University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

CommentSense: Multilabel Social Support Classification of Filipino-English Endocrinology Facebook Comments Using Machine Learning Classification Models

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Romaine Dara M. Regala

July 2023

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "CommentSense: Multilabel Social Support Classification of Filipino-English Endocrinology Facebook Comments Using Machine Learning Classification Models" prepared and submitted by Romaine Dara M. Regala in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Geoffrey A. Solano, Ph.D.
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Avegail D. Carpio, M.Sc.	_____	_____
2. Perlita E. Gasmen, M.Sc. (*cand.*)	_____	_____
3. Ma. Sheila A. Magboo, Ph.D. (*cand.*)	_____	_____
4. Vincent Peter C. Magboo, M.D., M.Sc.	_____	_____
5. Marbert John C. Marasigan, M.Sc. (*cand.*)	_____	_____
6. Richard Bryann L. Chua, Ph.D. (*cand.*)	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Vio Jianu C. Mojica, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences and Mathematics

Maria Constancia O. Carrillo, Ph.D.
Dean
College of Arts and Sciences

Abstract

The mental health implications of Diabetes can be mitigated through the presence of a social support system, and social media platforms offer a convenient avenue for fostering such support, even among healthcare professionals and patients. This study introduces novel machine learning models tailored for multilabel social support classification of Filipino-English Endocrinology Facebook comments. The objective is to effectively categorize comments into distinct types of support, including informational, emotional, appraisal, instrumental, and spam, thereby enabling healthcare professionals to efficiently manage their social media groups. The dataset underwent manual data cleaning and was subsequently divided into training and testing sets. Preprocessing techniques encompassing lowercasing, tokenization, and TF-IDF vectorization were employed on both sets. To address dataset imbalances, data augmentation techniques were implemented. Notably, the LP-SVM model emerged as the top performer and was seamlessly integrated into the CommentSense application. These findings enhance our comprehension of social support dynamics and furnish practitioners with a user-friendly tool for social support text classification.

Keywords: natural language processing, social support, diabetes, machine learning, multilabel social support classification, data augmentation

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Diabetes Mellitus is a chronic, metabolic disease from which patients are characterized by elevated levels of blood sugar. This disease affects about 422 million people and is one of the major causes of death worldwide as 1.5 million deaths are directly attributed to diabetes each year [1]. As the disease progress, 33% to 50% of people with diabetes would likely have diabetes distress as well as 20% of patients with diabetes are more likely to experience anxiety than those without. These psychological problems affect and hinder their recovery [2].

Social support has been shown to reduce the psychological and physiological effects of stress and may increase immune function [3]. Nowadays, social media platforms like Facebook can offer social support for everyone especially patients through health-related groups and pages. On average, Filipinos spent 4 hours and 6 minutes on social media mainly with Facebook [4], and with this, Dr. Iris-Isip Tan made a Facebook page in 2012 to experiment if photos with health messages would be viral. As time goes by, this page became a medium for providing reliable health information on endocrine disorders in the Filipino language.

With a following of more than 150,000 Facebook users, Dr. Iris-Isip Tan, an endocrinologist from Philippine General Hospital, gained interest in exploring social support on her Facebook page. By manually extracting comments on diabetes-related posts, pasting them into a Word document, and coding in NVivo she was able to label the data into the type of social support namely: 'informational', 'emotional', 'appraisal', and 'instrumental' as well as 'spam'.

Through the development of a system that automates the labeling and classification of comments on diabetes-related posts into distinct types of social support on a Facebook page, healthcare professionals, such as Dr. Iris-Isip Tan, can optimize

time management while gaining valuable insights into the predominant types of social support sought or provided by patients. This system enables efficient analysis, allowing doctors to discern the specific social support needs expressed by patients on the platform. By leveraging this understanding, doctors-page moderators can deliver targeted and timely support, enhancing both time management and patient care.

B. Statement of the Problem

Patients benefit from having an endocrinologist-moderated Facebook page because it offers informational assistance from a healthcare professional, but due to professional obligations and personal obligations, they lack the time to review and organize comments on their own moderated page. Considering patients and healthcare provider-moderator, this study aims to build a system that can effectively automate labeling/classifying comments on diabetes-related posts into different types of social support.

C. Objectives of the Study

This study aims to classify comments on diabetes-related posts on an Endocrinologist-Moderated Facebook page into different types of support using machine learning. Specifically, the study would aim to:

1. Develop a text classification model that can identify and/or categorize diabetes-related comments posted on a Facebook page into different types of social support

    (a) Manual data cleaning for spelling corrections

    (b) Splitting the dataset into 80% training and 20% testing

    (c) Apply data augmentation using OpenAI in order to handle class imbalance

(d) Benchmarking different multilabel text classification models, namely:

i. Logistic Regression

ii. Support Vector Machine

iii. Ensemble Classifier Chain based on Logistic Regression

iv. Ensemble Classifier Chain based on Support Vector Machine

v. MultiLayer Perceptron

(e) Evaluating the model using appropriate metrics

i. Precision

ii. Recall

iii. F1-score

iv. Hamming Loss

2. Develop a web application integrated with the model and allows the Facebook page-moderator to:

(a) Classify a single text

(b) Classify bulk of texts/comments through CSV format

(c) View list of classified comments

(d) Go to specific Facebook comment

(e) Export data in CSV

D. Significance of the Project

This study carries significant implications for both research purposes and practical applications in the field of diabetes management. By effectively classifying comments on a Facebook page related to diabetes, the study provides valuable insights into

the types of support that patients seek and offer in online communities. This understanding can greatly contribute to the development of tailored content and effective support programs that address the specific needs and preferences of diabetes patients.

Moreover, the classification system developed in this study can have practical benefits for professionals and page moderators. It enables them to efficiently analyze and categorize incoming comments, facilitating more targeted and personalized responses. By quickly identifying the type of support requested or offered, healthcare professionals can provide timely and appropriate information, guidance, and encouragement to patients. This not only enhances the quality of interactions on the Facebook page but also improves the overall management of the page by streamlining the moderation process.

Overall, the accurate classification of comments, whether for research purposes or to aid professionals in their responses, is crucial for optimizing the support and engagement experienced by diabetes patients on social media platforms. This study addresses this need, opening doors to more effective communication, improved content generation, and enhanced support programs for the management of diabetes.

E. Scope and Limitations

The scope of this study includes the development and evaluation of a text classification model that can accurately identify and categorize comments on diabetes-related posts on an Endocrinologist-Moderated Facebook page into different types of social support. Consequently, the research will be subject to the following limitations:

1. The study is limited to comments written in English and Filipino.

2. The study is limited to analyzing only the comments posted on the Facebook page, and may not take into account other factors that could affect the nature of requested or offered support by patients such as demographic or disease stage.

3. The study is limited to comments posted on a specific Facebook page, and the results may not generalize to other social media platforms.

4. The dataset's spelling has been subjected to manual correction, introducing potential human errors. On the other hand, grammar is not checked or corrected.

5. The CSV file to be fed to the system represents the extracted Facebook comments.

F. Assumptions

The following are the assumptions of the system:

1. The system has existing data that the text classifier can train on.

2. That the comments posted are representative of the types of support that diabetes patients are requesting and offering on social media.

3. The dataset is from a public Facebook page that can be accessed by everyone.

II. Review of Related Literature

It is evident that NLP is considered beneficial to every field. NLP helps in understanding and analyzing written texts and spoken for more objective and accurate analysis. This study aims to use NLP techniques with Hybrid deep learning in the classification of social support.

A. Diabetes Management

Diabetes is a tedious disease as every patient will find it challenging dealing with it and it takes practice in managing it everyday. The goal of Diabetes Management is to manage and keep glucose levels to normal [5] as well as other factors like diet and cholesterol. A study has shown effort in managing the diet of patients with Type 2 Diabetes by creating a program called Beyond Good Intention (BGI). It shows that the participants have greater improvement in dietary quality, especially those who have no nutrition goal baseline [6].

A..1 Social Support

Social support can be beneficial to diabetes management as it can reduce psychological repercussions of the challenges of the progressing disease [3]. There is a study that focused on the relationship between social support and self-care behavior in patients with diabetes where they perform a cross-sectional study on diabetes patients with over 30 years of age in Mashad, Iran. The result showed that 40.7% of the samples had good social support, 46.2% has average social support, and the rest had poor social support which shows that the mean score of social support was average, and the majority had moderate to poor self-care. They use Spearman's rank correlation coefficient between the domains of diabetes self-care behaviors and social support (healthy eating behaviors=0.758 correlation coefficient, physical activity=0.893 corre-

lation coefficient, blood glucose monitoring = 0.680 correlation coefficient, and proper medication use = 0.737) which indicates that social support and diabetes self-care behaviors are significantly and directly correlated [7].

A..2 Patient Engagement

Patient engagement is defined as the patient's desire and capability to actively choose to participate in care, with cooperation with a healthcare provider or institution, in order to maximize the outcomes or improve experiences of care [8]. We cannot argue that patient engagement is necessary for diabetes management but according to the study of Iqbal(2020), there are still barriers that should be overcome in this aspect. There are multiple barriers like access to services, understanding of referral requirements, and further practitioner/patient education. Improvement in these areas will hopefully lead to a change in diabetes care to prevent diabetes complications [9].

A..3 Health Communication

Health communication is an area of study that examines how the use of different communication strategies can keep people informed about their health and influence their behavior so they can live healthier lives. Public health experts also recognize health communication as important to different health programs [10]. A study focused on exploring the association between patients' perceptions of communication quality with their provider and a range of patients' outcomes in type 2 diabetes revealed that their higher perceived quality of provider-patient communication was associated with improved self-management, adherence to diabetes care, and greater well-being, perceived personal control, self-efficacy, and less diabetes distress. In addition to that it is also revealed that effective communication was more related to the provision of support [11].

B. Social Media

With the emergence of social media, understanding human behavior as well as mental health studies become a new interest. Platforms like Facebook, Instagram, Twitter, as well as Reddit, can be a new source of support for everybody, and the exchange of information can be easily accessed. A study on patient experience and cancer survivorship on social media reveals that Instagram can be a platform for patient experience and can be beneficial for heal and neck surgeons and other oncology providers in a sense that they can gain a better understanding of the daily experience of survivorship in order for them to provide support and comprehensive cancer care [12].

C. Multilabel Text Classification

With the significant growth of technology around the world there have been numerous efforts with the use of NLP to understand and analyze data. Text classification is one of the fundamental tasks in natural language processing that is used in numerous applications like sentiment analysis, and spam detection [13].

C..1 Learning Models

There are numerous studies utilizing text classification with various methods. A study on the applicability of machine learning methods to Multilabel medical text classification benchmarks different methods of handling multilabel text classification. Clinical documents written in Russia of more than 250 thousand patients were used for the research for which it includes four labels. The researchers performed some basic preprocessing like cleaning symbols and extra spaces, minimizing notes, correcting syntax, spelling correction, tokenization, and lastly, vectorizing both the training set and test sets using Bag of Words. The best-performing machine learning models are Logistic Regression and Support Vector Machine but the performance metrics become

better with the Ensemble Classifier Chains [14].

Model	Precision		Recall		F-measure	
	Micro	Macro	Micro	Macro	Micro	Macro
Shallow classifiers						
MNB	0.781	0.764	0.864	0.873	0.864	0.852
LR	**0.866**	**0.850**	**0.920**	0.915	**0.920**	**0.910**
Linear SVM	**0.865**	**0.849**	**0.919**	0.916	**0.919**	**0.909**
k-NN	0.694	0.715	0.803	0.827	0.803	0.809
Ensembles of classifier chains						
ECCLR	**0.867**	**0.852**	**0.925**	**0.921**	**0.922**	**0.912**
ECCSVM	**0.872**	**0.855**	**0.927**	**0.922**	**0.924**	**0.914**

Figure 1: Performance results of the classification models

Another study on multi-label Arabic text categorization also performed benchmark and baseline comparisons of multi-label learning algorithms. Multilabel text classification has two popular methods, which are the transformation-based methods (Binary Relevance, Classifier Chains, Calibrated Ranking by Pairwise Comparison, and Label Powerset) and the adaption-based methods (Multi-label k-Nearest Neighbors, Instance-Based Learning by Logistic Regression Multi-label, Binary Relevance kNN, and RFBoost), where the three well-known multiclass algorithms (Support Vector Machine, k-Nearest-Neighbors, and Random Forest) were used as the base learners for the transformation-based methods. The results showed that RFBoost significantly outperforms the other methods and the transformation-based methods perform well when the Support Vector Machine is used as a base classifier [15].

Classifier	Macro-Precision		Macro-Recall		Macro-F1		Micro-Precision		Micro-Recall		Micro-F1	
	score	f. size	score	f. size	score	f. size	score	f. size	score	f. size	score	f. size
Problem transformation approaches												
BR-*k*NN	**83.34**	2000	47.60	200	56.22	200	**85.66**	4000	55.26	200	64.77	200
BR-RF	82.80	4000	57.43	200	62.23	200	84.81	3,000	61.99	200	67.56	200
BR-SVM	76.89	500	61.24	4000	65.88	4000	82.39	200	64.20	4000	69.89	500
CLR-*k*NN	82.83	2000	47.43	200	55.57	200	85.03	4000	55.33	200	64.21	200
CLR-RF	82.14	3000	59.31	200	63.50	200	82.41	3000	63.98	200	68.38	200
CLR-SVM	76.05	500	64.59	4000	67.84	4000	81.56	200	67.30	4000	70.72	4000
CC-*k*NN	81.89	4000	50.92	200	57.16	200	80.55	4000	57.10	200	63.52	200
CC-RF	82.91	4000	57.91	200	61.78	200	84.80	4000	62.44	200	69.01	500
CC-SVM	73.12	500	62.02	2000	66.58	4000	76.67	500	65.08	4000	69.57	500
LP-*k*NN	74.37	500	52.16	200	57.99	200	71.27	200	58.08	200	64.00	200
LP-RF	68.42	500	52.80	200	57.80	200	72.19	500	59.68	200	65.28	500
LP-SVM	76.70	4000	**64.86**	4000	**69.79**	4000	79.72	4000	**67.39**	4000	**73.04**	4000
Algorithm adaptation approaches												
RFBoost	77.11	4000	63.74	500	**68.98**	4000	77.44	3000	**68.57**	2000	**72.57**	3000
ML*k*NN	79.21	3000	48.37	1000	56.39	1000	83.92	4000	53.52	1000	64.97	1000
BR*k*NN	**85.57**	1000	43.40	200	52.68	200	**90.10**	4000	51.41	200	62.83	200
IBLRML	79.01	500	47.50	200	56.44	200	84.43	2000	51.88	500	64.08	500

Figure 2: The best-obtained scores (%) of all compared methods.

A master's thesis focuses on a multi-label text classification task using a small dataset of bilingual short texts (emails) in both English and Swedish. The dataset consists of 5,800 emails that are categorized into 107 classes, with the majority of the emails containing both languages. Various models, including Support Vector Machines (SVM), Gated Recurrent Units (GRU), Convolutional Neural Network (CNN), Quasi Recurrent Neural Network (QRNN), and Transformers, were employed to handle this task.

The experimental results reveal that the SVM model outperforms the other models

in terms of the weighted average F1 score, achieving a score of 0.96. The CNN model follows with a score of 0.89, and the QRNN model obtains a score of 0.80. These findings highlight the effectiveness of the SVM model for the given multi-label text classification task on the bilingual email dataset [16].

	SVM	GRU	Very deep CNN	QRNN	Transformers
Weighted avg F1	0.96	0.67	0.89	0.80	0.53
Hamming loss	0.0077	0.0447	0.0235	0.0375	0.0461

Figure 3: Metrics comparison among the models

In summary, there are different methods one can utilize to effectively perform text classification to understand a specific topic. The lack of research about social support types being requested and offered online is evidence that this can be a gap available for future studies.

III. Theoretical Framework

A. Natural Language Processing

Natural Language Processing (NLP) has existed for more than 50 years and has a variety of real-world applications in different fields, including medical research, search engines, and even business intelligence [17]. NLP is a subfield of Artificial Intelligence that aims to process, analyze, and natural language data (spoken and written). The goal of NLP is to enable machines to interact with humans in a way that resembles natural human communication [18].

