AN APPROACH FOR CATEGORIZING ONLINE NEWS ARTICLES IN TRINIDAD AND TOBAGO BASED ON THE GROUP BY CLASS CLASSIFICATION ALGORITHM

A Thesis
Submitted in Fulfilment of the Requirements for the Degree of
Masters of Science Computer Science - Cloud Technologies Specialization

of
The University of the West Indies

by
David Charles
2019

Department of Computing and Information Technology
Faculty of Science and Technology
St. Augustine Campus

This thesis entitled:
AN APPROACH FOR CATEGORIZING ONLINE NEWS ARTICLES IN TRINIDAD AND TOBAGO BASED ON THE GROUP BY CLASS CLASSIFICATION ALGORITHM
written by David Charles
has been approved for the Department of Computing and Information Technology

Mr.Person

Dr. Phaedra Mohammed

Dr. Place advisory member name here

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

ABSTRACT

An Approach for Categorizing Online News Articles In Trinidad and Tobago Based On The Group By Class Classification Algorithm

David Charles

Online news articles contain information that describes a range of topics relevant to socio-economic factors such as education, crime and culture. Extracting and analyzing the information within a large collection of news articles can be challenging for machines due to the unstructured representation of the data. Common attempts to make the information within these articles more accessible include: tagging, where users manually assign categorical tags to news articles, and/or the inclusion of text search, where articles that have the words specified in some search query are returned. The challenge with these approaches is that articles may not be consistently or efficiently annotated with meta data, and search queries based only on word matching may not always return desired results. In this project, the goal was to develop an extensible solution for improving information organization and retrieval of unstructured, unlabeled online articles. By achieving this goal new useful information can be extracted from these articles. Online news articles in Trinidad and Tobago made up the data set, and these articles had no category labels. Steps taken to achieve the goal included: developing a text classification model for categorizing articles, choosing a method for storing and retrieving articles, and developing an API (application programming interface) for interfacing with the article database and text classification model. The contributions of this research includes a new machine learning algorithm called the: Group By Class(GBC) classification algorithm based on the vector space model, the GBC module that implements the algorithm, and an API that utilizes the GBC module and other modules for classifying, retrieving, and storing articles. A Web application was also developed to illustrate some of the main functionality provided by the API.

Keywords:Text Classification Model, Vector Space Model, Incremental learning, Online News Articles.

Acknowledgements

Place acknowledgements here

Dedication

Write dedication here.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

In this digital age there are large amounts of news articles created each day, most of which are electronic in nature (Mouine et al. 2016). Electronic news articles, also called online articles, contain much information about a wide variety of topics which generally correspond to the standard topics specified by the International Press Telecommunication Council(IPTC). The IPTC, one of the main standardization bodies in the journalism domain is an international consortium of newspaper distributors, editors, and news agencies (Bacan, Pandzic, and Gulija 2005). The IPTC topic categories are often used for tagging and organizing articles however this task can be difficult to achieve manually. Manual classification is done by a human annotator who interprets and categorizes the content. Although quality results are usually achieved, this approach quickly becomes very expensive and time-consuming due to the vast amount of online news articles that can be created each day in comparison to limited amount of manpower. There are many approaches for automatic text classification. One approach is the use of machine learning (ML) which is capable of automatically learning and improving from experience without explicitly being programmed. For classifying articles using ML, a classification ML algorithm is chosen and then pre-labeled content(e.g. articles) are used by the ML algorithm to produce a trained model. The trained model is then used for classifying new content by making predictions of which tags are associated with particular articles. Some of the state of the art machine learning approaches for generating text classification models includes Naive Bayes Algorithms, Support Vector Machines, and Artificial Neural Networks. These are further specified as supervised machine learning since they require labeled data for training. Each of these approaches have varying features and limitations (Nikam 2015), however they all suffer from making poor predictions when faced with completely new data. A solution would be for the model to have the capability of learning from new data it encounters. The ability of a model to have new data be added to it without having to be retrained on the entire data set is called *incremental learning*. There are many examples in the research literature that demonstrates ways in which models based on Naive Bayes (Ren, Lian, and Zou 2014), Support Vector Machines(Laskov et al. 2006), and Artificial Neural Networks(Polikar et al. 2001) can support incremental learning. Although the implementation of these algorithms from scratch is possible, it is often preferred by developers to utilize a standard library that already provide most of the desired functionality. One such library is the python library "SciKit-Learn". SciKit-Learn is used to build machine learning models, and provides support for classification algorithms such as: Naive Bayes and Support Vector Machines. SciKit-learn, however, has yet to implement the incremental learning strategies presented by the research community.

The goal of this project was to develop an extensible solution for improving information organization and retrieval of unstructured, unlabeled online articles in Trinidad and Tobago. The solution proposed included the generation and usage of a text classification model for categorizing the online articles. The initial text classification model was trained using online news articles from Trinidad and Tobago. These articles were selected by random with the only requirement being that each article contain at least two words from one of the categories of the IPTC topics list. To make the solution extensible, a machine learning classification algorithm that supports incremental learning had to be used. In this project a novel algorithm called the Group By Class (GBC) algorithm was created. It is based on the vector space model and was used to generate the classification model. This algorithm was created for the purpose of simplifying the process of incremental learning and its accuracy, precision and recall were compared with the Naive Bayes algorithm with favourable results. Models generated using the GBC Algorithm provide an easy, intuitive approach for incremental learning. In this project a python module called the GBC module was developed. It was used for generating text classification models that easily support incremental learning. To illustrate the incremental learning functionality as well as the benefits of automated article categorization, a RESTful application programming interface (API) was developed along with a web application called the GBC System. The web application via the use of the API, allow a user to classify articles, as well as retrieve and filter by category and query string.

1.2 Background and Problem Definition

Manual classification of news articles can be time consuming and costly. An automated solution can assist annotators with the classification of these articles. Articles can contain both geographical and categorical information and hence can be used to get categorical information about a given location, this is called geographical information retrieval (GIR) (Hu, Ye, and Shaw 2017). GIR using news articles is possible if the articles are first categorized which however is a difficult task to do manually for a large data set of articles. There exists solutions for document classification however the classification of news articles is a multi-label text classification problem. Existing solutions requires a lot of effort for implementing multi-label text classification. In most of the state of the art solutions for document classification, a model is trained only once and then used for classifying new documents. However as time passes, the model's classification performance can decrease because it can't adapt to changing data. A solution to this problem is incremental learning. However implementing incremental learning using current state of the art solution is not a simple task and requires much effort.

1.3 Project Goal and Objectives

The goal of this project was to develop an extensible solution for improving information organization and retrieval of unstructured, unlabeled online articles written in Trinidad and Tobago. This goal was achieved by training a text classification model and using it as part of a system called the GBC System. The GBC System organizes articles by automatically assigning categorical labels to them. It also allows for articles to be retrieved from a database and filtered by category or query string(text within the article). The solution is also extensible due to the modularity of the system and the capability of the GBC model to learn incrementally. The project objectives included:

- Developing a text classification model that allows for easy implementation of Incremental learning and Multi-Label Text Classification.
- Developing an approach for gathering online news articles which would be used as training data.
- Comparing the performance of the GBC classifier with one of the state of the art classifiers: Naive Bayes.
- Developing a system (GBC System) through which the classification services and all other services and functionality can be explored by the user.

1.4 Project Significance

The first significance of this project is that the classification of online news articles is a Document Classification project. By providing a solution for automated classification of online news articles, the same solution or slight variations of it can be applied to other document classification problems such as spam filtering and sentiment analysis. Another significance of this project is the fact that online news articles provide a wealth of information within its unstructured content. By developing a system that would make online articles more suited for analysis, new opportunities such as city network research (Hu, Ye, and Shaw 2017) can be introduced that may result in improving one's understanding of the happenings in their society and also can influence the decisions made by the collective society. In addition to providing a solution to the document classification problem, the novel text classification model presented in this project also tackles the problem of *incremental learning* which is the ability of a model to improve without it having to be retrained on old training data (RR Ade and Deshmukh 2013), and *multi-label text classification*, which is the task of classifying a document with multiple categories or labels (Sucar et al. 2014).

1.5 Thesis Organization

The remainder of this thesis is organized as follows:

- Chapter 2 contains literature reviews on automated classification of articles, and the Naive Bayes classifier with focus on incremental learning, multilabel classification, and performance.

- Chapter 3 outlines the overall GBC system architecture and also provides detailed descriptions for the suite of GBC algorithms for training, performing classification and incremental learning.

- Chapter 4 presents the various modules and technologies developed and used for implementing the GBC system solution.

- Chapter 5 details the tests and experiments that were done, along with a discussion of the results obtained.

- This thesis concludes in Chapter 6 with discussions on the GBC system, the GBC classification model, limitations, and future work.

Chapter 2

Literature Review

2.1 Automated Classification of Articles

The two major benefits of automated classification of news articles includes the potential information that can be obtained through analysis of these categorized articles and the reduced labour of having to categorize the articles manually. Ee and Lim (2001) developed a system called 'Categorizer' which allowed users to create/specify their own personalized category names and a set of keywords for each personalized category describing the content of the category. After the user provides the categories and keywords; Categorizer would use the information provided by the user to train a classification model using the support vector machine (Jakkula 2006) algorithm and news articles from the Reuters dataset (Lewis 1987). The user would then be able to request news under their new category and then most recently downloaded news from the Reuters news database would be retrieved, both the personalized categories and the document vectors representing the news articles gets passed to the support vector machine classifier. The classifier returns the results sorted by score values in HTML format that gets displayed on the web page. The intended users of 'Categorizer' were individuals interested in browsing and simply reading news articles that they have specific interest in. The creators of 'Categorizer' retrained the model from scratch each time new data was added to the data set.

Reis et al. (2019) developed an approach for detecting fake news articles. They trained multiple models using news articles which they had manually labeled as 'fake' and 'not fake'. The models they trained and compared included k-Nearest Neighbors (KNN), Naive Bayes (NB), Random Forests (RF), Support Vector Machine with RBF kernel(SVM), and XGBoost (XGB). The best results they obtained was from RF and XGB classifiers which statistically tied with 85% and 86% accuracy respectively. In their research Reis et al. (2019) stated that fact checkers can use their solution as an auxiliary tool for automatic fake news detection thereby helping with identifying content that is more likely to be fake.

Hu, Ye, and Shaw (2017) in their research, developed a computational framework for extracting and analyzing semantic relatedness between cities using news articles. According to the authors news articles cover public opinions, socio-economic activities, human interests, and culture that exist only within people perceptions. The authors stated that extracting semantic relatedness from news articles can allow individuals to understand the relationships among cities which can be beneficial for city network research. To achieve their goals, Hu, Ye, and Shaw (2017) trained a Labeled Latent Dirichlet Allocation (LLDA) model (Ramage et al. 2009). The dataset the researchers used to train the model included news articles retrieved from the Guardian [1] using its API. The Guardian is a newspaper organization that publishes news articles about the world . The labels that were assigned to the training dataset was based on tags that were assigned by news producers from the Guardian and the IPTC topics (Bacan, Pandzic, and Gulija 2005). After the model was trained on all the articles, it was applied to articles that contain city information. These included articles that contained co-occurrence of multiple cities, and articles that contained only individual cities. The results of their research indicated that human interests (such as the article categories) can link cities together even if the cities are far away. Hu, Ye, and Shaw (2017) explained that their

1. https://www.theguardian.com

proposed framework can be useful in city network research for understanding relations between cities. They also stated that their framework could be used for exploration of the evolution of the semantic relatedness of some ancient cities.

2.2 Naive Bayes Classifier Background And Performance

The Naive Bayes Classifier is based on a family of statistical algorithms that can be used for document classification. Variation of the Naive Bayes classifier models include: the Gaussian Naive Bayes, Bernoulli Naive Bayes and Multinomial Naive Bayes (McCallum, Nigam, et al. 1998). The Naive Bayes classifier is a popular approach used for classification and there are existing implementations available in libraries such as the Scikit-learn library (Pedregosa et al. 2011). This makes these classifiers easy to use without having to implement algorithms from scratch. The Naive Bayes model most commonly used for document classification is the Multinomial Naive Bayes. McCal-lum, Nigam, et al. (1998) explain that in the multinomial model, a text document is treated as a sequence of ordered word events that are taken from the same vocabulary with the assumption that document lengths are independent of category/class. All of the Naive Bayes classifiers are based on the Bayes Theorem which finds the probability that an event will occur given the probability of an event that has already occurred. The mathematical formulation for Bayes Theorem is given below:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

The Naive Bayes Classifiers performance have been tested and investigated by many researchers. Androutsopoulos et al. (2000) utilized Naive Bayes for performing anti-spam filtering with personal e-mail messages as their dataset. Their findings indicated that the Naive Bayesian filter by far outperformed a keyword-based filter even with little training data (very small corpora). Research presented by Abbas et al. (2019) used the multinomial Naive Bayes(MNB) classification algorithm for sentiment analysis. The classifier was used for categorizing movie reviews based on overall sentiment (positive/negative) each with a negative or positive sentiment label. They stated that significant results in text categorization performance was achieved with the multinomial Naive Bayes(MNB).

2.3 Naive Bayes Multi-Label Learning and Classification

In multi-label classification the aim is to correctly assign unseen instances (documents) to sets of different labels/classes (Sucar et al. 2014). While for multi-label learning the training set comprises of instances(documents)that each has association with a set of categories (Zhang, Peña, and Robles 2009). The multinomial Naive Bayes model is capable of Multi-label classification and learning. Sucar et al. (2014) did research involving multi-label classification with Bayesian network. In their research they indicated that the task of multi-label classification has often been addressed by either creating a compound class variable with all possible combinations of categories/labels (label power-set methods) or by training independent classifiers for each category (binary relevance methods). They also highlighted the short comings of the two methods stating that 'label power-set methods' tend to be more computationally complex, while binary relevance methods ignores dependencies among the different classes/categories. They mentioned that Chain classifiers were a solution that was recently introduced to address the short comings of the two approaches to multi-label classification. With Chain classification each classifier in the chain learns and predicts the label of one class given the attributes and all the predictions of the previous classifiers in the chain (Read et al. 2011). Hence to address these short comings for multi-label classification, they presented a new approach for chaining Bayesian classifiers by combining strengths of classifier chains and Bayesian networks. Bayesian networks and Naive Bayes models are not the same thing. Bayesian networks are graphical models representing sets of variables and their conditional dependencies (all the dependencies are modeled) whereas in a Naive Bayes model all the features are conditionally independent of each other. Bayesian networks are often used as a means of improving Naive Bayes for classification (Jiang et al. 2007).

2.4 Naive Bayes Classifier and Incremental Learning

According to RR Adeand Deshmukh (2013) incremental learning is a machine learning paradigm whereby new samples of training data are continuously used for extending an existing model's knowledge without retraining the model on the entire dataset. RR Adeand Deshmukh (2013) stated that an incremental learning algorithm can be defined as one that satisfies the following: “it will preserve previously acquired knowledge”, “it does not require access to the original data”, “it should be dynamic in nature with the changing environment”, “it should be capable of generating new classes or clusters when required” and “it should be able to learn and update with every new data-labeled or unlabeled”. Read et al. (2012) stated in their paper that incremental learning can be divided into two main approaches: batch incremental approaches which gather examples in batches to train models; and instance incremental approaches that learn from each example as it arrives. They also provided an in-depth analysis comparing both approaches. Zhong et al. (2017) stated that as time goes on, the data distribution and data features of a dataset changes and hence the classification model that is trained based on historical data may no longer apply to some new data. Hence there is an important need for classification models to support incremental learning. Ren, Lian, and Zou (2014) highlight problems with three kinds of incremental Naive Bayesian classification algorithms and in order to overcome the problems, they presented the concept of Classification Contribution Degree. The results of their research demonstrated that their new algorithm proposed had better data completeness along with higher classification performance.

Ade and Deshmukh (2014) stated that the Naive Bayes algorithm is usually used for batch learning, because the algorithm performed poorly when each training sample was handled separately. In batch learning the algorithm performs the majority of its computation on the accumulated information on all of the training instance(the batch of training instances). It does not perform the majority of its computations after observing each training instance. Incremental learning with Naive Bayes though possible is not commonly implemented (Alcobé 2004). Alcobe (2004) in their research they stated that there were few works on incremental Bayesian network learning. Kotsiantis (2013) attempted to increase prediction accuracy of an incremental version of Naive Bayes model using instance based learning. Instance based learning is a family of machine learning algorithms that compare new instances with instances already seen in training which have been stored in memory.With instance based learning the stored training instances are stored in persistent storage like a database. These training instance represents the model itself. When new data has to be classified, training instances are looked up in storage for instance that most closely resembles the new data/instance. The limitations of instance based learning is that a lot of memory is usually required and making predictions is computationally intensive (Aha, Kibler, and Albert 1991). Incremental learning in the Naive Bayes classifier is not a simple task as indicated by the variations and amount of research that are done and still are being done on the topic.

Chapter 3

System Architecture

The goal of this project was to develop an extensible solution for improving information organization and retrieval of unstructured, unlabeled online articles. To achieve this goal the GBC system was developed to automate the classification of online articles and to provide a means by which articles can be retrieved and filtered by either the article content or category. The system allows users to submit online news articles via either plain text or the URL of the site containing the article. Upon submission, the plain text or content located at the URL is categorized and the results are displayed for the user. The system is designed to ensure that only online articles in Trinidad and Tobago are added to the database on submission. The system provides an API that upon submission of new data, returns meta data on all of the categories related to the article, the top three categories, all the terms/words which were considered for categorization, and the category confidence scores for each category. To make the solution extensible the novel GBC text classification model that supports incremental learning was trained and used by the GBC System.

3.1 The Model View Controller Architecture

The GBC system design, shown in Figure 3.1, was based on the Model View Controller Architecture(MVC) (Leff and Rayfield 2001). The 3 main components in the GBC system design are: the Architecture View, Architecture Model, and Architecture Controller. The phrase 'Architecture Model' would be used when referring to the model of the MVC architecture where as the phrase 'GBC model' would be used when referring to the GBC text classification model that was trained and used by the GBC system.

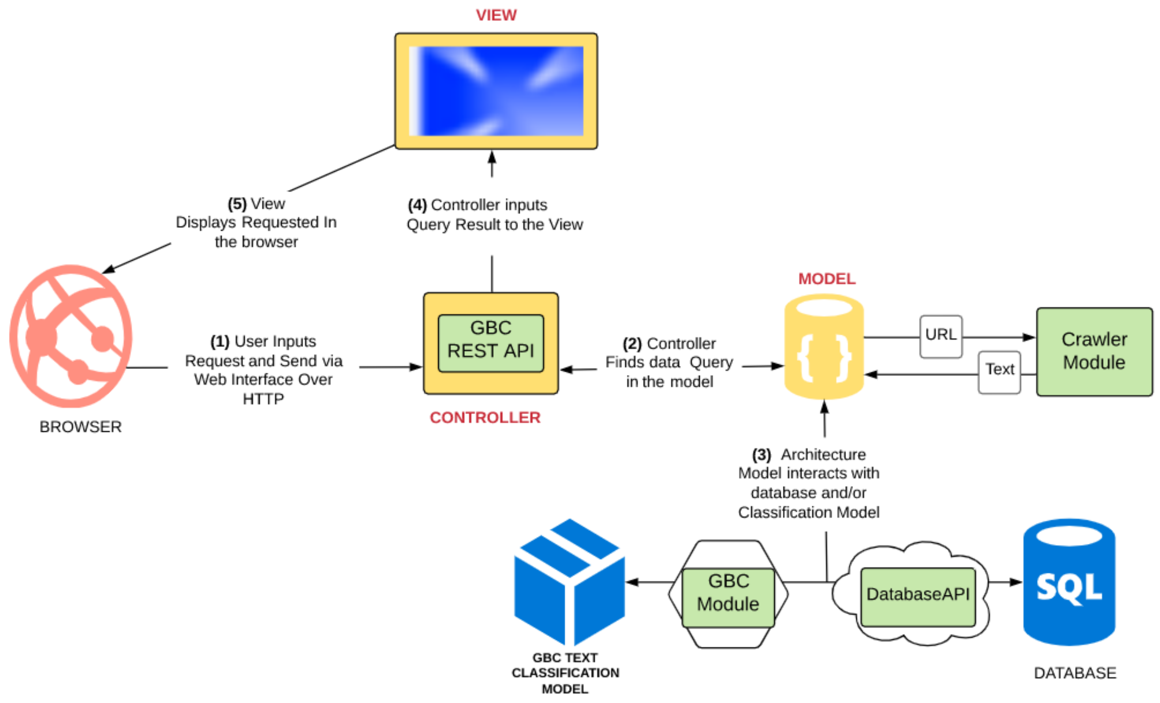

VIEW
(5) View Displays Requested In the browser
(4) Controller inputs Query Result to the View
MODEL
BROWSER
(1) User Inputs Request and Send via Web Interface Over HTTP
GBC REST API
CONTROLLER
(2) Controller Finds data Query in the model
URL
Text
Crawler Module
(3) Architecture Model interacts with database and/or Classification Model
GBC Module
DatabaseAPI
SQL
GBC TEXT CLASSIFICATION MODEL
DATABASE

Figure 3.1: GBC System MVC Architecture

Other components, shown in Figure 3.1, used by the GBC system include: the 'Crawler module' which was used by the Architecture Controller for extracting content from a URL, the 'DatabaseAPI' used by the Architecture Model for interacting with the database, and the 'GBC module' used by the Architecture Model for interacting with the GBC model. The Architecture Controller and the Architecture View interacts with each other via the 'GBC REST API' provided by the Architecture Controller.

3.2 Architecture Model

In MVC, the Architecture Model encapsulates the data and business/domain logic. All computation for accessing data that may or may not be stored in some form of persistent storage is done by the model. In the GBC system, the model handles requests coming from the controller, and as illustrated in Figure 3.1, interacts with the database and GBC model to satisfy a request.

3.2.1 Architecture Model Interaction with Database

The Architecture Model interacts with a database through a RESTful Application Programming Interface called the DatabaseAPI which is illustrated in Figure 3.2. DatabaseAPI was developed to be utilized only by the model and allows the model to perform create, read, update, or delete operations on any table in the database via HTTP requests. The DatabaseAPI supports other functionality, based on Flask Restless (Maia 2015), which the model leverages when sending results to the controller. Flask-Restless, discussed further in Chapter 4, is an extension of the flask micro-framework that allows for the creation of RESTful JSON APIs. The addition or retrieval of articles from the database is done by the model which uses the DatabaseAPI for database interaction.

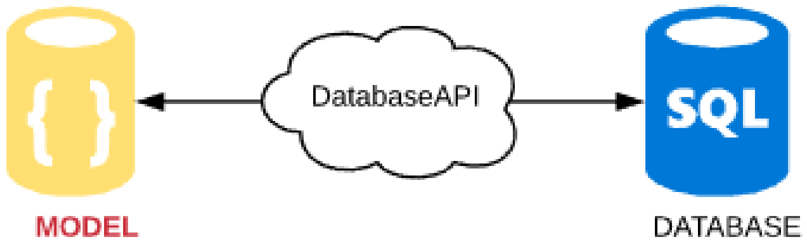


Figure 3.2: Interaction Between Architecture Model and Database via API

3.2.2 Architecture Model Interaction with Crawler Module

As discussed earlier, the model uses the DatabaseAPI to add article text to the database. However, because only content from online articles in Trinidad and Tobago are valid to be added to the database, the Architecture Model needs to first extract the information at the URL and then add the extracted content to the database. It does so using the Crawler module. Via the Architecture View the user can submit either a URL of an online article or actual textual information. This request is sent by the browser to the Architecture Controller which interprets and makes a request for the data from the Architecture Model. Any parameters the Architecture Model may need to satisfy the request are passed as well. The Architecture Model therefore can either receive plain text or URLs which may or may not be from an online news article. A check is done on the URL by the Architecture Model to ensure that only textual information from online articles in Trinidad and Tobago is added to the database. The URL therefore received by the Architecture Model is what determines whether or not the content at that URL is valid to be added to the database.

3.2.3 Architecture Model Interaction With GBC Model

The Architecture Model interacted with the GBC model through the use of the Group By Class (GBC) module illustrated in Figure 3.3. The GBC module implements the three (3) group by class algorithms described in section 3.5 for: training the GBC text classification model, performing classification using the GBC model, and updating the GBC model (incremental learning). Collected news articles were used as training data for training the GBC text classification model shown in Figure 3.1. After the GBC model was trained it was then added to the system to be utilized by the Architecture model. The two scenarios whereby the Architecture Model interacts with the GBC model are: for categorizing documents and updating the GBC text classification model (Incremental Learning). Though possible, the Architecture Model of the GBC system does not use the GBC module at any time to retrain the GBC model from scratch. It instead takes advantage of the incremental learning capability of the GBC model. Hence over time the GBC model can adapt to changing data as new data is added to the GBC system. The GBC model implemented by the GBC system is updated using an instance incremental approach (Read et al. 2012) whereby as new training samples enter the system the GBC model gets updated.

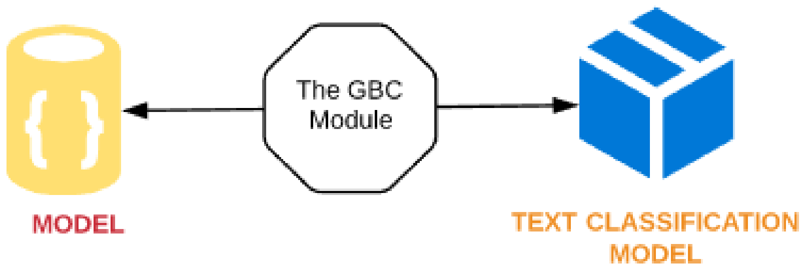


Figure 3.3: Architecture Model Interaction with GBC Model Via GBC Module

3.2.3.1 Categorizing Documents

The GBC model is stored on the server as a JSON file. With each request that requires the use of the GBC model, the GBC module is used by the Architecture Model to load the GBC model. The Architecture Model via the GBC module then uses the GBC model to categorize the textual content. More details on how the GBC module is used for performing classification is given in chapter 4 section 4.2.4.

3.2.3.2 Updating the GBC Model (Incremental Learning)

The system was built to support incremental learning. Incremental learning is an approach whereby a model can be updated with new label data without having to be retrained on past data (Ruping 2001). With online news articles, the style of writing and even the words used in articles on different topics can change over time. A model trained using current news articles can become less relevant over time due to the change in words associated with the various categories. Hence, one common approach is to retrain the model over time so that it can be up to date. Repeated retraining, however, will eventually become too expensive due to the increasing processing time and storage requirements that comes with the increasing size of the data set. Incremental learning mitigates this issue. In this project the classification model presented, allows for labeled data to be added to it without needing to be retrained on the entire data set. The algorithm and steps describing how incremental learning is achieved is further discussed in section 3.8.

3.3 Architecture Controller

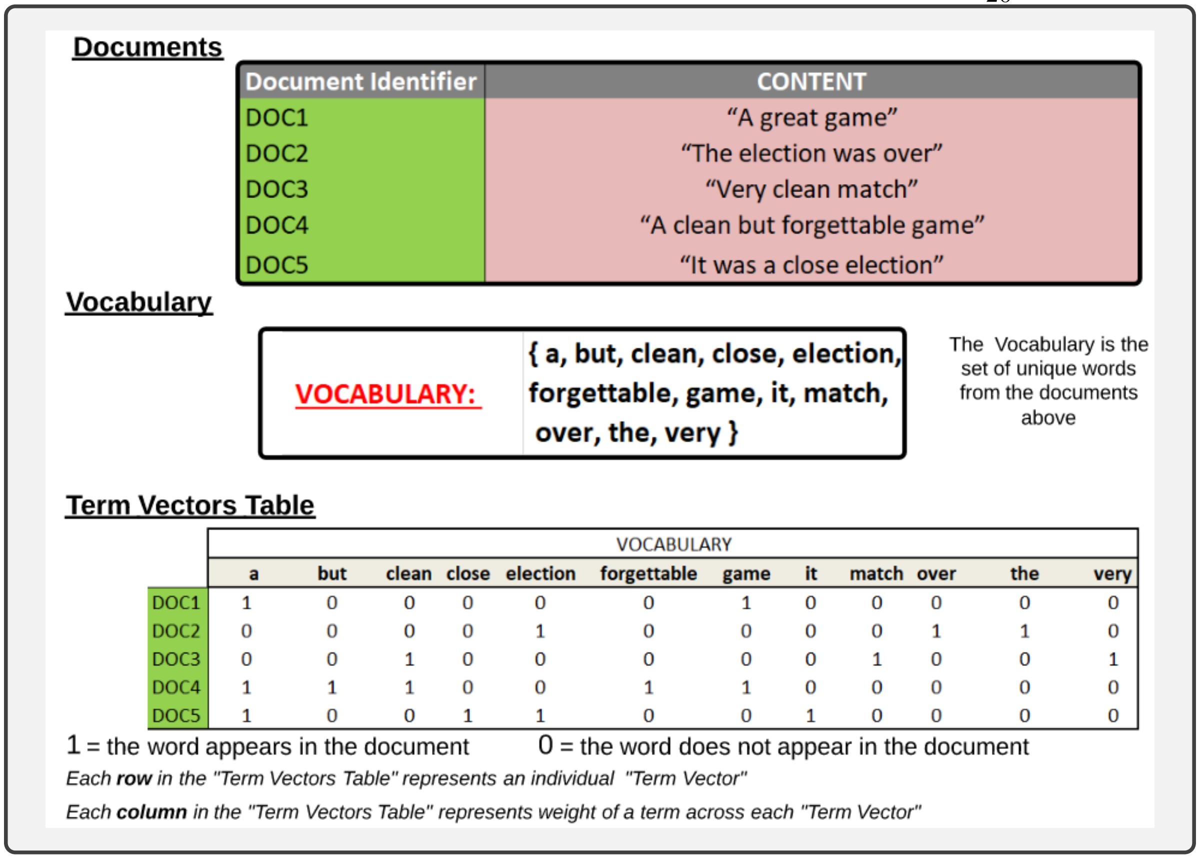
The Architecture Controller is the coordinator between the Architecture Model and the Architecture View. It receives requests from the user then invokes various CRUD(create, read, update, delete) operations on the model. In the MVC architecture illustrated in Figure 3.1, the Architecture Controller receives request from a web application running in the browser. The Architecture Controller then interacts with the Architecture Model to satisfy the data query request made by the Architecture View. A data query can be one of the CRUD operations. The Architecture Controller in Figure 3.1 provides a REST API by which the View sends requests and receives results. This API is called the “GBC REST API”. The main purpose of the GBC API is to allow users to interact with the classifier as a service. To illustrate the GBC API a web application was built that used it. The API can be accessed by other developers for developing their own applications.

