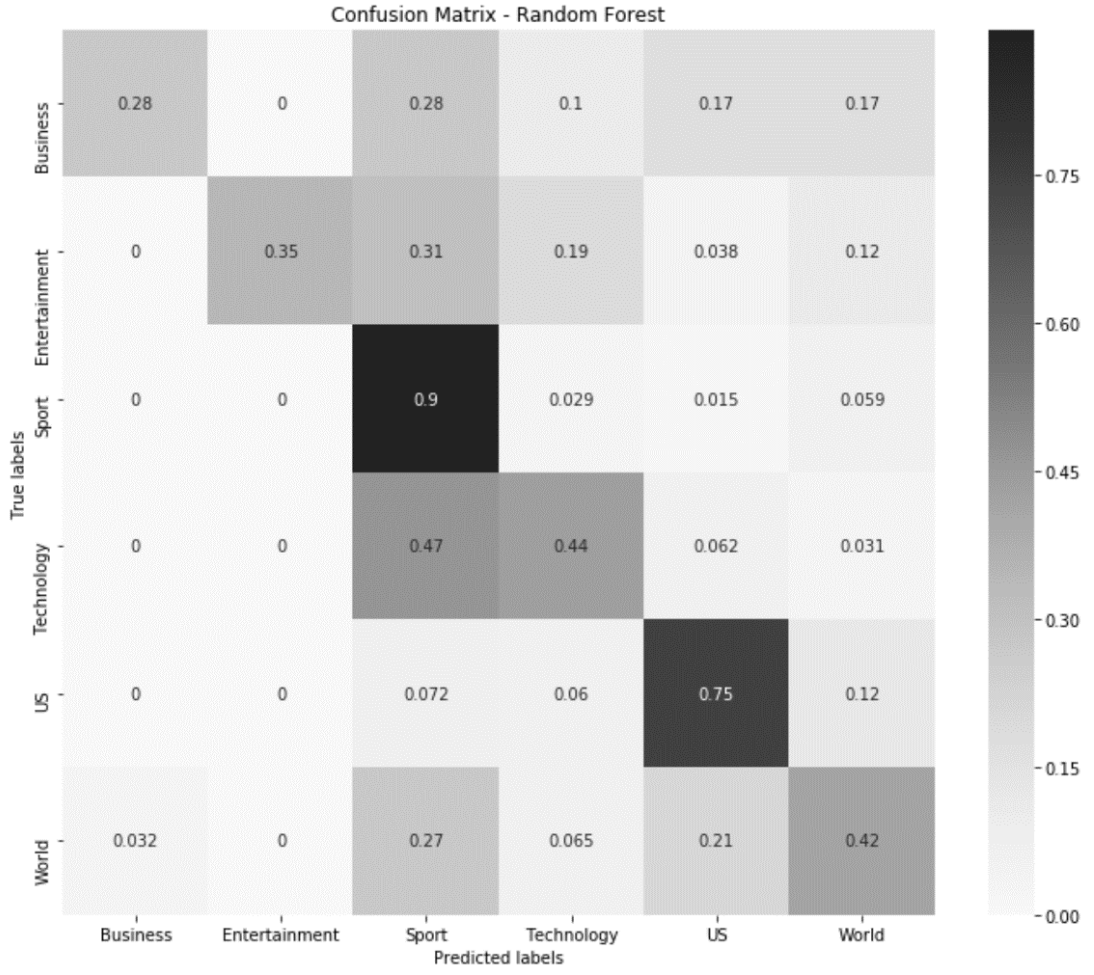

Figure 19: Random Forest Confusion Matrix

5.2 Ensemble (Gradient Boosting)

Another ensemble classification model is Gradient Boosting. Gradient Boosting builds an additive model in a stage-wise fashion like other boosting methods and allows optimization of arbitrary differentiable loss function. It also produces a prediction model based on weaks prediction models such as decision trees. Overall, this classification model is not the worst as compared to Decision Tree, but it is also not the best classification we have.

	Precision	Recall	F1 Score	Accuracy
Business	0.33	0.24	0.28	
Entertainment	0.57	0.50	0.53	
Sport	0.54	0.76	0.63	
Technology	0.42	0.31	0.36	0.57
US	0.62	0.73	0.67	
World	0.73	0.44	0.55	
Weighted Average	0.57	0.57	0.55	