B. Text classification

Text classification refers to a technique in Natural Language Processing that focuses on categorizing text documents into two or more categories. The most common form of text classification is binary classification, which assigns a document to one of two categories. The process of text classification involves extracting a set of features that describe the document and then applying an algorithm to use these features to classify the document into its appropriate category [19].

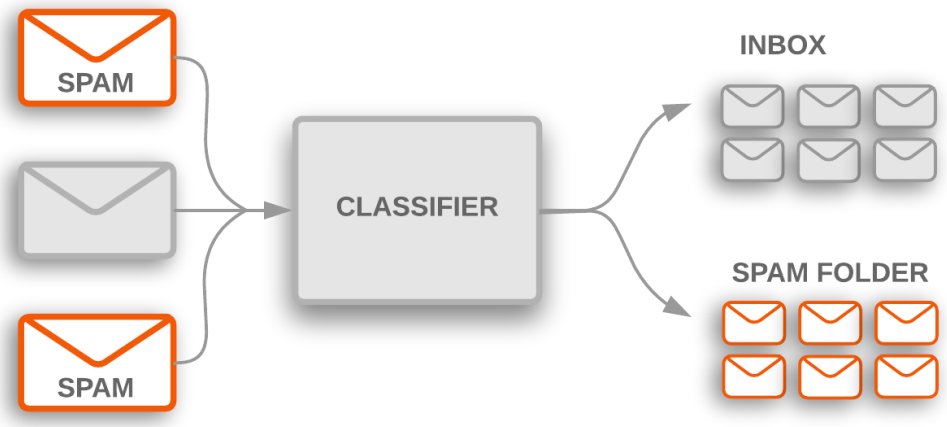

Figure 4: Example of a Text Classification Application

C. Label Powerset

The label powerset method takes a multi-label dataset and changes it to a single multi-class dataset by treating each label combination as a separate class. It achieves multi-label categorization by associating an instance with a class composed of a set of labels. In our example, we assign a class to each unique label set (C001, C110, C011, C101). After that, a multi-class classifier is trained to assign an instance to one of the classes listed above [20].

D. Classifier Chain

This method is comparable to binary relevance. However, it considers label correlation. This method employs a chain of classifiers, with each classifier taking as input the predictions of all previous classifiers. The number of classifiers is the same as the number of classes [21].

E. Support Vector Machine

The Support Vector Machine (SVM) is a powerful supervised machine learning algorithm that excels in classification tasks. It utilizes labeled input data, divided into two distinct classes, and aims to predict the classification of a query sample. By transforming the data into a higher-dimensional space, SVM employs a support vector classifier to determine a threshold, known as a hyperplane, which effectively separates the two classes with minimal error. This technique enables SVM to make accurate predictions and effectively classify new samples based on their characteristics [22].

F. Hamming Loss

In multiclass classification, the evaluation of model performance is often measured using the Hamming loss, which quantifies the similarity between the true labels (y_true) and the predicted labels (y_pred). This loss can be viewed as the Hamming distance between the two label sets, or equivalently, as a subset zero-one loss when the normalized parameter is set to True [23].

In contrast, multilabel classification introduces a different interpretation of the Hamming loss compared to the subset zero-one loss. The zero-one loss considers an entire set of labels for a specific sample as incorrect if it does not perfectly match the true set of labels. On the other hand, the Hamming loss takes a more lenient approach and penalizes only the individual labels that are incorrectly predicted [23].

IV. Design and Implementation

A. Dataset

The dataset utilized for modeling consists of manually annotated/labeled comments extracted from Dr. Iris-Isip Tan's Facebook page. The dataset includes a total of 9,767 comments that were posted on 196 diabetes-related posts from 1 January 2018 to 31 March 2021, spanning a period of several years. The dataset used for modeling is the manually annotated/labeled comments from Dr. Iris-Isip Tan's Facebook page. These are the labels available in the dataset:

- informational - sharing of peer experiences or research
- emotional - enhancing a sense of belonging or as an outlet to vent frustrations
- appraisal - prompting self-reflection
- instrumental - sharing of resources
- spam - anything not related to diabetes

B. Use Cases

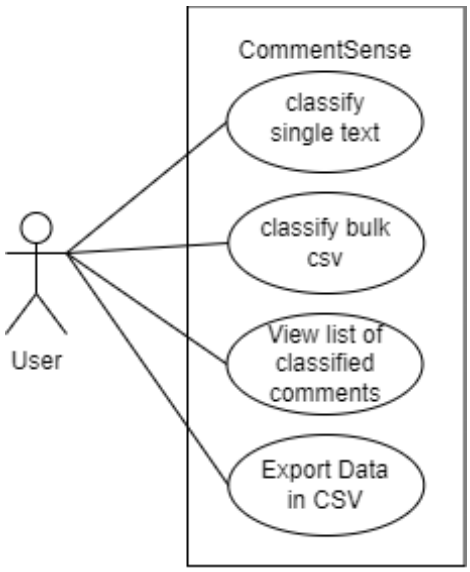

Figure 5: Use-Case Diagram of System

The use-case diagram (Figure 5) presents the system to be used by the Facebook moderator. The Facebook moderator should be able to classify a single text or classify bulk comments inside a CSV file. The Facebook moderator should also see the list of classified comments. Lastly, the Facebook moderator should be able to export the data to a CSV file.

C. System Design

C..1 Context Diagram

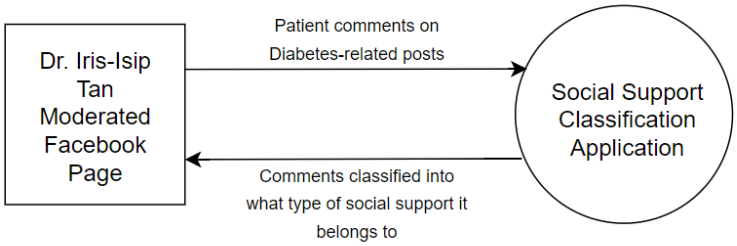

Figure 6: Context Diagram

Figure 6 represents how the system works. Patient comments on Diabetes-related

posts on Dr. Iris-Isip Tan's Facebook page are the data to be processed by the system. The system then outputs the classified comments based on what type of social support it belongs to.

C..2 Data Flow Diagram

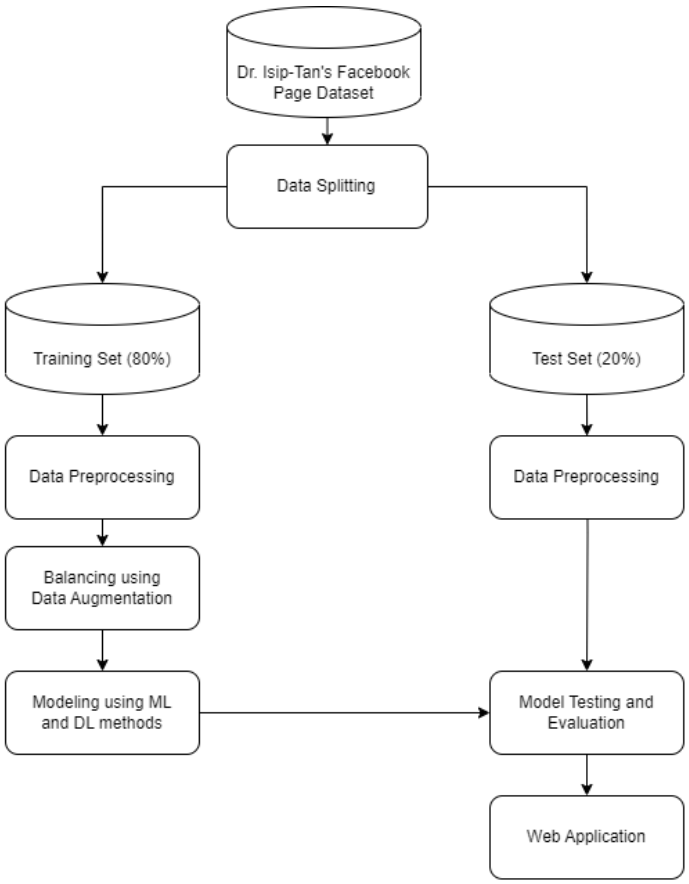

Figure 7: Data Flow Diagram

Figure 7 shows the data flow from the dataset to the web application development. The dataset is from Dr. Iris-Isip Tan's labeled comments extracted from her moderated Facebook page. We will perform data splitting of 80% training set and 20% test set where both of them will be subject to data preprocessing. After data preprocessing, the training set will be balanced using data augmentation utilizing OpenAI. After rebalancing,It will now proceed to modeling using LP-Logistic Re-

gression, LP-Support Vector Machine, ECCLR, ECCSVM, and MLP. After training the model, accuracy, precision, recall, F1-score, and Hamming loss will be used to evaluate the model. Lastly, the best-trained model is used for the development of the web application.

D. System Architecture

The system makes use of Python as its main programming language. For the development of web application, the system also makes use of SQLite for its database server while using Django as its framework. For modeling, python machine learning libraries like scikit-multilearn and scikit-learn will be utilized as well as libraries like pandas, numpy, and joblib. These are the detailed list of the important libraries.

- Django==4.2.3
- emoji==2.6.0
- joblib==1.3.1
- nltk==3.8.1
- numpy==1.25.0
- pandas==2.0.3
- requests==2.31.0
- scikit-learn==1.3.0
- scikit-multilearn==0.2.0
- scikit-optimize==0.9.0
- scipy==1.11.1

E. Technical Architecture

Initially, the system should be accessible via a link that works on any browser and can be viewed either from a computer or a smartphone but during the course of development and time restrictions it is only deployed using localhost through Django.

Minimum System Requirements

1. Computer
    - Browser (Chrome, Edge, Safari, etc)
    - Windows 7 or higher
    - Stable internet connection

V. Results

A. Exploratory Data Analysis

In Chapter 4, the dataset utilized in this study originated from Dr. Iris-Isip Tan's Facebook page called "Endocrine Witch." The dataset consisted of comments from diabetes-related posts, covering the period from 1st of January 2018 to 31st of March 2021. These comments were manually extracted and compiled into Word Documents, which were later coded in NVivo for further analysis. Each document in the dataset was labeled according to its respective category or theme. Notably, comments with multiple labels (multilabeled comments) were given particular attention and included in the dataset.