3.4 Architecture View

The Architecture View in Figure 3.1, is a web application running in the browser. It provides a graphical user interface through which the user can interact with the GBC API. It is responsible for displaying results from the API and also allows the user to input and send requests using this API. Hence via the web application the GBC API was used for classifying articles, accessing the GBC text classification model, and for adding and retrieving articles from the database. Articles displayed in the view can be filtered by either category or query string of the article content. The web application via its dashboard also accepts a URL or textual content as input and returns the categories associated with the submitted. The top three categories are displayed in a pie chart, and all associated categories are displayed in a bar chart.

3.5 The Group By Class (GBC) Text Classification Model

The Group By Class (GBC) algorithm is a supervised Machine Learning(ML) algorithm that was developed for generating text classification models that supports incremental learning. Supervised ML algorithms require labeled training data to generate classification models which are then used to categorize or tag new unseen data. The GBC algorithm is based on the vector space model. The vector space model is an algebraic model that can be used for representing any object in general e.g. textual documents as vectors (Arguello 2013). An example of how textual documents can be represented as vectors is shown in Figure 3.4.



Documents

Document Identifier	CONTENT
DOC1	“A great game”
DOC2	“The election was over”
DOC3	“Very clean match”
DOC4	“A clean but forgettable game”
DOC5	“It was a close election”

Vocabulary

VOCABULARY: { a, but, clean, close, election, forgettable, game, it, match, over, the, very }

The Vocabulary is the set of unique words from the documents above

Term Vectors Table

	VOCABULARY											
	a	but	clean	close	election	forgettable	game	it	match	over	the	very
DOC1	1	0	0	0	0	0	1	0	0	0	0	0
DOC2	0	0	0	0	1	0	0	0	0	1	1	0
DOC3	0	0	1	0	0	0	0	0	1	0	0	1
DOC4	1	1	1	0	0	1	1	0	0	0	0	0
DOC5	1	0	0	1	1	0	0	1	0	0	0	0

1 = the word appears in the document   0 = the word does not appear in the document

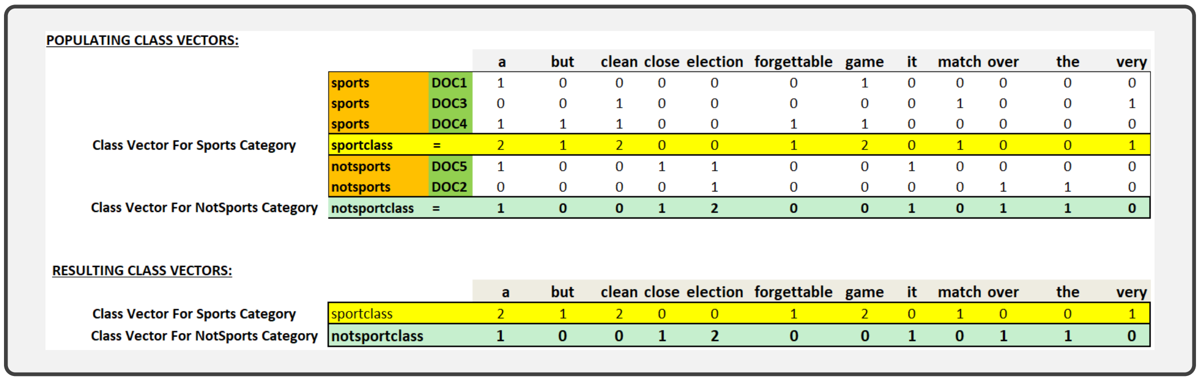
*Each **row** in the "Term Vectors Table" represents an individual "Term Vector"*

*Each **column** in the "Term Vectors Table" represents weight of a term across each "Term Vector"*

Figure 3.4: Term Vectors Binary Text Representation

In Figure 3.4 a vocabulary was created based on the unique words across all documents. Each document was then represented using a vector that records whether or not a word in the document was present in the vocabulary. The vectors generated are the binary text representation of the documents. These vectors are sometimes called term vectors (Arguello 2013) and are used as input to machine learning algorithms (Naresh and Sreepada 2019). In Figure 3.4, each row in the “Term Vectors Table” represent a term vector and its associated document.

A GBC model is a collection of class vectors. Class Vectors are similar to Term Vectors however they are meant to represent a group of documents in the same class or category rather than just a single document. The number of class vectors is equal to the number of labels involved in the training of the model, where each label has a corresponding class vector. Figure 3.5 illustrates the class vectors ‘sportclass’ and ‘notsportclass’ obtained using the sample data from Figure 3.6.



POPULATING CLASS VECTORS:

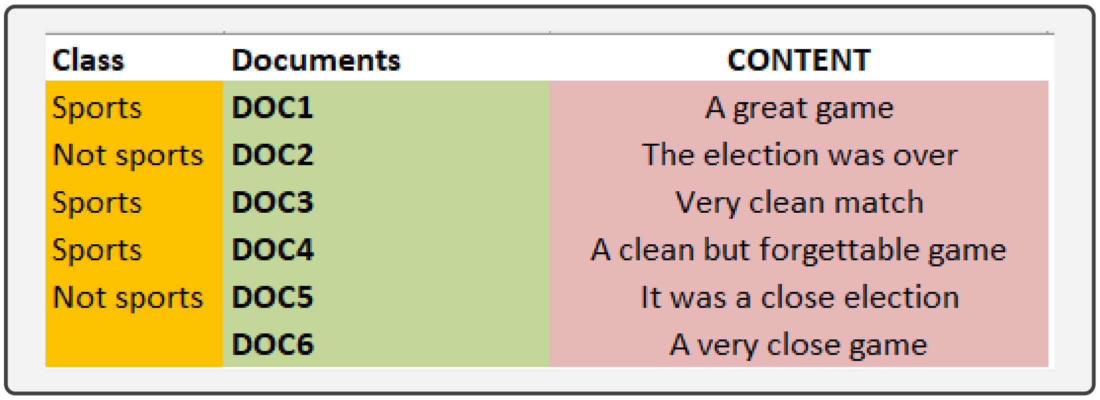
			a	but	clean	close	election	forgettable	game	it	match	over	the	very
	sports	DOC1	1	0	0	0	0	0	1	0	0	0	0	0
	sports	DOC3	0	0	1	0	0	0	0	0	1	0	0	1
	sports	DOC4	1	1	1	0	0	1	1	0	0	0	0	0
Class Vector For Sports Category	sportclass	=	2	1	2	0	0	1	2	0	1	0	0	1
	notsports	DOC5	1	0	0	1	1	0	0	1	0	0	0	0
	notsports	DOC2	0	0	0	0	1	0	0	0	0	1	1	0
Class Vector For NotSports Category	notsportclass	=	1	0	0	1	2	0	0	1	0	1	1	0

RESULTING CLASS VECTORS:

		a	but	clean	close	election	forgettable	game	it	match	over	the	very
Class Vector For Sports Category	sportclass	2	1	2	0	0	1	2	0	1	0	0	1
Class Vector For NotSports Category	notsportclass	1	0	0	1	2	0	0	1	0	1	1	0

Figure 3.5: Populated Class Vectors

The sample training data shown in Figure 3.6 will be used for illustrating the training steps involved in training a GBC model. The data shows the category and content for each document. The sixth document ‘DOC6’ in Figure 3.6 was purposely not assigned to any category.



Class	Documents	CONTENT
Sports	DOC1	A great game
Not sports	DOC2	The election was over
Sports	DOC3	Very clean match
Sports	DOC4	A clean but forgettable game
Not sports	DOC5	It was a close election
	DOC6	A very close game

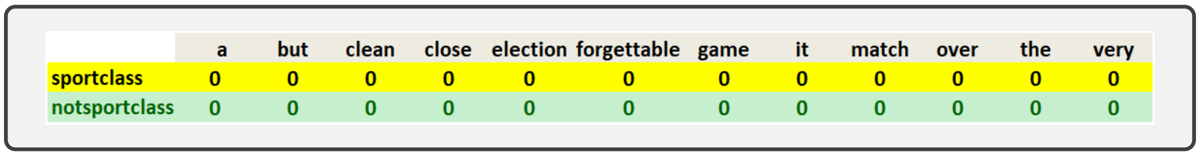
Figure 3.6: Documents and Categories

3.6 Group By Class Model (GBC) Training Steps

Training the GBC text classification model involves three major steps: initializing the class vectors, populating the class vectors, and standardizing the class vectors. At the end of the training steps multiple standardized class vectors would be generated, which together makes up the text classification model.

3.6.1 (STEP 1) Initializing the Class Vectors

In Figure 3.7 there are only two class vectors corresponding to the given labels in this example. The class vectors are initialized based on the vocabulary derived from the label documents in Figure 3.6, and the categories associated with the documents. All of the values within the class vectors are set to zero, because no term vectors representing respective documents have been merged with any of the class vectors.



	a	but	clean	close	election	forgettable	game	it	match	over	the	very
sportclass	0	0	0	0	0	0	0	0	0	0	0	0
notsportclass	0	0	0	0	0	0	0	0	0	0	0	0

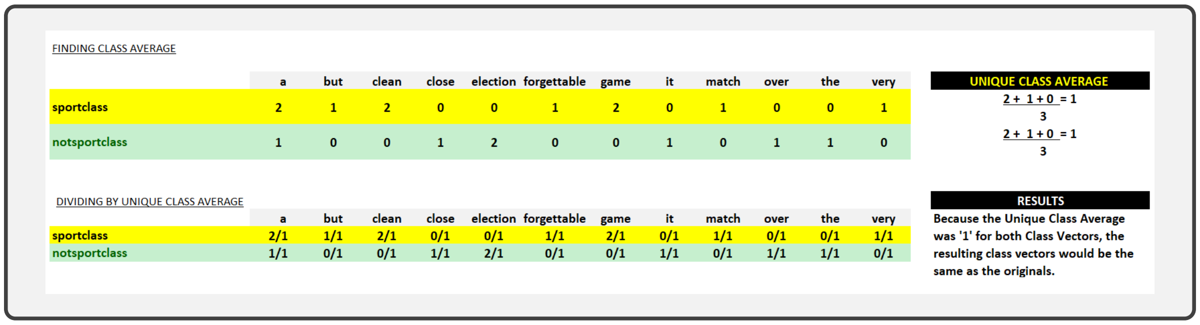
Figure 3.7: Initialized Class Vectors

3.6.2 (STEP 2) Populating The Class Vectors

The Class Vectors are populated by merging each document's term vector with its corresponding class vector as illustrated in Figure 3.5. As shown in the figure, merging term vectors involves adding the corresponding weights across all of the term vectors and the resulting sum is represented in the class vector. In Figure 3.5 the two class vectors "sportclass" and "notsportclass" are populated. Note that in this step if a document was labeled with multiple categories then it would be merged with multiple class vectors corresponding to each of those categories, this step is hence called the multi-label learning step (Zhang, Peña, and Robles 2009).

3.6.3 (STEP 3) Standardizing Class Vectors

There are two main steps involved in standardizing a class vector. Firstly the average of the term weights in the class vector is found. Only unique weights are used to find this average because term weights in a class vector that have the same weight values are considered to be of the same value. This average is called the "unique class average". Secondly, after the unique class average is calculated for a given class vector, each term weight in the class vector is divided by the unique class average for that class vector. The resulting class vectors are now standardized and can be used for classifying documents. The Class Vectors produced together make up the text classification model. The standardizing steps are illustrated in Figure 3.8. If there were significantly more documents categorized as "sports" versus documents categorized as "not sports" then the class vector trained on the largest set of documents will always be chosen as the most relevant class for any new document being categorized and vice versa. This is why it was necessary to standardize the class vectors. The process of standardizing the class



FINDING CLASS AVERAGE

	a	but	clean	close	election	forgettable	game	it	match	over	the	very
sportclass	2	1	2	0	0	1	2	0	1	0	0	1
notsportclass	1	0	0	1	2	0	0	1	0	1	1	0

UNIQUE CLASS AVERAGE

$\frac{2+1+0}{3} = 1$

$\frac{2+1+0}{3} = 1$

DIVIDING BY UNIQUE CLASS AVERAGE

	a	but	clean	close	election	forgettable	game	it	match	over	the	very
sportclass	2/1	1/1	2/1	0/1	0/1	1/1	2/1	0/1	1/1	0/1	0/1	1/1
notsportclass	1/1	0/1	0/1	1/1	2/1	0/1	0/1	1/1	0/1	1/1	1/1	0/1

RESULTS

Because the Unique Class Average was '1' for both Class Vectors, the resulting class vectors would be the same as the originals.

Figure 3.8: Standardized Class Vectors

vector is also referred to as feature scaling (Bollegala 2017). In this project dividing by the 'unique class average' was the approach explored for feature scaling. This approach was used because of its simplicity and the favourable results which were obtained while experimenting with this technique. Other feature normalization (Singh, Verma, and Thoke 2015) approaches will be explored in the future.

3.6.4 GBC Algorithm For Training The Text Classification Model

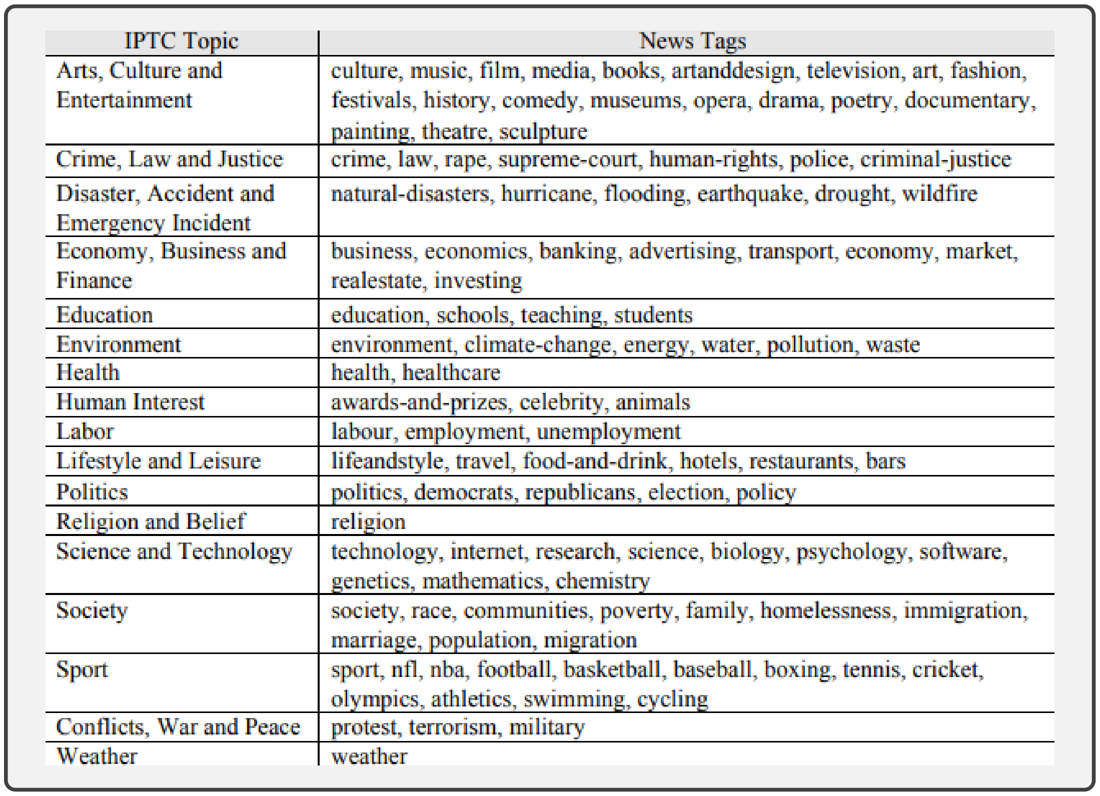
GBC Algorithm 1 below combines the three steps for training the text classification model.

Algorithm 1 Group By Class (*GBC*) Algorithm For Training GBC Model

```
1: Initialize:
       numC = number of categories
       numD = number of documents
       $L = \{l_1, ..., \mathrm{l}_{\mathrm{numC}}\}$, where L is the set of labels/categories
       $C = \{\vec{\mathbf{c}}_1, ..., \vec{\mathbf{c}}_{\mathrm{numC}}\}$, where C is the set of class vectors
       $D = \{d_1, ..., \mathrm{d}_{\mathrm{numD}}\}$, where D is the set of labeled documents
2: $V \leftarrow$ GetVocab($D$), where V is all unique words from set D
3: for $d_i \in D$ do
4:     $DocumentLabels \leftarrow$ GetDocumentLabels($d_i$)
5:     for $l_j \in L$ do
6:         if $l_j \in DocumentLabels$ then
7:             $\vec{\mathbf{t}} \leftarrow$ ConvertDocumentToBinaryTermVector($d_i$ , $V$)
8:             $\vec{\mathbf{c}}_{\mathrm{j}} \leftarrow$ MergeClassVectorWithTermVector($\vec{\mathbf{c}}_{\mathrm{j}}$, $\vec{\mathbf{t}}$)
9:         end if
10:    end for
11: end for
12: for $\vec{\mathbf{c}}_{\mathrm{j}} \in C$ do
13:    if NotEmptyClassVector($\vec{\mathbf{c}}_{\mathrm{j}}$) then
14:        $AvgUQ \leftarrow$ GetUniqueClassAverage($\vec{\mathbf{c}}_{\mathrm{j}}$)
15:        $\vec{\mathbf{c}}_{\mathrm{j}} \leftarrow$ DivideVectorElementsByAverage($\vec{\mathbf{c}}_{\mathrm{j}}$ , $AvgUQ$)
16:    end if
17: end for
```

3.7 Implicit Tagging

In this project the goal was to develop an extensible, solution for improving information organization and retrieval of unstructured, unlabeled online articles. As mentioned in the goal, the articles were not labeled prior to training. An approach that was called *implicit tagging* was introduced to address this problem. The term *implicit tagging* have been defined by Soleymani and Pantic (2012) . as an effortless process by which content is tagged based on users' spontaneous reactions. In this project *implicit tagging* was defined as the process by which an unlabeled article is tagged based on whether or not it contains two or more key words associated with a particular category. Implicit tagging in this project was inspired by research done by Ko and Seo (2009), in their research they explained an approach for obtaining labeled data from unlabeled data for text classification. The categories (IPTC Topic) and the key words (News Tags) for the respective categories are shown in Figure 3.9 below (Bacan, Pandzic, and Gulija 2005). Implicit tagging cannot replace the text classification model due to the limited amount of keywords per category compared to a text classification model. The GBC module supports implicit tagging by allowing tags to be assigned to articles at run-time. These temporarily tagged articles are then used as training data. Hence, when the function "GetDocumentLabels" in Algorithm 1 is called, the labels being retrieved are implicit labels.



IPTC Topic	News Tags
Arts, Culture and Entertainment	culture, music, film, media, books, artanddesign, television, art, fashion, festivals, history, comedy, museums, opera, drama, poetry, documentary, painting, theatre, sculpture
Crime, Law and Justice	crime, law, rape, supreme-court, human-rights, police, criminal-justice
Disaster, Accident and Emergency Incident	natural-disasters, hurricane, flooding, earthquake, drought, wildfire
Economy, Business and Finance	business, economics, banking, advertising, transport, economy, market, realestate, investing
Education	education, schools, teaching, students
Environment	environment, climate-change, energy, water, pollution, waste
Health	health, healthcare
Human Interest	awards-and-prizes, celebrity, animals
Labor	labour, employment, unemployment
Lifestyle and Leisure	lifeandstyle, travel, food-and-drink, hotels, restaurants, bars
Politics	politics, democrats, republicans, election, policy
Religion and Belief	religion
Science and Technology	technology, internet, research, science, biology, psychology, software, genetics, mathematics, chemistry
Society	society, race, communities, poverty, family, homelessness, immigration, marriage, population, migration
Sport	sport, nfl, nba, football, basketball, baseball, boxing, tennis, cricket, olympics, athletics, swimming, cycling
Conflicts, War and Peace	protest, terrorism, military
Weather	weather

Figure 3.9: International Press Telecommunication Council(IPTC) Topics

3.8 Incremental Learning

The topic of incremental learning was briefly introduced early on in this chapter. Incremental learning is the ability of a classification model to add new data without needing to retrain the model on the entire data set. The Group By Class text classification model supports incremental learning via the following steps:

(1) Get the document labels from the new training document.

(2) Find class vectors whose labels are the same as one of the new document labels.

(3) Destandardize identified class vectors by multiplying each term weight within each identified class vector by the unique class average for the respective class vector.

(4) Convert the labeled document to a term vector as illustrated in Figure 3.4.

(5) Add the term vector to each identified class vectors as shown in Figure 3.5.

(6) Restandardize the modified identified class vectors, by performing the same standardization steps illustrated in Figure 3.8.

Algorithm 2 GBC Algorithm For Incremental Learning

```
1: Initialize:
       newLD = a new labeled document
       $L = \{l_1, ..., \mathrm{l_n}\}$, where L is the set of labels/categories
       $C = \{\vec{\mathbf{c_1}}, ..., \vec{\mathbf{c}}_\mathrm{n}\}$, where C is the set of standardized class vectors
2: $NewDocumentLabels \leftarrow$ GetDocumentLabels($newLD$)
3: $V \leftarrow$ GetVocab($newLD$), where V is unique words from newLD
4: $\vec{\mathbf{t}} \leftarrow$ ConvertDocumentToBinaryTermVector($newLD$ , $V$)
5: for $l_j \in L$ do
6:     if $l_j \in NewDocumentLabels$ then
7:         $\vec{\mathbf{c}}_\mathrm{j} \leftarrow$ DestandardizeClassVector($\vec{\mathbf{c}}_\mathrm{j}$)
8:         $\vec{\mathbf{c}}_\mathrm{j} \leftarrow$ MergeClassVectorWithTermVector($\vec{\mathbf{c}}_\mathrm{j}$, $\vec{\mathbf{t}}$)
9:         $\vec{\mathbf{c}}_\mathrm{j} \leftarrow$ RestandardizeClassVector($\vec{\mathbf{c}}_\mathrm{j}$)
10:    end if
11: end for
```

3.9 Group By Class Model Classification Steps

The GBC Classification model calculates the similarity between two documents using the dot product (Inner Product) of their term vectors as per the vector space model The result of a dot product calculation is a scalar value which represents how strongly related two documents are to each other (Widnall 2009). The dot product definition is shown in Figure 3.10.

$$\boxed{\vec{\mathbf{A}} \cdot \vec{\mathbf{B}} = \sum a_i b_i = a_1 b_1 + a_2 b_2 + a_3 b_3}$$

Figure 3.10: Dot Product Definition

The steps for classifying a new document using the GBC Classification model are as follows:

(1) Find related class vectors, these are all class vectors that have a weight value for at least one of the terms present in the new document.

(2) Penalize the related class vectors

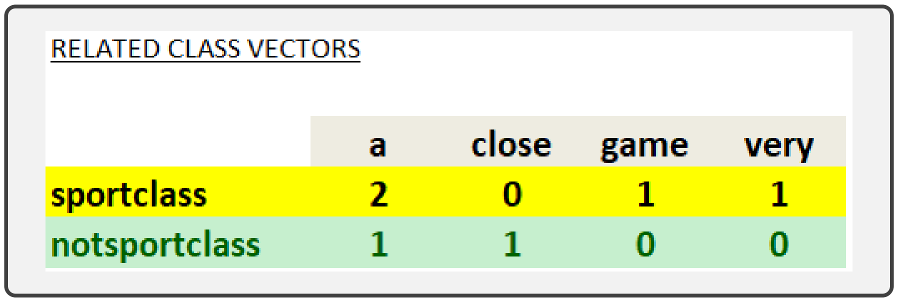
(3) Represent the new document as a query vector (the name given to the term vector of a document to be classified).

(4) Predict the category of document by finding the dot product between the query vector and each of the penalized class vectors. The class vectors that gives the highest result is the category that best represents the query vector, and hence also the new document.

Details of the classification steps are given below using ‘DOC6’ as the new document from Figure 3.6 and the standardized Class Vector from Figure 3.8.

3.9.1 (STEP 1) Find Related Class Vectors

Related class vectors have positive values for term weights since terms are present in the new document. Consider the new document ‘DOC6’ shown in Figure 3.6. Both class vectors from Figure 3.8 would be related because each has term weights for at least one word in the new document. For example ‘sportclass’ has a term weight value for the word ‘very’ and both ‘sportclass’ and ‘notsportclass’ have a term weight for the word/term ‘a’. Notice that the case of the word or letter does not matter. Also for the related class vectors the vocabulary size of the ‘sportclass’ and ‘notsportclass’ are smaller because of the vocabulary size of the new document ‘DOC6’ to be classified.



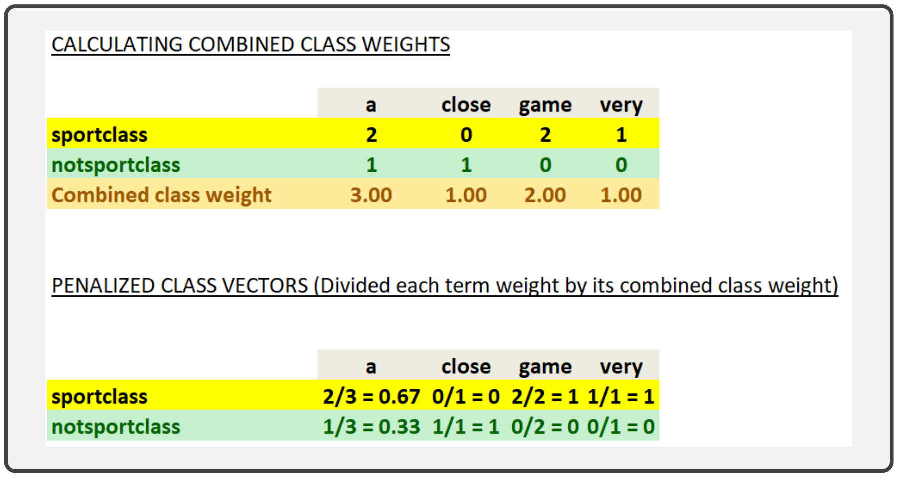
RELATED CLASS VECTORS

	a	close	game	very
sportclass	2	0	1	1
notsportclass	1	1	0	0

Figure 3.11: The Related Class Vectors

3.9.2 (STEP 2) Penalize the Related Class Vectors

Sometimes the related class vectors returned may have term weights for the same word or letter. One such example is ‘a’ which is a word that is known to appear frequently in documents. Words like ‘a’ are sometimes called stop words and are known to affect classification performance. In the machine learning community one common practice of dealing with stop words is to simply remove them or ignore them. However stop word removal is not enough to improve classification performance. It has been shown that some text classification models like the Naive Bayes Classifier uses algorithms like TF-IDF for scaling words by their importance (Kibriya et al. 2004). When Naive Bayes utilizes TF-IDF weights, its classification performance improves. In this project a similar concept of scaling terms by importance was used. The GBC Model was designed so that words that repeat across class vectors are penalized in each class vector they appear in. This was done by firstly calculating the 'combined class weight' for each term associated with the related class vectors. Then secondly, penalize weights of terms that repeat across class vectors, by dividing each term weight by its combined class weight. The penalization step is illustrated in Figure 3.12.


