Interactive Text Classification & Evaluation

Client/Instructor: Mohamed Farag

Jarvis Ly, Jason Tran, Subeom Kwon, and Thinh Truong

CS 4624 Multimedia, Hypertext, and Information Access

Virginia Tech

Blacksburg VA 24061

April 27, 2023

Table of Contents

List of Figures

List of Tables

Abstract

Text classification is a critical task in natural language processing that assigns predefined categories or labels to text documents. It has become even more important than ever with the rapid growth in the sheer number of text documents with the introduction of social media. It is highly practical to have a machine classify these documents rather than a human manually identifying the contents of a document. Starting in 2007, a project from the Google Summer of Code program released a free Python machine-learning library that featured many classification, regression, and clustering tools. This project will be based on this library to perform the necessary text classification from generating a model to outputting a prediction given text.

The goal of this project is to create an interactive text classifier with the web application, user, and developer manuals as deliverables. Our team will work closely with a client to ensure that our application is on track and fits their needs. The main objective is to develop a web application that allows the user to interact with a machine-learning text classification model by tracking its correctness based on the principle of supervised machine learning. The UI should display keywords that were used to classify the text and highlight them to the user. The interactive portion of the application comes from the fact that the user will be able to classify the text themselves, mark down whether the highlighted text is right or wrong and save the document for future reference.

This project is the first of its kind this Spring 2023 semester and no previous groups in other semesters have done a project like this. Our group will have to start from scratch, and use tools and technologies that are unfamiliar to us but have the willingness to learn them. Our approach in building this application is to use a similar stack to MERN but instead of Expressjs, we opt to use Flask as our back-end server to handle our scikit-learn machine learning script. We use Reactjs as the front-end framework, and Nodejs to run the application and use other features. Lastly, we use MongoDB as our database to store documents, classifications, and other important attributes. We hope that our project will provide valuable insight into the effectiveness and power of machine learning and allow those who wish to continue our project to be able to with ease through reading our user and developer manuals in this report.

1 Introduction

1.1 Background

Text classification is the process of categorizing text documents into predefined classes or categories based on their content. The need for text classification stems from the increasing amount of textual data available in various forms such as emails, social media posts, news articles, and product reviews. Text is one of the most common types of unstructured data where analyzing, understanding, organizing, and sorting through text data is difficult and time-consuming. With the advent of machine learning, text classifiers can significantly expedite and automate this process.

1.2 Objective

The primary objective of this project is to develop a system that allows users to interactively evaluate text classifier performance conducted on a user-supplied webpage. The web application will serve as a binary text classifier, meaning that it will determine whether or not the text data belongs to a particular category. The system will display the classifier results to the user as well as what words within the supplied webpage contributed to reaching the classifier output. The user will then be able to interact with the output by providing their own highlights and annotations as to whether or not certain words were effectively used in determining the classifier result.

1.3 Deliverables

The purpose of our application is to provide an interactive binary text classifier on user-supplied webpages. We intend to achieve a fully functional web application by completing the deliverables listed below:

- A Python script which conducts the text preprocessing and text classification
- A database that stores classifier output and user annotations
- User Interface which allows file uploading and text annotations.

1.4 Team Acknowledgements

Our team consisted of Jason Tran, Thinh Truong, Subeom Kwon, and Jarvis Ly. All team members are seniors majoring in Computer Science at Virginia Tech graduating in Spring 2023.

2 Requirements

In the following section, we describe what needs to be accomplished in order to have a fully functional product as requested by the client. The final product shall be a web application that shows the content of a given user-uploaded text file, allow a text classifier to be applied to the text of the text file and show the output of the text classifier prediction.

2.1 Text Preprocessing

We utilize scikit-learn which is a Python library that provides the tools that we will use to perform the preprocessing. To feed the text data into the machine learning model, we must first turn the text content into numerical feature vectors. This was conducted via a bag of words representation which stores the unique words and their occurrences within the text. We then filter stopwords within the text which are common words found in text that would not have much significance in classifying text. Finally, we downscale the weights for words using tf-idf which is “Term Frequency times Inverse Document Frequency”. This involves dividing the number of occurrences for each word by the total number of words, allowing us to balance the weights for each word within the text data. Having completed these steps, we can now feed our data to the classifier for training.

2.2 Training and Testing the Machine Learning Model

To achieve proficient accuracy with the model, we must supply an adequately sized dataset for the purpose of training and testing the model. For our dataset, we used the 20 newsgroups text dataset for our training and testing. This dataset comprises around 18,000 newsgroups posts on 20 topics. This dataset is split into two subsets: one for training and one for testing.

2.3 User Interface

Upon launch of the web application, the user will be greeted with the option to enter text manually or upload a text file. After having done so, the text data from the file will be extracted and displayed to the user. After running the classifier, the classifier results are displayed to the user. Additionally, The user has the option to sign in via Google and save previous classifications to the database. These previous classifications will be displayed via a sidebar.