To ensure the accuracy of the data, the comments underwent a manual review to identify and correct any spelling errors. After this meticulous data-cleaning process, the resulting dataset contained 2941 rows. The figure below presents an overview of the dataset's information.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2942 entries, 0 to 2941
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   id       2942 non-null   object
 1   label    2942 non-null   object
 2   comment  2942 non-null   object
dtypes: object(3)
memory usage: 69.1+ KB
```

Figure 8: Information of the dataset

Figure 8 shows that the dataset has 3 columns mainly: id with type object, label with type object, and comment with type object. The comment column will be the feature variable while the label column is the target variable.

For training and test split, the data is partitioned into an 80-20 split, where 80% is for the training set and 20% is for the testing set.

B. Handling Class Imbalance using OpenAI Data Augmentation

Upon analyzing the dataset, a class imbalance is observed among the labels. Such class imbalance in a multilabel text classification cannot be handled by the Imbalanced-learn Python library for which novel class balancing methods is provided (SMOTE, Random Undersampling, etc). In order to handle class imbalance, OpenAI's API is utilized to create augmented sentences as needed by the training set.

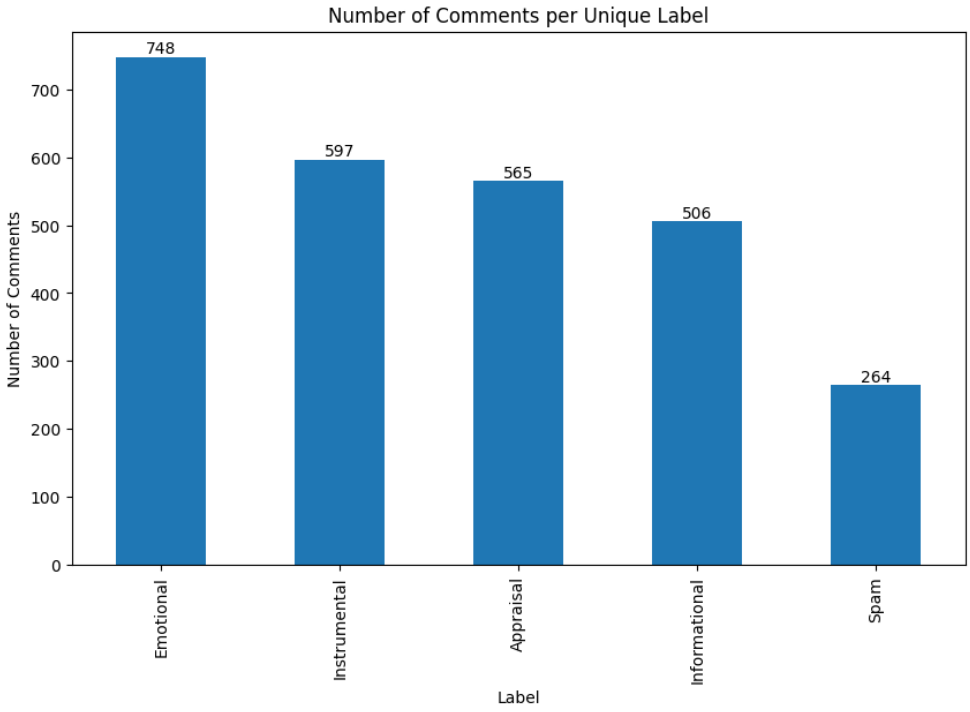

Figure 9: Trainset imbalance

In Figure 9, the imbalance is visible among the labels as the Emotional count is relatively high compared to others and the Spam label count is relatively low among the others. There is at most 484 difference in the count of the emotional labels and

spam labels.

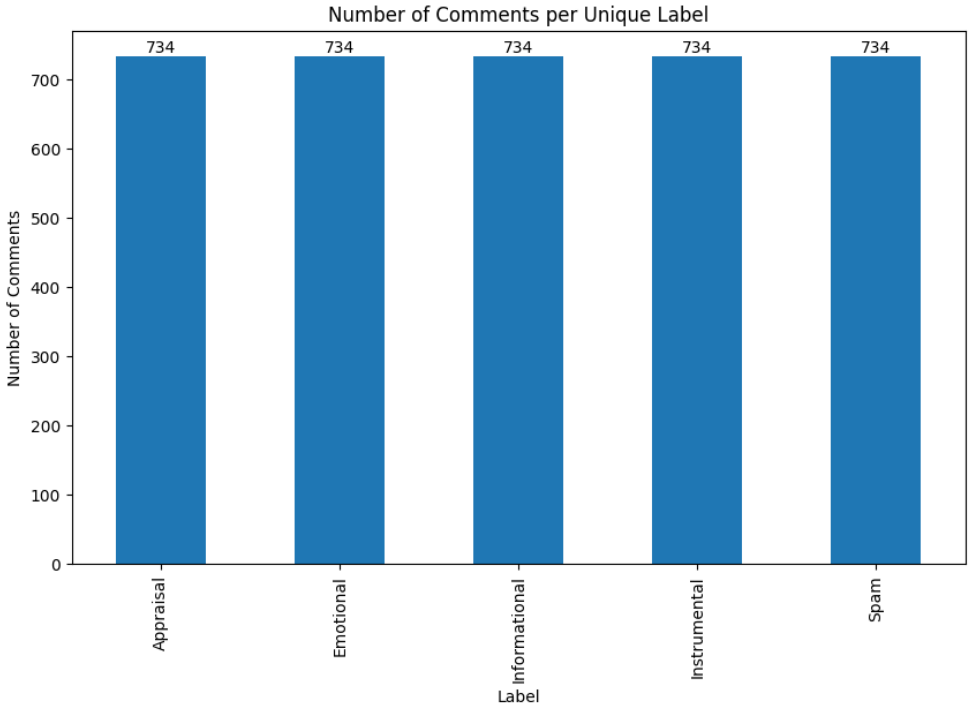

Figure 10: Rebalanced trainset using Data Augmentation

The duplicates or noise in the Emotional label are manually removed leaving a count of 734 emotional labeled comments. This becomes the basis of how much augmented data should be added per label in the training set.

C. Data Preprocessing

Both the imbalanced dataset and the balanced dataset were processed in a similar manner, employing a set of preprocessing methods. For the training set, several steps were applied, including lowercasing letters, removing numbers, eliminating single characters, removing multiple spaces, tokenization, removing stop words, and TF-IDF vectorization. These steps aimed to standardize the textual data and extract meaningful features for training purposes. Similarly, for the test set, preprocessing steps such as removing multiple spaces and converting the data to binary using the MultiLabelBinarizer were performed. These preprocessing techniques were essential

in ensuring data consistency and preparing the dataset for subsequent analysis and modeling tasks.

D. Model Evaluation

The model training process is divided into two distinct batches: training with imbalanced data and training with a balanced training set. For evaluating the performance of each model, both the Classification Report and the hamming loss were utilized.

	Imbalanced			Balanced		
Labels	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Appraisal	0.68	0.63	**0.66**	**0.69**	**0.64**	**0.66**
Emotional	0.79	**0.77**	0.78	**0.83**	0.76	**0.79**
Informational	0.56	0.44	0.49	**0.55**	**0.45**	**0.50**
Instrumental	0.69	**0.61**	**0.65**	**0.70**	0.56	0.63
Spam	**0.86**	0.48	0.62	0.78	**0.62**	**0.70**
Micro AVG	0.70	**0.61**	0.65	**0.71**	**0.61**	**0.66**
Macro AVG	**0.72**	0.59	0.64	0.71	**0.61**	**0.65**
Weighted AVG	0.70	**0.61**	0.65	**0.71**	**0.61**	**0.66**
Samples AVG	0.70	**0.65**	**0.67**	0.71	**0.65**	**0.67**

Table 1: Classification Report For Imbalance and Balance dataset - Label Powerset Logistic Regression

Table 1, shows that most of the label in LP-Logistic Regression performs a little bit better in augmented data. It is notable that the most constant labels are Emotional and Spam. On the other hand, Instrumental Label seems to decrease in metrics when augmented data is applied. Overall, this model performs much better in imbalanced data.

	Imbalanced			Balanced		
Labels	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Appraisal	0.66	0.64	0.65	**0.68**	**0.66**	**0.67**
Emotional	0.86	**0.75**	0.80	**0.89**	**0.75**	**0.81**
Informational	**0.56**	**0.47**	**0.51**	0.54	0.46	0.50
Instrumental	0.69	**0.64**	0.66	**0.75**	**0.64**	**0.69**
Spam	**0.96**	0.41	0.57	0.90	**0.56**	**0.69**
Micro AVG	0.71	0.61	0.66	**0.74**	**0.63**	**0.68**
Macro AVG	**0.75**	0.58	0.64	0.74	**0.63**	**0.68**
Weighted AVG	0.73	0.61	0.66	**0.75**	**0.63**	**0.68**
Samples AVG	0.71	0.65	0.67	**0.74**	**0.67**	**0.69**

Table 2: Classification Report For Imbalance and Balance dataset - Label Powerset Support Vector Machine

Table 2, shows that most of the label in LP-Support Vector Machine model performs a little bit better in augmented data, particularly Appraisal, Emotional, and Instrumental. Just like in LP-LR model, it is notable that the most constant label is Emotional. On the other hand, Informational Label seems to decrease in metrics when augmented data is applied. Overall, this model performs much better in imbalanced data.

	Imbalanced			Balanced		
Labels	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Appraisal	0.70	**0.48**	**0.57**	**0.72**	0.47	**0.57**
Emotional	0.86	**0.74**	**0.80**	**0.88**	0.72	0.79
Informational	**0.59**	0.43	0.50	0.58	**0.45**	**0.51**
Instrumental	**0.56**	**0.72**	**0.63**	**0.56**	0.65	0.61
Spam	**0.74**	0.48	0.58	0.58	**0.66**	**0.62**
Micro AVG	**0.67**	**0.59**	**0.63**	**0.67**	**0.59**	0.62
Macro AVG	**0.69**	0.57	**0.62**	0.67	**0.59**	**0.62**
Weighted AVG	**0.69**	**0.59**	**0.63**	0.68	**0.59**	**0.63**
Samples AVG	**0.68**	**0.62**	**0.64**	**0.67**	0.62	**0.63**

Table 3: Classification Report For Imbalance and Balance dataset - Ensemble Classifier Chain Logistic Regression

Table 3, shows that most of the label in Ensemble Classifier Chain Logistic Regres-

sion does not perform well in augmented data than the two earlier models, particularly Instrumental. Overall, this model performs slightly better in imbalanced data.

	Imbalanced			Balanced		
Labels	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Appraisal	0.75	**0.43**	**0.55**	**0.76**	0.42	0.54
Emotional	0.84	**0.76**	**0.80**	**0.85**	**0.76**	**0.80**
Informational	**0.52**	0.48	0.50	**0.52**	**0.50**	**0.51**
Instrumental	0.60	0.70	**0.65**	0.62	**0.71**	**0.66**
Spam	**0.85**	0.52	**0.64**	0.77	**0.53**	0.63
Micro AVG	**0.69**	**0.60**	**0.64**	**0.69**	**0.60**	**0.64**
Macro AVG	**0.71**	0.58	**0.63**	**0.71**	0.58	**0.63**
Weighted AVG	0.70	**0.60**	0.63	**0.71**	**0.60**	**0.64**
Samples AVG	**0.69**	**0.63**	**0.65**	**0.69**	**0.63**	**0.65**

Table 4: Classification Report For Imbalance and Balance dataset - Ensemble Classifier Chain Support Vector Machine

Table 4, shows that most of the label in the Ensemble Classifier Chain Support Vector Machine model performs a little bit better in augmented data, particularly Emotional, and Informational. Just like in other models, it is notable that the most constant label is Emotional. On the other hand, Appraisal and Informational Label seems to decrease in metrics when augmented data is applied. Overall, this model performs much better in balanced data.

	Imbalanced			Balanced		
Labels	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Appraisal	**0.66**	**0.51**	**0.58**	0.64	**0.51**	0.57
Emotional	0.76	**0.76**	0.76	**0.82**	0.72	**0.77**
Informational	**0.59**	**0.41**	**0.49**	0.52	0.35	0.42
Instrumental	**0.62**	0.49	**0.55**	0.59	**0.50**	0.54
Spam	**0.72**	0.44	0.54	0.63	**0.56**	**0.60**
Micro AVG	**0.67**	**0.54**	**0.60**	0.65	0.53	0.59
Macro AVG	**0.67**	0.52	**0.58**	0.64	**0.53**	**0.58**
Weighted AVG	**0.67**	**0.54**	**0.59**	0.65	0.53	0.58
Samples AVG	**0.58**	**0.58**	**0.57**	0.57	**0.58**	0.56

Table 5: Classification Report For Imbalance and Balance dataset - MultiLayer Perceptron

Table 5, shows that most of the label in the MultiLayer Perceptron model does not perform well in augmented data, particularly Appraisal, Instrumental and Informational. Just like in other models, it is notable that the most constant label is Emotional. Overall, this model performs much better in imbalanced data.

Metrics	LP-LR	LP-SVM	ECC-LR	ECC-SVM	MLP
Weighted AVG(Precision)	0.70	**0.73**	0.69	0.69	0.67
Weighted AVG(Recall)	**0.61**	**0.61**	0.59	0.60	0.54
Weighted AVG(F1-Score)	**0.71**	0.66	0.63	0.63	0.59
Hamming Loss	0.152	**0.149**	0.163	0.159	0.170

Table 6: Compare models Imbalance

Metrics	LP-LR	LP-SVM	ECC-LR	ECC-SVM	MLP
Weighted AVG(Precision)	0.71	**0.75**	0.68	0.71	0.65
Weighted AVG(Recall)	0.61	**0.63**	0.59	0.60	0.53
Weighted AVG(F1-Score)	0.66	**0.68**	0.63	0.64	0.58
Hamming Loss	0.150	**0.140**	0.165	0.157	0.176

Table 7: Compare models Balance

Upon analyzing tables 6 and 7, even though with below average metrics, Label Powerset Support Vector Machine performs better than the other models.

E. Web-Based Applilcation

E..1 Single Text

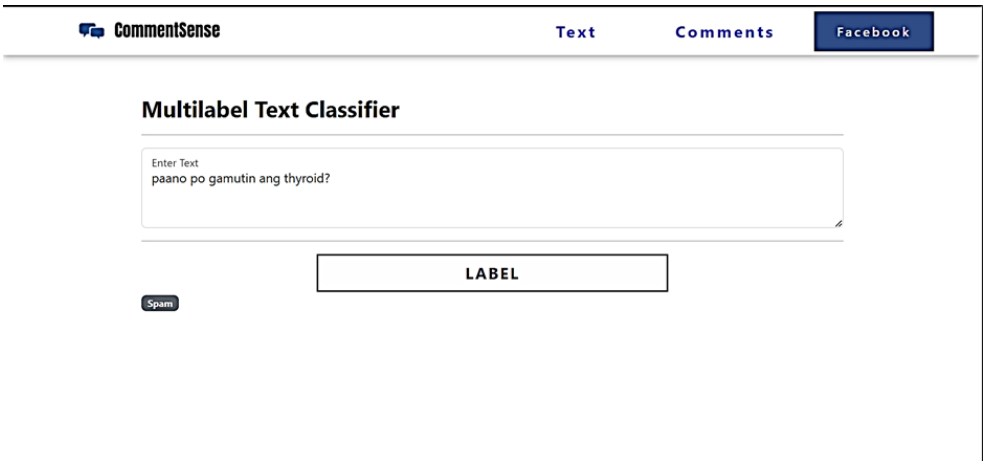

Figure 11: Single Text Classifier

This screen also serves as a landing page for the application. The user can input a single text string and the system will output its label/s.

E..2 Bulk Comments

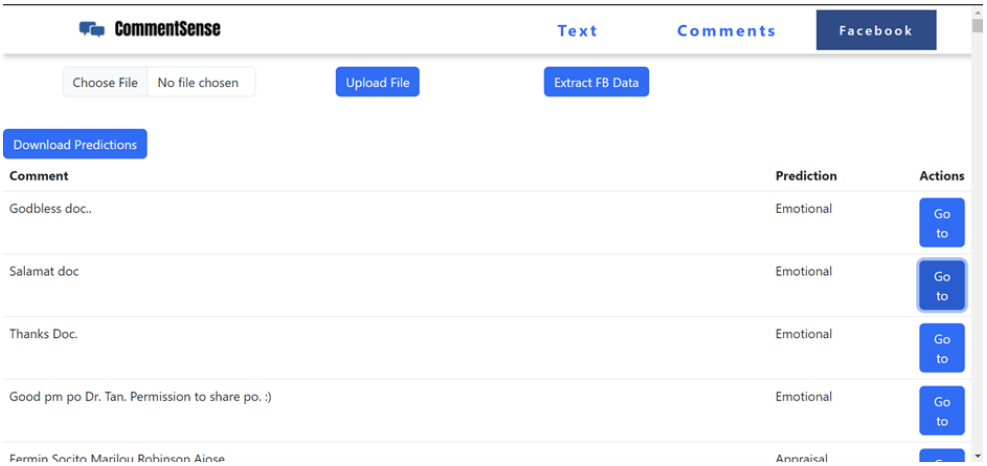

Figure 12: Bulk Text Classifier

This screen can be accessed by clicking on the "Comments" option in the navigation bar. Upon reaching this page, users can conveniently upload a CSV file by utilizing the "Upload File" button, and the system will generate a corresponding CSV file with predictions that can be downloaded. In case the Facebook data is not currently accessible, users have the option to click on the "Extract FB Data" button, enabling them to extract real-time information. Once the extraction process is complete, users can proceed to download the extracted data, facilitated by the presence of a dedicated "Download Extracted Data" button. To classify comment rows effectively, both the extracted data CSV and the user's existing CSV files must be uploaded to the system. On this page, users can conveniently view the list of predictions derived from the CSV file. Furthermore, by including each comment's permalink in the data extraction, users can navigate to the specific page comment effortlessly.

VI. Discussions

This study focuses on applying various learning models (LP-LR, LP-SVM, ECC-LR, ECC-SVM, and MLP) to classify texts or comments from a Facebook page moderated by an endocrinologist, specifically related to diabetes. The goal is to categorize these comments into different types of social support, including informational (sharing of peer experiences or research), emotional (enhancing a sense of belonging or as an outlet to vent frustrations), appraisal (prompting self-reflection), instrumental (sharing of resources), and spam (anything not related to diabetes). The evaluation process revealed that the Label Powerset Support Vector Machine model performed the best and was integrated into a user-friendly application called CommentSense.

The study timeline was significantly influenced by the dataset, which required manual cleaning, revision, and augmentation. The initial dataset had a poor structure, posing challenges for the models to achieve optimal performance. However, data augmentation was applied to improve the initial metrics.

To evaluate the performance of each model, a classification report was utilized, providing a detailed analysis of the metrics. Examining the performance of each model across different labels, the Emotional label consistently achieved the highest metrics, followed by Appraisal or Spam. The Instrumental label had relatively lower metrics, and the Informational label performed last.

Overall, CommentSense provides an intuitive interface for users to effectively classify texts into various social support categories and identify spam. While Emotional support stands out as the most consistent performer, further analysis of each model's performance sheds light on the strengths and weaknesses across different support categories.

VII. Conclusions

This study presents innovative machine learning models for multilabel social support classification of Filipino-English Endocrinology Facebook Comments. The data, collected and labeled by Dr. Iris-Isip Tan, underwent manual preprocessing, including inputting the data into Excel, performing data cleaning tasks such as spelling corrections, removing whitespace, eliminating Dr. Iris-Isip Tan's replies, and removing commenters' aliases. The dataset was then split into a training set (80%) and a testing set (20%) for further analysis.

Both the training set and test set underwent preprocessing techniques. For the training set, a series of steps were applied, including lowercase conversion, number removal, elimination of single characters, removal of multiple spaces, tokenization, stop word removal, and TF-IDF vectorization. Similarly, the testing set underwent preprocessing steps such as removing multiple spaces and converting the data to binary using the MultiLabelBinarizer.

The researcher observed class imbalance in the dataset and addressed it through data augmentation using OpenAI. The model training process consisted of two batches: one for the imbalanced dataset and one for the balanced dataset. The models were trained and evaluated using the classification report, which provided insights into the performance per label and overall weighted averages, as well as the hamming loss. The results indicated that the LP-SVM model outperformed the other models, followed by the LP-LR model. However, it was noted that the metrics of the LP-SVM model were still lower than expected. As a result, the LP-SVM model with a threshold of 0.3 was integrated into the CommentSense application, allowing users to classify texts and comments efficiently which can facilitate a better understanding of how social support is requested or offered.

VIII. Recommendations

In order to improve the performance of the models used in this study, it is recommended to have a clean and well-structured dataset that can clearly represent the labels. The comments should be tagged individually and not be thread or bulk in order for the models to understand the patterns clearly. In addition to that, it is recommended to create an automation that can automate spelling checks to minimize human error.

Exploring ensemble methods and incorporating contextual word representations offer promising avenues for enhancing the study's outcomes thus it is recommended. Ensemble methods, which involve combining multiple classifiers or utilizing ensemble learning techniques, have the potential to significantly improve the overall performance of the models. By leveraging the diversity and complementary strengths of different classifiers, ensemble methods can enhance accuracy and robustness. Furthermore, incorporating contextual word embeddings into the machine learning models enables a deeper understanding of the comments and texts, capturing rich semantic and contextual information. Contextual word representations have demonstrated their effectiveness in improving classification accuracy and robustness across various domains. By exploring ensemble methods and contextual word representations in a study like this, further improvements can be achieved, advancing the understanding and categorization of social support in the domain of Endocrinology and other domains.

Furthermore, extending the application of CommentSense to other social media platforms commonly used by individuals seeking diabetes-related support is recommended. By expanding the application's reach, a more comprehensive and holistic approach to social support classification and spam detection can be achieved, catering to a broader user base.

IX. Bibliography

[1] WHO, "Diabetes," *World Health Organization*, n.d.

[2] CDC, "Diabetes: Mental health," *Centers for Disease Control Prevention*, 2022.

[3] E. E. Bakken, "Social support." Available at https://www.takingcharge.csh.umn.edu/social-support, 2022.

[4] S. Amurthalingam, "Social media statistics in the philippines [2022]." Available at https://www.meltwater.com/en/blog/social-media-statistics-philippines, 2022.

[5] U. Health, "Diabetes mellitus treatments." Available at https://www.ucsfhealth.org/conditions/diabetes-mellitus/treatment, n.d.

[6] L. A. van der Velde, J. C. Kiefte-de Jong, G. E. Rutten, and R. C. Vos, "Effectiveness of the beyond good intentions program on improving dietary quality among people with type 2 diabetes mellitus: A randomized controlled trial," *Frontiers in nutrition*, vol. 8, p. 583125, 2021.

[7] D. Sarpooshi, M. Mahdizadeh, A. Jaferi, H. Robatsarpooshi, M. Haddadi, and N. Peyman, "The relationship between social support and self-care behavior in patients with diabetes mellitus," *Family Medicine & Primary Care Review*, vol. 23, no. 2, pp. 227–231, 2021.

[8] T. Higgins, E. Larson, and R. Schnall, "Unraveling the meaning of patient engagement: a concept analysis," *Patient Education and Counseling*, vol. 100, no. 1, pp. 30–36, 2017.

[9] M. Iqbal, "What are the barriers to patient engagement in managing type 2 diabetes? how can we overcome them?: A review article," *Asian Journal of Research and Reports in Endocrinology*, pp. 1–5, 2020.

[10] T. University, "10 strategies for effective health communication." Available at https://publichealth.tulane.edu/blog/health-communication-effective-strategies, 2020.

[11] M. Peimani, E. Nasli-Esfahani, and R. Sadeghi, "Patients' perceptions of patient–provider communication and diabetes care: A systematic review of quantitative and qualitative studies," *Chronic illness*, vol. 16, no. 1, pp. 3–22, 2020.

[12] R. W. Gao, J. D. Smith, and K. M. Malloy, "Head and neck cancer and social media: the patient experience and cancer survivorship," *The Laryngoscope*, vol. 131, no. 4, pp. E1214–E1219, 2021.

[13] Z. Deutschman, "Multi-label text classification." Available at https://towardsdatascience.com/multi-label-text-classification-5c505fdedca8, 2019.

[14] I. Lenivtceva, E. Slasten, M. Kashina, and G. Kopanitsa, "Applicability of machine learning methods to multi-label medical text classification," in *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part IV 20*, pp. 509–522, Springer, 2020.

[15] B. Al-Salemi, M. Ayob, G. Kendall, and S. A. M. Noah, "Multi-label arabic text categorization: A benchmark and baseline comparison of multi-label learning algorithms," *Information Processing & Management*, vol. 56, no. 1, pp. 212–227, 2019.

[16] S. Harsha Kadam and K. Paniskaki, "Text analysis for email multi label classification," 2020.

[17] B. Lutkevich, "natural language processing (nlp)." Available at https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP, 2021.

[18] Y. Vasiliev, *Natural Language Processing with Python and SpaCy: A Practical Introduction*. No Starch Press, 2020.

[19] G. Miner, D. Delen, J. Elder, A. Fast, T. Hill, and R. A. Nisbet, "Chapter 7 - text classification and categorization," in *Practical Text Mining and Statistical Analysis for Non-structured Text Data Applications*, pp. 881–892, Boston: Academic Press, 2012.

[20] Y. Motro, "Overview on multi-label classification," Dec 2022.

[21] J. A. George, "An introduction to multi-label text classification," Dec 2020.

[22] K. Jain, "What is support vector machine?," Mar 2021.

[23] G. Tsoumakas and I. Katakis, "Multi-label classification," *International Journal of Data Warehousing and Mining*, vol. 3, no. 3, p. 1–13, 2007.

X. Appendix

A. Source Code

Label Powerset Logistic Regression with (Imbalanced)

source–code/lrNotA.py