CALCULATING COMBINED CLASS WEIGHTS

	a	close	game	very
sportclass	2	0	2	1
notsportclass	1	1	0	0
Combined class weight	3.00	1.00	2.00	1.00

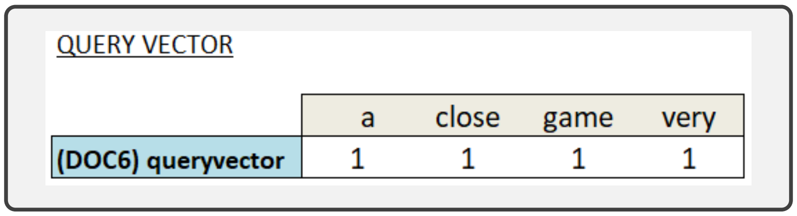
PENALIZED CLASS VECTORS (Divided each term weight by its combined class weight)

	a	close	game	very
sportclass	2/3 = 0.67	0/1 = 0	2/2 = 1	1/1 = 1
notsportclass	1/3 = 0.33	1/1 = 1	0/2 = 0	0/1 = 0

Figure 3.12: The Penalized Class Vectors

3.9.3 (STEP 3) Represent New Document as a Query Vector

In this step the new document from Figure 3.6 is represented as a query vector because it will be used to predict the category of the document by finding the dot product between it and the penalized class vectors in the last classification step.


QUERY VECTOR

	a	close	game	very
(DOC6) queryvector	1	1	1	1

Figure 3.13: The Query Vector

3.9.4 (STEP 4) Predict Category Of New Document

Predicting the category of the new document means finding which class vector the query vector representing the new document is most similar to. As discussed earlier this can be done by calculating the dot product of the query vector, shown in Figure 3.13 with each of the penalized class vectors from Figure 3.12. The formulation and results are given below:

Let QVocabSize = total unique words in new document

Let $\overrightarrow{\mathbf{CV}}$ = any class vector

Let $\vec{\mathbf{Q}}$ = queryvector, Let $\vec{\mathbf{S}}$= sportclass, Let $\vec{\mathbf{N}}$ = notsportclass

Formula For Dot Product:

$\vec{\mathbf{Q}} \cdot \overrightarrow{\mathbf{CV}} = \sum_{i=1}^{QVocabSize} q_i cv_i =>$

$\vec{\mathbf{Q}} \cdot \vec{\mathbf{S}} = (1 * 0.67) + (1 * 0) + (1 * 1) + (1 * 1) = 2.67$

$\vec{\mathbf{Q}} \cdot \vec{\mathbf{N}} = (1 * 0.33) + (1 * 1) + (1 * 0) + (1 * 0) = 1.33$

The largest result is 2.67 which indicates that the new document is most related to the ‘sportclass’ category compared to the ‘notsportclass’. Hence the new document would be categorized as ‘sportsclass’. Note in this step, though the highest class vector was chosen, multiple class vectors which represents multiple categories are returned. This illustrates the multi-label classification capability of the GBC model whereby an instance (document) can be associated with multiple categories to varying degrees. This step is hence also called the multi-label classification step (Sucar et al. 2014).

3.9.5 GBC Algorithm For Classifying Documents

Algorithm 3 GBC Algorithm For Classifying Documents

```
1: Initialize:
       newD = a new document to be labeled
       $C = \{\vec{\mathbf{c}}_1,...,\vec{\mathbf{c}}_n\}$, where C is the set of standardized class vectors
       P = [...], where P is a list of all predicted category weights
       L = [$l_1...l_n$], where L is a list of categories
2: $V \leftarrow$ GetVocab($newD$), where V is all unique words from newD
3: $\vec{\mathbf{t}} \leftarrow$ ConvertDocumentToBinaryTermVector($newD$ , $V$)
4: R $\leftarrow$GetRelatedClassVectors($C$,$V$) where R $\subseteq$ C
5: for $\vec{\mathbf{r}}_j \in R$ do
6:     P[$j$] $\leftarrow$ dotProduct($\vec{\mathbf{t}}$,$\vec{\mathbf{r}}_j$)
7: end for
8: positionOfMaxLabel $\leftarrow$ getMaxindex(P) , get list index with the highest weight
9: PredictedTag = L[positionOfMaxLabel]
```

3.10 Algorithm Function Descriptions

Table 3.1: Algorithms 1-3 Functions and Descriptions

Algorithms	Functions	Descriptions
Algorithm 1	DivideVectorElementsByAverage($\vec{\mathbf{c}}_j$,AUQ)	divides each element in class vector by its unique class average
Algorithm 1	GetUniqueClassAverage($\vec{\mathbf{c}}_j$)	returns unique class average for passed in class vector
Algorithm 1	NotEmptyClassVector($\vec{\mathbf{c}}_j$)	returns true if class vector has at least one term with a non zero weight value
Algorithm 2	RestandardizeClassVector($\vec{\mathbf{c}}_j$)	standardizes class vector by performing the same standardization steps illustrated in Figure 3.8 and discussed in Section 3.6.3
Algorithm 1, 2	MergeClassVectorWithTermVector($\vec{\mathbf{c}}_j$, $\vec{\mathbf{t}}$)	Add the term vector to each identified class vectors (step 5 section 3.8)
Algorithm 2	DestandardizeClassVector($\vec{\mathbf{c}}_j$)	returns the destandardized class vector mentioned in step 3 of Section 3.8
Algorithm 1, 2	GetDocumentLabels($newLD$)	returns all labels that a document was tagged with
Algorithm 1, 2, 3	GetVocab($newD$)	returns vocabulary of a document or set of documents submitted
Algorithm 1, 2, 3	ConvertDocumentToBinaryTermVector(d,v)	returns term vector for document
Algorithm 3	GetRelatedClassVectors(C,V)	returns the set of class vectors that contains at least one word from the new document
Algorithm 3	dotProduct($\vec{\mathbf{t}}$,$\vec{\mathbf{r}}_j$)	Returns the dot product of two vectors
Algorithm 3	getMaxindex(P)	get list index with the highest weight

Chapter 4

System Implementation

As discussed in Chapter 3 the GBC system was designed based on the Model, View, Controller (MVC) architecture pattern. Modules were presented that were used within the architecture. Persistent storage such as the database and text classification model were also mentioned in Chapter 3. This chapter describes the tools, techniques, frameworks, types of databases and the approach used for building the components GBC system.

4.1 Backend and Frontend

The terms *backend* and *frontend* are two popular words used in software development. Any part of the system design process that relates directly to the browser is called *frontend*, while *backend* refers to the programs and scripts that work behind the scenes (Robbins 2012). The frontend, also called the client-side of the system, is mainly the part of the system the user directly interacts with. The backend, also called the server-side, is not typically accessed directly by the user. It is responsible for the storage and manipulation of data and runs on the server side. The GBC system implementation was separated into backend[1] and frontend[2] implementation. The components associated with the backend implementation included: the controller, model, modules, and data sources(database and classification model). The frontend component consisted of

1. https://github.com/DavidDexterCharles/GroupByClassModel/tree/master/BackEnd
2. https://github.com/DavidDexterCharles/GroupByClassModel/tree/master/FrontEnd

the view of the MVC architecture.

4.2 GBC System Backend Implementation

The backend components of the system were hosted on an Amazon EC2 instance mentioned in Table A.2. The backend implementation comprised of a MySQL Database, a Text Classification Model, and two applications that serves as an API for interacting with the database (DatabaseAPI) and as the API used by the frontend (GBC REST API). Modules such as the 'Crawler' and 'GBC module' developed to provide important functionality required by the GBC system. The GBC system and modules were developed using the python programming language and the Flask micro web framework.

4.2.1 Database Design

In implementing the system, a MySQL database described in Table A.2 was used. An ERD diagrm of the database design shown in Figure 4.1. The main tables used were 'article' and 'topicmodel' which stored online articles and category keywords respectively. In developing the system a text classification model had to first be trained. The training data and the implicit tags (section 3.7, Chapter3), were all stored in the database and used for training the classification model.

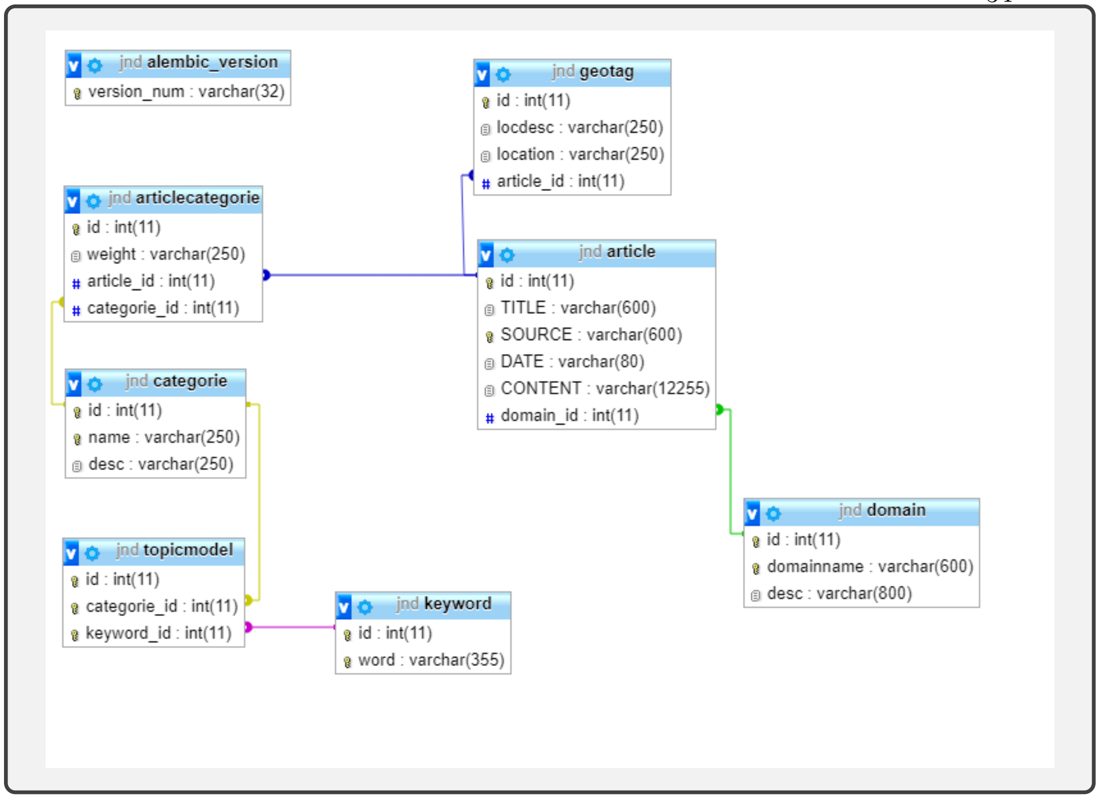


Figure 4.1: Database ERD

4.2.2 The DatabaseAPI

The DatabaseAPI [3] was developed using the Flask micro framework [4] . It was used by the Architecture Model of the GBC system to perform CRUD(Create, Read, Update, and Delete) operations on the database. Two common approaches used for performing CRUD on a database include:

- Writing adhoc SQL Queries to perform all the CRUD operations on the database.
- Using an Object-relational Mapper (ORM) to access, address and manipulate objects without having to consider the relationship between those objects and their data sources.

SQLAlchemy is an example of an ORM that associates instances of Python classes

3. https://github.com/DavidDexterCharles/GroupByClassModel/tree/master/BackEnd/DatabaseAPI
4. https://flask.palletsprojects.com/en/1.1.x/

can be associated with full database tables and the rows contained within. The ORM provides a system that transparently synchronizes all state changes between each object and their related rows.

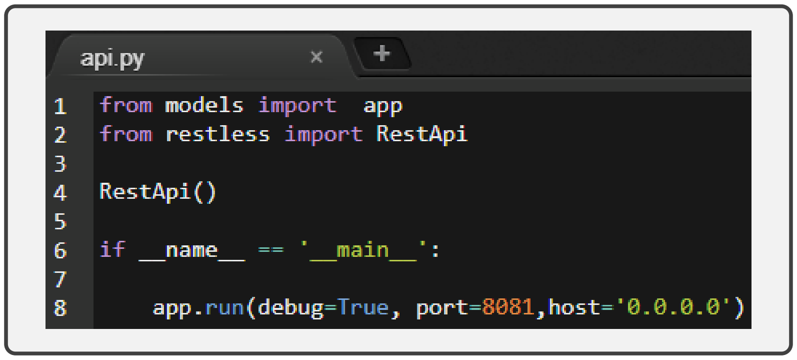
The DatabaseAPI used "Flask-Restless" [5] which is a Flask extension that uses models (instances of Python classes) which have been defined using SQLAlchemy or Flask-SQLAlchemy (flask extension of SQLAlchemy), for the creation of ReSTful JSON APIs. Each '.py' file in Python can be considered to be a module, hence the structure of the DatabaseAPI would be described from the perspective of modules. The DatabaseAPI consists of three (3) main modules:

(1) 'api.py'

(2) 'models.py'

(3) 'restless.py'

The first module, 'api.py', is the entry point to the application and is used to start the server for the DatabaseAPI. Code snippet for this module is shown in Figure 4.2. This module is started by running the following command in terminal 'python api.py', note the version of Python used is Python 3. When started, the REST API would be hosted on port 8081 as illustrated.

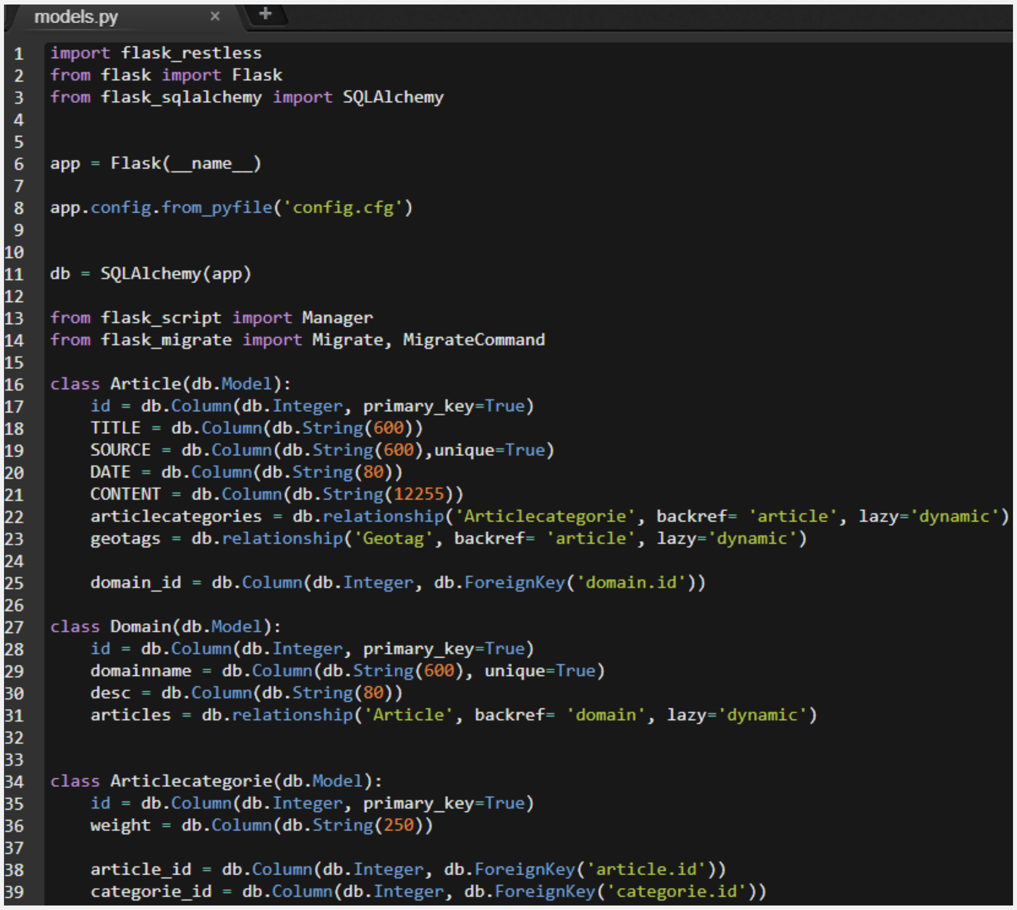


```
from models import  app
from restless import RestApi

RestApi()

if __name__ == '__main__':

    app.run(debug=True, port=8081,host='0.0.0.0')
```

Figure 4.2: First Module 'api.py' in DatabaseAPI

5. https://flask-restless.readthedocs.io/en/stable/index.html

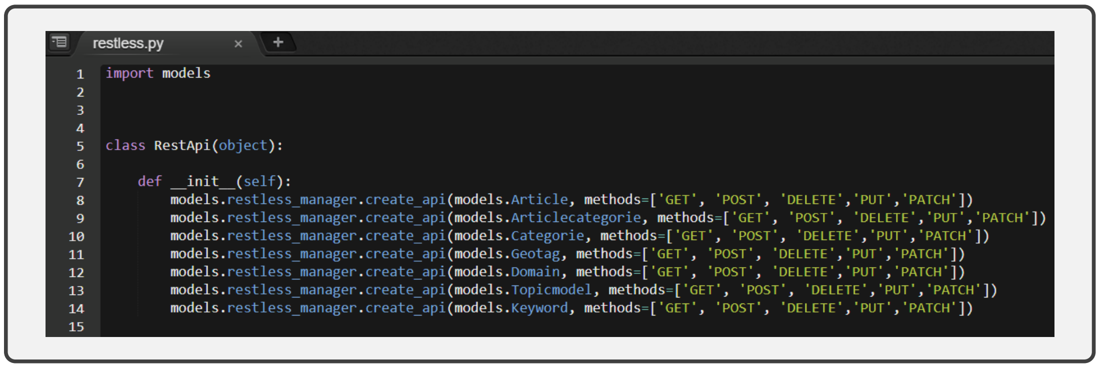
The second module, ‘models.py’, contains all of the Python classes which have been described using Flask-SQLAlchemy. These Python classes map to tables within the MySQL Database and provide ORM functionality. Figure 4.3 illustrates three Python classes that correspond to three tables in the database (article, domain, and articlecategorie). The ‘config.cfg’shown imported on line 8 in Figure 4.3 contains the connection information that enables the DatabaseAPI Flask application to communicate with the database.



```
1  import flask_restless
2  from flask import Flask
3  from flask_sqlalchemy import SQLAlchemy
4
5
6  app = Flask(__name__)
7
8  app.config.from_pyfile('config.cfg')
9
10
11 db = SQLAlchemy(app)
12
13 from flask_script import Manager
14 from flask_migrate import Migrate, MigrateCommand
15
16 class Article(db.Model):
17     id = db.Column(db.Integer, primary_key=True)
18     TITLE = db.Column(db.String(600))
19     SOURCE = db.Column(db.String(600),unique=True)
20     DATE = db.Column(db.String(80))
21     CONTENT = db.Column(db.String(12255))
22     articlecategories = db.relationship('Articlecategorie', backref= 'article', lazy='dynamic')
23     geotags = db.relationship('Geotag', backref= 'article', lazy='dynamic')
24
25     domain_id = db.Column(db.Integer, db.ForeignKey('domain.id'))
26
27 class Domain(db.Model):
28     id = db.Column(db.Integer, primary_key=True)
29     domainname = db.Column(db.String(600), unique=True)
30     desc = db.Column(db.String(80))
31     articles = db.relationship('Article', backref= 'domain', lazy='dynamic')
32
33
34 class Articlecategorie(db.Model):
35     id = db.Column(db.Integer, primary_key=True)
36     weight = db.Column(db.String(250))
37
38     article_id = db.Column(db.Integer, db.ForeignKey('article.id'))
39     categorie_id = db.Column(db.Integer, db.ForeignKey('categorie.id'))
```

Figure 4.3: Second Module ‘models.py’ in DatabaseAPI

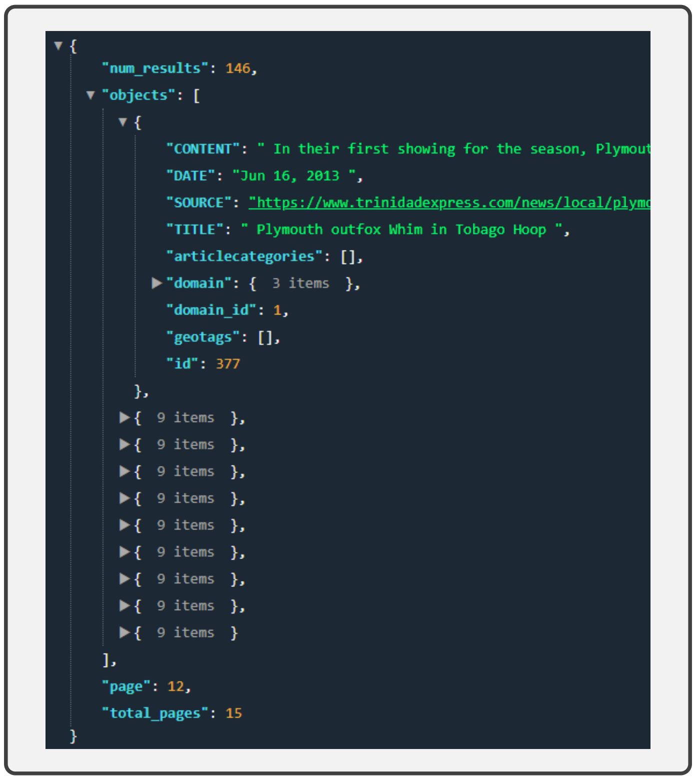
The third module, ‘restless.py’, has a small amount of code but provides powerful restful capability by automatically providing API end points that can be consumed by other services. This is possible due to the Flask-Restless import statements shown in Figure 4.3 on line 1. The module ‘restless.py’ utilizes the ORM (‘models.py’) to dynamically create these endpoints. Figure 4.4 shows the code for the ‘restless.py’ module and illustrates how the classes imported from ‘models.py’ were used to perform HTTP requests such as GET, POST, PUT and DELETE on all the tables. The DatabaseAPI is not made public,to prevent unauthorized updates/and deletes on the database, and hence can only be accessed on the local machine. In Figure 4.2 the constructor method ‘RestApi()’ on line 4 is imported from the ‘restless.py’ module and enables the ReSTful functionality.



```python
import models



class RestApi(object):

    def __init__(self):
        models.restless_manager.create_api(models.Article, methods=['GET', 'POST', 'DELETE','PUT','PATCH'])
        models.restless_manager.create_api(models.Articlecategorie, methods=['GET', 'POST', 'DELETE','PUT','PATCH'])
        models.restless_manager.create_api(models.Categorie, methods=['GET', 'POST', 'DELETE','PUT','PATCH'])
        models.restless_manager.create_api(models.Geotag, methods=['GET', 'POST', 'DELETE','PUT','PATCH'])
        models.restless_manager.create_api(models.Domain, methods=['GET', 'POST', 'DELETE','PUT','PATCH'])
        models.restless_manager.create_api(models.Topicmodel, methods=['GET', 'POST', 'DELETE','PUT','PATCH'])
        models.restless_manager.create_api(models.Keyword, methods=['GET', 'POST', 'DELETE','PUT','PATCH'])

```

Figure 4.4: Third Module ‘restless.py’ in DatabaseAPI

Figure 4.5 illustrates a GET request example endpoint that was generated by the DatabaseAPI to retrieve article information from the database. The API by default returns a JSON object. Instructions on how to use the Flask-Restless API and make other possible API calls are available in the Flask-Restless documentation (Maia 2015).



```
▼{
    "num_results": 146,
  ▼"objects": [
    ▼{
        "CONTENT": " In their first showing for the season, Plymout
        "DATE": "Jun 16, 2013 ",
        "SOURCE": "https://www.trinidadexpress.com/news/local/plymo
        "TITLE": " Plymouth outfox Whim in Tobago Hoop ",
        "articlecategories": [],
      ▶"domain": { 3 items },
        "domain_id": 1,
        "geotags": [],
        "id": 377
     },
    ▶{ 9 items },
    ▶{ 9 items },
    ▶{ 9 items },
    ▶{ 9 items },
    ▶{ 9 items },
    ▶{ 9 items },
    ▶{ 9 items },
    ▶{ 9 items },
    ▶{ 9 items }
    ],
    "page": 12,
    "total_pages": 15
}
```

Figure 4.5: Response For a Get Request: ‘http://0.0.0.0:8081/api/article?page=12’

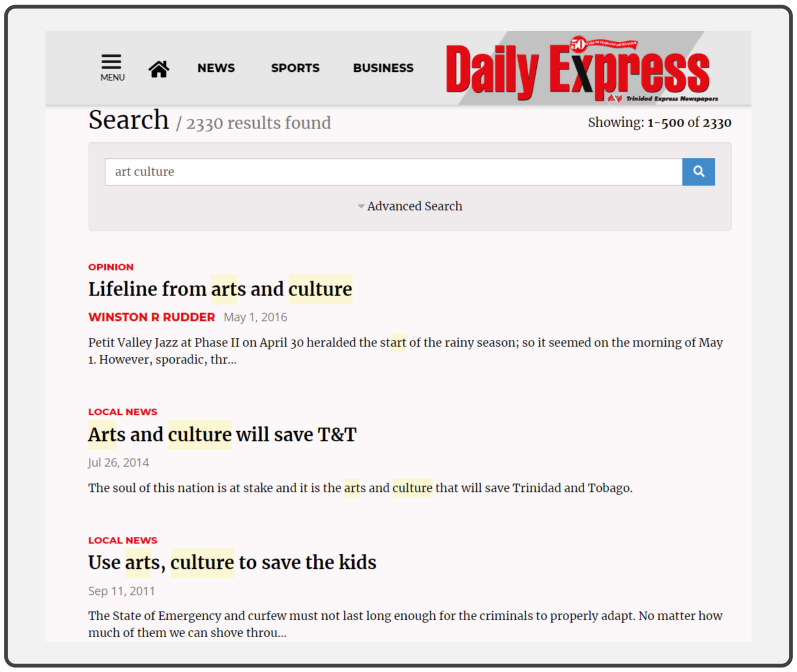
4.2.3 Collecting Training Data

The module ‘crawler.py’ was primarily designed for extracting textual information from online articles and web pages. The code for the crawler can be found at the github repository [6] associated with this project. All collected training data was eventually stored in the database in section 4.2.1. For illustrative purpose the training data is shown as being stored in JSON files. In order to train the text classification model, labeled online articles were needed, however there were challenges with acquiring the labeled data directly from news agencies in Trinidad and Tobago. Articles that were already labeled were not free for the public and the majority free access articles on the web were not labeled. Hence, an approach for gathering acceptable training data was developed using the Crawler. This involved collecting only articles that contained

6. https://github.com/DavidDexterCharles/GroupByClassModel/blob/master/BackEnd/crawler/crawler.py

keywords corresponding to the IPTC topics. Articles that contained at least two keywords from at least one of the IPTC categories were considered to be associated with the respective category. This method of associating articles with categories based on keyword presence was called *implicit tagging* as discussed in Chapter 3. The following steps describe how online articles were retrieved:

- First a manual search was done on the news article website for potential training data as illustrated in Figure 4.6. In this example the search parameters included the keywords 'art' and 'culture' which correspond to the IPTC news tags in Figure 3.9. Articles containing these words would therefore be implicitly tagged as art. Though the implicit tags were not guaranteed to be fully correct, through trial and error experiments it was found that most articles that contained two or more keywords for a particular IPTC topic were related to said topic. Similar search parameters were specified for the other IPTC topics.



Search / 2330 results found

Showing: 1-500 of 2330

art culture

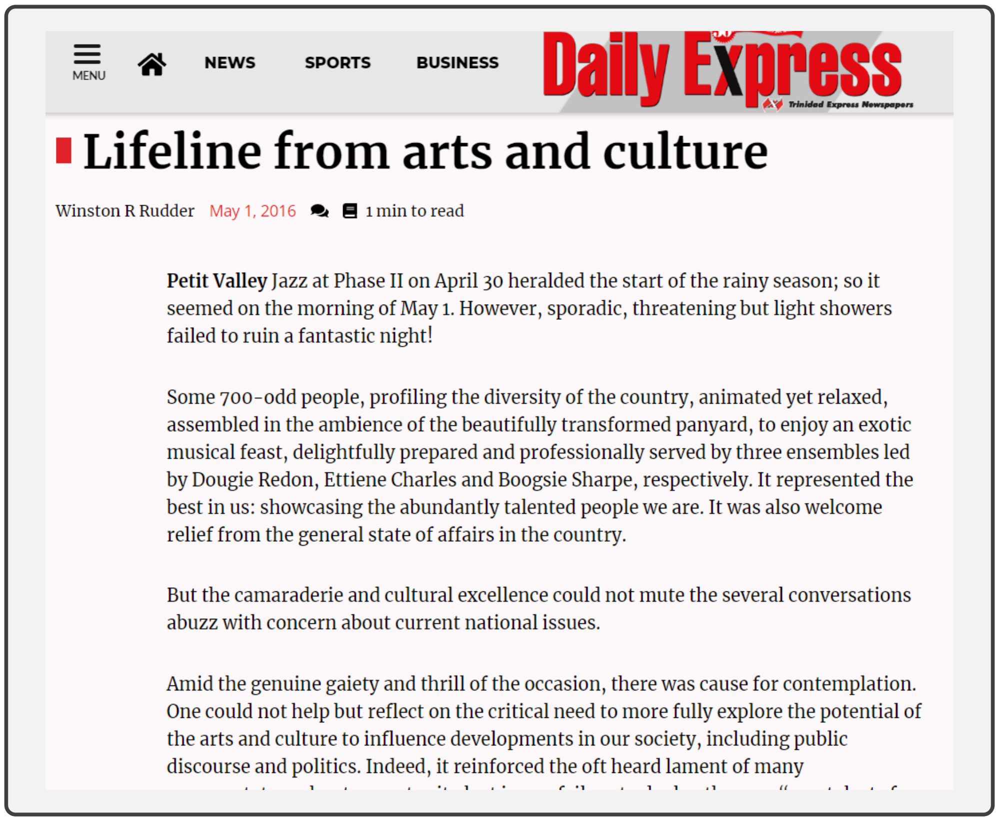
OPINION
Lifeline from arts and culture
WINSTON R RUDDER May 1, 2016
Petit Valley Jazz at Phase II on April 30 heralded the start of the rainy season; so it seemed on the morning of May 1. However, sporadic, thr...