Table 15: Performance Metrics for Gradient Boosting

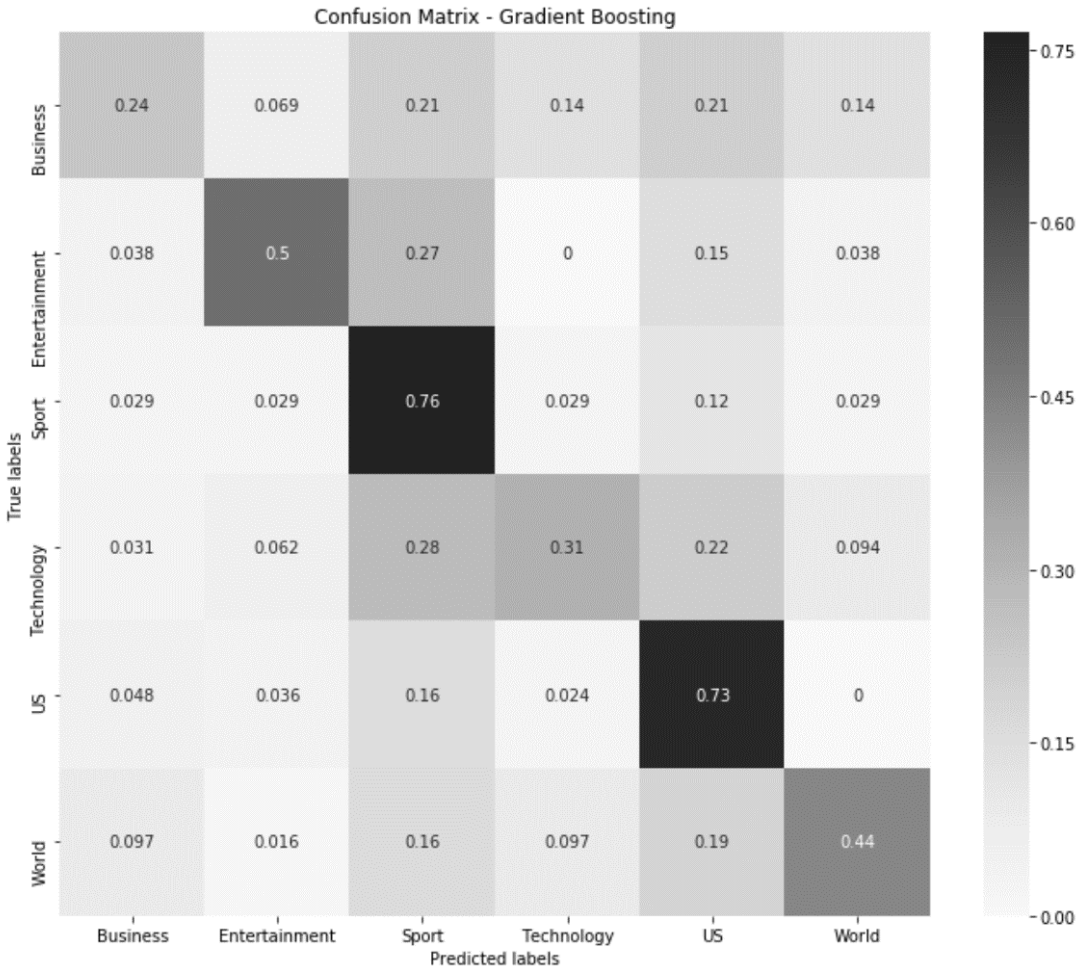

Figure 20: Gradient Boosting Confusion Matrix

5.3 OneVsRest (Linear Support Vector Machine)

At first, we used the Support Vector Machine classification model of the SVC class which uses OneVsOneClassifier. OneVsOneClassifier works by constructing each classifier with a pair of classes. During prediction, the class which received the most vote gets selected. If there is an equal number of votes for both classes, it will select the class with the higher aggregate classification confidence by summing both the confidence level computed by the underlying binary classifier.

OneVsRestClassifier works by fitting one classifier per class. For each classifier, the class is fitted against all the other classes. With the advantage of interpretability, it is possible to gain knowledge about the class based on its corresponding classifier.

In the end, we achieved a better result from Linear Support Vector Machine with OneVsRestClassifier as compared to Support Vector Machine with OneVsOneClassifier. Therefore, we decided to change the strategy from OneVsOneClassifier to OneVsRestClassifier which uses LinearSVC class.

Compared to the rest of the classification models, Linear Support Vector Machine achieved the highest recall score, F1-score and accuracy, which is why we have chosen to use this classification model in our web application.

	Precision	Recall	F1 Score	Accuracy
Business	0.57	0.45	0.50	
Entertainment	0.70	0.62	0.65	
Sport	0.94	0.87	0.90	
Technology	0.78	0.56	0.65	0.73
US	0.66	0.88	0.75	
World	0.68	0.63	0.66	
Weighted Average	0.73	0.73	0.72	

Table 16: Performance Metrics for OneVsRest (Linear Support Vector Machine)

	Precision	Recall	F1-Score	Accuracy
Support Vector Machine	0.76	0.62	0.58	0.62
Linear Support Vector Machine	**0.73**	**0.73**	**0.72**	**0.73**

Table 17: Comparison between Support Vector Machine and Linear Support Vector Machine