```
1
2  """## Import Libraries for Data Preprocessing"""
3
4  import numpy as np
5  import pandas as pd
6  from subprocess import check_output
7  import matplotlib.pyplot as plt
8  from scipy.stats import bernoulli
9  import seaborn as sns
10 import nltk
11 import re
12 from nltk.tokenize import word_tokenize
13 from nltk.corpus import stopwords
14 nltk.download('punkt')
15 nltk.download('stopwords')
16
17 """## Import Libraries for Modeling"""
18
19 from sklearn.feature_extraction.text import TfidfVectorizer
20 from sklearn.preprocessing import MultiLabelBinarizer
21 from sklearn import metrics
22 from sklearn.metrics import classification_report
23
24 from skmultilearn.problem_transform import LabelPowerset
25 from sklearn.linear_model import LogisticRegression
26 from sklearn.metrics import f1_score, accuracy_score, hamming_loss, precision_score, recall_score
27
28 """### OThers"""
29
30 import warnings
31 warnings.filterwarnings("ignore")
32
33 from wordcloud import WordCloud, STOPWORDS
34 import plotly.express as px
35 import plotly.graph_objects as go
36 import plotly.figure_factory as ff
37 from plotly.subplots import make_subplots
38
39 trainset = pd.read_csv('traindata.csv')
40 testset = pd.read_csv('testdata.csv')
41 print(trainset.shape)
42 trainset.head()
43
44 print(trainset.info())
45
46 """# Data Preprocessing
47
48 ## Converting Capital to small
49 ### TrainSet
50 """
51
52 trainset['comment'] = trainset['comment'].apply(lambda x : x.lower())
53 trainset
54
55 """### TestSet"""
56
57 testset['comment'] = testset['comment'].apply(lambda x : x.lower())
58 testset
59
60 """### Regular Expression
61
62 ### TrainSet
63 """
64
65 trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
66 trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
67 trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
68 trainset
69
70 """### TestSet"""
71
72 testset['comment'] = testset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
73 testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
74 testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
75 testset
76
77 """### Tokenize comments
```

```
78
79  ### TrainSet
80  """
81
82  trainset ['comment'] = trainset['comment'].apply(word_tokenize)
83  trainset
84
85  """### TestSet"""
86
87  testset ['comment'] = testset['comment'].apply(word_tokenize)
88  testset
89
90  """## Removing Stopwords
91
92  ### TrainSet
93  """
94
95  # remove stop words
96  stop_words = set(stopwords.words("english"))
97  trainset ['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
98
99  with open("stopwords-tl.txt", "r") as f:
100     stop_words = f.read().split("\n")
101 trainset ['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
102 trainset
103
104 """### TestSet"""
105
106 # remove stop words
107 stop_words = set(stopwords.words("english"))
108 testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
109
110 with open("stopwords-tl.txt", "r") as f:
111     stop_words = f.read().split("\n")
112 testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
113 testset
114
115 """### Formatting labels
116
117 ### TrainSet
118 """
119
120 trainset ['label'] = trainset['label'].map(lambda x : x.split(','))
121 trainset
122
123 """## TestSet"""
124
125 testset ['label'] = testset['label'].map(lambda x : x.split(','))
126 testset
127
128 """### TrainSet"""
129
130 trainset ['label'] = trainset['label'].map(lambda x : ' '.join(x))
131 trainset
132
133 """### TestSet"""
134
135 testset ['label'] = testset['label'].map(lambda x : ' '.join(x))
136 testset
137
138 """### TrainSet"""
139
140 # clean label to numpy
141 labels = trainset['label'].values
142 labels = [[l for l in clean.split()] for clean in labels]
143 type(labels)
144
145 """### TestSet"""
146
147 # clean label to numpy
148 labeltest = testset['label'].values
149 labeltest = [[l for l in clean.split()] for clean in labeltest]
150 type(labeltest)
151
152 """### TrainSet"""
153
154 # Convert the labels column into binary label columns
155 mlb = MultiLabelBinarizer()
156 y_train = mlb.fit_transform(labels)
157
158 # Create a DataFrame with the binary label columns
159 y_train_df = pd.DataFrame(y_train, columns=mlb.classes_)
160
161 # Count the number of comments per unique label
162 y_train_counts = y_train_df.sum().sort_values(ascending=False)
163
164 """### TestSet"""
165
166 # Convert the labels column into binary label columns
167 y_test = mlb.fit_transform(labeltest)
168
169 # Create a DataFrame with the binary label columns
```

```
y_test_df = pd.DataFrame(y_test, columns=mlb.classes_)

# Count the number of comments per unique label
y_test_counts = y_test_df.sum().sort_values(ascending=False)

"""## Train Test"""

X_train = trainset['comment']
type(X_train)

type(y_train)

X_test = testset['comment']
type(X_test)

type(y_test)

"""## Convert Text to TF-IDF representation"""

vectorizer = TfidfVectorizer()

X_train_text = [' '.join(comment) for comment in X_train]  # Convert tokenized comments back to string format
X_train_tfidf = vectorizer.fit_transform(X_train_text)

X_test_text = [' '.join(comment) for comment in X_test]  # Convert tokenized comments back to string format
X_test_tfidf = vectorizer.transform(X_test_text)

print(X_train.shape)
print(y_train.shape)

"""## Logistic Regression"""

lp_classifier = LabelPowerset(LogisticRegression(C=10.0, penalty='l2', solver='liblinear'))
lp_classifier.fit(X_train_tfidf, y_train)
lp_predictions = lp_classifier.predict(X_test_tfidf)

# # Calculate the evaluation metrics with the best model

classification_rep = classification_report(y_test, lp_predictions, target_names=mlb.classes_)
print("Classification Report:")
print(classification_rep)

#Hamming Loss
lp_hamming_loss = hamming_loss(y_test, lp_predictions)
print("Hamming Loss:", lp_hamming_loss)
```

Label Powerset Logistic Regression with (Balanced)(Trainset)

source–code/lrA.py

```
1
2   """## Import Libraries for Data Preprocessing"""
3
4   import numpy as np
5   import pandas as pd
6   from subprocess import check_output
7   import matplotlib.pyplot as plt
8   from scipy.stats import bernoulli
9   import seaborn as sns
10  import nltk
11  import re
12  from nltk.tokenize import word_tokenize
13  from nltk.corpus import stopwords
14  nltk.download('punkt')
15  nltk.download('stopwords')
16
17  """## Import Libraries for Modeling"""
18
19  from sklearn.feature_extraction.text import TfidfVectorizer
20  from sklearn.preprocessing import MultiLabelBinarizer
21  from sklearn import metrics
22  from sklearn.metrics import classification_report
23
24  from skmultilearn.problem_transform import LabelPowerset
25  from sklearn.linear_model import LogisticRegression
26  from sklearn.metrics import f1_score, accuracy_score, hamming_loss, precision_score, recall_score
27
28  """### OThers"""
29
30  import warnings
31  warnings.filterwarnings("ignore")
32
33  from wordcloud import WordCloud, STOPWORDS
34  import plotly.express as px
35  import plotly.graph_objects as go
36  import plotly.figure_factory as ff
37  from plotly.subplots import make_subplots
38
39  trainset = pd.read_csv('aug.csv')
40  testset = pd.read_csv('testdata.csv')
41  print(trainset.shape)
```

```
trainset.head()

print(trainset.info())

"""# Data Preprocessing

## Converting Capital to small
### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x : x.lower())
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x : x.lower())
testset

"""### Regular Expression

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
testset

"""### Tokenize comments

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(word_tokenize)
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(word_tokenize)
testset

"""## Removing Stopwords

### TrainSet
"""

# remove stop words
stop_words = set(stopwords.words("english"))
trainset['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])

with open("stopwords-tl.txt", "r") as f:
    stop_words = f.read().split("\n")
trainset['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
trainset

"""### TestSet"""

# remove stop words
stop_words = set(stopwords.words("english"))
testset['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])

with open("stopwords-tl.txt", "r") as f:
    stop_words = f.read().split("\n")
testset['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
testset

"""### Formatting labels

### TrainSet
"""

trainset['label'] = trainset['label'].map(lambda x : x.split(','))
trainset

"""## TestSet"""

testset['label'] = testset['label'].map(lambda x : x.split(','))
testset

"""### TrainSet"""

trainset['label'] = trainset['label'].map(lambda x : ' '.join(x))
trainset

"""### TestSet"""
```

```
134
135  testset['label'] = testset['label'].map(lambda x : ' '.join(x))
136  testset
137
138  """### TrainSet"""
139
140  # clean label to numpy
141  labels = trainset['label'].values
142  labels = [[l for l in clean.split()] for clean in labels]
143  type(labels)
144
145  """### TestSet"""
146
147  # clean label to numpy
148  labeltest = testset['label'].values
149  labeltest = [[l for l in clean.split()] for clean in labeltest]
150  type(labeltest)
151
152  """### TrainSet"""
153
154  # Convert the labels column into binary label columns
155  mlb = MultiLabelBinarizer()
156  y_train = mlb.fit_transform(labels)
157
158  # Create a DataFrame with the binary label columns
159  y_train_df = pd.DataFrame(y_train, columns=mlb.classes_)
160
161  # Count the number of comments per unique label
162  y_train_counts = y_train_df.sum().sort_values(ascending=False)
163
164  """### TestSet"""
165
166  # Convert the labels column into binary label columns
167  y_test = mlb.fit_transform(labeltest)
168
169  # Create a DataFrame with the binary label columns
170  y_test_df = pd.DataFrame(y_test, columns=mlb.classes_)
171
172  # Count the number of comments per unique label
173  y_test_counts = y_test_df.sum().sort_values(ascending=False)
174
175  """## Train Test"""
176
177  X_train = trainset['comment']
178  type(X_train)
179
180  type(y_train)
181
182  X_test = testset['comment']
183  type(X_test)
184
185  type(y_test)
186
187  """## Convert Text to TF-IDF representation"""
188
189  vectorizer = TfidfVectorizer()
190
191  X_train_text = [' '.join(comment) for comment in X_train]  # Convert tokenized comments back to string format
192  X_train_tfidf = vectorizer.fit_transform(X_train_text)
193
194  X_test_text = [' '.join(comment) for comment in X_test]  # Convert tokenized comments back to string format
195  X_test_tfidf = vectorizer.transform(X_test_text)
196
197  print(X_train.shape)
198  print(y_train.shape)
199
200  """## Logistic Regression"""
201
202  lp_classifier = LabelPowerset(LogisticRegression(C=10.0, penalty='l2', solver='liblinear'))
203  lp_classifier.fit(X_train_tfidf, y_train)
204  lp_predictions = lp_classifier.predict(X_test_tfidf)
205
206  # # Calculate the evaluation metrics with the best model
207
208  classification_rep = classification_report(y_test, lp_predictions, target_names=mlb.classes_)
209  print("Classification Report:")
210  print(classification_rep)
211
212  #Hamming Loss
213  lp_hamming_loss = hamming_loss(y_test, lp_predictions)
214  print("Hamming Loss:", lp_hamming_loss)
```

Label Powerset Support Vector Machine (Imbalanced)

source–code/svmNotA.py

```
1
2  """## Import Libraries for Data Preprocessing"""
3
4  import numpy as np
5  import pandas as pd
```

```
from subprocess import check_output
import matplotlib.pyplot as plt
from scipy.stats import bernoulli
import seaborn as sns
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')

"""## Import Libraries for Modeling"""

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn import metrics
from sklearn.metrics import classification_report

from skmultilearn.problem_transform import LabelPowerset
from sklearn.svm import SVC
from sklearn.metrics import f1_score, accuracy_score, hamming_loss, precision_score, recall_score

"""### OThers"""

import warnings
warnings.filterwarnings("ignore")

from wordcloud import WordCloud, STOPWORDS
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots

trainset = pd.read_csv('traindata.csv')
testset = pd.read_csv('testdata.csv')
print(trainset.shape)
trainset.head()

print(trainset.info())

"""# Data Preprocessing

## Converting Capital to small
### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x : x.lower())
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x : x.lower())
testset

"""### Regular Expression

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
testset

"""### Tokenize comments

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(word_tokenize)
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(word_tokenize)
testset

"""## Removing Stopwords

### TrainSet
"""

# remove stop words
stop_words = set(stopwords.words("english"))
trainset['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
```