LOCAL NEWS
Arts and culture will save T&T
Jul 26, 2014
The soul of this nation is at stake and it is the arts and culture that will save Trinidad and Tobago.

LOCAL NEWS
Use arts, culture to save the kids
Sep 11, 2011
The State of Emergency and curfew must not last long enough for the criminals to properly adapt. No matter how much of them we can shove throu...

Figure 4.6: Daily Express News Articles Returned For Keyword Search 'Art and Culture'

- Figure 4.6 shows that 500 articles were returned on a single page for a search query. Each article title was hyper-linked to a specific article.Figure 4.7 shows an example of what happens when the hyperlink for the title ‘Lifeline from arts and culture’ is invoked. Using the listing functionality provided by the online news website, the HTML code (page source) for each result was collected and stored in a text file with a name corresponding to a specified IPTC topic. The HTML stored in the text file contains all of the URL present on the web page. The information returned from the search query would vary slightly almost on a daily basis, so to maintain consistency during testing and experiment phase, the HTML web pages were downloaded and stored in text files. The crawler then used the downloaded web content to traverse and extract the data associated with each article URL.



Lifeline from arts and culture

Winston R Rudder May 1, 2016 1 min to read

Petit Valley Jazz at Phase II on April 30 heralded the start of the rainy season; so it seemed on the morning of May 1. However, sporadic, threatening but light showers failed to ruin a fantastic night!

Some 700-odd people, profiling the diversity of the country, animated yet relaxed, assembled in the ambience of the beautifully transformed panyard, to enjoy an exotic musical feast, delightfully prepared and professionally served by three ensembles led by Dougie Redon, Ettiene Charles and Boogsie Sharpe, respectively. It represented the best in us: showcasing the abundantly talented people we are. It was also welcome relief from the general state of affairs in the country.

But the camaraderie and cultural excellence could not mute the several conversations abuzz with concern about current national issues.

Amid the genuine gaiety and thrill of the occasion, there was cause for contemplation. One could not help but reflect on the critical need to more fully explore the potential of the arts and culture to influence developments in our society, including public discourse and politics. Indeed, it reinforced the oft heard lament of many

Figure 4.7: Sample Article

- Figure 4.8 illustrates how the crawler module works. Only the categories ‘crime’ and ‘art’ were used for this example. The ‘pagesource’ array list, declared on line 5, stored the paths to the text files. Each text file contained HTML code for a web page like the one shown in Figure 4.6, and corresponded to an IPTC topic. For example, the ‘crime.txt’ file located in the ‘pagesource’ folder on line 6, contained html code for a web page that had crime related urls. The web page had a listing of articles which were returned based on a search query. The search query would have contained two or more key words corresponding to the IPTC crime category. As shown in Figure 4.8, the variable ‘domain’ on line 8 stores the domain of the news website and is used by the crawler. Other variables included: ‘articlecontent’, ‘publicationdate’, and ‘articletitle’. These variables stored identifiers that specify the location on the web page the required information would be extracted from. For example, the article shown in Figure 4.7 is a web page. The HTML code for this article was examined and it was found that the content of the article was wrapped in paragraph tags:

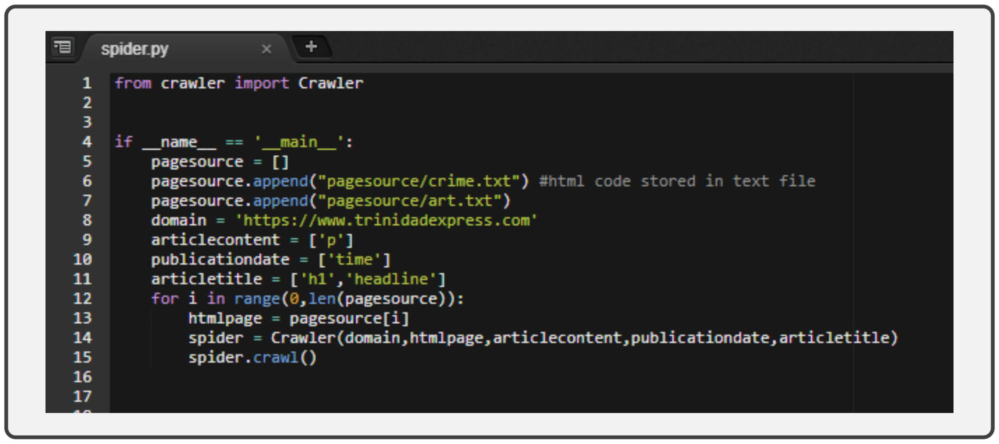
  ```
  <p> Petit Valley Jazz at Phase II on .... </p>
  ```

  and hence the variable ‘articlecontent’ was assigned the list ‘[p]’. A list is used because other element attributes can be used to identify a piece of content on a web page. The list format is as follows: ‘[html tag, class, id]. For example the variable ‘articlecontent’ is assigned as list ‘[h1,headline]’. This is because the title of the content was identified as follows :

  ```
  <h1 class=‘‘headline">Lifeline from arts and culture</h1>
  ```

  The variables once assigned, are passed as parameters to the Crawler constructor on line 14 in Figure 4.8. For each page source containing the web page, all the URLs on the respective pages are visited. For each URL visited only content that matches the specified parameters gets extracted. As shown in Figure 4.8

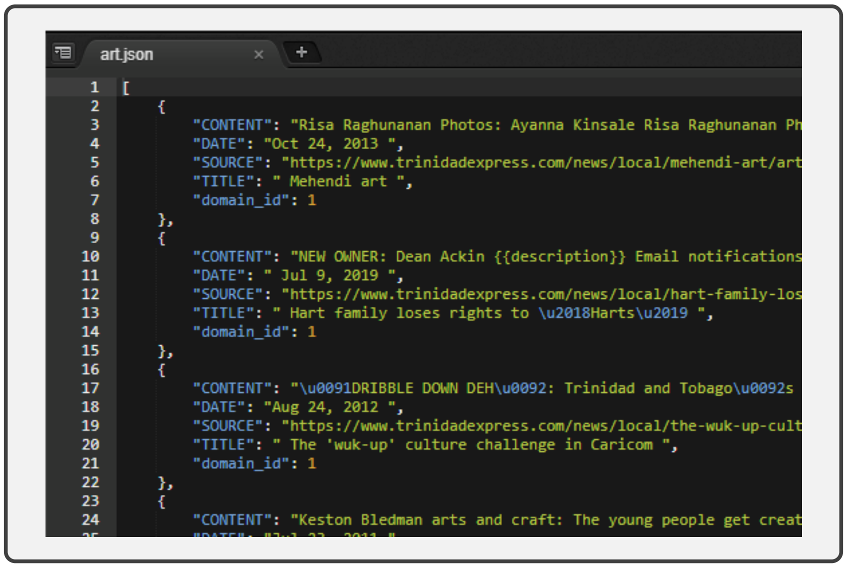
line 14, a crawler object called 'spider' is created and for each page source the method 'crawl()' is called using the 'spider' object. Each time 'spider.crawl()' is called results similar to that shown in Figure 4.9 is obtained. These results are then merged into one JSON file and used as the training data for training the text classification model. Only articles from the domain shown on line 8 in Figure 4.8 were used for training the initial classification model.



spider.py

```
 1  from crawler import Crawler
 2
 3
 4  if __name__ == '__main__':
 5      pagesource = []
 6      pagesource.append("pagesource/crime.txt") #html code stored in text file
 7      pagesource.append("pagesource/art.txt")
 8      domain = 'https://www.trinidadexpress.com'
 9      articlecontent = ['p']
10      publicationdate = ['time']
11      articletitle = ['h1','headline']
12      for i in range(0,len(pagesource)):
13          htmlpage = pagesource[i]
14          spider = Crawler(domain,htmlpage,articlecontent,publicationdate,articletitle)
15          spider.crawl()
16
17
```

Figure 4.8: Crawler Module Example Code



art.json

```
[
    {
        "CONTENT": "Risa Raghunanan Photos: Ayanna Kinsale Risa Raghunanan Ph
        "DATE": "Oct 24, 2013 ",
        "SOURCE": "https://www.trinidadexpress.com/news/local/mehendi-art/art
        "TITLE": " Mehendi art ",
        "domain_id": 1
    },
    {
        "CONTENT": "NEW OWNER: Dean Ackin {{description}} Email notifications
        "DATE": " Jul 9, 2019 ",
        "SOURCE": "https://www.trinidadexpress.com/news/local/hart-family-los
        "TITLE": " Hart family loses rights to \u2018Harts\u2019 ",
        "domain_id": 1
    },
    {
        "CONTENT": "\u0091DRIBBLE DOWN DEH\u0092: Trinidad and Tobago\u0092s
        "DATE": "Aug 24, 2012 ",
        "SOURCE": "https://www.trinidadexpress.com/news/local/the-wuk-up-cult
        "TITLE": " The 'wuk-up' culture challenge in Caricom ",
        "domain_id": 1
    },
    {
        "CONTENT": "Keston Bledman arts and craft: The young people get creat
```

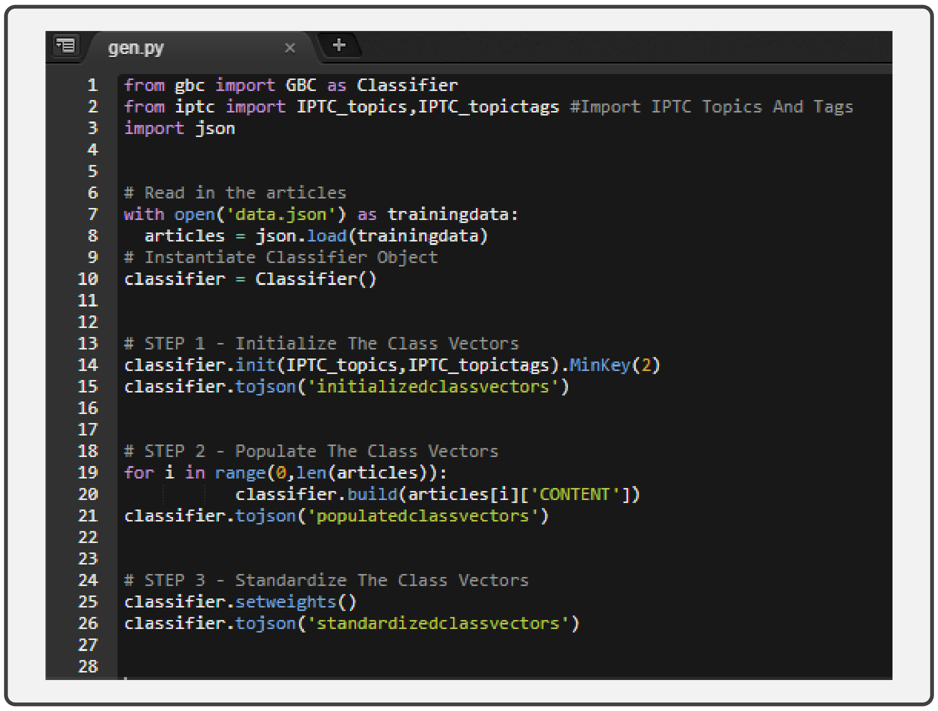
Figure 4.9: Result for 'spider.crawl()' where html page = 'pagesource/art.txt'

4.2.4 GBC Text Classification Model And The GBC Module

The module ‘gbc.py’ [7] also called the GBC module was developed to provide functionality for training a text classification model, updating the model, and performing classification tasks using the model. The GBC module implements the suite of Group By Class algorithms for training, incremental learning, and classification.

4.2.4.1 Training Model Using GBC Module

The steps for training the text classification model have already been discussed in detail in Chapter 3 section 3.6. In this chapter the same steps are reiterated using the GBC module and the training data collected. Snapshot of the training data ‘data.json’ is shown in Figure B of the appendix.



```
 1  from gbc import GBC as Classifier
 2  from iptc import IPTC_topics,IPTC_topictags #Import IPTC Topics And Tags
 3  import json
 4
 5
 6  # Read in the articles
 7  with open('data.json') as trainingdata:
 8    articles = json.load(trainingdata)
 9  # Instantiate Classifier Object
10  classifier = Classifier()
11
12
13  # STEP 1 - Initialize The Class Vectors
14  classifier.init(IPTC_topics,IPTC_topictags).MinKey(2)
15  classifier.tojson('initializedclassvectors')
16
17
18  # STEP 2 - Populate The Class Vectors
19  for i in range(0,len(articles)):
20              classifier.build(articles[i]['CONTENT'])
21  classifier.tojson('populatedclassvectors')
22
23
24  # STEP 3 - Standardize The Class Vectors
25  classifier.setweights()
26  classifier.tojson('standardizedclassvectors')
27
28
```

Figure 4.10: Script ‘gen.py’ For Training Classification Model Using GBC Module

As shown in Figure 4.10 above, the first step is to initialize the class vectors using the code shown on lines 13-14. The function *Minkey()* on line 14 is used to specify how

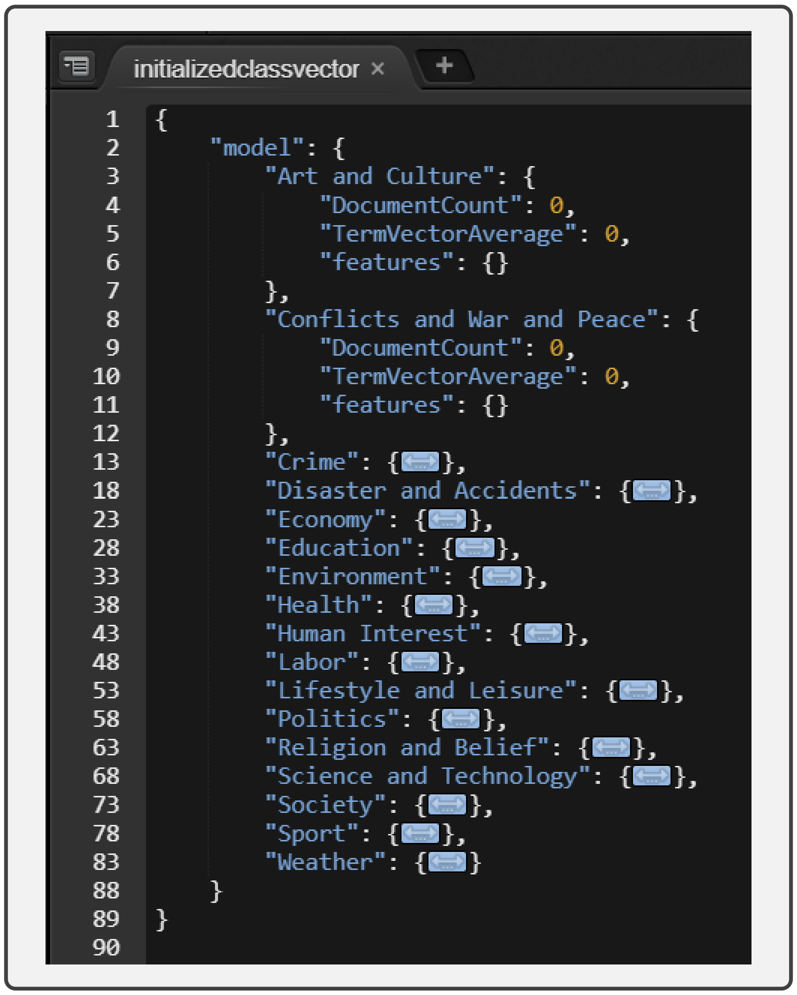
7. https://github.com/DavidDexterCharles/GroupByClassModel/blob/master/gbcDEMO/gbc.py

many keywords/tags, for example IPTC topic tags, needs to be present in a particular document in order for the document to belong to a particular category. This is the implicit tagging functionality of the GBC module which was discussed in Section 3.7. The imported 'iptc' [8] module on line 2 in Figure 4.10 is shown in Figure B of appendix. Consider a document containing only the phrase "I enjoy the art of the Caribbean' and the word 'art' is a valid key word from the IPTC topics belonging to the 'Art and Culture' category as shown in Figure B of appendix. The document would be implicitly tagged as 'Art and Culture' if the *Minkey()* function was invoked using a parameter value of '1'. If the function is invoked as *Minkey(2)* then this means that the document needs to have at least two keywords from any given category in order to be tagged under that category. The document with text "I enjoy the art of the Caribbean" would therefore fail to be tagged as 'Art and Culture' if *Minkey(2)* was called. If another key word from the same category is found to be in the document then the document will be tagged as 'Art and Culture' since it now has at least 2 key words for the respective category. For example the document: "I enjoy the art and culture of the Caribbean" will be implicitly tagged as 'Art and Culture' if *Minkey(2)* was specified since culture is also a key word in the 'Art and Culture' category. During the process of gathering the training data it was ensured that all documents gathered had keywords present with respect to the IPTC topics as discussed in Section 4.2.3. The parameter for *Minkey()* must be greater than zero.

The code on line 15 in Figure 4.10 generates the model containing the initialized class vectors. Figure 4.11 shows the generated model in JSON format, which contains the initialized class vectors such as "Art and Culture" to "Weather". The class vectors are initialized but not populated. They currently have no features indicated by the empty *features* object which is responsible for storing terms and the corresponding weights as discussed in Section 3.6.1. The *DocumentCount* represents the number of

8. https://github.com/DavidDexterCharles/GroupByClassModel/blob/master/gbcDEMO/iptc.py

documents that were converted to term vectors and merged with the respective class vector, and the *TermVectorAverage* stores the 'unique class average' as discussed in Section 3.6.3.

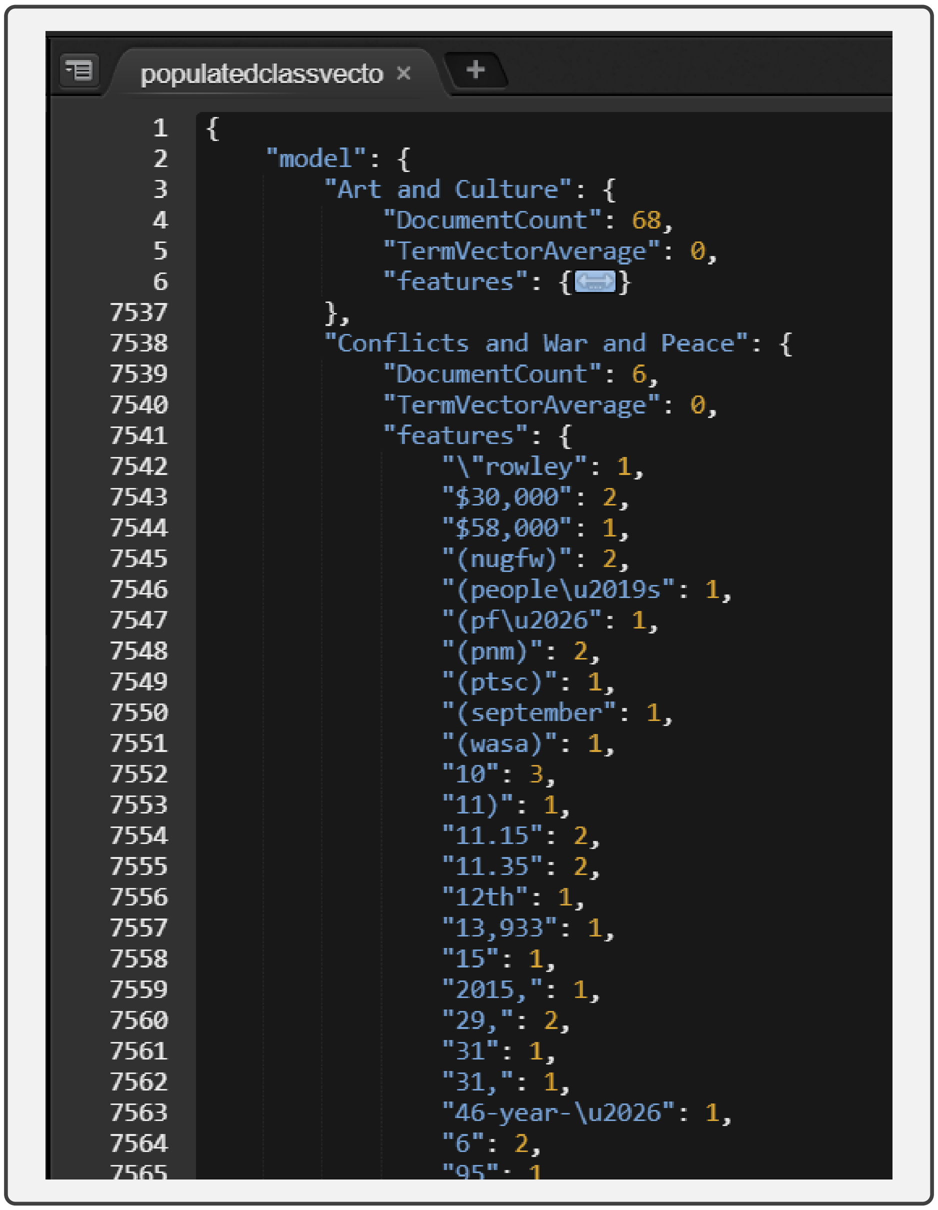


```
{
    "model": {
        "Art and Culture": {
            "DocumentCount": 0,
            "TermVectorAverage": 0,
            "features": {}
        },
        "Conflicts and War and Peace": {
            "DocumentCount": 0,
            "TermVectorAverage": 0,
            "features": {}
        },
        "Crime": {↔},
        "Disaster and Accidents": {↔},
        "Economy": {↔},
        "Education": {↔},
        "Environment": {↔},
        "Health": {↔},
        "Human Interest": {↔},
        "Labor": {↔},
        "Lifestyle and Leisure": {↔},
        "Politics": {↔},
        "Religion and Belief": {↔},
        "Science and Technology": {↔},
        "Society": {↔},
        "Sport": {↔},
        "Weather": {↔}
    }
}
```

Figure 4.11: Model 1 Containing Initialized Class Vectors From Training Data

The next step shown on lines 18-20 of Figure 4.10 populates the class vectors. The *build()* function on line 20, accepts a text document as input and checks which classes or categories the document belongs to, in this example the document belongs to only one category/class ('Conflicts and War and Peace'). It then converts the document to a term vector and merges the term vector with the respective class vector. Details about populating the class vector can be found in Section (3.6.2). Line 21 of Figure

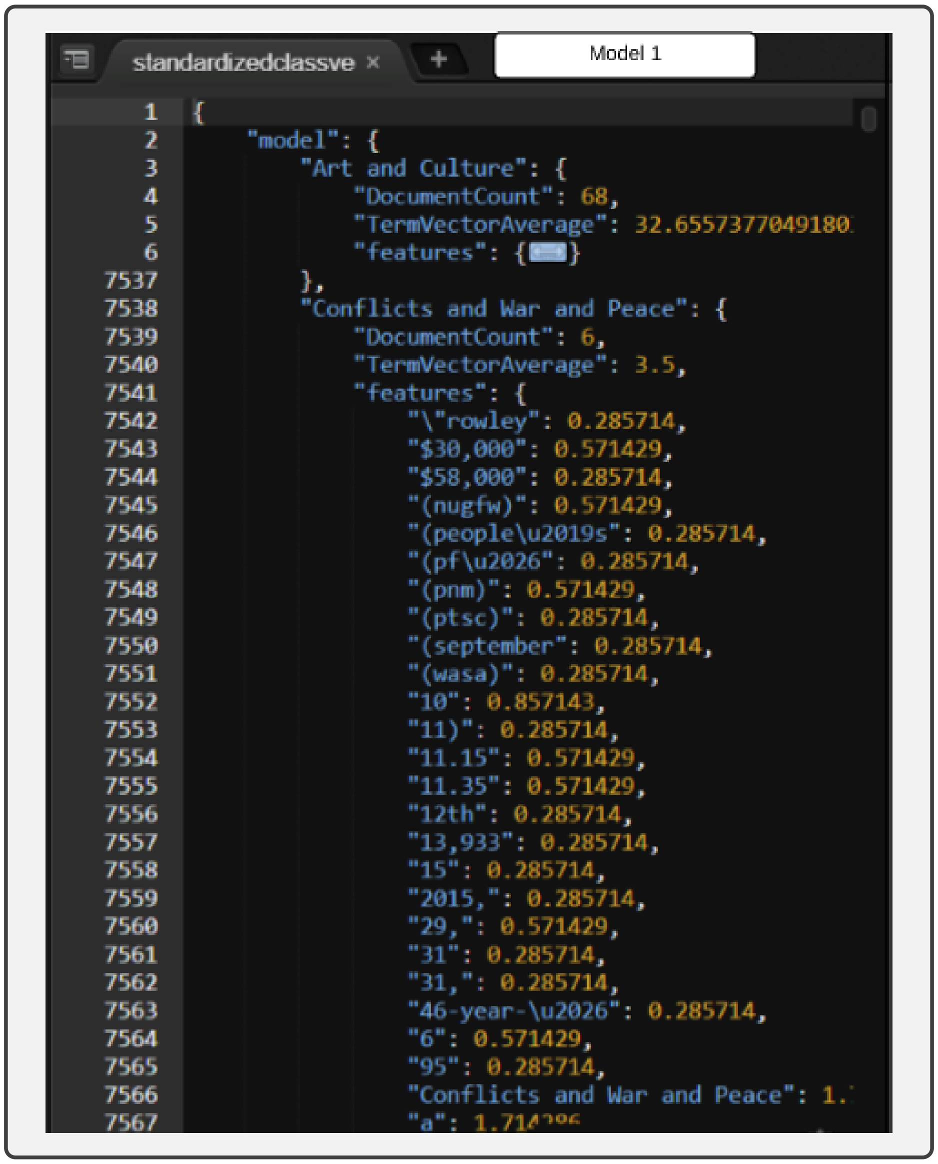
4.10 outputs the model as a JSON file with the populated class vectors as shown in Figure 4.12. Within the *features* object , shown in Figure 4.12, the terms are sorted in alphabetical order. The *features* object also does not store terms with weights of zero, because this means that the term is not present in the class vector. In Figure 4.12 the class vector 'Conflicts and War and Peace' can be seen to have term weights for terms like ' "rowley ' and 'wasa'.



```
{
    "model": {
        "Art and Culture": {
            "DocumentCount": 68,
            "TermVectorAverage": 0,
            "features": {↔}
        },
        "Conflicts and War and Peace": {
            "DocumentCount": 6,
            "TermVectorAverage": 0,
            "features": {
                "\"rowley": 1,
                "$30,000": 2,
                "$58,000": 1,
                "(nugfw)": 2,
                "(people\u2019s": 1,
                "(pf\u2026": 1,
                "(pnm)": 2,
                "(ptsc)": 1,
                "(september": 1,
                "(wasa)": 1,
                "10": 3,
                "11)": 1,
                "11.15": 2,
                "11.35": 2,
                "12th": 1,
                "13,933": 1,
                "15": 1,
                "2015,": 1,
                "29,": 2,
                "31": 1,
                "31,": 1,
                "46-year-\u2026": 1,
                "6": 2,
                "95": 1
```

Figure 4.12: Model 1 Containing Populated Class Vectors From Training Data

The last step for generating the classification model is to standardize the class vectors. This is done using the code snippet on line 25 of Figure 4.10. The *setweights()* function standardizes all the class vectors in the model using the same approach discussed in Section 3.6.3. The output for the model containing the standardized class vectors is shown in Figure 4.13.