3 Design

3.1 High-Level Application Architecture

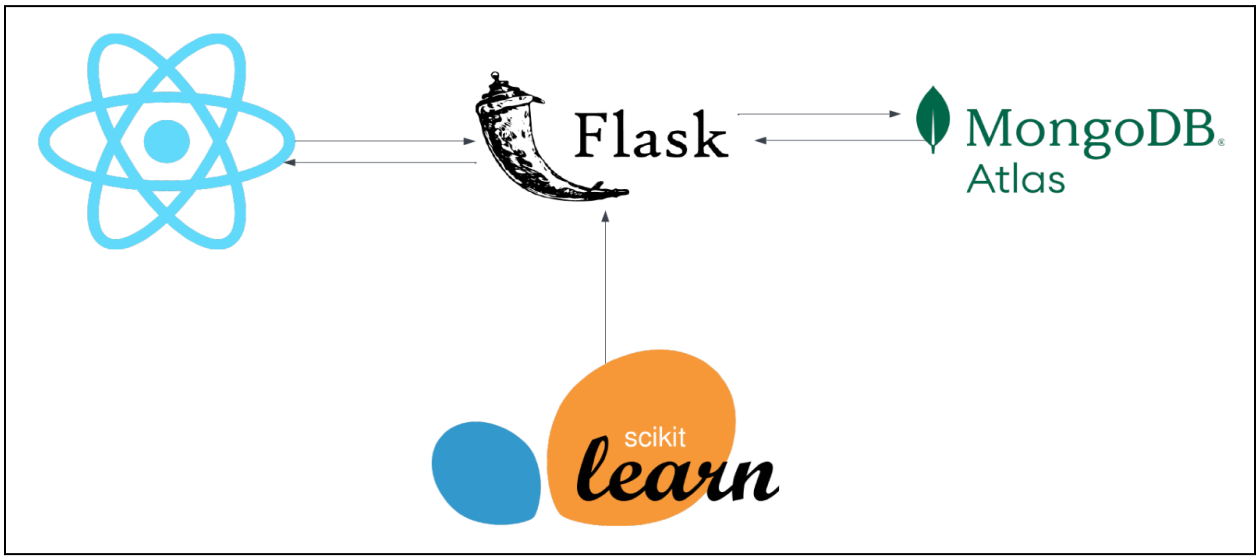

Figure 1: System Architecture

3.2 Default App Layout Wireframe

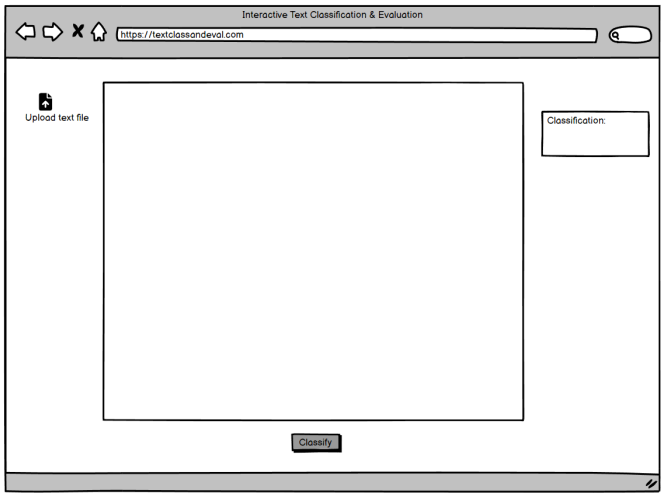

Figure 2: Home Page

When the application is first loaded, several elements are presented to the user by default. The user can view a file upload button, a text box, a classification box, and the classify button. At the start, the user only has access to the upload button.

3.3 File Upload Wireframe

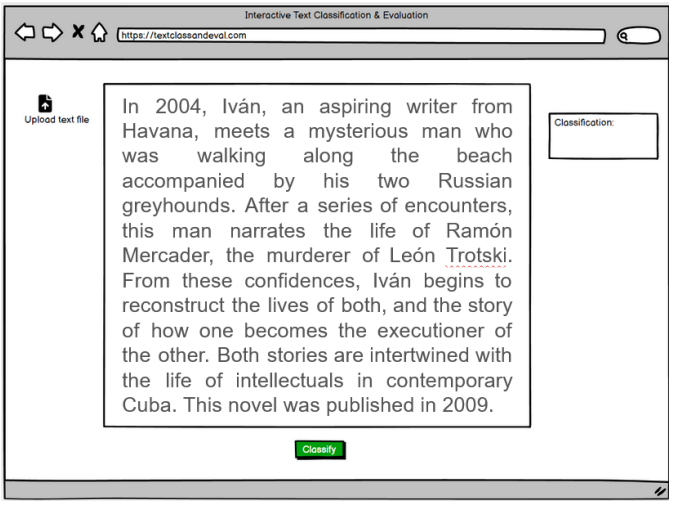

Figure 3: Page After a File Has Been Uploaded

Once the valid file has been uploaded, the file's text will appear in the text box. Additionally, the classify button is now accessible to the user.