```
98
99  with open("stopwords-tl.txt", "r") as f:
100     stop_words = f.read().split("\n")
101 trainset['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
102 trainset
103
104 """### TestSet"""
105
106 # remove stop words
107 stop_words = set(stopwords.words("english"))
108 testset['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
109
110 with open("stopwords-tl.txt", "r") as f:
111     stop_words = f.read().split("\n")
112 testset['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
113 testset
114
115 """### Formatting labels
116
117 ### TrainSet
118 """
119
120 trainset['label'] = trainset['label'].map(lambda x : x.split(','))
121 trainset
122
123 """## TestSet"""
124
125 testset['label'] = testset['label'].map(lambda x : x.split(','))
126 testset
127
128 """### TrainSet"""
129
130 trainset['label'] = trainset['label'].map(lambda x : ' '.join(x))
131 trainset
132
133 """### TestSet"""
134
135 testset['label'] = testset['label'].map(lambda x : ' '.join(x))
136 testset
137
138 """### TrainSet"""
139
140 # clean label to numpy
141 labels = trainset['label'].values
142 labels = [[l for l in clean.split()] for clean in labels]
143 type(labels)
144
145 """### TestSet"""
146
147 # clean label to numpy
148 labeltest = testset['label'].values
149 labeltest = [[l for l in clean.split()] for clean in labeltest]
150 type(labeltest)
151
152 """### TrainSet"""
153
154 # Convert the labels column into binary label columns
155 mlb = MultiLabelBinarizer()
156 y_train = mlb.fit_transform(labels)
157
158 # Create a DataFrame with the binary label columns
159 y_train_df = pd.DataFrame(y_train, columns=mlb.classes_)
160
161 # Count the number of comments per unique label
162 y_train_counts = y_train_df.sum().sort_values(ascending=False)
163
164 """### TestSet"""
165
166 # Convert the labels column into binary label columns
167 y_test = mlb.fit_transform(labeltest)
168
169 # Create a DataFrame with the binary label columns
170 y_test_df = pd.DataFrame(y_test, columns=mlb.classes_)
171
172 # Count the number of comments per unique label
173 y_test_counts = y_test_df.sum().sort_values(ascending=False)
174
175 """## Train Test"""
176
177 X_train = trainset['comment']
178 type(X_train)
179
180 type(y_train)
181
182 X_test = testset['comment']
183 type(X_test)
184
185 type(y_test)
186
187 """## Convert Text to TF-IDF representation"""
188
189 vectorizer = TfidfVectorizer()
```

```
190
191 X_train_text = [' '.join(comment) for comment in X_train] # Convert tokenized comments back to string format
192 X_train_tfidf = vectorizer.fit_transform(X_train_text)
193
194 X_test_text = [' '.join(comment) for comment in X_test] # Convert tokenized comments back to string format
195 X_test_tfidf = vectorizer.transform(X_test_text)
196
197 print(X_train.shape)
198 print(y_train.shape)
199
200 """## Support Vector Machine"""
201
202 lpsvm_classifier = LabelPowerset(SVC(kernel='linear'))
203 lpsvm_classifier.fit(X_train_tfidf, y_train)
204 lpsvm_predictions = lpsvm_classifier.predict(X_test_tfidf)
205
206 # # Calculate the evaluation metrics with the best model
207
208 classification_rep = classification_report(y_test, lpsvm_predictions, target_names=mlb.classes_)
209 print("Classification Report:")
210 print(classification_rep)
211
212 #Hamming Loss
213 lp_hamming_loss = hamming_loss(y_test, lpsvm_predictions)
214 print("Hamming Loss:", lp_hamming_loss)
```

Label Powerset Support Vector Machine (Balanced)

source–code/svmA.py

```
1
2 """## Import Libraries for Data Preprocessing"""
3
4 import numpy as np
5 import pandas as pd
6 from subprocess import check_output
7 import matplotlib.pyplot as plt
8 from scipy.stats import bernoulli
9 import seaborn as sns
10 import nltk
11 import re
12 from nltk.tokenize import word_tokenize
13 from nltk.corpus import stopwords
14 nltk.download('punkt')
15 nltk.download('stopwords')
16
17 """## Import Libraries for Modeling"""
18
19 from sklearn.feature_extraction.text import TfidfVectorizer
20 from sklearn.preprocessing import MultiLabelBinarizer
21 from sklearn import metrics
22 from sklearn.metrics import classification_report
23
24 from skmultilearn.problem_transform import LabelPowerset
25 from sklearn.svm import SVC
26 from sklearn.metrics import f1_score, accuracy_score, hamming_loss, precision_score, recall_score
27
28 """### OThers"""
29
30 import warnings
31 warnings.filterwarnings("ignore")
32
33 from wordcloud import WordCloud, STOPWORDS
34 import plotly.express as px
35 import plotly.graph_objects as go
36 import plotly.figure_factory as ff
37 from plotly.subplots import make_subplots
38 from joblib import dump,load
39
40 trainset = pd.read_csv('aug.csv')
41 testset = pd.read_csv('testdata.csv')
42 print(trainset.shape)
43 trainset.head()
44
45 print(trainset.info())
46
47 """# Data Preprocessing
48
49 ## Converting Capital to small
50 ### TrainSet
51 """
52
53 trainset['comment'] = trainset['comment'].apply(lambda x : x.lower())
54 trainset
55
56 """### TestSet"""
57
58 testset['comment'] = testset['comment'].apply(lambda x : x.lower())
59 testset
60
61 """### Regular Expression
```

```
62
63  ### TrainSet
64  """
65
66  trainset ['comment'] = trainset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
67  trainset ['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
68  trainset ['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
69  trainset
70
71  """### TestSet"""
72
73  testset ['comment'] = testset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
74  testset ['comment'] = testset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
75  testset ['comment'] = testset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
76  testset
77
78  """### Tokenize comments
79
80  ### TrainSet
81  """
82
83  trainset ['comment'] = trainset['comment'].apply(word_tokenize)
84  trainset
85
86  """### TestSet"""
87
88  testset ['comment'] = testset['comment'].apply(word_tokenize)
89  testset
90
91  """## Removing Stopwords
92
93  ### TrainSet
94  """
95
96  # remove stop words
97  stop_words = set(stopwords.words("english"))
98  trainset ['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
99
100 with open("stopwords-tl.txt", "r") as f:
101     stop_words = f.read().split("\n")
102 trainset ['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
103 trainset
104
105 """### TestSet"""
106
107 # remove stop words
108 stop_words = set(stopwords.words("english"))
109 testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
110
111 with open("stopwords-tl.txt", "r") as f:
112     stop_words = f.read().split("\n")
113 testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
114 testset
115
116 """### Formatting labels
117
118 ### TrainSet
119 """
120
121 trainset ['label'] = trainset['label'].map(lambda x : x.split(','))
122 trainset
123
124 """## TestSet"""
125
126 testset ['label'] = testset['label'].map(lambda x : x.split(','))
127 testset
128
129 """### TrainSet"""
130
131 trainset ['label'] = trainset['label'].map(lambda x : ' '.join(x))
132 trainset
133
134 """### TestSet"""
135
136 testset ['label'] = testset['label'].map(lambda x : ' '.join(x))
137 testset
138
139 """### TrainSet"""
140
141 # clean label to numpy
142 labels = trainset['label'].values
143 labels = [[l for l in clean.split()] for clean in labels]
144 type(labels)
145
146 """### TestSet"""
147
148 # clean label to numpy
149 labeltest = testset['label'].values
150 labeltest = [[l for l in clean.split()] for clean in labeltest]
151 type(labeltest)
152
153 """### TrainSet"""
```

```
154
155 # Convert the labels column into binary label columns
156 mlb = MultiLabelBinarizer()
157 y_train = mlb.fit_transform(labels)
158
159 # Create a DataFrame with the binary label columns
160 y_train_df = pd.DataFrame(y_train, columns=mlb.classes_)
161
162 # Count the number of comments per unique label
163 y_train_counts = y_train_df.sum().sort_values(ascending=False)
164
165 """### TestSet"""
166
167 # Convert the labels column into binary label columns
168 y_test = mlb.fit_transform(labeltest)
169
170 # Create a DataFrame with the binary label columns
171 y_test_df = pd.DataFrame(y_test, columns=mlb.classes_)
172
173 # Count the number of comments per unique label
174 y_test_counts = y_test_df.sum().sort_values(ascending=False)
175
176 """## Train Test"""
177
178 X_train = trainset['comment']
179 type(X_train)
180
181 type(y_train)
182
183 X_test = testset['comment']
184 type(X_test)
185
186 type(y_test)
187
188 """## Convert Text to TF-IDF representation"""
189
190 vectorizer = TfidfVectorizer()
191
192 X_train_text = [' '.join(comment) for comment in X_train] # Convert tokenized comments back to string format
193 X_train_tfidf = vectorizer.fit_transform(X_train_text)
194
195 X_test_text = [' '.join(comment) for comment in X_test] # Convert tokenized comments back to string format
196 X_test_tfidf = vectorizer.transform(X_test_text)
197
198 print(X_train.shape)
199 print(y_train.shape)
200
201 """## Support Vector Machine"""
202
203 lpsvm_classifier = LabelPowerset(SVC(kernel='linear'))
204 lpsvm_classifier.fit(X_train_tfidf, y_train)
205 lpsvm_predictions = lpsvm_classifier.predict(X_test_tfidf)
206
207 # # Calculate the evaluation metrics with the best model
208
209 classification_rep = classification_report(y_test, lpsvm_predictions, target_names=mlb.classes_)
210 print("Classification Report:")
211 print(classification_rep)
212
213 #Hamming Loss
214 lp_hamming_loss = hamming_loss(y_test, lpsvm_predictions)
215 print("Hamming Loss:", lp_hamming_loss)
216
217 # Save as the model
218 # dump(lpsvm_classifier, 'SvmA.joblib')
219 # model = load('SvmA.joblib')
220
221 # import pickle
222 # # Save the vectorizer and transformed training data
223 # with open('vectorizer.pkl', 'wb') as file:
224 #     pickle.dump(vectorizer, file)
225 # # Save the vectorizer and transformed training data
226 # with open('mlb.pkl', 'wb') as file:
227 #     pickle.dump(mlb, file)
```

Ensemble Classifier Chain - Logistic Regression (Imbalanced)

source–code/eccLR.py

```
1
2 """## Import Libraries for Data Preprocessing"""
3
4 import numpy as np
5 import pandas as pd
6 from subprocess import check_output
7 import matplotlib.pyplot as plt
8 from scipy.stats import bernoulli
9 import seaborn as sns
10 import nltk
11 import re
12 from nltk.tokenize import word_tokenize
```

```
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')

"""## Import Libraries for Modeling"""

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn import metrics
from sklearn.metrics import  classification_report

from skmultilearn.problem_transform import LabelPowerset, ClassifierChain
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score, hamming_loss, precision_score,  recall_score

"""### OThers"""

import warnings
warnings.filterwarnings("ignore")

from wordcloud import WordCloud, STOPWORDS
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory  as  ff
from plotly.subplots import make_subplots

trainset  = pd.read_csv('traindata.csv')
testset  = pd.read_csv('testdata.csv')
print(trainset.shape)
trainset.head()

print(trainset.info())

"""# Data Preprocessing

## Converting Capital to small
### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x : x.lower())
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x : x.lower())
testset

"""### Regular Expression

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+', ' ',  x)) # Removes multiple spaces
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+', ' ',  x)) # Removes multiple spaces
testset

"""### Tokenize comments

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(word_tokenize)
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(word_tokenize)
testset

"""## Removing Stopwords

### TrainSet
"""

# remove stop words
stop_words = set(stopwords.words("english"))
trainset['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])

with open("stopwords-tl.txt", "r") as f:
    stop_words = f.read().split("\n")
trainset['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
trainset

"""### TestSet"""
```

```
105
106  # remove stop words
107  stop_words = set(stopwords.words("english"))
108  testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
109
110  with open("stopwords-tl.txt", "r") as f:
111      stop_words = f.read().split("\n")
112  testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
113  testset
114
115  """### Formatting labels
116
117  ### TrainSet
118  """
119
120  trainset ['label'] = trainset['label'].map(lambda x : x.split(','))
121  trainset
122
123  """## TestSet"""
124
125  testset ['label'] = testset['label'].map(lambda x : x.split(','))
126  testset
127
128  """### TrainSet"""
129
130  trainset ['label'] = trainset['label'].map(lambda x : ' '.join(x))
131  trainset
132
133  """### TestSet"""
134
135  testset ['label'] = testset['label'].map(lambda x : ' '.join(x))
136  testset
137
138  """### TrainSet"""
139
140  # clean label to numpy
141  labels = trainset['label'].values
142  labels = [[l for l in clean.split()] for clean in labels]
143  type(labels)
144
145  """### TestSet"""
146
147  # clean label to numpy
148  labeltest = testset['label'].values
149  labeltest = [[l for l in clean.split()] for clean in labeltest]
150  type(labeltest)
151
152  """### TrainSet"""
153
154  # Convert the labels column into binary label columns
155  mlb = MultiLabelBinarizer()
156  y_train = mlb.fit_transform(labels)
157
158  # Create a DataFrame with the binary label columns
159  y_train_df = pd.DataFrame(y_train, columns=mlb.classes_)
160
161  # Count the number of comments per unique label
162  y_train_counts = y_train_df.sum().sort_values(ascending=False)
163
164  """### TestSet"""
165
166  # Convert the labels column into binary label columns
167  y_test = mlb.fit_transform(labeltest)
168
169  # Create a DataFrame with the binary label columns
170  y_test_df = pd.DataFrame(y_test, columns=mlb.classes_)
171
172  # Count the number of comments per unique label
173  y_test_counts = y_test_df.sum().sort_values(ascending=False)
174
175  """## Train Test"""
176
177  X_train = trainset['comment']
178  type(X_train)
179
180  type(y_train)
181
182  X_test = testset['comment']
183  type(X_test)
184
185  type(y_test)
186
187  """## Convert Text to TF-IDF representation"""
188
189  vectorizer = TfidfVectorizer()
190
191  X_train_text = [' '.join(comment) for comment in X_train]  # Convert tokenized comments back to string format
192  X_train_tfidf = vectorizer.fit_transform(X_train_text)
193
194  X_test_text = [' '.join(comment) for comment in X_test]  # Convert tokenized comments back to string format
195  X_test_tfidf = vectorizer.transform(X_test_text)
196
```

```
print(X_train.shape)
print(y_train.shape)

"""## Logistic Regression"""

 ecc_lr_classifier  = ClassifierChain(LogisticRegression(C=10.0, penalty='l2', solver='liblinear'))
 ecc_lr_classifier . fit ( X_train_tfidf , y_train)
ecc_lr_predictions  =  ecc_lr_classifier .predict( X_test_tfidf )

# # Calculate the evaluation metrics with the best model

 classification_rep  =  classification_report (y_test ,  ecc_lr_predictions , target_names=mlb.classes_)
print(" Classification  Report:")
print( classification_rep )

#Hamming Loss
lp_hamming_loss = hamming_loss(y_test, ecc_lr_predictions)
print("Hamming Loss:", lp_hamming_loss)
```

Ensemble Classifier Chain - Logistic Regression (Balanced)

source–code/eccLRA.py

```

"""## Import Libraries for Data Preprocessing"""

import numpy as np
import pandas as pd
from subprocess import check_output
import matplotlib.pyplot as plt
from scipy.stats import bernoulli
import seaborn as sns
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')

"""## Import Libraries for Modeling"""

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn import metrics
from sklearn.metrics import  classification_report

from skmultilearn.problem_transform import LabelPowerset, ClassifierChain
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score, hamming_loss, precision_score,  recall_score

"""### OThers"""

import warnings
warnings.filterwarnings("ignore")

from wordcloud import WordCloud, STOPWORDS
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots

trainset = pd.read_csv('aug.csv')
testset = pd.read_csv('testdata.csv')
print(trainset.shape)
trainset.head()

print(trainset.info())

"""# Data Preprocessing

## Converting Capital to small
### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x : x.lower())
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x : x.lower())
testset