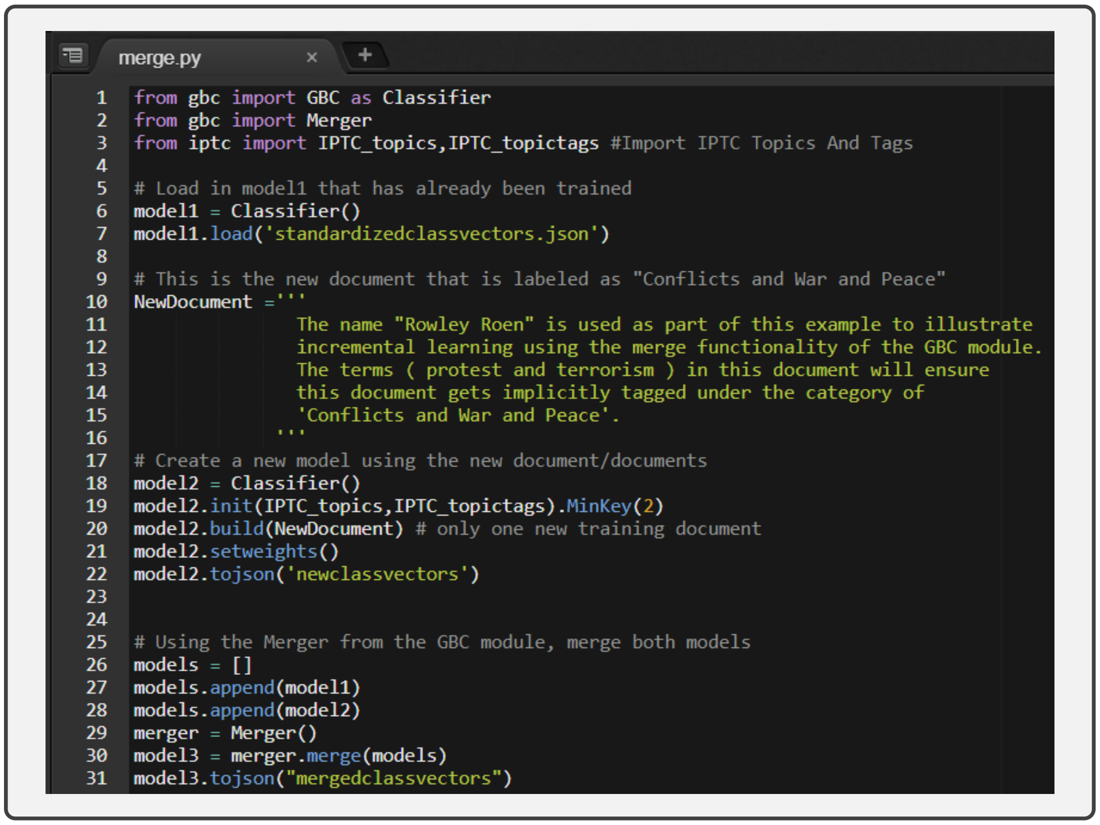
```
{
    "model": {
        "Art and Culture": {
            "DocumentCount": 68,
            "TermVectorAverage": 32.6557377049180
            "features": {}
        },
        "Conflicts and War and Peace": {
            "DocumentCount": 6,
            "TermVectorAverage": 3.5,
            "features": {
                "\"rowley": 0.285714,
                "$30,000": 0.571429,
                "$58,000": 0.285714,
                "(nugfw)": 0.571429,
                "(people\u2019s": 0.285714,
                "(pf\u2026": 0.285714,
                "(pnm)": 0.571429,
                "(ptsc)": 0.285714,
                "(september": 0.285714,
                "(wasa)": 0.285714,
                "10": 0.857143,
                "11)": 0.285714,
                "11.15": 0.571429,
                "11.35": 0.571429,
                "12th": 0.285714,
                "13,933": 0.285714,
                "15": 0.285714,
                "2015,": 0.285714,
                "29,": 0.571429,
                "31": 0.285714,
                "31,": 0.285714,
                "46-year-\u2026": 0.285714,
                "6": 0.571429,
                "95": 0.285714,
                "Conflicts and War and Peace": 1.
                "a": 1.71
```

Figure 4.13: Model 1 Containing Standardized Class Vectors From Training Data

4.2.4.2 GBC Module Incremental Learning and Classification

After the text classification model was trained it was then incorporated into the GBC system and was used by the Architecture Model. As discussed in Section 3.2.3 Architecture Model only interacts with the GBC text classification model when a classification task needs to be addressed or when the GBC text classification model needs to be updated via incremental learning.

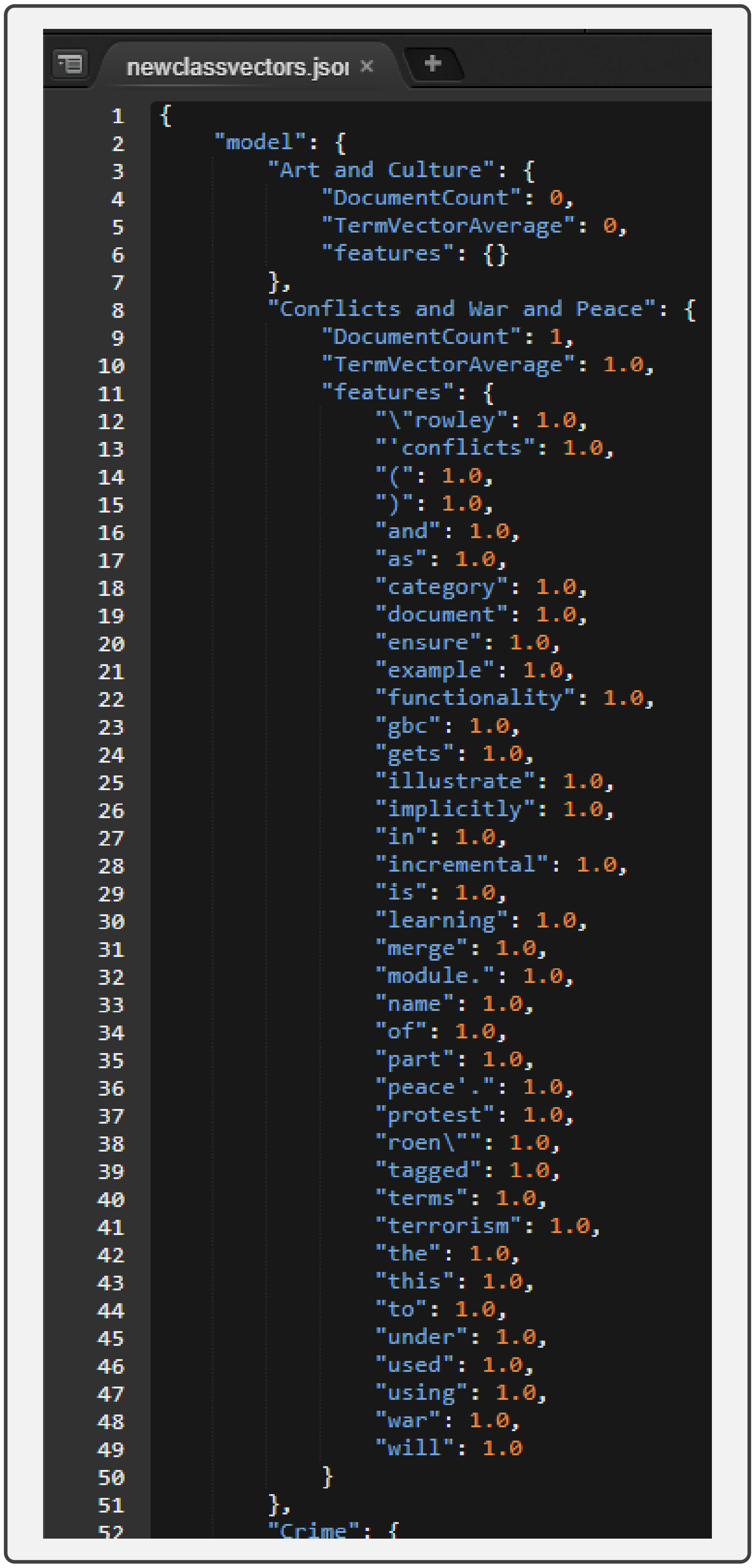
- **Incremental Learning Using The GBC Module:** Figure 4.14 illustrates how the GBC incremental learning algorithm was implemented using the GBC module. The general approach involves the merging of class vectors. The



```python
from gbc import GBC as Classifier
from gbc import Merger
from iptc import IPTC_topics,IPTC_topictags #Import IPTC Topics And Tags

# Load in model1 that has already been trained
model1 = Classifier()
model1.load('standardizedclassvectors.json')

# This is the new document that is labeled as "Conflicts and War and Peace"
NewDocument ='''
               The name "Rowley Roen" is used as part of this example to illustrate
               incremental learning using the merge functionality of the GBC module.
               The terms ( protest and terrorism ) in this document will ensure
               this document gets implicitly tagged under the category of
               'Conflicts and War and Peace'.
             '''
# Create a new model using the new document/documents
model2 = Classifier()
model2.init(IPTC_topics,IPTC_topictags).MinKey(2)
model2.build(NewDocument) # only one new training document
model2.setweights()
model2.tojson('newclassvectors')


# Using the Merger from the GBC module, merge both models
models = []
models.append(model1)
models.append(model2)
merger = Merger()
model3 = merger.merge(models)
model3.tojson("mergedclassvectors")
```

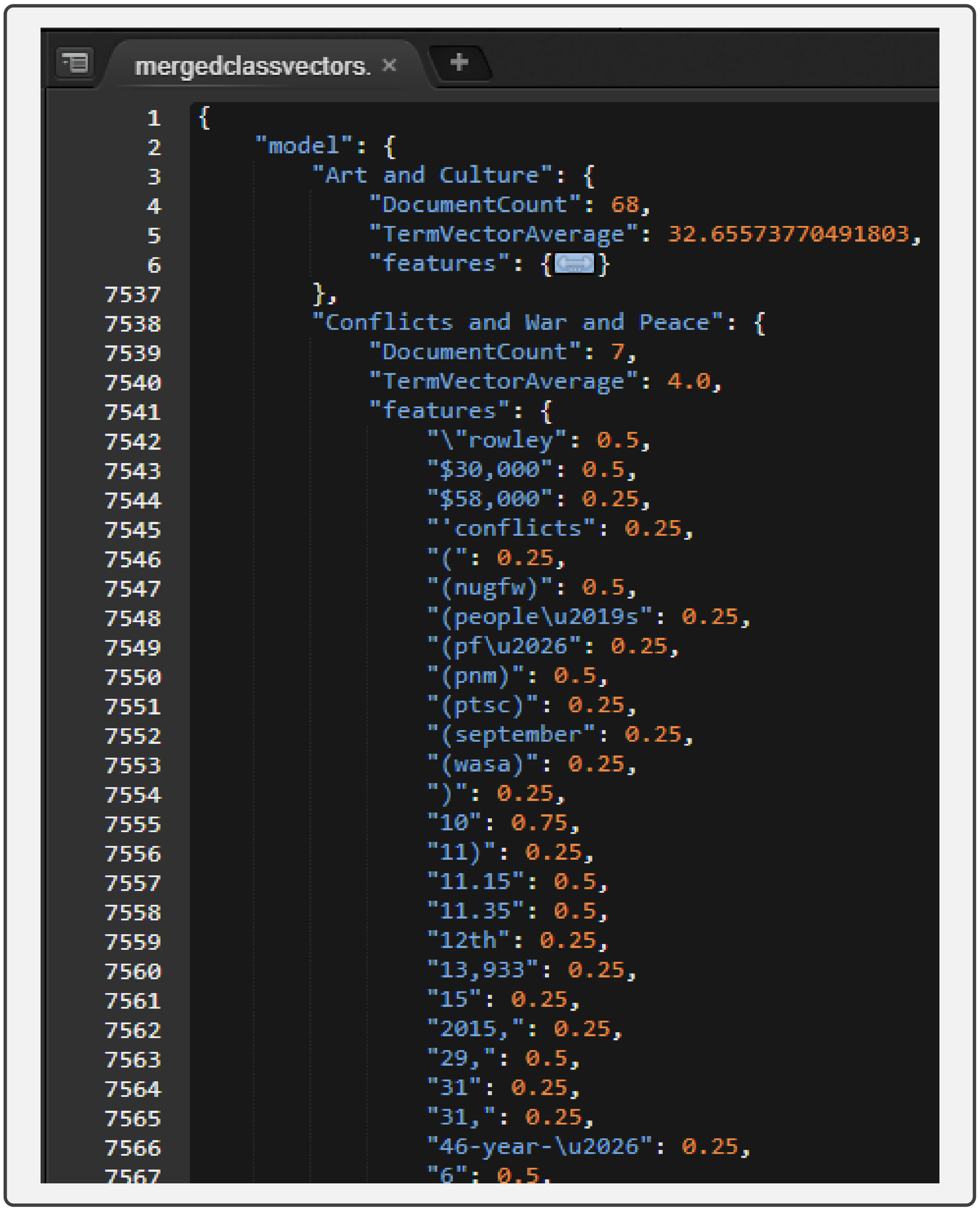
Figure 4.14: Script 'merge.py' For Merging Classification Models Using GBC Module

merge.py script shown in Figure 4.14 uses the GBC module which at the very core level of its 'Merger' functionality utilizes the incremental learning algorithm from section 3.8. The ‘Merger’ functionality imported from the GBC module (‘gbc.py’) merges classification models and not just individual class vectors. On lines 9-16 in Figure 4.14 some textual content for a new document was stored in a variable called ‘NewDocument’ that will be added to the text classification model. Though initially the new document is unlabeled, at run time it would be implicitly labeled as shown on line 19 of Figure 4.14. To merge the new document with the loaded model ‘model1’ shown in Figure 4.14 on line 7, a new model ‘model2’ is generated using the new document as shown in Figure 4.14 on line 20, then both are merged to give ‘model3’ on lines 25-31 in Figure 4.14. This approach is taken because instead of one new document being added it could have been multiple documents. In such a scenario it is more feasible to create a new model with these new documents then merge the new model with the old. ‘Model 3’ shown in Figure 4.16 shows the result of merging ‘Models 1 and 2’ and their class vectors of the same category ‘Conflicts and War and Peace’. Only category ‘Conflicts and War and Peace’ was affected because the new document corresponding to ‘Model 2’ (Figure 4.15) was implicitly tagged with only that category. For ‘Model 1’ (Figure 4.13), the *DocumentCount* with original value of ‘6’, and *TermVectorAverage* with an original value of ‘3.5’ on lines 7539 and 7540 respectively are updated to new values shown in ‘Model 3’ due to the merge between ‘Model 1’ and ‘Model 2’. In ‘Model 3’ the *Document-Count* increases to ‘7’ due to *DocumentCount* of ‘Model 1’ equal to ‘6’ been added with *DocumentCount* of ‘Model 2’ equal to ‘1’. The *TermVectorAverage* (unique class average) is updated from ‘3.5’ to ‘4.0’ due to the *destandardization* and *restandardization* steps of the incremental learning algorithm involved in merging class vector of ‘Model 1’ (features object of Model 1 on lines 7541) with the class vector of ‘Model 2’ (features object of Model 2 on lines 11).


newclassvectors.json

```
{
    "model": {
        "Art and Culture": {
            "DocumentCount": 0,
            "TermVectorAverage": 0,
            "features": {}
        },
        "Conflicts and War and Peace": {
            "DocumentCount": 1,
            "TermVectorAverage": 1.0,
            "features": {
                "\"rowley": 1.0,
                "'conflicts": 1.0,
                "(": 1.0,
                ")": 1.0,
                "and": 1.0,
                "as": 1.0,
                "category": 1.0,
                "document": 1.0,
                "ensure": 1.0,
                "example": 1.0,
                "functionality": 1.0,
                "gbc": 1.0,
                "gets": 1.0,
                "illustrate": 1.0,
                "implicitly": 1.0,
                "in": 1.0,
                "incremental": 1.0,
                "is": 1.0,
                "learning": 1.0,
                "merge": 1.0,
                "module.": 1.0,
                "name": 1.0,
                "of": 1.0,
                "part": 1.0,
                "peace'.": 1.0,
                "protest": 1.0,
                "roen\"": 1.0,
                "tagged": 1.0,
                "terms": 1.0,
                "terrorism": 1.0,
                "the": 1.0,
                "this": 1.0,
                "to": 1.0,
                "under": 1.0,
                "used": 1.0,
                "using": 1.0,
                "war": 1.0,
                "will": 1.0
            }
        },
        "Crime": {
```

Figure 4.15: Model 2 Containing Standardized Class Vectors From The New Document

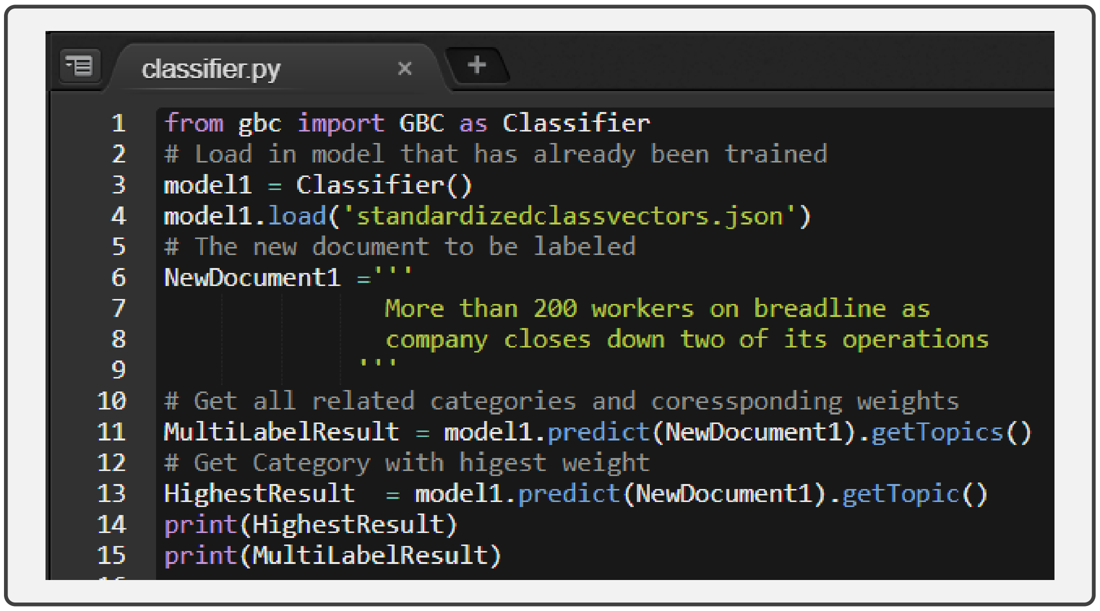


mergedclassvectors. ×

```
{
    "model": {
        "Art and Culture": {
            "DocumentCount": 68,
            "TermVectorAverage": 32.65573770491803,
            "features": {⟺}
        },
        "Conflicts and War and Peace": {
            "DocumentCount": 7,
            "TermVectorAverage": 4.0,
            "features": {
                "\"rowley": 0.5,
                "$30,000": 0.5,
                "$58,000": 0.25,
                "'conflicts": 0.25,
                "(": 0.25,
                "(nugfw)": 0.5,
                "(people\u2019s": 0.25,
                "(pf\u2026": 0.25,
                "(pnm)": 0.5,
                "(ptsc)": 0.25,
                "(september": 0.25,
                "(wasa)": 0.25,
                ")": 0.25,
                "10": 0.75,
                "11)": 0.25,
                "11.15": 0.5,
                "11.35": 0.5,
                "12th": 0.25,
                "13,933": 0.25,
                "15": 0.25,
                "2015,": 0.25,
                "29,": 0.5,
                "31": 0.25,
                "31,": 0.25,
                "46-year-\u2026": 0.25,
                "6": 0.5,
```

Figure 4.16: Model 3 Result Of Merging Model 1 and Model 2

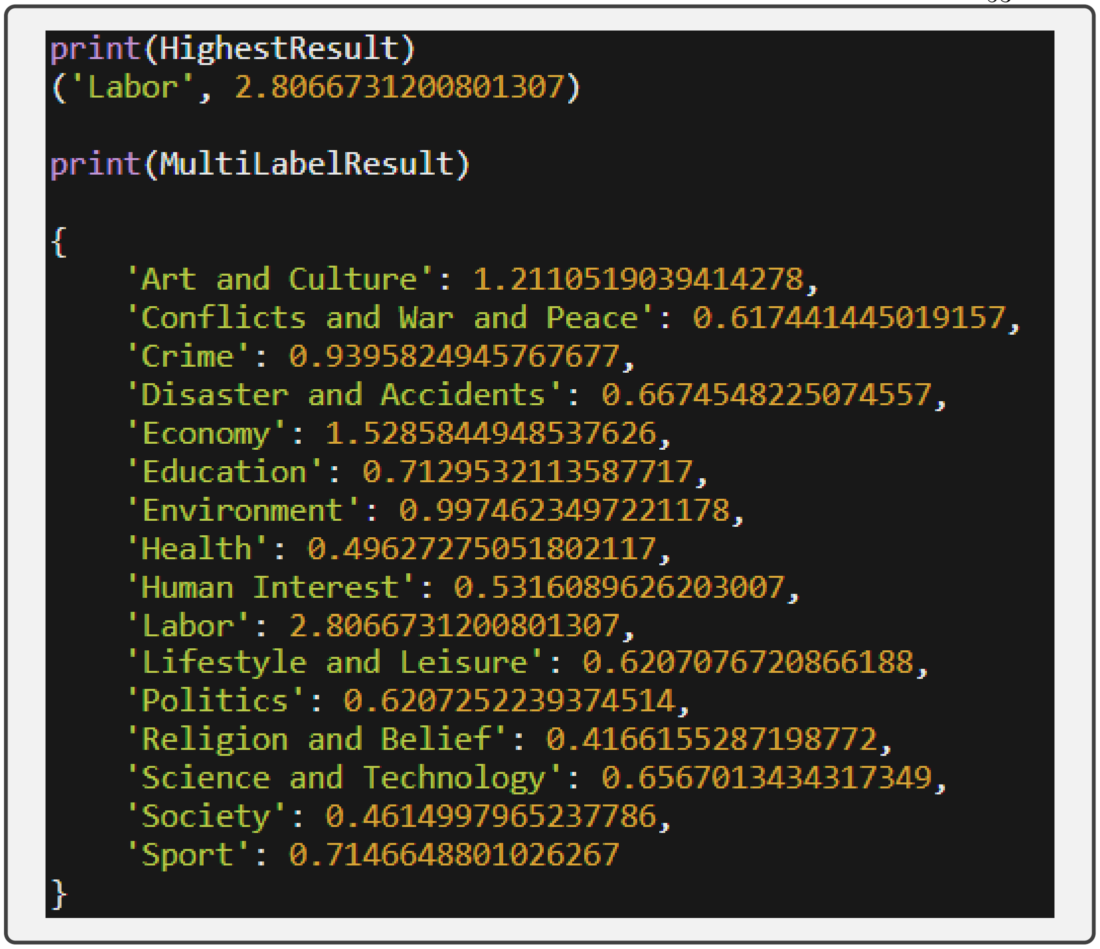
- **Document Classification Using The GBC Module:** The Classification algorithm for the GBC text classification model is shown in section 3.9.5. Using the GBC module, an already trained classification model is loaded as illustrated on line 4 of Figure 4.17. The new document 'NewDocument1' can then be categorized using the GBC module as illustrated from lines 10-13 in the following Figure 4.17.



```
1   from gbc import GBC as Classifier
2   # Load in model that has already been trained
3   model1 = Classifier()
4   model1.load('standardizedclassvectors.json')
5   # The new document to be labeled
6   NewDocument1 ='''
7                More than 200 workers on breadline as
8                company closes down two of its operations
9               '''
10  # Get all related categories and coressponding weights
11  MultiLabelResult = model1.predict(NewDocument1).getTopics()
12  # Get Category with higest weight
13  HighestResult  = model1.predict(NewDocument1).getTopic()
14  print(HighestResult)
15  print(MultiLabelResult)
```

Figure 4.17: Script 'classifier.py' Showing Document Classification Using GBC Module

The GBC module allows for the classifier to return either the category with the best weighting value 'HighestResult', or all related categories and their weights 'MultilabelResult'. The category predictions are shown in Figure 4.18.



```
print(HighestResult)
('Labor', 2.8066731200801307)

print(MultiLabelResult)

{
    'Art and Culture': 1.2110519039414278,
    'Conflicts and War and Peace': 0.617441445019157,
    'Crime': 0.9395824945767677,
    'Disaster and Accidents': 0.6674548225074557,
    'Economy': 1.5285844948537626,
    'Education': 0.7129532113587717,
    'Environment': 0.9974623497221178,
    'Health': 0.49627275051802117,
    'Human Interest': 0.5316089626203007,
    'Labor': 2.8066731200801307,
    'Lifestyle and Leisure': 0.6207076720866188,
    'Politics': 0.6207252239374514,
    'Religion and Belief': 0.4166155287198772,
    'Science and Technology': 0.6567013434317349,
    'Society': 0.4614997965237786,
    'Sport': 0.7146648801026267
}
```

Figure 4.18: Prediction Results For NewDocument1 In Figure 4.17

4.2.5 Model, Controller And Connected Modules

Figure 4.19 highlights the main classes which make up the backend implementation. All of the modules are imported and used by a class called *Modules*. The *Modules* class is used by the three classes: *Categorie*, *Article*, and *Topicmodel* which provide all of the functionality for interacting with the database and the GBC text classification model. The *Model* class uses multiple inheritance which is a feature the Python programming language supports. This feature allows the 'Model' object to access or inherit behaviours and characteristics from more than one parent class such as *Categorie*, *Article*, and *Topicmodel*. The *Controller* class uses the 'Model' class to get access to all the information that may be requested by the ‘View’. The variables and functions for GBC, Merger, and Crawler were not included in order to simplify the UML diagram.