3.4 Classification Wireframe

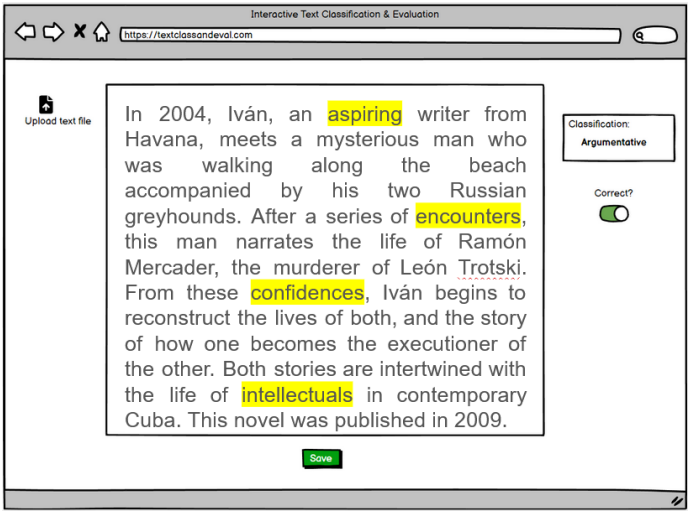

Figure 4: Page After the Text Has Been Classified

After the classify button has been pressed, the user will be presented with the classification result from the machine learning model. Words that were significant to the classification result are highlighted as well. Additionally, a correctness switch is now accessible and is on by default.

3.5 Evaluation Wireframe

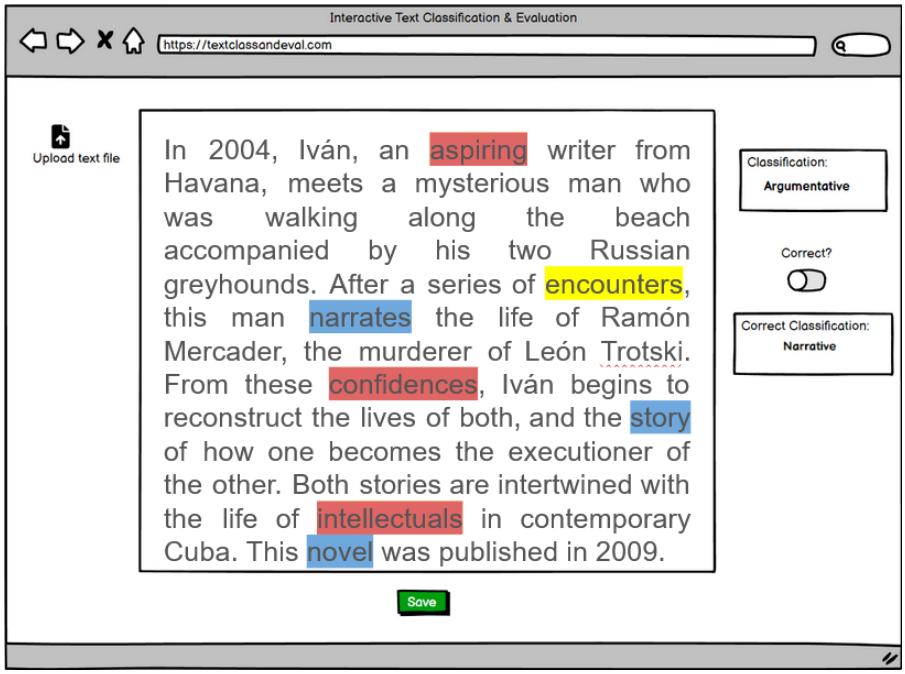

Figure 5: Page with User Evaluation Edits

If the user is dissatisfied with the classification result, new evaluation options will be available after the correctness switch is turned off. The user will be able to remove and add new highlighted words and even keep existing ones. Red highlighted words signify removed words and blue signifies newly added words. Furthermore, the user can provide the correct classification in the new classification box.

4 Implementation

4.1 General Flow of the Application

The Front-end of the classifier displays a text-field that allows the user to input the text they want to classify. If the user clicks on the submit button, it makes a POST request to the /classify endpoint using axios. This /classify endpoint is created by Flask library in the classify.py script.