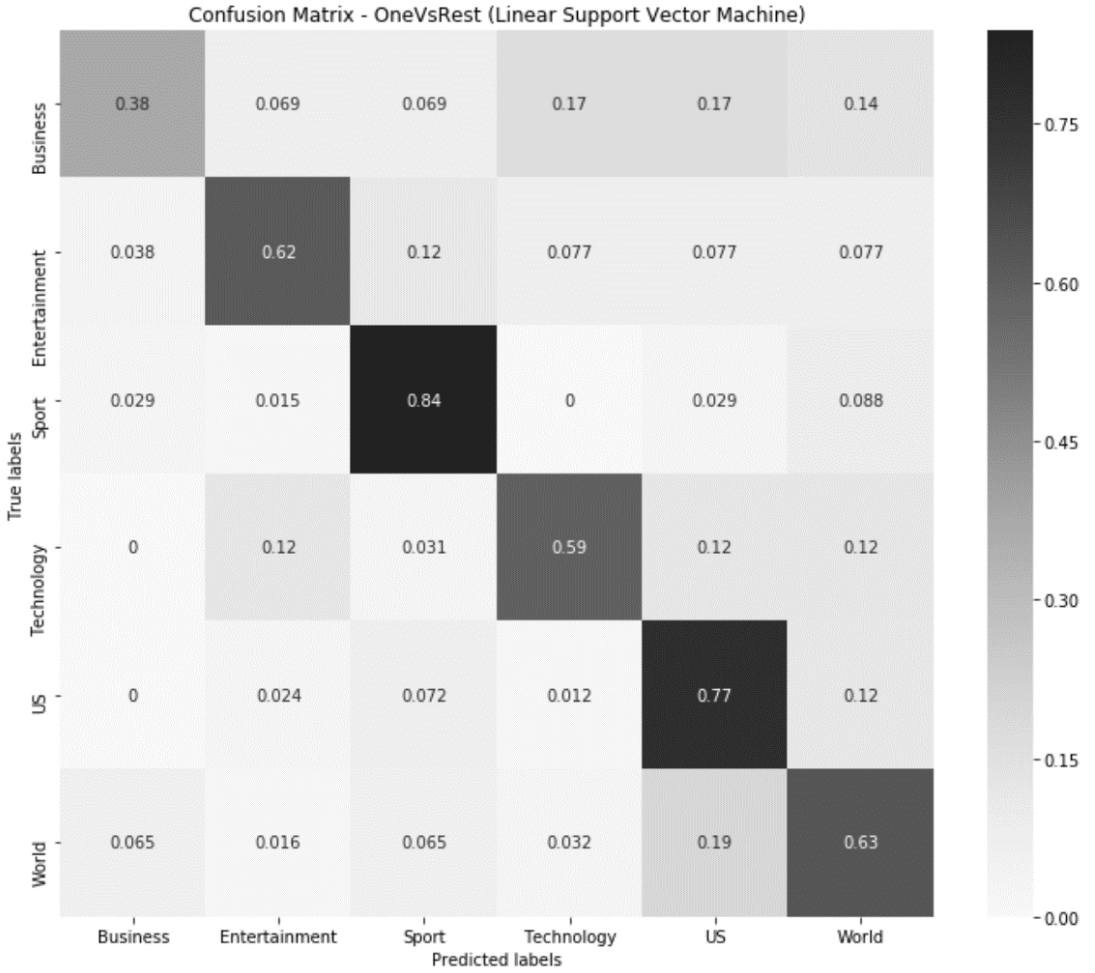

Figure 21: Linear Support Vector Machine Confusion Matrix

5.4 Conclusion

As seen from the following table, Linear Support Vector Machine had the highest performance out of not only the 3 classifiers we tested in this section, but also highest amongst all the 9 classifiers we trained. We went ahead and used this classifier to perform classification for the web application.

	Precision	Recall	F1-Score	Accuracy
Random Forest	0.64	0.60	0.58	0.60
Linear Support Vector Machine	**0.73**	**0.73**	**0.72**	**0.73**
Gradient Boosting	0.57	0.57	0.55	0.57

Table 18: Performance overview of the improved 3 models

6 Submission Files

Link to video: https://www.youtube.com/watch?v=jRqb7Z6IR5M

First folder contains data files: https://drive.google.com/open?id=1TPGqx_iZqzBM5iNt_FlKf_p_D8wThlUA

- Crawled text data: Crawled Tweets.xlsx
- 5 queries from question 2 and their results: /Queries/Query[1-5].json
- Manual classifications: Manual Classification.xlsx
- Automatic classification results: Automatic Classification Results.csv
- Folder containing pipeline used for initial indexing: /Pipeline

Second folder contains application files: https://drive.google.com/open?id=1t4EW7zC0DKENrbPGrXl8HF8FrcleJKEt

- Ready-to-run web application: /webapp
- Solr with initial 10,000 indexed tweets: /solr-7.7.1
- Folder structure for automated indexing: /Data
- Script for automated indexing: Updater.py
- Document with a guide on how to start the applications: Readme.pdf