"""### Regular Expression

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
trainset
```

```
69
70  """### TestSet"""
71
72  testset ['comment'] = testset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
73  testset ['comment'] = testset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
74  testset ['comment'] = testset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
75  testset
76
77  """### Tokenize comments
78
79  ### TrainSet
80  """
81
82  trainset ['comment'] = trainset['comment'].apply(word_tokenize)
83  trainset
84
85  """### TestSet"""
86
87  testset ['comment'] = testset['comment'].apply(word_tokenize)
88  testset
89
90  """## Removing Stopwords
91
92  ### TrainSet
93  """
94
95  # remove stop words
96  stop_words = set(stopwords.words("english"))
97  trainset ['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
98
99  with open("stopwords-tl.txt", "r") as f:
100     stop_words = f.read().split("\n")
101 trainset ['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
102 trainset
103
104 """### TestSet"""
105
106 # remove stop words
107 stop_words = set(stopwords.words("english"))
108 testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
109
110 with open("stopwords-tl.txt", "r") as f:
111     stop_words = f.read().split("\n")
112 testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
113 testset
114
115 """### Formatting labels
116
117 ### TrainSet
118 """
119
120 trainset ['label'] = trainset['label'].map(lambda x : x.split(','))
121 trainset
122
123 """## TestSet"""
124
125 testset ['label'] = testset['label'].map(lambda x : x.split(','))
126 testset
127
128 """### TrainSet"""
129
130 trainset ['label'] = trainset['label'].map(lambda x : ' '.join(x))
131 trainset
132
133 """### TestSet"""
134
135 testset ['label'] = testset['label'].map(lambda x : ' '.join(x))
136 testset
137
138 """### TrainSet"""
139
140 # clean label to numpy
141 labels = trainset['label'].values
142 labels = [[l for l in clean.split()] for clean in labels]
143 type(labels)
144
145 """### TestSet"""
146
147 # clean label to numpy
148 labeltest = testset['label'].values
149 labeltest = [[l for l in clean.split()] for clean in labeltest]
150 type(labeltest)
151
152 """### TrainSet"""
153
154 # Convert the labels column into binary label columns
155 mlb = MultiLabelBinarizer()
156 y_train = mlb.fit_transform(labels)
157
158 # Create a DataFrame with the binary label columns
159 y_train_df = pd.DataFrame(y_train, columns=mlb.classes_)
160
```

```
# Count the number of comments per unique label
y_train_counts = y_train_df.sum().sort_values(ascending=False)

"""### TestSet"""

# Convert the labels column into binary label columns
y_test = mlb.fit_transform(labeltest)

# Create a DataFrame with the binary label columns
y_test_df = pd.DataFrame(y_test, columns=mlb.classes_)

# Count the number of comments per unique label
y_test_counts = y_test_df.sum().sort_values(ascending=False)

"""## Train Test"""

X_train = trainset['comment']
type(X_train)

type(y_train)

X_test = testset['comment']
type(X_test)

type(y_test)

"""## Convert Text to TF-IDF representation"""

vectorizer = TfidfVectorizer()

X_train_text = [' '.join(comment) for comment in X_train] # Convert tokenized comments back to string format
X_train_tfidf = vectorizer.fit_transform(X_train_text)

X_test_text = [' '.join(comment) for comment in X_test] # Convert tokenized comments back to string format
X_test_tfidf = vectorizer.transform(X_test_text)

print(X_train.shape)
print(y_train.shape)

"""## Logistic Regression"""

 ecc_lr_classifier = ClassifierChain(LogisticRegression(C=10.0, penalty='l2', solver='liblinear'))
 ecc_lr_classifier.fit(X_train_tfidf, y_train)
ecc_lr_predictions = ecc_lr_classifier.predict(X_test_tfidf)

# # Calculate the evaluation metrics with the best model

classification_rep = classification_report(y_test, ecc_lr_predictions, target_names=mlb.classes_)
print("Classification Report:")
print(classification_rep)

#Hamming Loss
lp_hamming_loss = hamming_loss(y_test, ecc_lr_predictions)
print("Hamming Loss:", lp_hamming_loss)
```

Ensemble Classifier Chain - Support Vector Machine (Imbalanced)

source–code/eccSVM.py

```

"""## Import Libraries for Data Preprocessing"""

import numpy as np
import pandas as pd
from subprocess import check_output
import matplotlib.pyplot as plt
from scipy.stats import bernoulli
import seaborn as sns
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')

"""## Import Libraries for Modeling"""

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn import metrics
from sklearn.metrics import classification_report

from skmultilearn.problem_transform import LabelPowerset, ClassifierChain
from sklearn.svm import SVC
from sklearn.metrics import f1_score, accuracy_score, hamming_loss, precision_score, recall_score

"""### OThers"""

import warnings
warnings.filterwarnings("ignore")

```

```
from wordcloud import WordCloud, STOPWORDS
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots

trainset = pd.read_csv('traindata.csv')
testset = pd.read_csv('testdata.csv')
print(trainset.shape)
trainset.head()

print(trainset.info())

"""# Data Preprocessing

## Converting Capital to small
### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x : x.lower())
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x : x.lower())
testset

"""### Regular Expression

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
testset

"""### Tokenize comments

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(word_tokenize)
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(word_tokenize)
testset

"""## Removing Stopwords

### TrainSet
"""

# remove stop words
stop_words = set(stopwords.words("english"))
trainset['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])

with open("stopwords-tl.txt", "r") as f:
    stop_words = f.read().split("\n")
trainset['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
trainset

"""### TestSet"""

# remove stop words
stop_words = set(stopwords.words("english"))
testset['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])

with open("stopwords-tl.txt", "r") as f:
    stop_words = f.read().split("\n")
testset['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
testset

"""### Formatting labels

### TrainSet
"""

trainset['label'] = trainset['label'].map(lambda x : x.split(','))
trainset

"""## TestSet"""
```

```
testset ['label'] = testset['label'].map(lambda x : x.split(','))
testset

"""### TrainSet"""

trainset['label'] = trainset['label'].map(lambda x : ' '.join(x))
trainset

"""### TestSet"""

testset['label'] = testset['label'].map(lambda x : ' '.join(x))
testset

"""### TrainSet"""

# clean label to numpy
labels = trainset['label'].values
labels = [[l for l in clean.split()] for clean in labels]
type(labels)

"""### TestSet"""

# clean label to numpy
labeltest = testset['label'].values
labeltest = [[l for l in clean.split()] for clean in labeltest]
type(labeltest)

"""### TrainSet"""

# Convert the labels column into binary label columns
mlb = MultiLabelBinarizer()
y_train = mlb.fit_transform(labels)

# Create a DataFrame with the binary label columns
y_train_df = pd.DataFrame(y_train, columns=mlb.classes_)

# Count the number of comments per unique label
y_train_counts = y_train_df.sum().sort_values(ascending=False)

"""### TestSet"""

# Convert the labels column into binary label columns
y_test = mlb.fit_transform(labeltest)

# Create a DataFrame with the binary label columns
y_test_df = pd.DataFrame(y_test, columns=mlb.classes_)

# Count the number of comments per unique label
y_test_counts = y_test_df.sum().sort_values(ascending=False)

"""## Train Test"""

X_train = trainset['comment']
type(X_train)

type(y_train)

X_test = testset['comment']
type(X_test)

type(y_test)

"""## Convert Text to TF-IDF representation"""

vectorizer = TfidfVectorizer()

X_train_text = [' '.join(comment) for comment in X_train] # Convert tokenized comments back to string format
X_train_tfidf = vectorizer.fit_transform(X_train_text)

X_test_text = [' '.join(comment) for comment in X_test] # Convert tokenized comments back to string format
X_test_tfidf = vectorizer.transform(X_test_text)

print(X_train.shape)
print(y_train.shape)

"""## Ensemble Classifier Chain SVM"""

eccsvm_classifier = ClassifierChain(SVC(C=10.0, gamma='scale', kernel='rbf'))
eccsvm_classifier.fit(X_train_tfidf, y_train)
eccsvm_predictions = eccsvm_classifier.predict(X_test_tfidf)

# # Calculate the evaluation metrics with the best model

classification_rep = classification_report(y_test, eccsvm_predictions, target_names=mlb.classes_)
print("Classification Report:")
print(classification_rep)

#Hamming Loss
lp_hamming_loss = hamming_loss(y_test, eccsvm_predictions)
print("Hamming Loss:", lp_hamming_loss)
```

Ensemble Classifier Chain - Support Vector Machine (Balanced)

```
1
2  """## Import Libraries for Data Preprocessing"""
3
4  import numpy as np
5  import pandas as pd
6  from subprocess import check_output
7  import matplotlib.pyplot as plt
8  from scipy.stats import bernoulli
9  import seaborn as sns
10 import nltk
11 import re
12 from nltk.tokenize import word_tokenize
13 from nltk.corpus import stopwords
14 nltk.download('punkt')
15 nltk.download('stopwords')
16
17 """## Import Libraries for Modeling"""
18
19 from sklearn.feature_extraction.text import TfidfVectorizer
20 from sklearn.preprocessing import MultiLabelBinarizer
21 from sklearn import metrics
22 from sklearn.metrics import  classification_report
23
24 from skmultilearn.problem_transform import LabelPowerset, ClassifierChain
25 from sklearn.svm import SVC
26 from sklearn.metrics import f1_score, accuracy_score, hamming_loss, precision_score,  recall_score
27
28 """### OThers"""
29
30 import warnings
31 warnings.filterwarnings("ignore")
32
33 from wordcloud import WordCloud, STOPWORDS
34 import plotly.express as px
35 import plotly.graph_objects as go
36 import plotly.figure_factory as ff
37 from plotly.subplots import make_subplots
38
39 trainset = pd.read_csv('aug.csv')
40 testset = pd.read_csv('testdata.csv')
41 print(trainset.shape)
42 trainset.head()
43
44 print(trainset.info())
45
46 """# Data Preprocessing
47
48 ## Converting Capital to small
49 ### TrainSet
50 """
51
52 trainset ['comment'] = trainset['comment'].apply(lambda x : x.lower())
53 trainset
54
55 """### TestSet"""
56
57 testset ['comment'] = testset['comment'].apply(lambda x : x.lower())
58 testset
59
60 """### Regular Expression
61
62 ### TrainSet
63 """
64
65 trainset ['comment'] = trainset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
66 trainset ['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
67 trainset ['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
68 trainset
69
70 """### TestSet"""
71
72 testset ['comment'] = testset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
73 testset ['comment'] = testset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
74 testset ['comment'] = testset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
75 testset
76
77 """### Tokenize comments
78
79 ### TrainSet
80 """
81
82 trainset ['comment'] = trainset['comment'].apply(word_tokenize)
83 trainset
84
85 """### TestSet"""
86
87 testset ['comment'] = testset['comment'].apply(word_tokenize)
88 testset
89
90 """## Removing Stopwords
```

```
91
92  ### TrainSet
93  """
94
95  # remove stop words
96  stop_words = set(stopwords.words("english"))
97  trainset ['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
98
99  with open("stopwords-tl.txt", "r") as f:
100     stop_words = f.read().split("\n")
101 trainset ['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
102 trainset
103
104 """### TestSet"""
105
106 # remove stop words
107 stop_words = set(stopwords.words("english"))
108 testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
109
110 with open("stopwords-tl.txt", "r") as f:
111     stop_words = f.read().split("\n")
112 testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
113 testset
114
115 """### Formatting labels
116
117 ### TrainSet
118 """
119
120 trainset ['label'] = trainset['label'].map(lambda x : x.split(','))
121 trainset
122
123 """## TestSet"""
124
125 testset ['label'] = testset['label'].map(lambda x : x.split(','))
126 testset
127
128 """### TrainSet"""
129
130 trainset ['label'] = trainset['label'].map(lambda x : ' '.join(x))
131 trainset
132
133 """### TestSet"""
134
135 testset ['label'] = testset['label'].map(lambda x : ' '.join(x))
136 testset
137
138 """### TrainSet"""
139
140 # clean label to numpy
141 labels = trainset['label'].values
142 labels = [[l for l in clean.split()] for clean in labels]
143 type(labels)
144
145 """### TestSet"""
146
147 # clean label to numpy
148 labeltest = testset['label'].values
149 labeltest = [[l for l in clean.split()] for clean in labeltest]
150 type(labeltest)
151
152 """### TrainSet"""
153
154 # Convert the labels column into binary label columns
155 mlb = MultiLabelBinarizer()
156 y_train = mlb.fit_transform(labels)
157
158 # Create a DataFrame with the binary label columns
159 y_train_df = pd.DataFrame(y_train, columns=mlb.classes_)
160
161 # Count the number of comments per unique label
162 y_train_counts = y_train_df.sum().sort_values(ascending=False)
163
164 """### TestSet"""
165
166 # Convert the labels column into binary label columns
167 y_test = mlb.fit_transform(labeltest)
168
169 # Create a DataFrame with the binary label columns
170 y_test_df = pd.DataFrame(y_test, columns=mlb.classes_)
171
172 # Count the number of comments per unique label
173 y_test_counts = y_test_df.sum().sort_values(ascending=False)
174
175 """## Train Test"""
176
177 X_train = trainset['comment']
178 type(X_train)
179
180 type(y_train)
181
182 X_test = testset['comment']
```

```
type(X_test)

type(y_test)

"""## Convert Text to TF-IDF representation"""

vectorizer = TfidfVectorizer()

X_train_text = [' '.join(comment) for comment in X_train]  # Convert tokenized comments back to string format
X_train_tfidf = vectorizer.fit_transform(X_train_text)

X_test_text = [' '.join(comment) for comment in X_test]  # Convert tokenized comments back to string format
X_test_tfidf = vectorizer.transform(X_test_text)

print(X_train.shape)
print(y_train.shape)

"""## Ensemble Classifier Chain SVM"""

eccsvm_classifier = ClassifierChain(SVC(C=10.0, gamma='scale', kernel='rbf'))
eccsvm_classifier.fit(X_train_tfidf, y_train)
eccsvm_predictions = eccsvm_classifier.predict(X_test_tfidf)

# # Calculate the evaluation metrics with the best model

classification_rep = classification_report(y_test, eccsvm_predictions, target_names=mlb.classes_)
print("Classification Report:")
print(classification_rep)

#Hamming Loss
lp_hamming_loss = hamming_loss(y_test, eccsvm_predictions)
print("Hamming Loss:", lp_hamming_loss)
```

MLP (Imbalanced)

source–code/MLP.py

```

"""## Import Libraries for Data Preprocessing"""

import numpy as np
import pandas as pd
from subprocess import check_output
import matplotlib.pyplot as plt
from scipy.stats import bernoulli
import seaborn as sns
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')

"""## Import Libraries for Modeling"""

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn import metrics
from sklearn.metrics import classification_report

from sklearn.neural_network import MLPClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import f1_score, accuracy_score, hamming_loss, precision_score, recall_score

"""### OThers"""

import warnings
warnings.filterwarnings("ignore")

from wordcloud import WordCloud, STOPWORDS
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots

trainset = pd.read_csv('traindata.csv')
testset = pd.read_csv('testdata.csv')
print(trainset.shape)
trainset.head()

print(trainset.info())

"""# Data Preprocessing

## Converting Capital to small
### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x : x.lower())
trainset

```

```
"""### TestSet"""

testset ['comment'] = testset['comment'].apply(lambda x : x.lower())
testset

"""### Regular Expression

### TrainSet
"""

trainset ['comment'] = trainset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
trainset ['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
trainset ['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
trainset

"""### TestSet"""

testset ['comment'] = testset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
testset ['comment'] = testset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
testset ['comment'] = testset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
testset

"""### Tokenize comments

### TrainSet
"""

trainset ['comment'] = trainset['comment'].apply(word_tokenize)
trainset

"""### TestSet"""

testset ['comment'] = testset['comment'].apply(word_tokenize)
testset

"""## Removing Stopwords

### TrainSet
"""

# remove stop words
stop_words = set(stopwords.words("english"))
trainset ['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])

with open("stopwords-tl.txt", "r") as f:
    stop_words = f.read().split("\n")
trainset ['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
trainset

"""### TestSet"""

# remove stop words
stop_words = set(stopwords.words("english"))
testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])

with open("stopwords-tl.txt", "r") as f:
    stop_words = f.read().split("\n")
testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
testset

"""### Formatting labels

### TrainSet
"""

trainset ['label'] = trainset['label'].map(lambda x : x.split(','))
trainset

"""## TestSet"""

testset ['label'] = testset['label'].map(lambda x : x.split(','))
testset

"""### TrainSet"""

trainset ['label'] = trainset['label'].map(lambda x : ' '.join(x))
trainset

"""### TestSet"""

testset ['label'] = testset['label'].map(lambda x : ' '.join(x))
testset

"""### TrainSet"""

# clean label to numpy
labels = trainset['label'].values
labels = [[l for l in clean.split()] for clean in labels]
type(labels)