POST request with user input:

```
const [responseData, setResponseData] = useState('');
const [inputText, setInputText] = useState('');

const handleClick = async () => {
        try {
        // Make POST request to the Flask API
        const response = await axios.post('http://localhost:5002/classify', { text: inputText });
        const data = response.data;
        console.log(data);
        setResponseData(data); // Update state with response data
        } catch (error) {
        console.error(error);
        }
};
const handleInputChange = (event) => {
        setInputText(event.target.value);
};
```

4.2 train_and_save_model.py (Machine Learning Model Training / Testing)

This script constitutes the training and testing for the machine learning model. We first start the script by defining a category that the AI can classify the text into.

```
categories = ['soc.religion.christian','comp.graphics', 'sci.med',
        'sci.electronics','sci.space', 'sci.crypt','rec.sport.baseball',
        'rec.sport.hockey', 'rec.autos','talk.politics.guns']
```

Subsequently, the train and test datasets are fetched from the “20_newsgroup” corpus, and a pipeline consisting of two essential steps is constructed. The first step involves leveraging the TF-IDF (term frequency-inverse document frequency) technique, which involves assigning numerical values based on the word’s significance.

The succeeding step in the pipeline entails the implementation of the actual classifier model that will be deployed to classify the provided text data. Specifically, a Naive Bayes classifier with a multinomial distribution is trained, which is suitable for text classification tasks incorporating TF-IDF values. Following this, the fit method of the Pipeline object is invoked to facilitate the training of the classifier on the training data. In this process, the pipeline first applies the TfidfVectorizer to the text data, which generates a numerical representation, and subsequently trains the MultinomialNB classifier, leveraging the provided target labels.

Ultimately, the trained model and the twenty_train object are saved to binary files using joblib.dump() function.

```
twenty_train = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, random_state=42)
twenty_test = fetch_20newsgroups(subset='test', categories=categories, shuffle=True, random_state=42)
text_clf = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('clf', MultinomialNB())
])
text_clf.fit(twenty_train.data, twenty_train.target)

predicted_test = text_clf.predict(twenty_test.data)
test_accuracy = accuracy_score(twenty_test.target, predicted_test)

joblib.dump(text_clf, 'text_classifier.joblib')
joblib.dump(twenty_train, 'twenty_train.joblib')
```

4.3 classifier.py (Back-end / Prediction)

The classifier.py script incorporates the Flask library to establish an API for text categorization. This API creates an endpoint that accepts textual input, which is subsequently classified into one of the predefined categories through the utilization of a pre-trained machine learning model. The classified category is then returned as a response. Ultimately, the front-end interface receives the outcome in JSON format, updating the state accordingly. Consequently, the classification result is presented to the user via the front-end display.

The following section provides an in-depth examination of the classifier.py implementation:

We start by creating a new instance of Flask class:

```
app = Flask(__name__)
```

Next, we load the saved pre-trained model and the twenty_train object and assign them to `text_clf` and `twenty_train` variables respectively:

```
text_clf = joblib.load('text_classifier.joblib')
```

```
twenty_train = joblib.load('twenty_train.joblib')
```

Subsequently, a category_map is constructed to enhance the readability of the classification results, thereby facilitating a more comprehensible presentation for the end user.

```
category_map = {
        'soc.religion.christian': 'Religion/Christian',
        'comp.graphics': 'Computer/Graphics',
        'sci.med': 'Science/Medicine',
        'sci.electronics': 'Science/Electronics',
        'sci.space': 'Science/Space',
        'sci.crypt': 'Science/Cryptocurrency',
        'rec.sport.baseball': 'Sports/Baseball',
        'rec.sport.hockey': 'Sports/Hockey',
        'rec.autos': 'Automobile',
        'talk.politics.guns': 'Politics/Guns',
}
```

Following this, the app.route decorator is employed to define a '/classify' endpoint, which serves as the communication channel between the back-end and front-end components. The 'methods' argument explicitly specifies that this endpoint will solely accommodate POST requests.

Within the classify_text() function, the text input from the front-end is acquired via a POST request and subsequently processed before being stored in the 'user_inputs' array. The classification is then executed using the pre-trained model, leveraging scikit-learn's integrated predict() function. Finally, the results are returned in a JSON format for further processing and display.

```
@app.route('/classify', methods=['POST'])
def classify_text():
        if request.method == 'POST':
        # Get the text from the request
        input_text = request.json['text']
        user_inputs = [input_text]

        # Perform the classification
        predicted = text_clf.predict(user_inputs)
        for category in predicted:
        output_string = category_map[twenty_train.target_names[category]]
        return jsonify(result=output_string)
```

Lastly, the script initiates the Flask application with debug mode enabled, listening for incoming requests on port 5002 (localhost:5002).

```
if __name__ == '__main__':
        app.run(debug=True, port=5002)
```