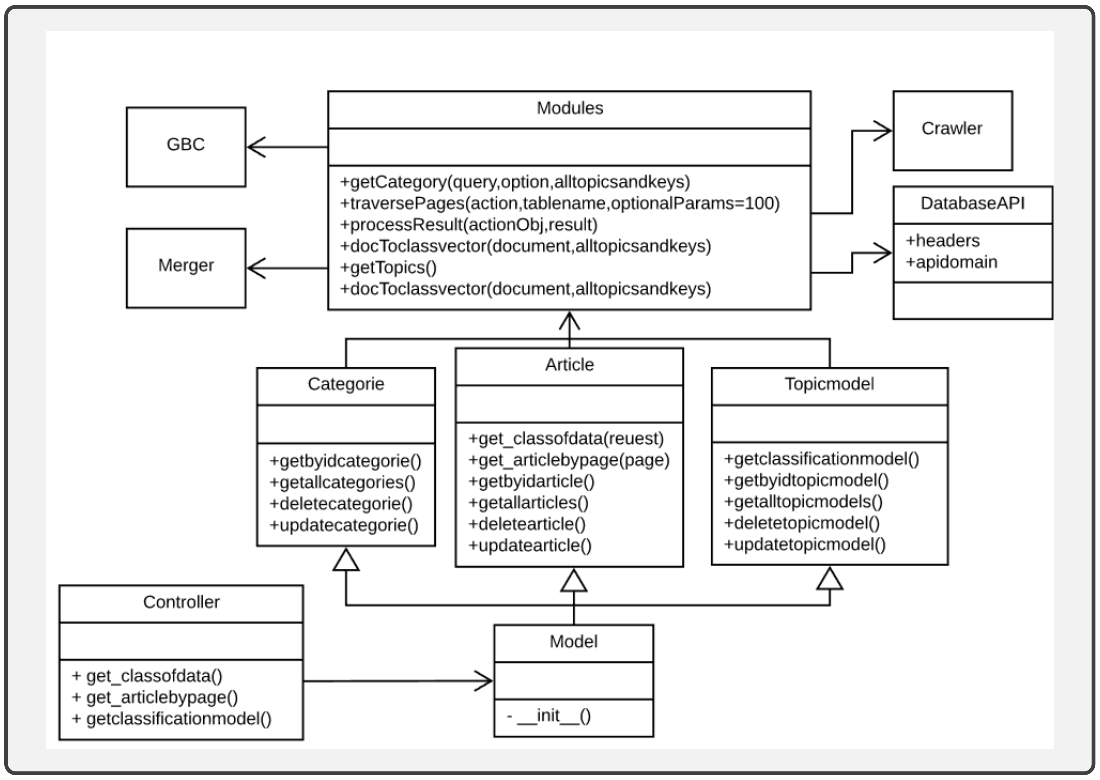
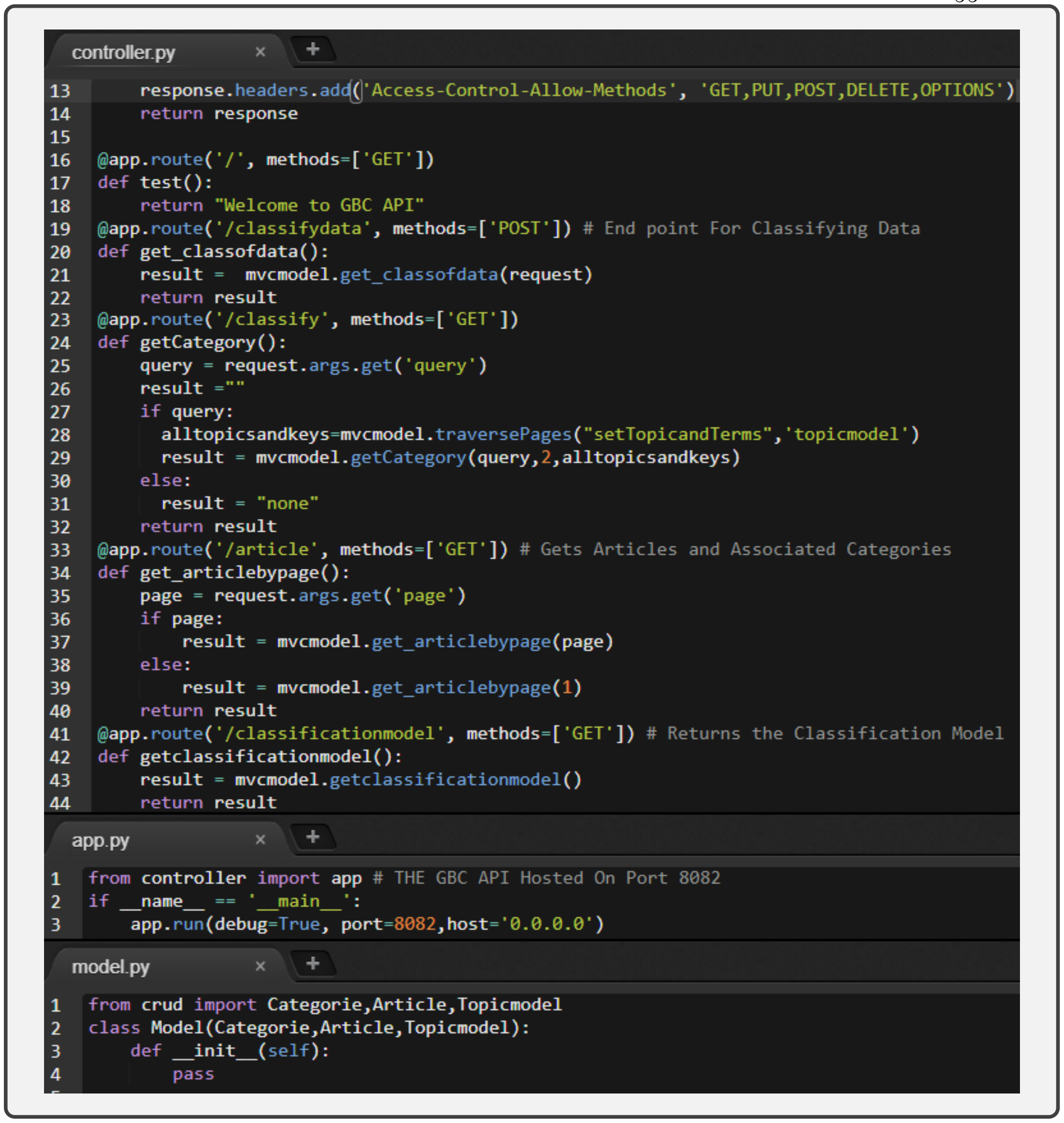


Figure 4.19: UML Diagram of The GBC System Backend Classes

Code Illustration for the ‘Model’ and ‘Controller’ is shown in Figure 4.20. As shown in Figure 4.20 there are three python scripts: ‘app.py’, ‘controller.py’ and ‘model.py’. The ‘app.py’ script is used to start the server, once started all the routes would be accessible via the port 8082 of the host machine. The ‘controller.py’ file handles all incoming request from the ‘View or Frontend’, it sends the query and gets response to and from the model through an instantiated ‘Model’ object shown on line 6 of ‘controller.py’ in Figure 4.20. ‘


controller.py

```
13     response.headers.add('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE,OPTIONS')
14     return response
15 
16 @app.route('/', methods=['GET'])
17 def test():
18     return "Welcome to GBC API"
19 @app.route('/classifydata', methods=['POST']) # End point For Classifying Data
20 def get_classofdata():
21     result =  mvcmodel.get_classofdata(request)
22     return result
23 @app.route('/classify', methods=['GET'])
24 def getCategory():
25     query = request.args.get('query')
26     result =""
27     if query:
28       alltopicsandkeys=mvcmodel.traversePages("setTopicandTerms",'topicmodel')
29       result = mvcmodel.getCategory(query,2,alltopicsandkeys)
30     else:
31       result = "none"
32     return result
33 @app.route('/article', methods=['GET']) # Gets Articles and Associated Categories
34 def get_articlebypage():
35     page = request.args.get('page')
36     if page:
37         result = mvcmodel.get_articlebypage(page)
38     else:
39         result = mvcmodel.get_articlebypage(1)
40     return result
41 @app.route('/classificationmodel', methods=['GET']) # Returns the Classification Model
42 def getclassificationmodel():
43     result = mvcmodel.getclassificationmodel()
44     return result
```

app.py

```
1 from controller import app # THE GBC API Hosted On Port 8082
2 if __name__ == '__main__':
3     app.run(debug=True, port=8082,host='0.0.0.0')
```

model.py

```
1 from crud import Categorie,Article,Topicmodel
2 class Model(Categorie,Article,Topicmodel):
3     def __init__(self):
4         pass
```

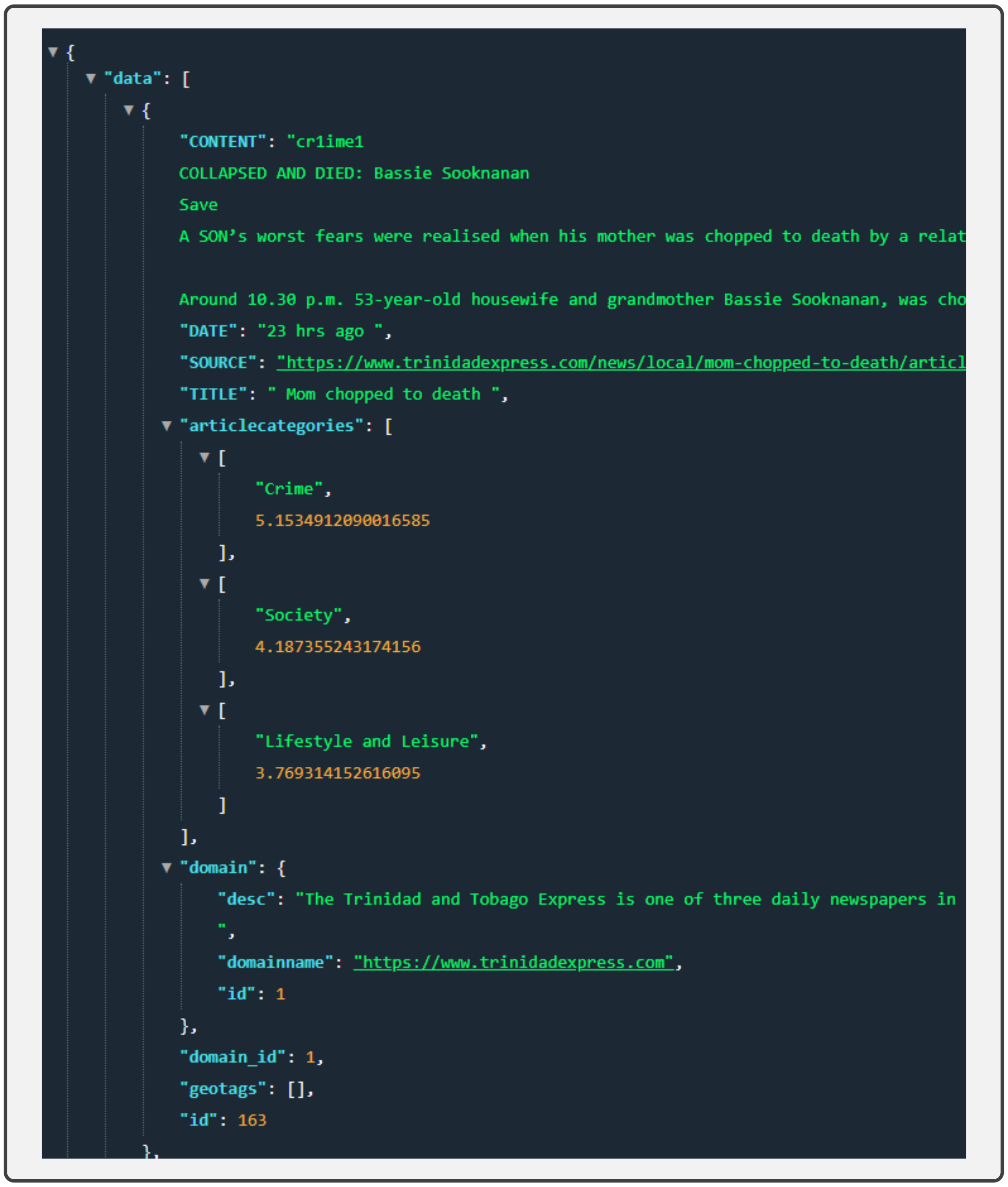
Figure 4.20: Model and Controller Code

4.2.5.1 The GBC REST API

The *controller.py* script shown in Figure 4.20 uses the following main routes */article*, '*/classificationmodel*, and */classifydata* on lines 33, 41, and 19 respectively to send and retrieve data.

- The article route */article* returns all articles and their top three categories. The route returns ten articles at a time and can be traversed by page using the

DatabaseAPI. An example of the article route is shown in Figure 4.21.

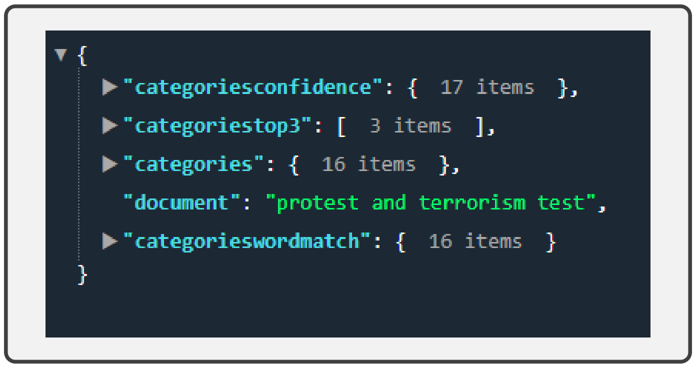


```
▼ {
  ▼ "data": [
    ▼ {
        "CONTENT": "cr1ime1
        COLLAPSED AND DIED: Bassie Sooknanan
        Save
        A SON’s worst fears were realised when his mother was chopped to death by a relat

        Around 10.30 p.m. 53-year-old housewife and grandmother Bassie Sooknanan, was cho
        "DATE": "23 hrs ago ",
        "SOURCE": "https://www.trinidadexpress.com/news/local/mom-chopped-to-death/articl
        "TITLE": " Mom chopped to death ",
      ▼ "articlecategories": [
        ▼ [
            "Crime",
            5.1534912090016585
          ],
        ▼ [
            "Society",
            4.187355243174156
          ],
        ▼ [
            "Lifestyle and Leisure",
            3.769314152616095
          ]
        ],
      ▼ "domain": {
          "desc": "The Trinidad and Tobago Express is one of three daily newspapers in
          ",
          "domainname": "https://www.trinidadexpress.com",
          "id": 1
        },
        "domain_id": 1,
        "geotags": [],
        "id": 163
      },
```

Figure 4.21: Article Route

- The classification model route */classificationmodel* returns the text classification model, and allows users to access and download the model for their own purposes. The model is mostly similar to the one in Figure 4.13 however it contain more information based on the application usage and the implemented incremental learning functionality.

- The classify data route */classifydata* processes articles and other textual data submitted via a POST request to the server. In Figure 4.20 there is also a */classify* route which is the GET request version of */classifydata*. The response from */classify* would be the same for */classifydata*. The difference between the two is that */classifydata* handles POST requests and can classify content based on a submitted URL rather than just textual information. The */classifydata* route also triggers the incremental learning step and updates the text classification model when necessary. Illustration of the response from this API endpoint is shown using the */classify* route since both results are the same but only the processing done behind the scenes and the method of request are different. When the following GET request is issued: */classify?query= protest and terrorism test*, the results shown in Figure 4.22 are returned.



```
▼ {
   ▶ "categoriesconfidence": {  17 items  },
   ▶ "categoriestop3": [  3 items  ],
   ▶ "categories": {  16 items  },
     "document": "protest and terrorism test",
   ▶ "categorieswordmatch": {  16 items  }
  }
```

Figure 4.22: Response For Classification Routes: */classifydata* and */classify*

Figure 4.23 shows the details of the *categoriesconfidence* object from Figure 4.22. The *categoriesconfidence* object shows what category or categories a piece of text implicitly gets tagged as. In Figure 4.23 the document ‘protest and terrorism test’ is implicitly categorized under only one category: ‘Conflicts and War and Peace’. This is because it contained at least two keywords from the ‘Conflicts and War and Peace’ IPTC category. The number ‘2’ assigned to the category ‘Conflicts and War and Peace’ is called the category confidence score. For a new document to be added to the text classification model during the incremental learning step, it must have an acceptable category confidence score for one or more categories. An acceptable category confidence score is determined by the value passed to the *MinKey()* function discussed in section 4.2.4.1. When a new text gets implicitly tagged its term vector gets merged with the text classification model based on the incremental learning algorithm in section 3.8 and by using the GBC Merger illustrated in Figure 4.14 .

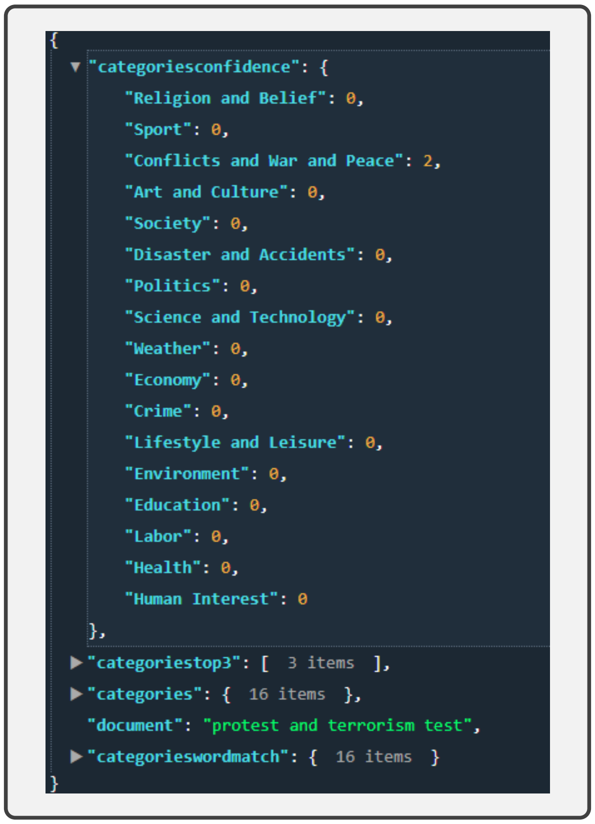
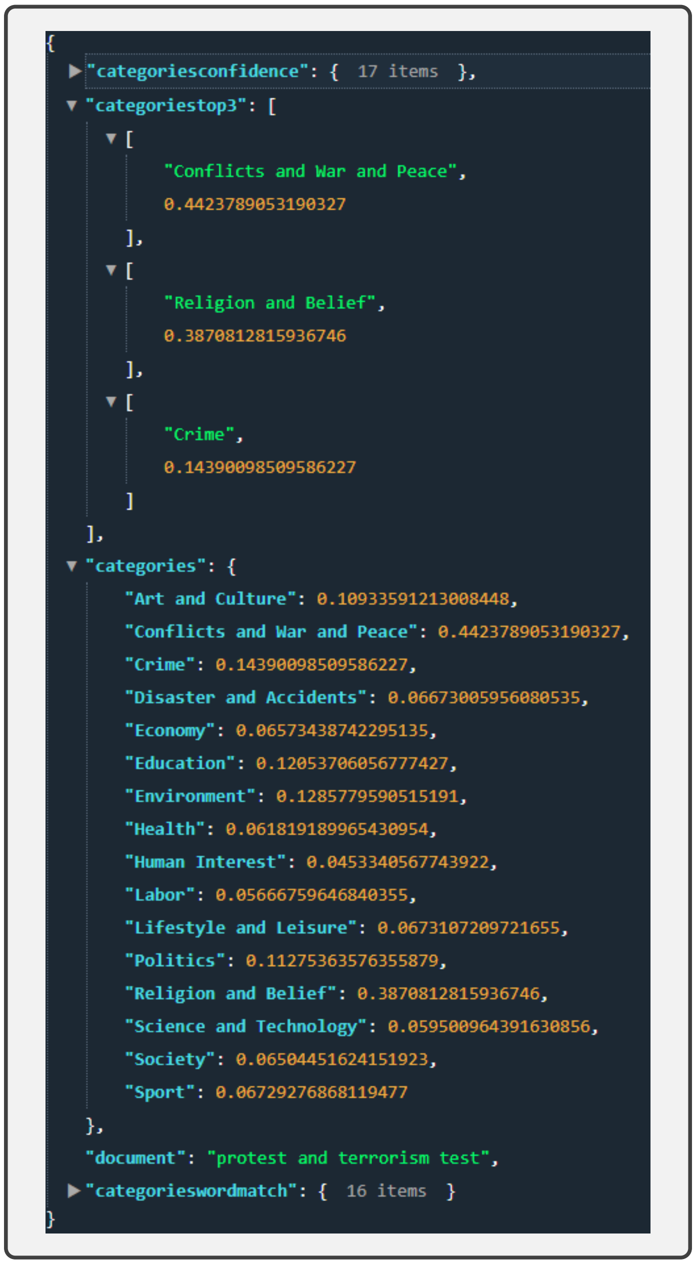

```
{
  ▼ "categoriesconfidence": {
      "Religion and Belief": 0,
      "Sport": 0,
      "Conflicts and War and Peace": 2,
      "Art and Culture": 0,
      "Society": 0,
      "Disaster and Accidents": 0,
      "Politics": 0,
      "Science and Technology": 0,
      "Weather": 0,
      "Economy": 0,
      "Crime": 0,
      "Lifestyle and Leisure": 0,
      "Environment": 0,
      "Education": 0,
      "Labor": 0,
      "Health": 0,
      "Human Interest": 0
  },
  ▶ "categoriestop3": [ 3 items ],
  ▶ "categories": { 16 items },
    "document": "protest and terrorism test",
  ▶ "categorieswordmatch": { 16 items }
}
```

Figure 4.23: Implicit Tagging Based On Category Confidence

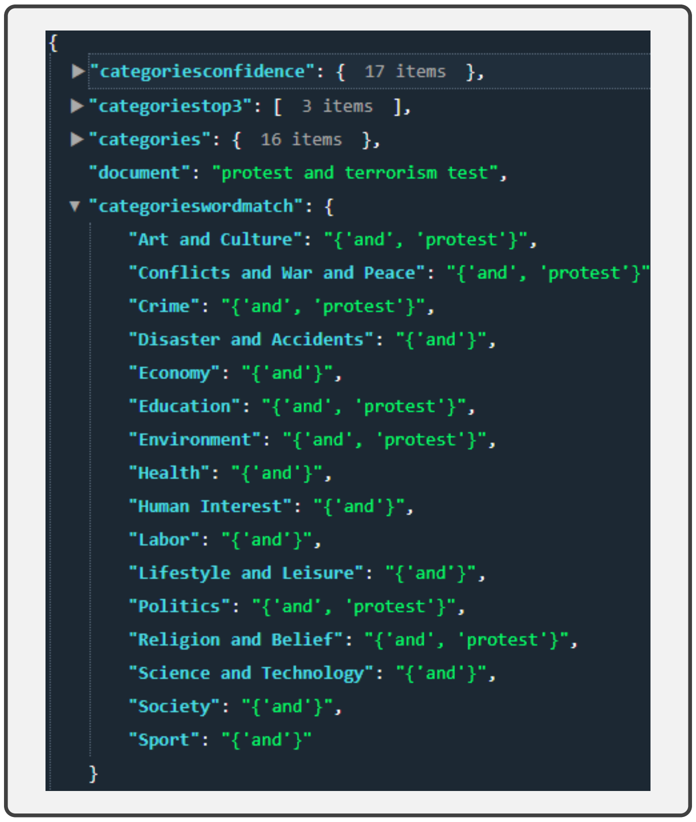
Figure 4.24 shows the details of the *categoriestop3* and *categories* objects from Figure 4.22. In Figure 4.24 the *categoriestop3* and *categories* displays the top 3 categories the document was classified as, and also returns the other related categories respectively.



```
{
  ▶ "categoriesconfidence": {  17 items  },
  ▼ "categoriestop3": [
      ▼ [
            "Conflicts and War and Peace",
            0.4423789053190327
        ],
      ▼ [
            "Religion and Belief",
            0.3870812815936746
        ],
      ▼ [
            "Crime",
            0.14390098509586227
        ]
    ],
  ▼ "categories": {
        "Art and Culture": 0.10933591213008448,
        "Conflicts and War and Peace": 0.4423789053190327,
        "Crime": 0.14390098509586227,
        "Disaster and Accidents": 0.06673005956080535,
        "Economy": 0.06573438742295135,
        "Education": 0.12053706056777427,
        "Environment": 0.1285779590515191,
        "Health": 0.061819189965430954,
        "Human Interest": 0.0453340567743922,
        "Labor": 0.05666759646840355,
        "Lifestyle and Leisure": 0.0673107209721655,
        "Politics": 0.11275363576355879,
        "Religion and Belief": 0.3870812815936746,
        "Science and Technology": 0.059500964391630856,
        "Society": 0.06504451624151923,
        "Sport": 0.06729276868119477
    },
    "document": "protest and terrorism test",
  ▶ "categorieswordmatch": {  16 items  }
}
```

Figure 4.24: Predicted Categories

In Figure 4.25 the details of the *categorieswordmatch* object from Figure 4.22 is shown. The *document* object shown in Figures (4.22 - 4.25) contains the textual content submitted or the content that would have been extracted from the URL source. The *categorieswordmatch* object in Figure 4.25 shows what terms from the *document* object was found in each of the class vectors. The results for the *categorieswordmatch* object corresponds to finding the related class vectors mentioned in section 3.9.1



```
{
  ▶"categoriesconfidence": {  17 items  },
  ▶"categoriestop3": [  3 items  ],
  ▶"categories": {  16 items  },
   "document": "protest and terrorism test",
  ▼"categorieswordmatch": {
      "Art and Culture": "{'and', 'protest'}",
      "Conflicts and War and Peace": "{'and', 'protest'}"
      "Crime": "{'and', 'protest'}",
      "Disaster and Accidents": "{'and'}",
      "Economy": "{'and'}",
      "Education": "{'and', 'protest'}",
      "Environment": "{'and', 'protest'}",
      "Health": "{'and'}",
      "Human Interest": "{'and'}",
      "Labor": "{'and'}",
      "Lifestyle and Leisure": "{'and'}",
      "Politics": "{'and', 'protest'}",
      "Religion and Belief": "{'and', 'protest'}",
      "Science and Technology": "{'and'}",
      "Society": "{'and'}",
      "Sport": "{'and'}"
   }
```

Figure 4.25: Related Class Vectors Via Word Match

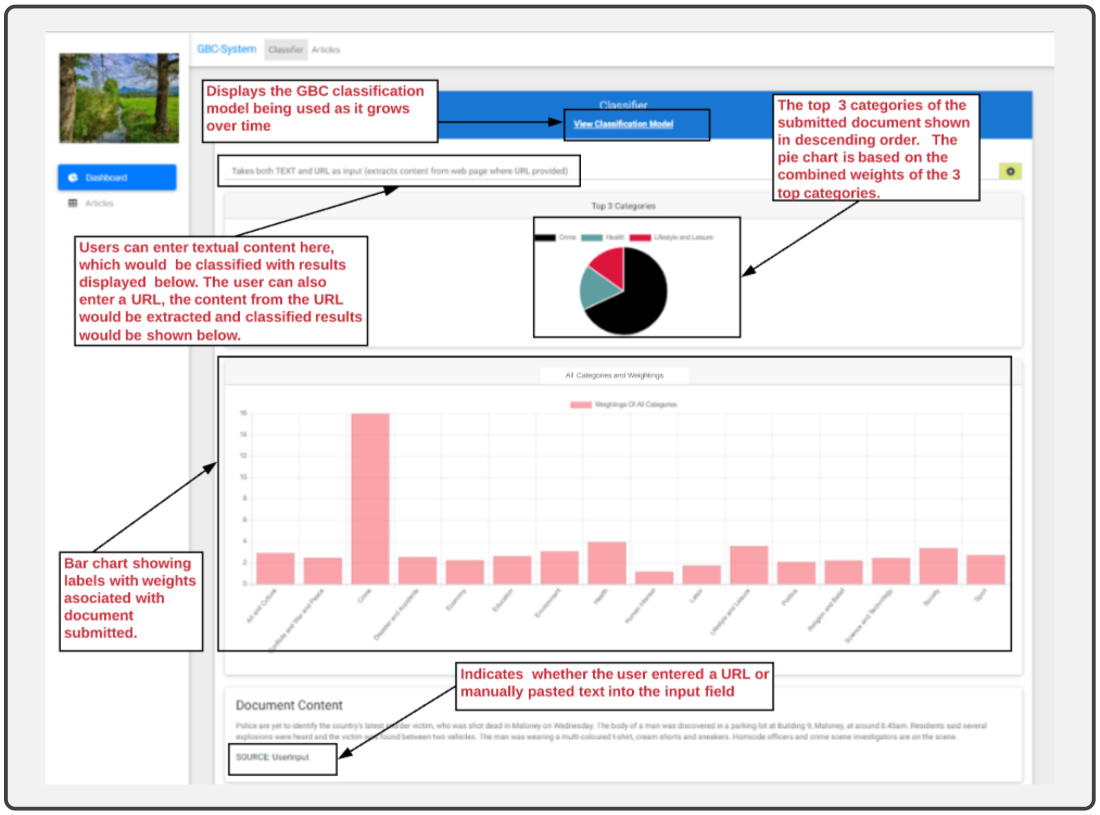
4.3 Frontend Implementation

The frontend of the GBC system was hosted on its own Amazon EC2 instance (A.4) and was developed using ‘Material Design for Bootstrap (Vue version)’ and ‘Vue.js’. These technologies are explained in table A.4. The frontend implementation consists of a user interface that can be split into two main views: ‘The ‘Classifier View’ and the

'Article Listing View'.

4.3.1 The 'Classifier View'

Figure 4.26 shows the 'Classifier View' and annotations of each component of the 'Classifier View'. The main purpose of the 'Classifier View' is to allow users to classify online news articles or other textual content. Results for the classification is shown in the 'Classifier View'. The 'classifier View' utilizes the */classifydata* route for getting content, like classification results, to be displayed to the user and for sending queries,'URL or Textual Content', to be classified. The 'Classifier View' also uses the */classificationmodel* route to display the GBC text classification model being used and updated.



Displays the GBC classification model being used as it grows over time

The top 3 categories of the submitted document shown in descending order. The pie chart is based on the combined weights of the 3 top categories.

Takes both TEXT and URL as input (extracts content from web page where URL provided)

Users can enter textual content here, which would be classified with results displayed below. The user can also enter a URL, the content from the URL would be extracted and classified results would be shown below.

Bar chart showing labels with weights asociated with document submitted.

Indicates whether the user entered a URL or manually pasted text into the input field

Document Content

Police are yet to identify the country's latest murder victim, who was shot dead in Maloney on Wednesday. The body of a man was discovered in a parking lot at Building 9, Maloney, at around 8.45am. Residents said several explosions were heard and the victim was found between two vehicles. The man was wearing a multi-coloured t-shirt, cream shorts and sneakers. Homicide officers and crime scene investigators are on the scene.

SOURCE: UserInput

Figure 4.26: Frontend Classifier View of The GBC System

4.3.1.1 Incremental Learning Frontend Illustration

An example of how incremental learning is conducted using the frontend is described in the following steps.

- STEP 1: Text is entered into the classifier views text-box in Figure 4.27. The text added is ‘Lowlands’ which is a location in Tobago. This text was chosen to illustrate that the model is initially unable to classify it because the model does not have any data relevant to it as shown in Figure 4.27 and 4.28 below.