"""### TestSet"""

```

```
# clean label to numpy
labeltest = testset['label'].values
labeltest = [[l for l in clean.split()] for clean in labeltest]
type(labeltest)

"""### TrainSet"""

# Convert the labels column into binary label columns
mlb = MultiLabelBinarizer()
y_train = mlb.fit_transform(labels)

# Create a DataFrame with the binary label columns
y_train_df = pd.DataFrame(y_train, columns=mlb.classes_)

# Count the number of comments per unique label
y_train_counts = y_train_df.sum().sort_values(ascending=False)

"""### TestSet"""

# Convert the labels column into binary label columns
y_test = mlb.fit_transform(labeltest)

# Create a DataFrame with the binary label columns
y_test_df = pd.DataFrame(y_test, columns=mlb.classes_)

# Count the number of comments per unique label
y_test_counts = y_test_df.sum().sort_values(ascending=False)

"""## Train Test"""

X_train = trainset['comment']
type(X_train)

type(y_train)

X_test = testset['comment']
type(X_test)

type(y_test)

"""## Convert Text to TF-IDF representation"""

vectorizer = TfidfVectorizer()

X_train_text = [' '.join(comment) for comment in X_train] # Convert tokenized comments back to string format
X_train_tfidf = vectorizer.fit_transform(X_train_text)

X_test_text = [' '.join(comment) for comment in X_test] # Convert tokenized comments back to string format
X_test_tfidf = vectorizer.transform(X_test_text)

print(X_train.shape)
print(y_train.shape)

# Create the base classifier
base_classifier = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', random_state=42)

# Create the One-vs-Rest Classifier
ovr_classifier = OneVsRestClassifier(base_classifier)

# Train the model
ovr_classifier.fit(X_train_tfidf, y_train)

# Make predictions
ovr_predictions = ovr_classifier.predict(X_test_tfidf)

# Calculate the evaluation metrics
classification_rep_ovr = classification_report(y_test, ovr_predictions, target_names=mlb.classes_)
print("Classification Report (One-vs-Rest MLP):")
print(classification_rep_ovr)

# Hamming Loss
ovr_hamming_loss = hamming_loss(y_test, ovr_predictions)
print("Hamming Loss (One-vs-Rest MLP):", ovr_hamming_loss)
```

MLP (Balanced)

source–code/MLPA.py

```

"""## Import Libraries for Data Preprocessing"""

import numpy as np
import pandas as pd
from subprocess import check_output
import matplotlib.pyplot as plt
from scipy.stats import bernoulli
import seaborn as sns
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
```

```
nltk.download('punkt')
nltk.download('stopwords')

"""## Import Libraries for Modeling"""

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn import metrics
from sklearn.metrics import  classification_report

from sklearn.neural_network import MLPClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import f1_score, accuracy_score, hamming_loss, precision_score,  recall_score

"""### OThers"""

import warnings
warnings.filterwarnings("ignore")

from wordcloud import WordCloud, STOPWORDS
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots

trainset = pd.read_csv('aug.csv')
testset = pd.read_csv('testdata.csv')
print(trainset.shape)
trainset.head()

print(trainset.info())

"""# Data Preprocessing

## Converting Capital to small
### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x : x.lower())
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x : x.lower())
testset

"""### Regular Expression

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
trainset['comment'] = trainset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(lambda x: re.sub('[^a-zA-Z]', ' ', x)) #Removes numbers
testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+[a-zA-Z]\s+', ' ', x)) # Removes single characters
testset['comment'] = testset['comment'].apply(lambda x: re.sub('\s+', ' ', x)) # Removes multiple spaces
testset

"""### Tokenize comments

### TrainSet
"""

trainset['comment'] = trainset['comment'].apply(word_tokenize)
trainset

"""### TestSet"""

testset['comment'] = testset['comment'].apply(word_tokenize)
testset

"""## Removing Stopwords

### TrainSet
"""

# remove stop words
stop_words = set(stopwords.words("english"))
trainset['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])

with open("stopwords-tl.txt", "r") as f:
    stop_words = f.read().split("\n")
trainset['comment'] = trainset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
trainset

"""### TestSet"""

```

```
# remove stop words
stop_words = set(stopwords.words("english"))
testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])

with open("stopwords-tl.txt", "r") as f:
    stop_words = f.read().split("\n")
testset ['comment'] = testset['comment'].apply(lambda x: [word for word in x if word not in stop_words])
testset

"""### Formatting labels

### TrainSet
"""

trainset ['label'] = trainset['label'].map(lambda x : x.split(','))
trainset

"""## TestSet"""

testset ['label'] = testset['label'].map(lambda x : x.split(','))
testset

"""### TrainSet"""

trainset ['label'] = trainset['label'].map(lambda x : ' '.join(x))
trainset

"""### TestSet"""

testset ['label'] = testset['label'].map(lambda x : ' '.join(x))
testset

"""### TrainSet"""

# clean label to numpy
labels = trainset['label'].values
labels = [[l for l in clean.split()] for clean in labels]
type(labels)

"""### TestSet"""

# clean label to numpy
labeltest = testset['label'].values
labeltest = [[l for l in clean.split()] for clean in labeltest]
type(labeltest)

"""### TrainSet"""

# Convert the labels column into binary label columns
mlb = MultiLabelBinarizer()
y_train = mlb.fit_transform(labels)

# Create a DataFrame with the binary label columns
y_train_df = pd.DataFrame(y_train, columns=mlb.classes_)

# Count the number of comments per unique label
y_train_counts = y_train_df.sum().sort_values(ascending=False)

"""### TestSet"""

# Convert the labels column into binary label columns
y_test = mlb.fit_transform(labeltest)

# Create a DataFrame with the binary label columns
y_test_df = pd.DataFrame(y_test, columns=mlb.classes_)

# Count the number of comments per unique label
y_test_counts = y_test_df.sum().sort_values(ascending=False)

"""## Train Test"""

X_train = trainset['comment']
type(X_train)

type(y_train)

X_test = testset['comment']
type(X_test)

type(y_test)

"""## Convert Text to TF-IDF representation"""

vectorizer = TfidfVectorizer()

X_train_text = [' '.join(comment) for comment in X_train]  # Convert tokenized comments back to string format
X_train_tfidf = vectorizer.fit_transform(X_train_text)

X_test_text = [' '.join(comment) for comment in X_test]  # Convert tokenized comments back to string format
X_test_tfidf = vectorizer.transform(X_test_text)

print(X_train.shape)
```

```
print(y_train.shape)

# Create the base classifier
base_classifier = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', random_state=42)

# Create the One-vs-Rest Classifier
ovr_classifier = OneVsRestClassifier(base_classifier)

# Train the model
ovr_classifier.fit(X_train_tfidf, y_train)

# Make predictions
ovr_predictions = ovr_classifier.predict(X_test_tfidf)

# Calculate the evaluation metrics
classification_rep_ovr = classification_report(y_test, ovr_predictions, target_names=mlb.classes_)
print("Classification Report (One-vs-Rest MLP):")
print(classification_rep_ovr)

# Hamming Loss
ovr_hamming_loss = hamming_loss(y_test, ovr_predictions)
print("Hamming Loss (One-vs-Rest MLP):", ovr_hamming_loss)
```

index HTML

source–code/index.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="{% static 'main.css' %}">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZL
</head>

<body>
    <!-- Start Navbar -->
    <header>

        <div class="container-fluid">

         <div class="nav-logo">
            <a href="{% url 'text'%}"><img src ="{% static 'CommentSense.png' %}"></a>
         </div>

            <div class="navb-items d-none d-xl-flex">

                <div class="item">
               <a href="{% url 'text'%}" style="text-decoration:none">Text</a>
                </div>

                <div class="item">
               <a href="{% url 'comment'%}" style="text-decoration:none">Comments</a>
                </div>

                <div class="item-button">
                    <a href="https://www.facebook.com/EndocrineWitch" type="button">Facebook</a>
                </div>
            </div>

            <!-- Button trigger modal -->
            <div class="mobile-toggler d-lg-none">
                <a href="#" data-bs-toggle="modal" data-bs-target="#navbModal">
                    <i class="fa-solid fa-bars"></i>
                </a>
            </div>

            <!-- Modal -->
            <div class="modal fade" id="navbModal" tabindex="-1" aria-labelledby="exampleModalLabel"
                aria-hidden="true">
                <div class="modal-dialog">
                    <div class="modal-content">
                        <div class="modal-header">
                            <img src ="{% static 'CommentSense.png' %}">
                            <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"><i class="fa-solid fa-xmark"><
                        </div>
                        <div class="modal-body">
                            <div class="modal-line">
                                <i class="fa-solid fa-font"></i><a href="{% url 'text'%}" style="text-decoration:none">Text</a>
                            </div>

                            <div class="modal-line">
                                <i class="fa-solid fa-comments"></i><a href="{% url 'comment'%}" style="text-decoration:none">Comments</a>
                            </div>
                            <a href="https://www.facebook.com/EndocrineWitch" class="navb-button" type="button">Facebook</a>
                        </div>
                    </div>
                </div>
```

```
            </div>

        </div>
    </header>
    <!-- End Navbar -->
{% block content %}
{% endblock content %}
    <script src="https://kit.fontawesome.com/af45447f8c.js" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6

</body>
</html>
```

text HTML

source–code/text.html

```
{% extends 'supportApp/index.html' %}
{% load static %}

{% block content %}
<title>Text Classifier</title>

<!-- Start Navbar -->

<form action="" method="post">
    <div class="container" style="width: 75%; margin-top: 30px; padding: 20px">
        <h3 style="font-weight: bolder;">Multilabel Text Classifier</h3>
        <hr>
        {% csrf_token %}
        <div class="form-floating">
            <!-- <textarea class="form-control" placeholder="Leave a comment here" id="floatingTextarea2" style="height: 100px"></textarea> -->
            {{form.text}}
            <label for="{{form.text.id_for_label}}">Enter Text</label>
        </div>
        <hr>
        <div class="d-grid col-6 mx-auto">
            <button class="navb-button" type="submit">LABEL</button>
        </div>
        <!-- {% for label_tuple in predictions %}
            {% for label in label_tuple %}
                <span class="badge bg-secondary">{{ label }}</span>
            {% endfor %}
        {% endfor %} -->
        {% for label_tuple in predictions %}
            {% for label in label_tuple %}
                {% if label == 'Appraisal' %}
                    <span class="badge bg-primary">{{ label }}</span>
                {% elif label == 'Informational' %}
                    <span class="badge bg-success">{{ label }}</span>
                {% elif label == 'Instrumental' %}
                    <span class="badge bg-warning text-dark">{{ label }}</span>
                {% elif label == 'Emotional' %}
                    <span class="badge bg-info text-dark">{{ label }}</span>
                {% elif label == 'Spam' %}
                    <span class="badge bg-secondary">{{ label }}</span>
                {% else %}
                    <span class="badge bg-dark">{{ label }}</span>
                {% endif %}
            {% endfor %}
        {% endfor %}
    </div>
</form>


<!-- End Navbar -->

{% endblock content %}
```

comments HTML

source–code/comments.html

```
{% extends 'supportApp/index.html' %}
{% load static %}

{% block content %}
<title>Facebook Comments Classifier</title>

<!-- Start Navbar -->
<div class="container px-4 text-center">
  <div class="row gx-5">
    <form class="d-flex" method="post" enctype="multipart/form-data">
      <div class="col">
        <div class="p-3">
          {% csrf_token %}
          {{form.csvcomments}}
        </div>
      </div>
```

```
        <div class="col">
          <div class="p-3">
            <button type="submit" class="btn btn-primary">Upload File</button>
          </div>
        </div>
        <!-- Add the Extract FB Data button -->
        <div class="col">
          <div class="p-3">
            <button type="button" class="btn btn-primary" id="extractBtn">Extract FB Data</button> <!-- Added button for extracting FB data --
          </div>
        </div>
        <div class="col">
          <div class="p-3">
              <a href="{% url 'download_extracted' %}" class="btn btn-primary" id="downloadBtn" style="display: none;">Download Extracted Data</a
          </div>
        </div>
      </form>
    </div>
  </div>
  <br>
  {% if predictions_filename %}
    <a href="{% url 'download_predictions' %}" class="btn btn-primary">Download Predictions</a>
  {% endif %}
  <br>
  {% if df|length > 0 %}
      <table class="table">
          <thead>
              <tr>
                  <th>Comment</th>
                  <th>Prediction</th>
                  {% if 'permalink_url' in df.columns %}
                      <th>Actions</th> <!-- Add a new column for actions -->
                  {% endif %}
              </tr>
          </thead>
          <tbody>
              {% for index, row in df.iterrows %}
              <tr>
                  <td>{{ row.comment }}</td>
                  <td>{{ row.prediction }}</td>
                  {% if 'permalink_url' in df.columns %}
                      <td>
                          <a href="{{ row.permalink_url }}" target="_blank" class="btn btn-primary">Go to</a>
                      </td>
                  {% endif %}
              </tr>
              {% endfor %}
          </tbody>
      </table>
  {% else %}
      <p>No predictions available.</p>
  {% endif %}

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    // AJAX request to trigger Python script
    $(document).ready(function() {
      $('#extractBtn').click(function() {
        var extractBtn = $(this);
        $.ajax({
          url: "{% url 'extract_fb_data' %}",
          type: "POST",
          headers: { "X-CSRFToken": "{{ csrf_token }}" },
          success: function(response) {
            console.log(response);
            // Show the "Download Extracted Data" button
            $('#downloadBtn').show();
            extractBtn.prop('disabled', true); // Disable the "Extract FB Data" button
          },
          error: function(xhr, status, error) {
            console.error(xhr.responseText);
          }
        });
      });
    });
  </script>

  {% endblock content %}
```

main css

source–code/main.css

```
@import url('https://fonts.googleapis.com/css2?family=Urbanist:wght@300;400;500;600;700&display=swap');

body{

    font-family: 'Urbanist', sans-serif;
    overflow-x: hidden;
}

```

```
 9  a{
10
11      text-decoration: none;
12      color: currentColor;
13  }
14
15  a:hover{
16
17      color: currentColor;
18  }
19
20  header{
21
22      width: 100vw;
23      height: 60px;
24      background-color: #fff;
25      box-shadow: 0px 3px 8px rgba(0, 0, 0, 25%);
26      display: flex;
27      align-items: center;
28  }
29
30  header .container-fluid{
31
32      width: 100%;
33      padding: 0 60px;
34      display: flex;
35      align-items: center;
36      justify-content: space-between;
37  }
38
39  @media(max-width:992px){
40
41      header .container-fluid{
42
43          padding: 0 5%;
44      }
45  }
46
47  header .navb-logo img{
48
49      width: 140px;
50      height: 66px;
51  }
52
53  header .navb-items{
54
55      display: flex;
56      align-items: center;
57      justify-content: flex-end;
58      letter-spacing: 3px;
59  }
60
61  header .item{
62
63      text-align: center;
64      margin-inline: 15%;
65      font-size: 20px;
66      font-weight: bold;
67      letter-spacing: 3px;
68      color: #102447;
69      padding: 5px 0;
70      transition: all .1s ease;
71      border-bottom: 0px solid #64d6f4;
72      border-top: 0px solid #64d6f4;
73      cursor: pointer;
74  }
75
76  header .item:hover{
77
78      border-bottom: 3px solid #64d6f4;
79      border-top: 3px solid #64d6f4;
80  }
81
82  header .item-button a{
83
84      background-color: #274d8a;
85      width: 150px;
86      height: 50px;
87      display: flex;
88      justify-content: center;
89      align-items: center;
90      font-size: 16px;
91      font-weight: 600;
92      color: #fff;
93      transition: all .5s ease;
94      text-decoration:none;
95  }
96
97  header .item-button a:hover{
98
99      background-color: #64d6f4;
100     text-decoration:none;
```

```
}

header .mobile-toggler{

    font-size: 30px;
}

/* modal */

.modal-dialog{

    margin: 0;
    width: 430px;
}

@media(max-width: 450px){

    .modal-dialog{

        width: 82%;
    }
}

.modal-content{

    border-radius: 0;
    height: 100vh;
    overflow-y: scroll;
    background-color: #102447;
}

.modal-header{

    display: flex;
    justify-content: space-between;
    align-items: center;
    width: 88%;
    margin: 0 auto;
    padding-bottom: 16px;
    border-bottom: 2px solid #fefefe;
}

.modal-header img{

    width: 140px;
    height: 66px;
    margin-top: 17.5px;
}

.modal-header .btn-close{

    background: transparent;
    opacity: 1;
}

.modal-header i{

    color: #fefefe;
    font-size: 30px;
}

.modal-body{

    width: 88%;
    margin: 0 auto;
    padding: 75px 0 0 0;
    flex: unset;
}

.modal-body .modal-line{

    width: 100%;
    display: flex;
    align-items: center;
    justify-content: flex-start;
    padding: 7px 0;
    margin-bottom: 50px;
    cursor: pointer;
    transition: all .5s ease;
    color: #274d8a;
    border-bottom: 1px solid #274d8a;
}

.modal-body .modal-line:hover{

    color: #64d6f4;
    border-bottom: 1px solid #64d6f4;
}

.modal-line a{

    font-size: 17px;
```

```
193         font-weight: 500;
194         letter-spacing: 2.5px;
195         color: #274d8a;
196     }
197
198     .modal-line i{
199
200         color: currentColor;
201         font-size: 30px;
202         width: 35px;
203         margin-right: 15px;
204         padding: 0 0 3px 3px;
205     }
206
207     .navb-button{
208
209         width: 100%;
210         height: 47px;
211         background-color: #fefefe;
212         display: flex;
213         justify-content: center;
214         align-items: center;
215         font-size: 20px;
216         font-weight: 700;
217         color: #274d8a;
218         letter-spacing: 2px;
219         transition: all .5s ease;
220     }
221
222     .navb-button:hover{
223
224         background-color: #274d8a;
225         color: #fefefe;
226     }
227
228     .section-1{
229
230         width: 100vw;
231         height: 95vh;
232         display: flex;
233         justify-content: center;
234         align-items: center;
235     }
236
237     .section-1 p{
238
239         font-size: 75px;
240         font-weight: 700;
241         color: #102447;
242         width: 90%;
243         text-align: center;
244     }
245
246
247     /* Add more CSS styles for other labels as needed */
248
249
250     @media(max-width:767px){
251
252         .section-1 p{
253
254             font-size: 50px;
255             text-align: start;
256             width: 70%;
257             margin: auto;
258         }
259     }
```