4.4 Database.py (Back-end / Prediction)

Database.py is designed to interact with a MongoDB database to save a user's text classification history. It first establishes a connection to the MongoDB server using the connection string URI stored in an environment variable (.env file). Then, it selects the 'User_History' database and defines a save_classification_history function to store information such as user ID, input text, classification results, important words, user's interpretation of the result, and any highlighted text by the user. The function inserts this data as a document into the 'User_History' collection within the database and returns the unique ObjectID of the inserted document.

```
MONGODB_URI = os.environ["MONGODB_URI"]
client = MongoClient(MONGODB_URI, tlsCAFile=certifi.where())
db = client.User_History

def save_classification_history(user_id, input_text, classifier_result, important_words, user_result,
user_highlight):
    history_collection = db.User_History
    history_record = {
        "user_id": user_id,
        "input_text": input_text,
        "classifier_result": classifier_result,
        "important_words": important_words,
        "user_result": user_result,
        "user_highlight": user_highlight
    }
    result = history_collection.insert_one(history_record)
    return result.inserted_id
```

5 Testing/Evaluation/Assessment

5.1 TF-IDF (Term-Frequency-Inverse Document Frequency)

TF-IDF uses the TfidfVectorizer() class from the sklearn.feature_extraction.text module. This step transforms the raw text into a numerical representation by calculating the Term Frequency-Inverse Document Frequency (TF-IDF) for each word in the text. This representation is useful for text analysis tasks, as it emphasizes the importance of each word in the context of the entire dataset.

TF-IDF is a product of two statistics: Term Frequency and Inverse Document Frequency.

Term Frequency Formula:

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

Where $f_{t,d}$ is the raw count of terms in a document.

Inverse document frequency Formula:

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Where *N* is the total number of documents.

5.2 Multinomial Naïve Bayes

We used Multinomial Naïve Bayes model for the classification. This step contains tokenization of raw texts, and includes important processes such as Feature Extraction and Prediction.

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_{i=1}^{n} x_i)!}{\prod_{i=1}^{n} x_i!} \prod_{i=1}^{n} {p_{ki}}^{x_i}$$

The multinomial naive Bayes classifier becomes a linear classifier when expressed in log-space:

$$\log p(C_k \mid \mathbf{x}) \propto \log\left(p(C_k)\prod_{i=1}^{n} {p_{ki}}^{x_i}\right)$$
$$= \log p(C_k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki}$$
$$= b + \mathbf{w}_k^\top \mathbf{x}$$

$$w_{ki} = \log p_{ki}$$

5.3 Testing and Evaluation of Text Classification Model

We used joblib to train and test the classifier and saved the model as a binary file so that we can easily load this classifier when we start the application. We then manually tested the application with texts from online articles and tested edge cases that might lead to errors (ex. putting one word as an input).

Test accuracy: 0.88
Classification Report:

	precision	recall	f1-score	support
comp.graphics	0.94	0.77	0.85	389
rec.autos	0.93	0.94	0.94	396
rec.sport.baseball	0.94	0.90	0.92	397
rec.sport.hockey	0.91	0.97	0.94	399
sci.crypt	0.70	0.97	0.82	396
sci.electronics	0.93	0.65	0.77	393
sci.med	0.97	0.76	0.85	396
sci.space	0.91	0.90	0.90	394
soc.religion.christian	0.76	0.98	0.86	398
talk.politics.guns	0.94	0.93	0.94	364
accuracy			0.88	3922
macro avg	0.89	0.88	0.88	3922
weighted avg	0.89	0.88	0.88	3922

Figure 6: Test Accuracy of Text Classifier

6 User's Manual

6.1 Installation

The source code for the application is available via GitHub at the link:
https://github.com/subeom7/text-classification-and-evaluation

For the most streamlined method of accessing the application, we would recommend using Git to obtain the project files, however using Git is not required.

If the user is installing the project manually, the user would access the project repository, click the clone button, then click download zip button. The user would then unzip the file.

If Git is not already installed, the user can reference the following link for instructions on how to install Git on their device: https://github.com/git-guides/install-git

If Git is already installed, the user would first clone the repository by copying the HTTPS repository link and typing the command "git clone https://github.com/subeom7/text-classification-and-evaluation.git" in the terminal. This will install all the source files into the current working directory.

All the project files can be found inside the folder titled Text-Classification-and-Evaluation. The user should enter the folder via the terminal by running *cd Text-Classification-and-Evaluation*. The user then would run the command "npm run install-all" to install all project dependencies. After having installed all dependencies, the user can run the application by using the command "npm run dev".

Note: If the user is encountering python: command not found error, a potential fix would be to enter the package.json file and edit the file to change instances of python to python3 or vice versa depending on your version of python installed.

DISCLAIMER: AT THE TIME OF THIS REPORT THE USER INTERFACE IS NOT STANDARDIZED FOR ALL PLATFORMS. IF USER ENCOUNTERS AND OVERLAPPING OF USER INTERFACE ELEMENTS, THE USER SHOULD DECREASE THEIR ZOOM PERCENTAGE WITHIN THEIR BROWSER.