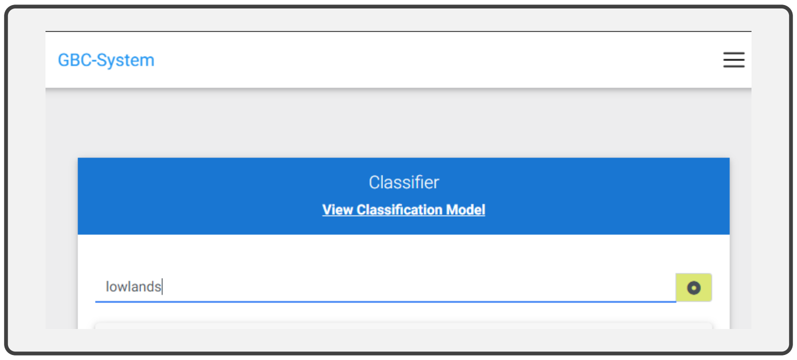


Figure 4.27: Step 1 enter text

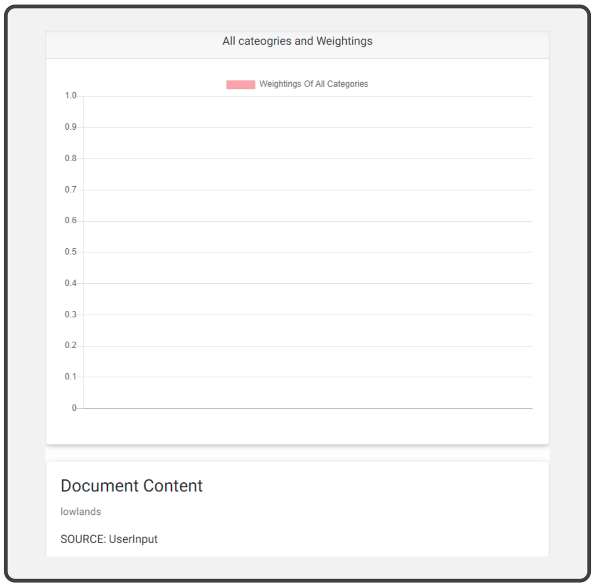
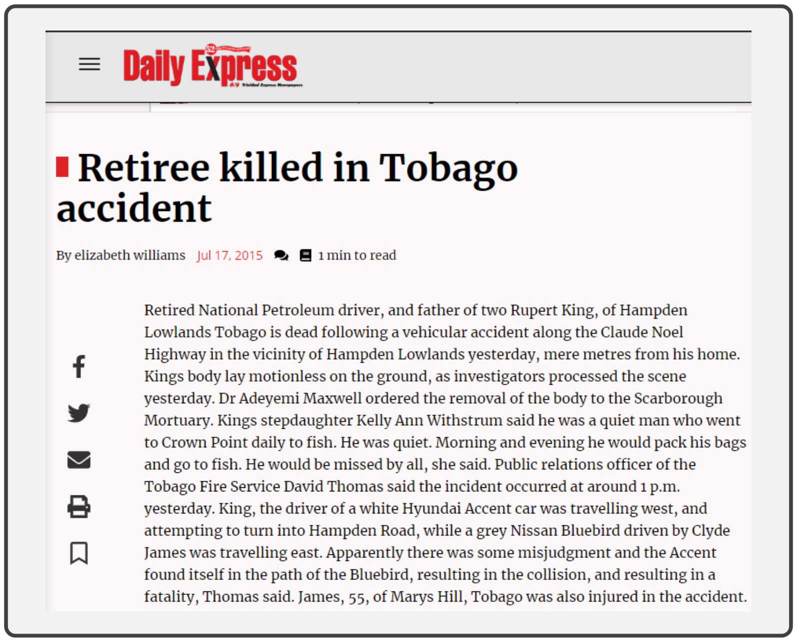


Figure 4.28: Step 1 Results for Lowlands Search

- STEP 2: The user searches for an article with the word ‘lowlands’ in it, or searches for an article that is about the word ‘lowlands’. The article chosen is shown in Figure 4.29.



Daily Express

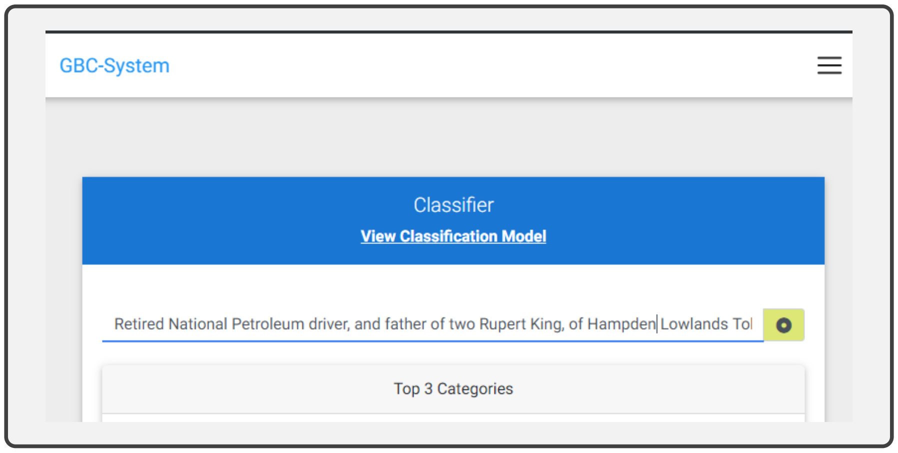
Retiree killed in Tobago accident

By elizabeth williams Jul 17, 2015 1 min to read

Retired National Petroleum driver, and father of two Rupert King, of Hampden Lowlands Tobago is dead following a vehicular accident along the Claude Noel Highway in the vicinity of Hampden Lowlands yesterday, mere metres from his home. Kings body lay motionless on the ground, as investigators processed the scene yesterday. Dr Adeyemi Maxwell ordered the removal of the body to the Scarborough Mortuary. Kings stepdaughter Kelly Ann Withstrum said he was a quiet man who went to Crown Point daily to fish. He was quiet. Morning and evening he would pack his bags and go to fish. He would be missed by all, she said. Public relations officer of the Tobago Fire Service David Thomas said the incident occurred at around 1 p.m. yesterday. King, the driver of a white Hyundai Accent car was travelling west, and attempting to turn into Hampden Road, while a grey Nissan Bluebird driven by Clyde James was travelling east. Apparently there was some misjudgment and the Accent found itself in the path of the Bluebird, resulting in the collision, and resulting in a fatality, Thomas said. James, 55, of Marys Hill, Tobago was also injured in the accident.

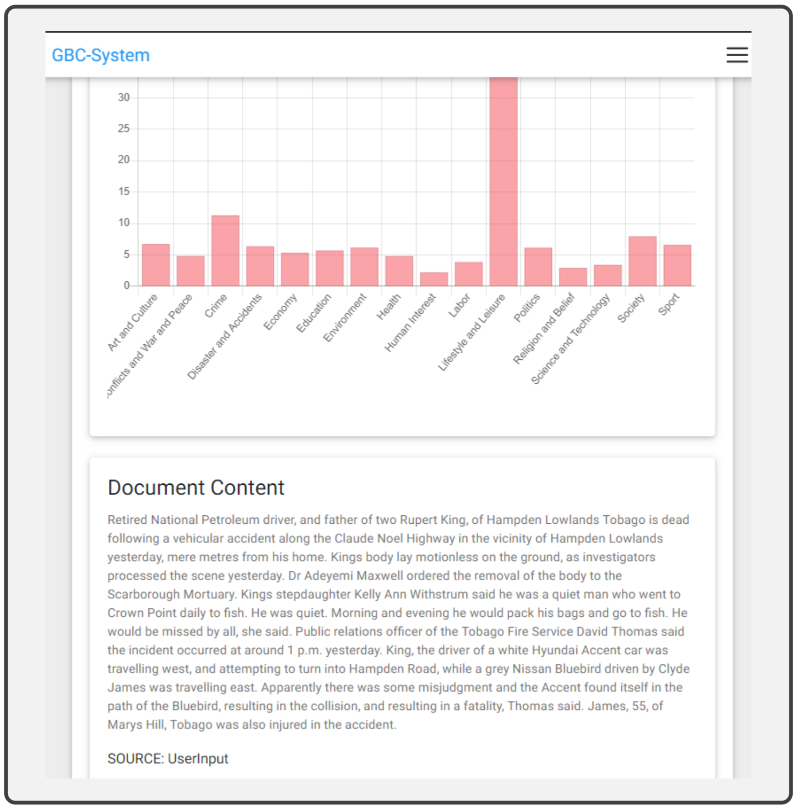
Figure 4.29: Step 2 Find Article about Lowlands

- STEP 3: The content of the article is copied and pasted into the classifier view text-box as shown in Figure 4.30.



Retired National Petroleum driver, and father of two Rupert King, of Hampden Lowlands Tol

Figure 4.30: Step 3 Submit Article Content Containing the term ‘lowlands’



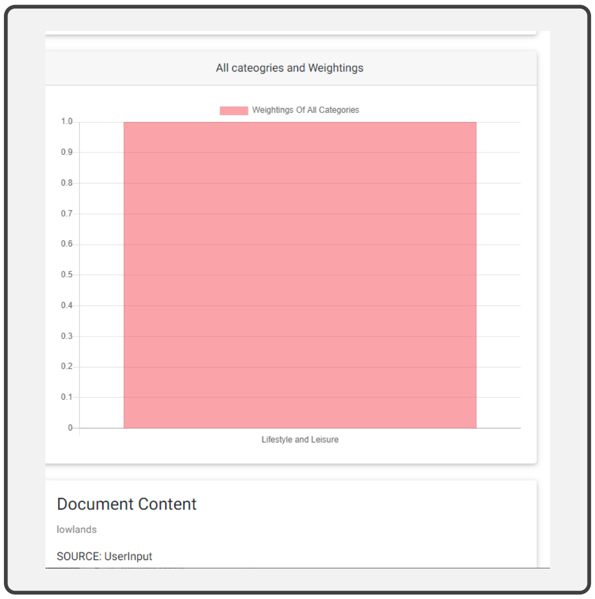
Document Content

Retired National Petroleum driver, and father of two Rupert King, of Hampden Lowlands Tobago is dead following a vehicular accident along the Claude Noel Highway in the vicinity of Hampden Lowlands yesterday, mere metres from his home. Kings body lay motionless on the ground, as investigators processed the scene yesterday. Dr Adeyemi Maxwell ordered the removal of the body to the Scarborough Mortuary. Kings stepdaughter Kelly Ann Withstrum said he was a quiet man who went to Crown Point daily to fish. He was quiet. Morning and evening he would pack his bags and go to fish. He would be missed by all, she said. Public relations officer of the Tobago Fire Service David Thomas said the incident occurred at around 1 p.m. yesterday. King, the driver of a white Hyundai Accent car was travelling west, and attempting to turn into Hampden Road, while a grey Nissan Bluebird driven by Clyde James was travelling east. Apparently there was some misjudgment and the Accent found itself in the path of the Bluebird, resulting in the collision, and resulting in a fatality, Thomas said. James, 55, of Marys Hill, Tobago was also injured in the accident.

SOURCE: UserInput

Figure 4.31: Step 3 Results

- STEP 4: Repeat step 1 by entering the word ‘lowlands’. Once more Figure 4.32 shows that ‘lowlands’ article was assigned to a category ‘lifestyle and leisure’. This result is now available because the previous article in Figure 4.29 was added to the GBC model.



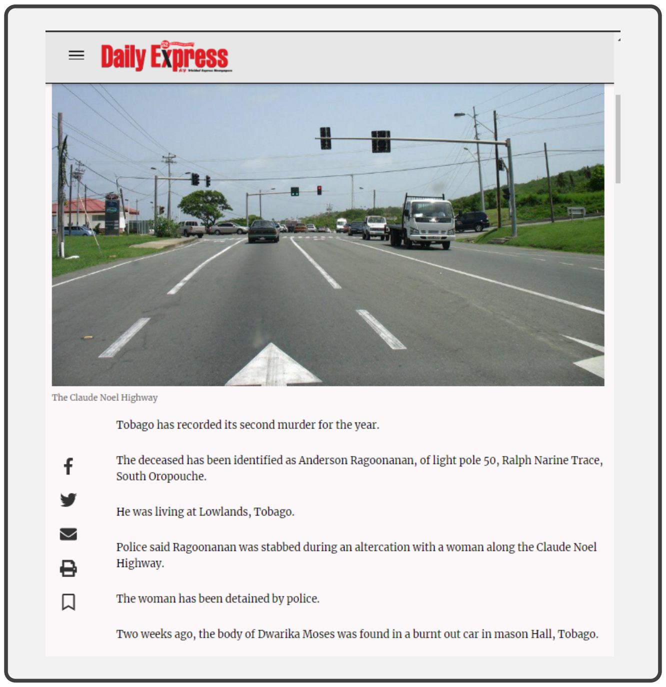
Document Content

lowlands

SOURCE: UserInput

Figure 4.32: Step 4 Category of 'lowlands'

- STEP 5: For further illustration, another article shown in Figure 4.33 is added. The classification results for the new article is shown in Figure 4.34.



The Claude Noel Highway

Tobago has recorded its second murder for the year.

The deceased has been identified as Anderson Ragoonanan, of light pole 50, Ralph Narine Trace, South Oropouche.

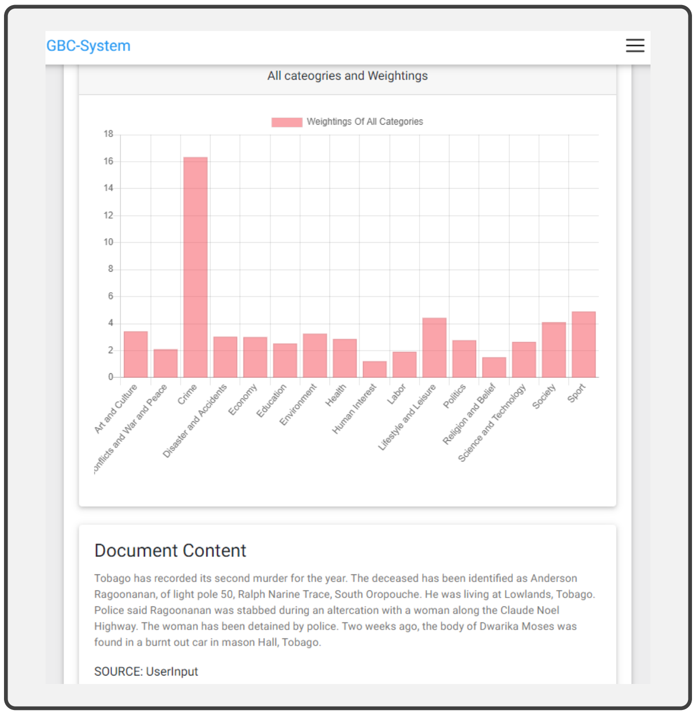
He was living at Lowlands, Tobago.

Police said Ragoonanan was stabbed during an altercation with a woman along the Claude Noel Highway.

The woman has been detained by police.

Two weeks ago, the body of Dwarika Moses was found in a burnt out car in mason Hall, Tobago.

Figure 4.33: Step 5 Enter Another Article Containing 'lowlands'



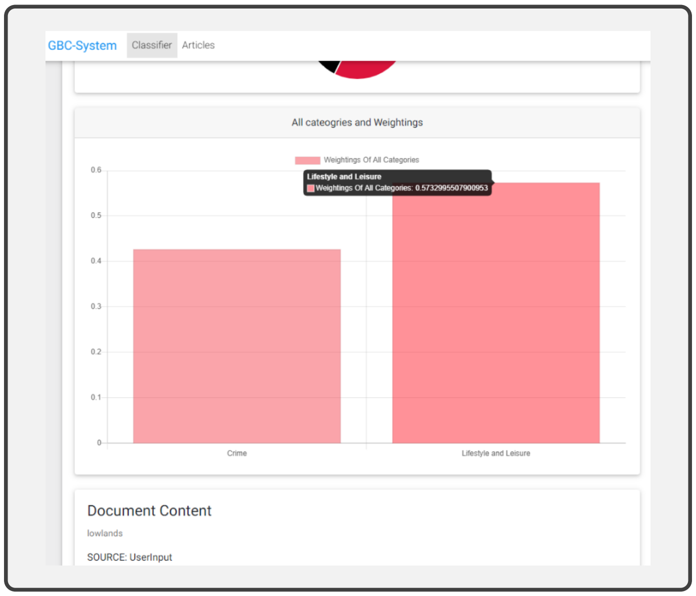
Document Content

Tobago has recorded its second murder for the year. The deceased has been identified as Anderson Ragoonanan, of light pole 50, Ralph Narine Trace, South Oropouche. He was living at Lowlands, Tobago. Police said Ragoonanan was stabbed during an altercation with a woman along the Claude Noel Highway. The woman has been detained by police. Two weeks ago, the body of Dwarika Moses was found in a burnt out car in mason Hall, Tobago.

SOURCE: UserInput

Figure 4.34: Step 5 Results

- STEP 6: After the new article was added in step 5, the text ‘lowlands’ is submitted again. The new classification results are shown in Figure 4.35.


GBC-System
Classifier
Articles
All cateogries and Weightings
Weightings Of All Categories
Lifestyle and Leisure
Weightings Of All Categories: 0.5732995507900953
0.6
0.5
0.4
0.3
0.2
0.1
0
Crime
Lifestyle and Leisure

Document Content

lowlands

SOURCE: UserInput

Figure 4.35: Step 6 New Result For Entering ‘lowlands’

Figure 4.35 shows the end results of the incremental learning process using the word ‘lowlands’. RR Adeand Deshmukh (2013) stated that an incremental learning algorithm should:

(1) Preserve previously acquired knowledge: this was demonstrated whereby the term ‘lowlands’ maintained its category of ‘lifestyle and leisure’ even as new data was added. Data was not lost or removed from classification model.

(2) Not require access to the original data: as new articles were added, the model was not retrained but was updated with the new data.

(3) Be dynamic in nature with the changing environment: As new data is added the model behaviour adapts to the new data. As shown with ‘lowlands’, the system was able to classify the term after new data was added.

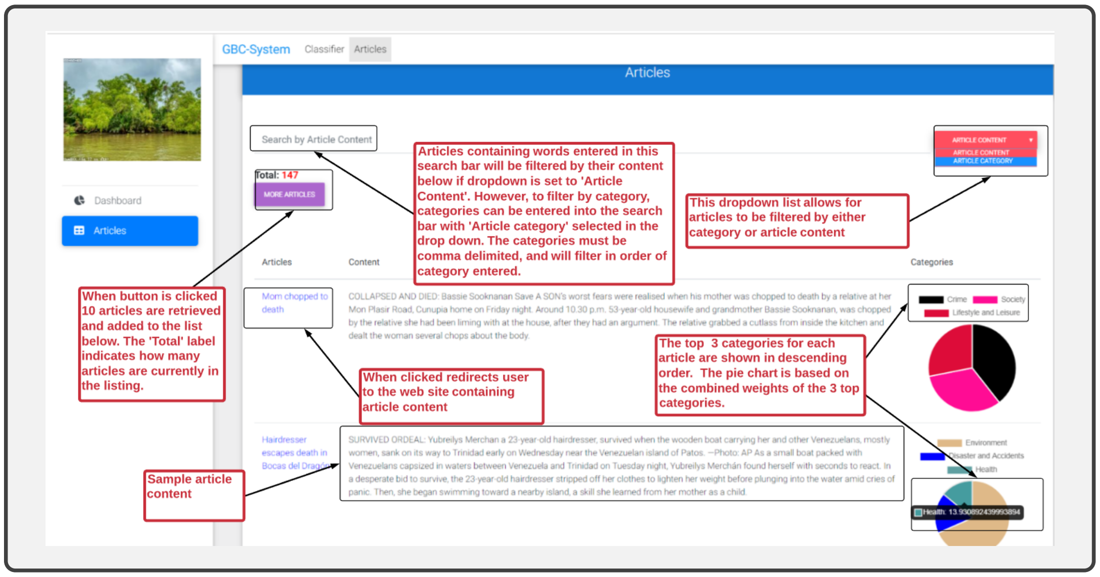
(4) Be capable of generating new classes: In Figure 4.35 after the second article was added, the term ‘lowlands’ was assigned to the ‘crime’ class in addition to the ‘lifestyle and leisure’ class. Also it is possible to add a new class (class vector) to the model without affecting the rest of the model because each class vector can exist independently of each other.

(5) Be able to learn and update with every new data: this was clearly demonstrated as ‘lowlands’ was assigned categories as new data was added. Also between Figure 4.35 and Figure 4.33 the weight of ‘lowlands’ in the ‘lifestyle and leisure’ category is shown to have increased due to the new data added.

In addition to the incremental learning behaviour it is shown that geographical information retrieval is possible as categorical information was returned for the term ‘lowlands’ which is a location in Trinidad and Tobago.

4.3.2 The ‘Article Listing View’

The ‘Article Listing View’ is illustrated in Figure 4.36, the figure contains annotations that describe the main components of the ‘Article Listing View’. The main purpose of the ‘Article Listing View’ is to allow users to view and filter articles by category and article content. These are online news articles that would have been added to the system by the system administrator or by clients that were testing out the GBC API. The ‘Article Listing View’ uses the ‘/article’ route for retrieving articles from the database. Each article being retrieved is always classified in real time. There is an option for the categories to be saved in the database however this feature was not utilized. Classifying the articles in real time ensures that the classifications are made based on the most recent version of the text classification model.



Articles containing words entered in this search bar will be filtered by their content below if dropdown is set to 'Article Content'. However, to filter by category, categories can be entered into the search bar with 'Article category' selected in the drop down. The categories must be comma delimited, and will filter in order of category entered.

This dropdown list allows for articles to be filtered by either category or article content

When button is clicked 10 articles are retrieved and added to the list below. The 'Total' label indicates how many articles are currently in the listing.

When clicked redirects user to the web site containing article content

The top 3 categories for each article are shown in descending order. The pie chart is based on the combined weights of the 3 top categories.

Sample article content

Mom chopped to death

COLLAPSED AND DIED: Bassie Sooknanan Save A SON's worst fears were realised when his mother was chopped to death by a relative at her Mon Plasir Road, Cunupia home on Friday night. Around 10.30 p.m. 53-year-old housewife and grandmother Bassie Sooknanan, was chopped by the relative she had been liming with at the house, after they had an argument. The relative grabbed a cutlass from inside the kitchen and dealt the woman several chops about the body.

Hairdresser escapes death in Bocas del Dragón

SURVIVED ORDEAL: Yubreilys Merchan a 23-year-old hairdresser, survived when the wooden boat carrying her and other Venezuelans, mostly women, sank on its way to Trinidad early on Wednesday near the Venezuelan island of Patos. —Photo: AP As a small boat packed with Venezuelans capsized in waters between Venezuela and Trinidad on Tuesday night, Yubreilys Merchán found herself with seconds to react. In a desperate bid to survive, the 23-year-old hairdresser stripped off her clothes to lighten her weight before plunging into the water amid cries of panic. Then, she began swimming toward a nearby island, a skill she learned from her mother as a child.

Figure 4.36: Frontend Article Listing View

4.3.3 Use Case Diagram

The following Use Case Diagram (Figure 4.37) illustrates how a user would go about interacting with the system.

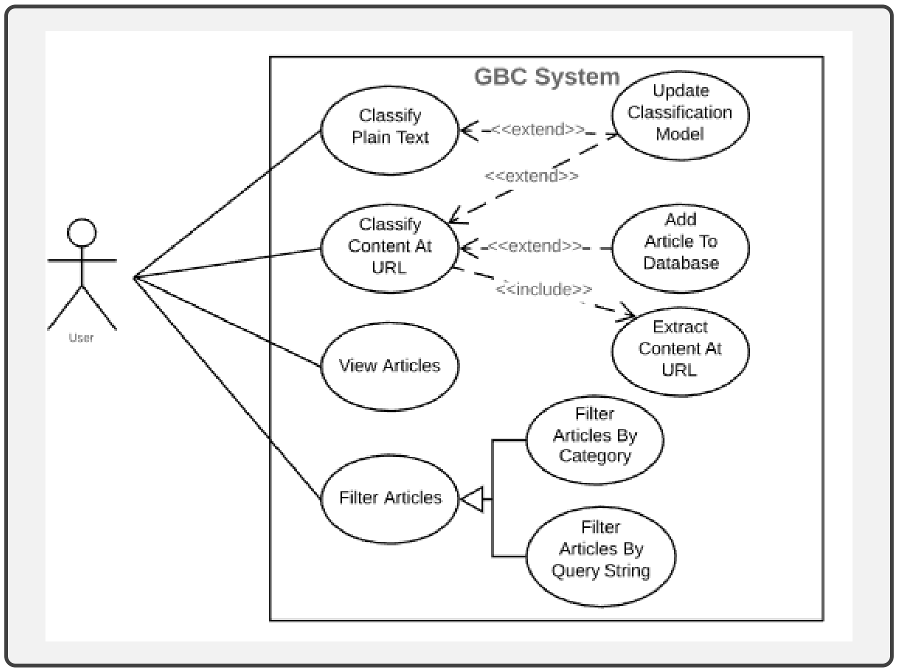


Figure 4.37: GBC System Use Case Diagram

Chapter 5

Tests and Experiments

5.1 The Data Sets

The GBC classifier/model was evaluated by comparing its classification performance with that of Multinomial Naive Bayes classifier. This classifier is one of the members of the family of Bayes classifiers and is often used for document classification. This evaluation required that both classifiers use the same data sets for training and classifying. One of the data sets used was the '20 newsgroups text data set' (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, et al.). This data set is free and easily accessible through the Scikit-learn [1] library. It contains approximately 18000 newsgroups posts on 20 topics. It is split in two subsets: one for performance evaluation and the other for training the model being evaluated. This data set has often been used in research for evaluating experimental models hence why it was chosen. The other data set which was used was the 'smsspam data set' [2] which contains sms messages that would have been labeled as either harmless or spam by humans. This data set was obtained from the 'UC Irvine Machine Learning Repository' (Dua and Graff 2017a). It is a collection of domain theories, databases, and data generators which are used by the machine learning community for empirically analyzing machine learning algorithms (Dua and Graff 2017b).

1. https://scikit-learn.org/0.19/datasets/
2. https://archive.ics.uci.edu/ml/datasets/sms+spam+collection

5.2 Tf-Idf Brief Summary

The Multinomial Naive Bayes model discussed in section 2.2, uses Tf-IDF weights to improve its classification performance (Kibriya et al. 2004). Tf-IDF stands for term frequency inverse document frequency. It is a numerical statistic that reflects the importance of a word within a document within a corpus or collection. When Naive Bayes is used to classify documents, each word in each document is counted as one. However when Tf-Idf is used, the TF-IDF weights of the words in documents are used instead of a frequency count.

5.3 Comparing GBC model With Multinomial Naive Bayes

The comparison between the GBC model and the Naive Bayes model was done by comparing the GBC model with both the Multinomial Naive Bayes with normal weights (word count) and the Multinomial Naive Bayes with Tf-IDF weights. The code and environment for both classifiers were written and configured respectively. The code used for testing[3] is available at the git-hub repository associated with this project[4] . Both classifiers were written in python, however the GBC module was used for training the GBC classification model, while functionality from scikit-learn (Buitinck et al. 2013) was used for training the Naive Bayes Classifier. Scikit-learn is a free software machine learning library for the Python programming language.

3. https://github.com/DavidDexterCharles/GroupByClassModel/tree/master/gbcTEST
4. https://github.com/DavidDexterCharles/GroupByClassModel

5.4 Classification Performance

The classification performance of a model is evaluated using the models accuracy, precision, recall and F1 score.

- Accuracy: is calculated as the ratio of the number of correct predictions made to the total number of predictions made
- Precision: (also called Positive Predictive Value), is calculated as the ratio of correct positive predictions to the total predicted positives.
- Recall: (also called Sensitivity, True Positive Rate, or Probability of Detection), is the ratio of correct positive predictions to the total positive examples.
- F1 Score: The F1 score is the weighted average of the precision and recall, this means that it considers both false negatives and false positives. An F1 score's best value is 1 which represents perfect recall and precision. The worst F1 score is at 0.

5.5 Testing With the 'fetch_20newsgroup' Dataset

The 'fetch_20newsgroup' had data already split into training and test data. The first testing phase involved training the models using the training data of the 'fetch_20newsgroup' data set. The models were used to make predictions on the test data of the 'fetch_20newsgroup' data set. The accuracy, precision, recall and F1 scores of both models were obtained and the results are shown in the following sub-sections. Table 5.1 summarizes the re-

						Weighted Average		
	Tf-IDF	Accuracy	Total	True	False	Precision	Recall	F1-Score
GBC	no	0.834	7532	6279	1253	0.84	0.83	0.83
Bayes	no	0.805	7532	6061	1471	0.81	0.80	0.79
Bayes	yes	0.838	7532	6310	1222	0.84	0.84	0.84

Table 5.1: Summary Results for Classifier Performance on 'fetch_20newsgroup' Dataset

sults from sections: 5.5.1, 5.5.2, and 5.5.3. The 'True' column shows the number of correct predictions while the 'False' column shows the number of incorrect predictions. The 'Total' column shows the number of documents used for testing the classification models.