Comment Extraction: sensitive data

source–code/comment.py

```
1   import requests
2   import csv
3   import re
4   import emoji
5
6   def remove_emojis(text):
7       text = emoji.demojize(text)
8       text = re.sub(r':[a-zA-Z_]+:', ' ', text)
9       return text.strip()
10
11  def get_fb_page_comments(page_id, access_token):
12      base_url = f"https://graph.facebook.com/{page_id}/posts"
13      params = {
14          "access_token": access_token,
15          "fields": "comments{id,message,permalink_url}",
16          "limit": 100
17      }
18
19      comments = []
```

```
20
21      while True:
22          response = requests.get(base_url, params=params)
23          data = response.json()
24
25          print(data)  # Print the response data for debugging
26
27          if "data" not in data:
28              break
29
30          for post_data in data["data"]:
31              if "comments" in post_data:
32                  post_comments = post_data["comments"]["data"]
33                  for comment in post_comments:
34                      comment_text = comment["message"]
35                      comment_text = remove_emojis(comment_text)
36                      comment["message"] = comment_text
37                      comment["comment_id"] = comment["id"]  # Add the comment ID to the comment
38                      comment["post_id"] = post_data["id"]  # Add the post ID to the comment
39                      comment["permalink_url"] = comment.get("permalink_url", "")  # Add the permalink URL to the comment
40                  comments.extend(post_comments)
41
42          if "paging" in data:
43              if "next" not in data["paging"]:
44                  break
45
46              next_url = data["paging"]["next"]
47              params = {}
48              url_parts = next_url.split("?")
49              if len(url_parts) > 1:
50                  query_string = url_parts[1]
51                  params = dict(q.split("=") for q in query_string.split("&"))
52          else:
53              break
54
55      return comments
56
57  # Usage example
58  page_id = "393074590777768" #sensitive data
59  access_token = "EAAKoA8VgyuwBAE2ZCsPeDPRifRpeE3GhZBBt1PNdQmwblaCvTdtySfe7uZAZCcrltrTRW4868twwErr68eGZBWLdcH8iq57cEYhQ0EZCg
60
61  page_comments = get_fb_page_comments(page_id, access_token)
62
63  csv_file = "page_comments.csv"
64  with open(csv_file, "w", newline="", encoding="utf-8") as file:
65      fieldnames = ["comment", "comment_id", "post_id", "permalink_url"]  # Updated fieldnames
66      writer = csv.DictWriter(file, fieldnames=fieldnames)
67      writer.writeheader()
68
69      for comment in page_comments:
70          comment_data = {
71              "comment": comment["message"],
72              "comment_id": comment["comment_id"],
73              "post_id": comment["post_id"],
74              "permalink_url": comment.get("permalink_url", f"https://www.facebook.com/{page_id}_{comment['post_id']}?comment_id={comment['comment
75          }
76          writer.writerow(comment_data)
77
78  print(f"Comments saved to {csv_file}.")
```

Comment Extraction: sensitive data

source–code/comment.py

```
1   import requests
2   import csv
3   import re
4   import emoji
5
6   def remove_emojis(text):
7       text = emoji.demojize(text)
8       text = re.sub(r':[a-zA-Z_]+:', ' ', text)
9       return text.strip()
10
11  def get_fb_page_comments(page_id, access_token):
12      base_url = f"https://graph.facebook.com/{page_id}/posts"
13      params = {
14          "access_token": access_token,
15          "fields": "comments{id,message,permalink_url}",
16          "limit": 100
17      }
18
19      comments = []
20
21      while True:
22          response = requests.get(base_url, params=params)
23          data = response.json()
24
25          print(data)  # Print the response data for debugging
26
27          if "data" not in data:
```

```
                break

            for post_data in data["data"]:
                if "comments" in post_data:
                    post_comments = post_data["comments"]["data"]
                    for comment in post_comments:
                        comment_text = comment["message"]
                        comment_text = remove_emojis(comment_text)
                        comment["message"] = comment_text
                        comment["comment_id"] = comment["id"]  # Add the comment ID to the comment
                        comment["post_id"] = post_data["id"]  # Add the post ID to the comment
                        comment["permalink_url"] = comment.get("permalink_url", "")  # Add the permalink URL to the comment
                    comments.extend(post_comments)

            if "paging" in data:
                if "next" not in data["paging"]:
                    break

                next_url = data["paging"]["next"]
                params = {}
                url_parts = next_url.split("?")
                if len(url_parts) > 1:
                    query_string = url_parts[1]
                    params = dict(q.split("=") for q in query_string.split("&"))
            else:
                break

        return comments

    # Usage example
    page_id = "393074590777768" #sensitive data
    access_token = "EAAKoA8VgyuwBAE2ZCsPeDPRifRpeE3GhZBBt1PNdQmwblaCvTdtySfe7uZAZCcrltrTRW4868twwErr68eGZBWLdcH8iq57cEYhQ0EZCg

    page_comments = get_fb_page_comments(page_id, access_token)

     csv_file = "page_comments.csv"
    with open(csv_file, "w", newline="", encoding="utf-8") as file:
        fieldnames = ["comment", "comment_id", "post_id", "permalink_url"]  # Updated fieldnames
        writer = csv.DictWriter(file, fieldnames=fieldnames)
        writer.writeheader()

        for comment in page_comments:
            comment_data = {
                "comment": comment["message"],
                "comment_id": comment["comment_id"],
                "post_id": comment["post_id"],
                "permalink_url": comment.get("permalink_url", f"https://www.facebook.com/{page_id}_{comment['post_id']}?comment_id={comment['comment
            }
            writer.writerow(comment_data)

    print(f"Comments saved to {csv_file}.")
```

App.py

source–code/apps.py

```
from django.apps import AppConfig
class SupportappConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'supportApp'
```

views.py

source–code/views.py

```
from django.shortcuts import render,redirect
from .forms import IndividualTextForm, BulkCommentsForm
from joblib import dump,load
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
import csv
import pandas as pd
from django.core.files.base import ContentFile
import datetime
import os
from django.conf import settings
from django.http import HttpResponse, HttpResponseBadRequest
import pickle
import subprocess
import requests
import csv
import re
import emoji

# TextCNN Text Classifier

def text(request):
    if request.method == "POST":
```

```
            form = IndividualTextForm(request.POST)
            if form.is_valid():
                text = form.cleaned_data['text']
                model = load('supportApp/model/SvmA.joblib')

                # Load the MultiLabelBinarizer
                with open('supportApp/model/vectorizer.pkl', 'rb') as le:
                    vectorizer = pickle.load(le)

                with open('supportApp/model/mlb.pkl', 'rb') as le:
                    mlb = pickle.load(le)

                # Preprocess the text input
                new_sequence = vectorizer.transform([text])
                # new_sequence = pad_sequences(new_sequence, padding = 'post',maxlen =64)

                # Make predictions
                new_pred = model.predict(new_sequence)

                # Threshold the predictions
                threshold = 0.3  # Adjust this threshold based on your problem and model performance
                thresholded_pred = (new_pred > = threshold).astype(int)

                # Decode the predicted labels
                predicted_labels = mlb.inverse_transform(thresholded_pred)

                # Print the predicted labels
                print("Predicted Labels:", predicted_labels)

                return render(request, 'supportApp/text.html', {'text': text, 'predictions': predicted_labels, 'form': form})
        else:
            form = IndividualTextForm()

        return render(request, 'supportApp/text.html', {'form': form})

def extract_fb_data(request):
        if request.method == 'POST':
            subprocess.run(['python', 'supportApp/comment.py'])  # Execute the Python script
            csv_le = "page_comments.csv"  # Replace with the actual path of the extracted CSV le

            if os.path.exists(csv_le):
                # Set the extracted lename in the session
                request.session['extracted_lename'] = csv_le

                return HttpResponse(csv_le, content_type='text/plain')

            return HttpResponseBadRequest('Extracted CSV le not found.')  # Return a bad request response if the le doesn't exist

        return HttpResponseBadRequest('Invalid request method.')

def download_extracted(request):
        extracted_lename = request.session.get('extracted_lename')
        if extracted_lename:
            le_path = os.path.join(settings.MEDIA_ROOT, extracted_lename)
            if os.path.exists(le_path):
                with open(le_path, 'rb') as le:
                    response = HttpResponse(le.read(), content_type='text/csv')
                    response['Content-Disposition'] = 'attachment; lename=' + extracted_lename  # Use 'extracted_lename' instead of 'predictions_lename
                    return response

        return HttpResponseBadRequest('Extracted CSV le not found.')
def comment(request):
        if request.method == "POST":
            form = BulkCommentsForm(request.POST, request.FILES)
            if form.is_valid():
                csv_le = form.cleaned_data['csvcomments']
                if csv_le:
                    # Read the CSV le using pandas
                    df = pd.read_csv(csv_le)
                    # Load the model
                    model = load('supportApp/model/SvmA.joblib')

                    # Load the MultiLabelBinarizer
                    with open('supportApp/model/vectorizer.pkl', 'rb') as le:
                        vectorizer = pickle.load(le)

                    with open('supportApp/model/mlb.pkl', 'rb') as le:
                        mlb = pickle.load(le)

                    # Check if 'permalink_url' column exists in the DataFrame
                    if 'permalink_url' in df.columns:
                        permalink_urls = df['permalink_url']
                        include_permalink = True
                    else:
                        include_permalink = False

                    # Iterate over each row in the DataFrame
                    predicted_labels = []
                    for index, row in df.iterrows():
                        comment_text = row['comment']
                        if pd.isna(comment_text):  # Check if comment_text is np.nan
                            predicted_labels.append('Unknown')
```

```
                        else:
                            # Preprocess the text input
                            new_sequence = vectorizer.transform([comment_text])

                            # Make predictions
                            new_pred = model.predict(new_sequence)

                            # Threshold the predictions
                            threshold = 0.3  # Adjust this threshold based on your problem and model performance
                            thresholded_pred = (new_pred >= threshold).astype(int)

                            # Decode the predicted labels and convert to formatted strings
                            predicted_labels.append(','.join ([',  '.join(label.split (','))  if  labels  and labels != [()]  else  'Unknown' for labels in mlb.invers

                    # Create a new DataFrame with comments, predictions, and permalink URLs (if applicable)
                    if include_permalink:
                        new_df = pd.DataFrame({'comment': df['comment'], 'prediction': predicted_labels, 'permalink_url':  permalink_urls})
                    else:
                        new_df = pd.DataFrame({'comment': df['comment'], 'prediction': predicted_labels})

                    filename = "predictions.csv"

                    # Save the new DataFrame to a CSV file
                    new_df.to_csv(filename,  index=False)

                    # Set the predictions filename in the session
                    request.session['predictions_filename'] = filename

                    # Render the template with the updated DataFrame and predictions
                    return render(request,  'supportApp/comments.html', {'df': new_df, 'predictions_filename':  filename,  'form':  form})
        else:
            form = BulkCommentsForm()

        # Check if predictions filename is set in the session
        predictions_filename = request.session.get('predictions_filename')
        if predictions_filename:
            # Remove the predictions file from the previous session
            os.remove(predictions_filename)
            # Clear the predictions filename from the session
            del request.session['predictions_filename']

        return render(request,  'supportApp/comments.html', {'form': form})

def download_predictions(request):
        predictions_filename = request.session.get('predictions_filename')
        if predictions_filename:
            file_path = os.path.join(settings.MEDIA_ROOT, predictions_filename)
            if os.path.exists(file_path):
                with open(file_path,  'rb') as file:
                    response = HttpResponse(file.read(), content_type='text/csv')
                    response['Content-Disposition'] = 'attachment; filename=' + predictions_filename
                    return response
        return render(request,  'supportApp/comments.html')
```

urls.py

source–code/urls.py

```
from django.urls import path
from .import views
from django.urls import re_path

urlpatterns = [
    path('',  views.text,  name='text'),
    re_path(r'^comment(?:/(?P<csv_filename>[^/]+))?/$', views.comment, name='comment'),
    path('extract_fb_data/',  views.extract_fb_data,  name='extract_fb_data'),
    path('download_predictions/', views.download_predictions, name='download_predictions'),
    path('download-csv/', views.download_extracted, name='download_extracted'),
]
```

XI. Acknowledgment

I would like to express my deepest gratitude to my SP adviser, Sir Geoff A. Solano, for his unwavering guidance, support, and encouragement throughout the entire journey of this SP. Their expertise, insightful feedback, and invaluable connections have been instrumental in shaping this SP and pushing me to achieve my best.

I would also like to extend my heartfelt appreciation to Sir Jasper Kyle Catapang for his valuable insights and assistance in shedding light on various aspects of this SP. His contributions have greatly enriched my understanding and have been invaluable in finishing this SP.

I am also grateful to Ms. Perlita Gasmen, Mrs. Sheila Magboo, and Sir Vincent Magboo for their valuable input, constructive criticism, and valuable suggestions that have significantly contributed to the refinement of this SP. Their expertise in their respective fields has brought a wealth of knowledge and perspective to my research, enabling me to develop a more comprehensive and better system.

To my family (my mom, dad, lola, and brother), friends, especially my dearest Amigos, and my kuya sponsor, I extend my sincere appreciation for standing by me during this challenging academic endeavor and for their unwavering support and encouragement. Your belief in my abilities and constant motivation has been a driving force, inspiring me to persevere and overcome obstacles along the way. Your love and understanding have provided the necessary strength and comfort during the ups and downs of this journey.

I would also like to express my heartfelt thanks to Dr. Iris-Isip Tan for her contributions to the base idea of this SP. Her dedication to helping others and providing valuable insights into the field of endocrinology has been truly inspiring. I wish her all the best in her profession and continued success in making a positive impact on people's lives.

Lastly, I would like to express my gratitude to God, for whom I am truly thankful.

His guidance, blessings, and unwavering presence have been my source of strength and inspiration throughout this academic endeavor.

This SP would not have been possible without the support and contributions of you remarkable individuals and many others who have played a part, no matter how big or small. I am truly humbled and honored to have had the opportunity to work with such exceptional mentors, advisors, and friends, and to have their unwavering support throughout this academic journey.

I sincerely thank all of you.