6.2 Guest Mode

The application has support for Google authentication, so the user has the option to utilize the text classifier in guest mode or while signed in. We recommend the user to sign in using a Google account to take advantage of all the features offered within the application, if the user wants to utilize the text classifier without having their results be archived, they have the option to do so.

6.3 User Mode

The user would click the Sign In option located at the top of the page and select a Google account of their choosing. After they sign in, they will be met with the following screen which now includes the user history side bar which is currently empty. The name of the user that is currently signed in will also be displayed on the top right corner of the screen and the user will have the option to sign out of the account if necessary.

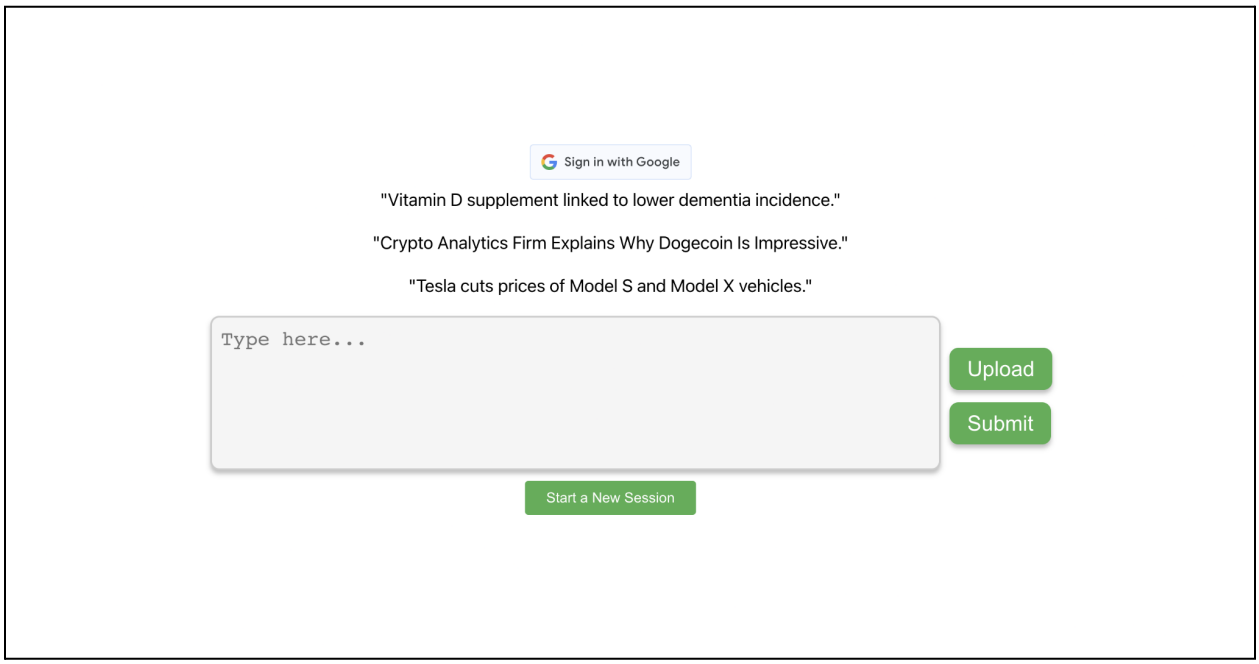

Figure 7: Default Page View

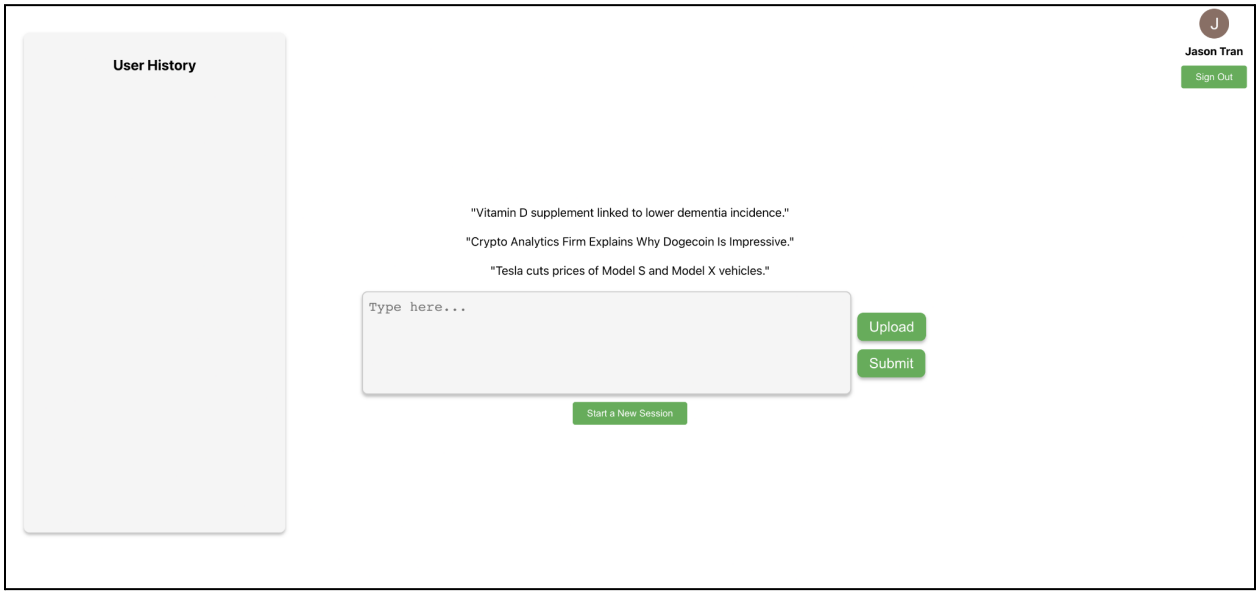

Figure 8: After Login

The user has the option to either manually input the text that they want classified in the provided text box or upload a zip file that contains *.txt* files to perform multiple classifications at once. We have provided sample inputs above the text field which the user can use to test out the application. However, they have the ability to enter whatever text input they desire. If the user uploads a zip file, a list of all the *.txt* files that are present in the zip file will be displayed. The user can click on the names of the various *.txt* files to view the different input text and outputs.

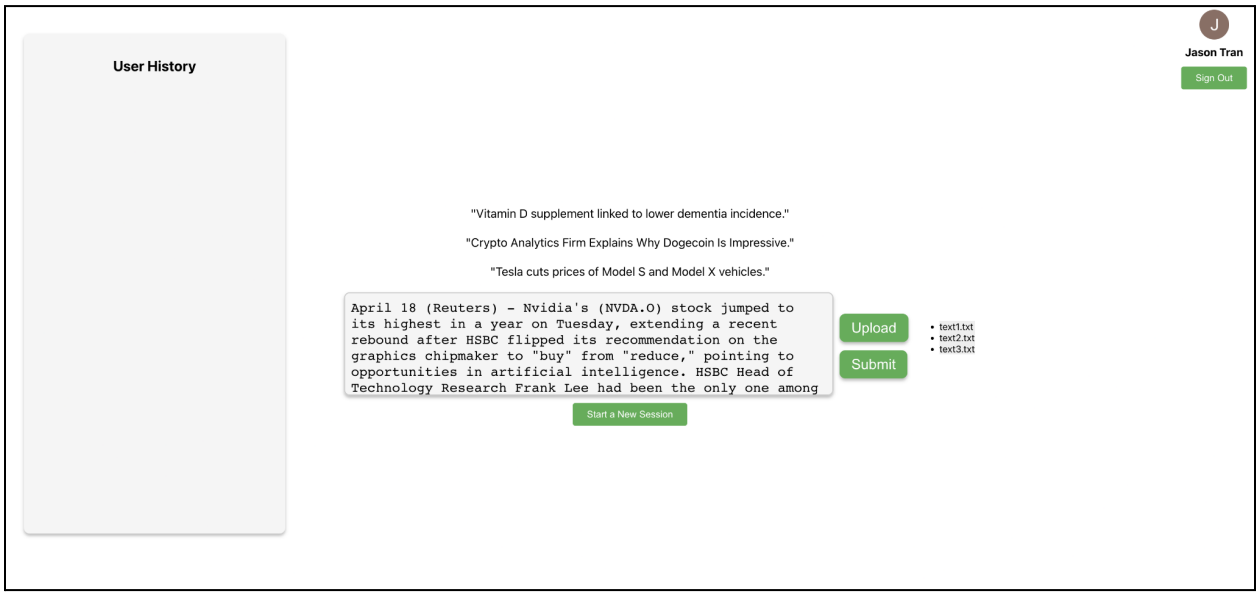

Figure 9: User Input via Zip File

Once the user has either manually input their text input or zip file, they can then click the submit button which will execute the text classifier. The outputs from the classification(s) will be displayed which includes:

AI Classification – what category the AI predicted the text belongs

Important Words – Words with heavy significance on the determination of the AI's classification

Your Classification – User selects what category the input text belongs

Highlights – A view of the input text with the important words highlighted

Figure 10: After Submit

The user would then select what category the input should belong to, at the current state of the application there are ten different categories in which the classifier can make its predictions. The user also has the option to highlight additional words, that were not listed in the important words, that they believe had high significance towards the determination of the AI's classification.

Figure 11: User Highlight / User Classification

The user can now save the output to the database which will then be displayed in the user history sidebar. This information will be maintained on the user's account and will persist until the user clears the user history or deletes specific entries. The information that will be saved to the database will include:

- Document ID
- User ID
- Text Input
- AI Classification
- AI Highlighted Words
- User Classification
- User Highlighted Words

Additionally, the user will have the ability to delete specific entries listed within the User History as well as have the ability to clear all previous entries saved to the database.