5.5.1 Performance Results GBC Model

```
                   class    precision   recall  f1-score   support

             alt.atheism       0.81       0.73      0.77       319

               sci.space       0.84       0.92      0.88       394

           comp.graphics       0.76       0.70      0.73       389

      talk.religion.misc       0.77       0.57      0.66       251

   comp.os.ms-windows.misc     0.81       0.63      0.71       394

  comp.sys.ibm.pc.hardware     0.73       0.76      0.74       392

     comp.sys.mac.hardware     0.90       0.75      0.82       385

            comp.windows.x     0.79       0.90      0.84       395

              misc.forsale     0.88       0.83      0.86       390

                rec.autos      0.92       0.90      0.91       396

          rec.motorcycles      0.96       0.96      0.96       398

       rec.sport.baseball      0.96       0.92      0.94       397

         rec.sport.hockey      0.90       0.99      0.94       399

                sci.crypt      0.75       0.95      0.84       396

          sci.electronics      0.84       0.66      0.74       393

                  sci.med      0.85       0.84      0.85       396

    soc.religion.christian     0.79       0.93      0.86       398

       talk.politics.guns      0.77       0.93      0.84       364

    talk.politics.mideast      0.82       0.99      0.89       376

       talk.politics.misc      0.82       0.66      0.73       310


                 accuracy                           0.83      7532

                macro avg      0.83       0.83      0.83      7532

             weighted avg      0.84       0.83      0.83      7532
```

5.5.2 Performance Results Multinomial Naive Bayes

```
                   class      precision   recall  f1-score   support

             alt.atheism         0.80      0.82      0.81       319

               sci.space         0.87      0.89      0.88       394

           comp.graphics         0.57      0.79      0.66       389

      talk.religion.misc         0.67      0.65      0.66       251

 comp.os.ms-windows.misc         0.69      0.03      0.05       394

comp.sys.ibm.pc.hardware         0.55      0.78      0.64       392

   comp.sys.mac.hardware         0.72      0.84      0.78       385

          comp.windows.x         0.81      0.73      0.77       395

            misc.forsale         0.80      0.85      0.82       390

               rec.autos         0.87      0.90      0.89       396

         rec.motorcycles         0.92      0.96      0.94       398

      rec.sport.baseball         0.95      0.93      0.94       397

        rec.sport.hockey         0.96      0.96      0.96       399

               sci.crypt         0.89      0.93      0.91       396

         sci.electronics         0.76      0.76      0.76       393

                 sci.med         0.89      0.83      0.86       396

  soc.religion.christian         0.90      0.93      0.91       398

      talk.politics.guns         0.80      0.89      0.84       364

   talk.politics.mideast         0.98      0.89      0.93       376

      talk.politics.misc         0.73      0.64      0.68       310


                accuracy                             0.80      7532

               macro avg         0.81      0.80      0.79      7532

            weighted avg         0.81      0.80      0.79      7532
```

5.5.3 Performance Results Multinomial Naive Bayes With Tf-IDF

```
                   class      precision    recall  f1-score   support

             alt.atheism          0.83       0.78      0.81        319

               sci.space          0.86       0.92      0.89        394

           comp.graphics          0.71       0.74      0.73        389

      talk.religion.misc          0.80       0.54      0.64        251

 comp.os.ms-windows.misc          0.75       0.64      0.69        394

comp.sys.ibm.pc.hardware          0.66       0.77      0.71        392

   comp.sys.mac.hardware          0.84       0.84      0.84        385

          comp.windows.x          0.85       0.77      0.80        395

            misc.forsale          0.84       0.78      0.81        390

               rec.autos          0.90       0.91      0.90        396

         rec.motorcycles          0.93       0.97      0.95        398

      rec.sport.baseball          0.95       0.94      0.95        397

        rec.sport.hockey          0.95       0.97      0.96        399

               sci.crypt          0.87       0.93      0.90        396

         sci.electronics          0.81       0.77      0.79        393

                 sci.med          0.90       0.84      0.87        396

  soc.religion.christian          0.79       0.95      0.87        398

      talk.politics.guns          0.74       0.92      0.82        364

   talk.politics.mideast          0.96       0.96      0.96        376

      talk.politics.misc          0.84       0.63      0.72        310


                accuracy                               0.84       7532

               macro avg          0.84       0.83      0.83       7532

            weighted avg          0.84       0.84      0.84       7532
```

5.6 Testing with the ‘smsspam’ Dataset

The results from section 5.6.2 and 5.6.1 showed that the Multinomial Naive Bayes outperformed GBC model but by 1 %. The results also showed that using TF-IDF weights in the Naive Bayes model with this dataset did not improve the models performance. Summary of results from section 5.6.2 and 5.6.1 is given in table 5.2 below.

						Weighted Average		
	Tf-IDF	Accuracy	Total	True	False	Precision	Recall	F1-Score
GBC	no	0.97	1115	1084	31	0.97	0.97	0.97
Bayes	no	0.98	1115	1092	23	0.98	0.98	0.98
Bayes	yes	0.98	1115	1092	23	0.98	0.98	0.98

Table 5.2: Results For ‘smsspam’ Dataset

5.6.1 Performance Group By Class Model

```
 class    precision  recall  f1-score   support

 ham        0.97       0.99     0.98       947

spam        0.98       0.85     0.91       168
```

The GBC model’s accuracy was 97% as shown in Table 5.2. The words *ham* and *spam* represent sms messages that were categorized as harmless and spam respectively.

5.6.2 Performance Multinomial Naive Bayes With and Without Tf-IDF

```
class   precision   recall  f1-score   support

ham       0.98       0.99     0.99       947

spam      0.96       0.90     0.93       168
```

Whether or not TF-IDF weightings were used, the Multinomial Naive Bayes model’s gave identical classification performance results on the ‘smsspam’ dataset. Hence why there is only one table shown in the results above. The accuracy of the model was 98%, as shown in table 5.2.

5.7 Discussion of Results

The GBC model was compared to the Multinomial Naive Bayes model in order to illustrate its performance and feasibility as a solution for document classification. From the obtained results, the GBC model outperformed only the 'Naive Bayes with no Tf-IDF weights on the 'fetch_20newsgroup' dataset. It came close to performing just as well as 'Naive Bayes with Tf-IDF weights. These results indicate that classification performance of both models are fairly the same and further confirm that TF-IDF weights can indeed improve a model's classification performance. In this project using TF-IDF weights with the GBC model has not yet been explored but this is an avenue for future work. The data sets used were large, standard data sets commonly used by the machine learning community for analyzing machine learning algorithms (Dua and Graff 2017b). To prevent any bias no cleaning of the data set, such as removing stop words, was done during training and classification. Also, for both datasets, the data was split into training and testing data so that the models would first be trained using the training data and then tested (classification testing) using the test data. Getting a high classification performance (precision, recall, and F1 scores) on a small data set or one that is uniform or predictable is easier than getting high classification performance on larger more chaotic datasets like the 'fetch20newsgroup' dataset. In general, the GBC classification model was able to give comparable performance results with those obtained from the Multinomial Naive Bayes model which shows favourable outcomes for the project.

Chapter 6

Discussion And Conclusion

6.1 Discussion of GBC Model

The true strengths of the GBC model is not mainly in its classification performance but in its implementation simplicity. The Naive Bayes classifier is capable of incremental learning however because of the way its algorithm works the task of updating Naive Bayes models require much effort and is actually an area involving plenty of research. Also with the Naive Bayes classifier in order for multilabel classification to be possible there are special techniques/approaches which are used and in some cases requires that multiple Naive Bayes models be created (Sucar et al. 2014). Multilabel classification is where a document such as an article can get tagged with more than one category. The Naive Bayes Classifier is designed to return only one category when classifying a document. The GBC machine learning algorithm automatically caters for multilabel classification because each class vectors generated are themselves individual models, and as illustrated in section 3.8 and section 4.3.1.1, updating a GBC text classification model is very simple and intuitive. There has been many research published that presented various techniques and improvements for incremental learning in Naive Bayes Classifiers as well as other state of the art approaches. However because models Like the Naive Bayes approaches focused heavily on classification performance and were not initially designed for incremental growth capability, research to find ways to make these models support incremental learning are still being done. In this project the GBC model presented was created for the purpose of being extensible by supporting incremental learning, and simple to use. The solution presented in this project also by nature of design allows for multilabel classification. With more research and experiments the classification performance of the model may even be improved over 'Naive Bayes (with Tf-IDF weights). In this project only two datasets were used however more datasets will be explored in future work.

6.2 Discussion of GBC System

The GBC system was developed to provide users with a means through which they can test the GBC model. However, the system can still be used in a real world scenario for categorizing news articles and other related documents. It comprised of a frontend and a backend. The backend included a database and two APIs: one for interacting with the database and the other for handling interaction between frontend and backend. The system was made public to users for testing via the frontend web interface. Most comments and feedback were positive with users stating that the system gave really good results and that the interface design was very simple and user friendly. Nikita Louison, a student pursuing her masters at the university of the west indies in computer science stated the following: 'The processing time was surprisingly quick, it has stuff now people paying money for that is slower, I also like that there is a break down of the classification further than the top three'. 'Nicholas Chamansingh, the current Head of Data Science Digicel Group Trinidad, stated the following: 'The system is very cool, so far the classification of the news articles I am adding is accurate.'

6.3 Limitations

In this project attempts were made to access datasets used, and code implementations done, by other researchers for comparative analysis. These were difficult to acquire because most researchers only published the results and did not include the datasets and code used to achieve the results. Another limitation was the dataset gathered from the local online news articles. The dataset is not truly substantial because there was still difficulty in extracting the data, this was due to CAPTCHA (completely automated public Turing test), used to tell computers and humans apart on the articles site that would sometimes prevent access to some of the articles. Hence the amount of local articles acquired for training was still limited.

6.4 Conclusion

In this project a novel machine learning text classification algorithm called the Group By Class (GBC) algorithm was presented. Using the GBC algorithm and some standard datasets, GBC classification models were generated and compared with the Multinomial Naive Bayes model, which is one of the state of the art classification models used for document classification. The results of the comparisons showed that the GBC classifier gave comparative results to the Bayes classifier. The GBC classifier also proved to provide certain benefits over the Multinomial Naive Bayes model which included the ease of incremental learning and multilabel classification, both these task are much more difficult to achieve in the Naive Bayes Model. The multilabel classification capability of the GBC Model allows a document to be classified based on the number of class vectors it is related to. Using this feature it was also illustrated that by entering a location name to be labeled, multiple categories associated with that location can be returned and hence categorical information about locations can also be obtained using the GBC Model. This can be useful for identifying bias reporting in news articles that report on places or can be used to identify the categories mostly associated with a given location based on the training data. In this project the solution presented provides automated classification, a method of extracting information about locations from news articles, and a classification model that adapts to changing data. The aim for future research is to improve the GBC classification model performance by experimenting with TF-IDF weights. Semantic relatedness between locations in Trinidad and Tobago using the GBC model is also another possible future project.

References

Abbas, Muhammad, Kamran Ali Memon, Abdul Aleem Jamali, Saleemullah Memon, and Anees Ahmed. 2019. "Multinomial Naive Bayes Classification Model for Sentiment Analysis." *IJCSNS* 19 (3): 62.

Ade, Roshani, and PR Deshmukh. 2014. "Instance-based vs Batch-based Incremental Learning Approach for Students Classification." *International Journal of Computer Applications* 106 (3).

Ade, RR, and PR Deshmukh. 2013. "Methods for incremental learning: a survey." *International Journal of Data Mining & Knowledge Management Process* 3 (4): 119.

Aha, David W, Dennis Kibler, and Marc K Albert. 1991. "Instance-based learning algorithms." *Machine learning* 6 (1): 37–66.

Alcobé, Josep Roure. 2004. "Incremental Augmented Naive Bayes Classifiers." In *Proceedings of the 16th European Conference on Artificial Intelligence,* 539–543. IOS Press.

Androutsopoulos, Ion, John Koutsias, Konstantinos V Chandrinos, and Constantine D Spyropoulos. 2000. "An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages." In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval,* 160–167. ACM.

Arguello, Jaime. 2013. "Vector space model." *Information Retrieval September* 25.

Bacan, Hrvoje, Igor S Pandzic, and Darko Gulija. 2005. "Automated news item categorization." In *Proceedings of the 19th Annual Conference of The Japanese Society for Artificial Intelligence,* 251–256. Citeseer.

Bollegala, Danushka. 2017. "Dynamic feature scaling for online learning of binary classifiers." *Knowledge-Based Systems* 129:97–105.

Buitinck, Lars, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, et al. 2013. "API design for machine learning software: experiences from the scikit-learn project." In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning,* 108–122.

Dua, Dheeru, and Casey Graff. 2017a. *UCI Machine Learning Repository.* University of California, Irvine, School of Information and Computer Sciences. https://archive.ics.uci.edu/ml/machine-learning-databases/00228/.

Dua, Dheeru, and Casey Graff. 2017b. *UCI Machine Learning Repository.* University of California, Irvine, School of Information and Computer Sciences. https://archive.ics.uci.edu/ml/about.html.

Ee, Chee-Hong Chan Aixin Sun, and Peng Lim. 2001. "Automated online news classification with personalization." In *4th international conference on asian digital libraries.*

Hu, Yingjie, Xinyue Ye, and Shih-Lung Shaw. 2017. "Extracting and analyzing semantic relatedness between cities using news articles." *International Journal of Geographical Information Science* 31 (12): 2427–2451.

Jakkula, Vikramaditya. 2006. "Tutorial on support vector machine (svm)." *School of EECS, Washington State University* 37.

Jiang, Liangxiao, Dianhong Wang, Zhihua Cai, and Xuesong Yan. 2007. "Survey of improving naive bayes for classification." In *International Conference on Advanced Data Mining and Applications,* 134–145. Springer.

Kibriya, Ashraf M, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. 2004. "Multinomial naive bayes for text categorization revisited." In *Australasian Joint Conference on Artificial Intelligence,* 488–499. Springer.

Ko, Youngjoong, and Jungyun Seo. 2009. "Text classification from unlabeled documents with bootstrapping and feature projection techniques." *Information Processing & Management* 45 (1): 70–83.

Kotsiantis, Sotiris. 2013. "Increasing the accuracy of incremental naive bayes classifier using instance based learning." *International Journal of Control, Automation and Systems* 11 (1): 159–166.

Laskov, Pavel, Christian Gehl, Stefan Krüger, and Klaus-Robert Müller. 2006. "Incremental support vector learning: Analysis, implementation and applications." *Journal of machine learning research* 7 (Sep): 1909–1936.

Leff, Avraham, and James T Rayfield. 2001. "Web-application development using the model/view/controller design pattern." In *Proceedings fifth ieee international enterprise distributed object computing conference,* 118–127. IEEE.

Lewis, David D. 1987. *UCI Machine Learning Repository: Reuters-21578 Text Categorization Collection Data Set.* https://archive.ics.uci.edu/ml/datasets/reuters-21578+text+categorization+collection. (Accessed on 07/22/2019).

Maia, Italo. 2015. *Building Web Applications with Flask.* Packt Publishing Ltd.

McCallum, Andrew, Kamal Nigam, et al. 1998. "A comparison of event models for naive bayes text classification." In *AAAI-98 workshop on learning for text categorization,* 752:41–48. 1. Citeseer.

Mouine, Mohamed, Diana Inkpen, Pierre-Olivier Charlebois, and Tri Ho. 2016. "Identifying Multiple Topics in Texts." *Int. J. Comput. Linguistics Appl.* 7 (2): 29–44.

Naresh, Annam, and Soudamini Sreepada. 2019. "Automatic Classification of Bing Answers User Verbatim Feedback." In *First International Conference on Artificial Intelligence and Cognitive Computing,* 449–458. Springer.

Nikam, Sagar S. 2015. "A comparative study of classification techniques in data mining algorithms." *Oriental journal of computer science & technology* 8 (1): 13–19.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research* 12:2825–2830.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_20newsgroups.html. (Accessed on 07/17/2019),

Polikar, Robi, Lalita Upda, Satish S Upda, and Vasant Honavar. 2001. "Learn++: An incremental learning algorithm for supervised neural networks." *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)* 31 (4): 497–508.

Ramage, Daniel, David Hall, Ramesh Nallapati, and Christopher D Manning. 2009. "Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora." In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1,* 248–256. Association for Computational Linguistics.

Read, Jesse, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. 2012. "Batch-incremental versus instance-incremental learning in dynamic and evolving data." In *International Symposium on Intelligent Data Analysis,* 313–323. Springer.

Read, Jesse, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. 2011. "Classifier chains for multi-label classification." *Machine learning* 85 (3): 333.

Reis, Julio CS, André Correia, Fabrício Murai, Adriano Veloso, Fabrício Benevenuto, and Erik Cambria. 2019. "Supervised Learning for Fake News Detection." *IEEE Intelligent Systems* 34 (2): 76–81.

Ren, Shuxia, Yangyang Lian, and Xiaojian Zou. 2014. "Incremental Naïve Bayesian Learning Algorithm based on Classification Contribution Degree." *JCP* 9 (8): 1967–1974.

Robbins, Jennifer Niederst. 2012. *Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics.* " O'Reilly Media, Inc.".

Ruping, Stefan. 2001. "Incremental learning with support vector machines." In *Proceedings 2001 IEEE International Conference on Data Mining,* 641–642. IEEE.

Singh, Bikesh Kumar, Kesari Verma, and AS Thoke. 2015. "Investigations on impact of feature normalization techniques on classifier's performance in breast tumor classification." *International Journal of Computer Applications* 116 (19).

Soleymani, Mohammad, and Maja Pantic. 2012. “Human-centered implicit tagging: Overview and perspectives.” In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC),* 3304–3309. IEEE.

Sucar, L Enrique, Concha Bielza, Eduardo F Morales, Pablo Hernandez-Leal, Julio H Zaragoza, and Pedro Larrañaga. 2014. “Multi-label classification with Bayesian network-based chain classifiers.” *Pattern Recognition Letters* 41:14–22.

Widnall, S. 2009. “Lecture l3-vectors, matrices and coordinate transformations.” *Dynamics:* 1–15.

Zhang, Min-Ling, José M Peña, and Victor Robles. 2009. “Feature selection for multi-label naive Bayes classification.” *Information Sciences* 179 (19): 3218–3229.

Zhong, Junwei, Zhenyan Liu, Yifei Zeng, Lijia Cui, and Zizheng Ji. 2017. “A survey on incremental learning.” In *Proceedings of the 5th International Conference on Computer, Automation and Power Electronics, Colombo, Sri Lanka,* 25–27.

Appendix A

Technologies Used

A.0.1 Technologies Used By The Backend

Backend		
TECHNOLOGY	PURPOSE	DESCRIPTION
Amazon Elastic Compute Cloud (Amazon EC2) and AWS Cloud9	The Database Server and Web Server were hosted on a separate EC2 instance from the frontend all coding and development was also done on this instance	Amazon EC2 is a web service for providing secure, re-sizable compute capacity in the cloud. Designed to make web-scale cloud computing easier for developers. AWS Cloud9 is a cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser. It includes a code editor, debugger, and terminal
MySQL Database	Articles and their associated information such as Title, domain from which the article was retrieved and publication date were stored in this type of database. The database was also used to store the IPTC topics and their associated key words. The IPTC Topics were stored in the 'categorie table' shown in Figure 4.1 and the relationship between the IPTC Topics and respective key words were shown via the bridge table 'topicmodel'	MySQL is an open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL).
Flask	Was used to create the DatabaseAPI and the GBC Rest API	Flask is a micro web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries.[3] It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself.
The 20 newsgroups dataset	Used for evaluating the text classification model	This data set comprises around 18000 newsgroups posts on 20 topics split in two subsets: one for training (or development) and the other one for testing (or for performance evaluation). Documentation regading this data set is at the following: ($https://scikit-learn.org/0.19/datasets$)

A.0.2 Technologies Used By The Backend Contd.

Backend		
TECHNOLOGY	PURPOSE	DESCRIPTION
The 'smsspam' dataset	Used for training and testing a GBC model for assessing the feasibility of the GBC algorithm	The data set was obtained from the 'UC Irvine Machine Learning Repository' (Dua and Graff 2017a). The UCI Machine Learning Repository is a collection of domain theories, databases, and data generators which are used by the machine learning community for the purpose of empirically analyzing machine learning algorithms (Dua and Graff 2017b)
Flask-Restless	Used to develop the DatabaseApi service	Flask-Restless provides simple generation of ReSTful APIs for database models defined using SQLAlchemy (or Flask-SQLAlchemy).

Table A.3: Backend Technologies Contd.

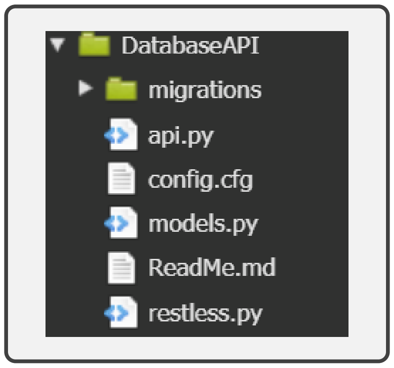
A.0.3 Technologies Used By The Frontend

Frontend		
TECHNOLOGY	PURPOSE	DESCRIPTION
Amazon Elastic Compute Cloud (Amazon EC2) and AWS Cloud9	The Frontend application was hosted on its own EC2 instance all coding and development was also done on this instance	Same as in table A.2
Material Design for Bootstrap	Used to develop the front end	An open source toolkit based on Bootstrap for developing Material Design apps with HTML, CSS, and JS
Vue.js	Used for developing front-end	A JavaScript framework with various optional tools for building user interfaces.
JSON Viewer Awesome	Used for displaying the JSON results in a format that is easy to traverse and read	Free Google chrome extension Offered by Rahul Baruri.

Table A.4: Frontend Technologies

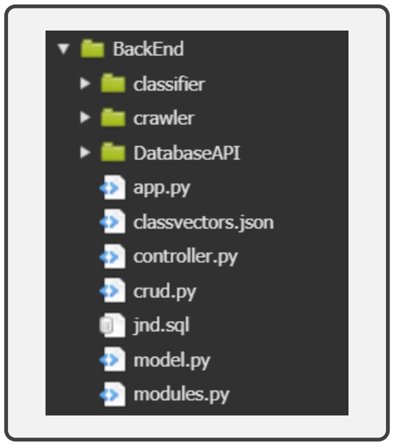
Appendix B

Backend Info



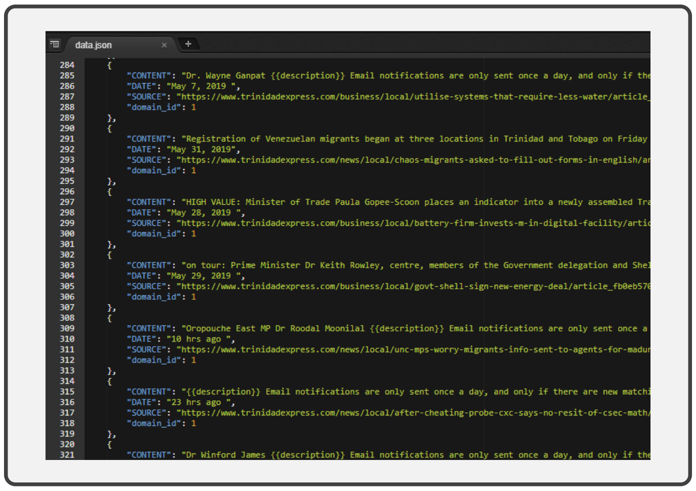
```
▼ DatabaseAPI
  ▶ migrations
    api.py
    config.cfg
    models.py
    ReadMe.md
    restless.py
```

Figure B.1: DatabaseAPI File Structure



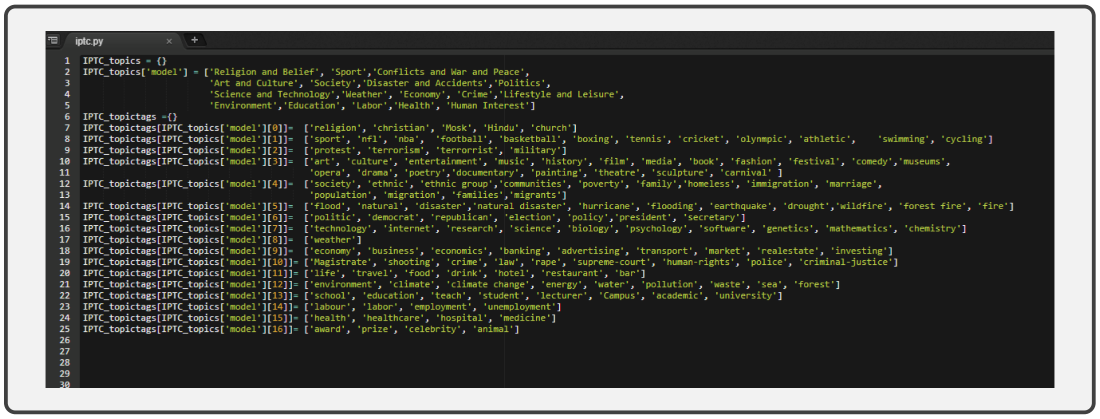
```
▼ BackEnd
  ▶ classifier
  ▶ crawler
  ▶ DatabaseAPI
    app.py
    classvectors.json
    controller.py
    crud.py
    jnd.sql
    model.py
    modules.py
```

Figure B.2: Backend File Structure



```
    {
        "CONTENT": "Dr. Wayne Ganpat {{description}} Email notifications are only sent once a day, and only if the
        "DATE": "May 7, 2019 ",
        "SOURCE": "https://www.trinidadexpress.com/business/local/utilise-systems-that-require-less-water/article_
        "domain_id": 1
    },
    {
        "CONTENT": "Registration of Venezuelan migrants began at three locations in Trinidad and Tobago on Friday
        "DATE": "May 31, 2019",
        "SOURCE": "https://www.trinidadexpress.com/news/local/chaos-migrants-asked-to-fill-out-forms-in-english/ar
        "domain_id": 1
    },
    {
        "CONTENT": "HIGH VALUE: Minister of Trade Paula Gopee-Scoon places an indicator into a newly assembled Tra
        "DATE": "May 28, 2019 ",
        "SOURCE": "https://www.trinidadexpress.com/business/local/battery-firm-invests-m-in-digital-facility/artic
        "domain_id": 1
    },
    {
        "CONTENT": "on tour: Prime Minister Dr Keith Rowley, centre, members of the Government delegation and Shel
        "DATE": "May 29, 2019 ",
        "SOURCE": "https://www.trinidadexpress.com/business/local/govt-shell-sign-new-energy-deal/article_fb0eb570
        "domain_id": 1
    },
    {
        "CONTENT": "Oropouche East MP Dr Roodal Moonilal {{description}} Email notifications are only sent once a
        "DATE": "10 hrs ago ",
        "SOURCE": "https://www.trinidadexpress.com/news/local/unc-mps-worry-migrants-info-sent-to-agents-for-madur
        "domain_id": 1
    },
    {
        "CONTENT": "{{description}} Email notifications are only sent once a day, and only if there are new matchi
        "DATE": "23 hrs ago ",
        "SOURCE": "https://www.trinidadexpress.com/news/local/after-cheating-probe-cxc-says-no-resit-of-csec-math/
        "domain_id": 1
    },
    {
        "CONTENT": "Dr Winford James {{description}} Email notifications are only sent once a day, and only if the
```

Figure B.3: Snapshot Of Training Data Format For ‘data.json’



```python
IPTC_topics = {}
IPTC_topics['model'] = ['Religion and Belief', 'Sport','Conflicts and War and Peace',
                        'Art and Culture', 'Society','Disaster and Accidents','Politics',
                        'Science and Technology','Weather', 'Economy', 'Crime','Lifestyle and Leisure',
                        'Environment','Education', 'Labor','Health', 'Human Interest']
IPTC_topictags ={}
IPTC_topictags[IPTC_topics['model'][0]]=  ['religion', 'christian', 'Mosk', 'Hindu', 'church']
IPTC_topictags[IPTC_topics['model'][1]]=  ['sport', 'nfl', 'nba',  'football', 'basketball', 'boxing', 'tennis', 'cricket', 'olynmpic', 'athletic',    'swimming', 'cycling']
IPTC_topictags[IPTC_topics['model'][2]]=  ['protest', 'terrorism', 'terrorrist', 'military']
IPTC_topictags[IPTC_topics['model'][3]]=  ['art', 'culture', 'entertainment', 'music', 'history', 'film', 'media', 'book', 'fashion', 'festival', 'comedy','museums',
                                            'opera', 'drama', 'poetry','documentary', 'painting', 'theatre', 'sculpture', 'carnival' ]
IPTC_topictags[IPTC_topics['model'][4]]=  ['society', 'ethnic', 'ethnic group','communities', 'poverty', 'family','homeless', 'immigration', 'marriage',
                                            'population', 'migration', 'families','migrants']
IPTC_topictags[IPTC_topics['model'][5]]=  ['flood', 'natural', 'disaster','natural disaster', 'hurricane', 'flooding', 'earthquake', 'drought','wildfire', 'forest fire', 'fire']
IPTC_topictags[IPTC_topics['model'][6]]=  ['politic', 'democrat', 'republican', 'election', 'policy','president', 'secretary']
IPTC_topictags[IPTC_topics['model'][7]]=  ['technology', 'internet', 'research', 'science', 'biology', 'psychology', 'software', 'genetics', 'mathematics', 'chemistry']
IPTC_topictags[IPTC_topics['model'][8]]=  ['weather']
IPTC_topictags[IPTC_topics['model'][9]]=  ['economy', 'business', 'economics', 'banking', 'advertising', 'transport', 'market', 'realestate', 'investing']
IPTC_topictags[IPTC_topics['model'][10]]= ['Magistrate', 'shooting', 'crime', 'law', 'rape', 'supreme-court', 'human-rights', 'police', 'criminal-justice']
IPTC_topictags[IPTC_topics['model'][11]]= ['life', 'travel', 'food', 'drink', 'hotel', 'restaurant', 'bar']
IPTC_topictags[IPTC_topics['model'][12]]= ['environment', 'climate', 'climate change', 'energy', 'water', 'pollution', 'waste', 'sea', 'forest']
IPTC_topictags[IPTC_topics['model'][13]]= ['school', 'education', 'teach', 'student', 'lecturer', 'Campus', 'academic', 'university']
IPTC_topictags[IPTC_topics['model'][14]]= ['labour', 'labor', 'employment', 'unemployment']
IPTC_topictags[IPTC_topics['model'][15]]= ['health', 'healthcare', 'hospital', 'medicine']
IPTC_topictags[IPTC_topics['model'][16]]= ['award', 'prize', 'celebrity', 'animal']
```

Figure B.4: Python script ‘iptc.py’ Corresponding To The IPTC Topics