Figure 12: User History

Once the user has finished their session, they can select the "start a new session" button which will allow the user to upload another zip file and/or manually enter another set of input text to perform any additional classifications that they desire.

7 Developer's Manual

The developer's manual below has been taken from the readme.MD file of

https://github.com/subeom7/Text-Classification-and-Evaluation

7.1 Setting Up the Project

Create an empty project.
Move into that project directory and clone the project using this command in the terminal:

git clone https://github.com/subeom7/Text-Classification-and-Evaluation.git

Next, if you are not in the main "Text-Classification-and-Evaluation" directory, change directory into main project directory using the following command:

cd Text-Classification-and-Evaluation

To install all the dependencies, run the following command in the project (Text-Classification-and-Evaluation) directory:

npm run install-all

If the above command does not work, run npm install and pip install -r requirements.txt inside Text-Classification-and-Evaluation directory and cd client directory and run npm install again.

npm install
pip install -r requirements.txt
cd client
npm install

7.2 Running the Project

To "concurrently" run client & server, use the following command. Make sure to run this command inside of the project (Text-Classification-and-Evaluation) directory:

npm run dev

To run the server. Use the following command. Open http://localhost:5002 to view it in your browser:

python server.py

To Run the client app (front-end) in the development mode, run the following command. Open http://localhost:3000 to view it in your browser. The page will reload when you make changes.

npm start

To Launch the test runner in the interactive watch mode. Run the following command:

npm test

To Build the app for production to the build folder, run the following commands. It correctly bundles React in production mode and optimizes the build for the best performance:

npm run build

7.3 Important Files

7.3.1 Front-End Files

Filename	Path/Location	Description
App.js	client/src	Main file App file that uses the custom components.
GoogleSignIn.js	client/src/components	Component to authorize user using encoded JWT tokens.
InputForm.js	client/src/components	Component which provides the UI/UX for the text field, upload button and submit button
OutputDisplay.js	client/src/components	Component which provides the UI and displays the AI Classification and Important Words
UserHistory.js	client/src/components	Component which provides the sidebar that displays the

		archived outputs from previous classifications. Also provides functionality for deleting specific entries and clearing user history.
UserPrediction.js	client/src/components	Component which provides the UI for allowing the user to select the classification they believe the input belongs to.

Table 1: Important Front-End Files

7.3.2 Back-End Files

Filename	**Description**
classifier.py	Loads the saved model and the twenty_train object file. Also provides the functions for performing the classification provided a user input, fetching the important words used in the classification, and saving the output to the database.
database.py	MongoDB database for storing, retrieving, updating, and deleting user text classification history records using a user's ID and Document ID as references. Since the back-end is implemented using Flask (python), we use pyMongo library to connect the back-end script with the MongoDB client.
server.py	Server script that provides endpoints for classifying text, saving classification results and user highlights to a MongoDB database, retrieving a user's classification history, and updating or deleting classification records
train_and_save_model.py	Actual model of the classifier, where testing and training is done. After testing and training, the model is saved as a serialize Python objects using joblib library. We later load this file in classifier.py

Table 2: Important Back-End Files

7.4 Training and Saving AI model

7.5 Dataset

We first start by fetching the 20_newsgroup data which is a collection of approximately 18,000 newsgroup documents, partitioned across 20 different newsgroups. We load the dataset twice, for training and testing.

The **subset** parameter is used to specify whether to load the training data ('train') or testing data ('test').

The **categories** parameter is used to specify a list of newsgroup categories to include in the dataset. In our case we predeclared the categories array and set it to the categories parameter. If not provided, all categories will be used.

The **shuffle** parameter is set to True, indicating that the dataset will be shuffled before being returned. We shuffle the dataset to ensure a better distribution of classes, avoid overfitting, maintain consistency in cross-validation and overall improve model performance and accuracy.

Finally, The **random_state** parameter is set to 42 to ensure that the shuffling is deterministic and the results can be replicated in subsequent runs.

```
categories = ['soc.religion.christian','comp.graphics', 'sci.med',
'Sci.electronics','sci.space','sci.crypt','rec.sport.baseball','rec.sport.
hockey', 'rec.autos','talk.politics.guns']
```

```
twenty_train = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, random_state=42)
twenty_test = fetch_20newsgroups(subset='test', categories=categories, shuffle=True, random_state=42)
```

7.6 Creating a Pipeline

Next, we define a text classification pipeline using the **“Pipeline”** class from the sklearn.pipeline module. The pipeline consists of two steps. The first step is Text feature extraction with the TfidfVectorizer(). This step converts the input text documents into a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features. TF-IDF is a numerical statistic that reflects how important a word is to a document in a collection or corpus. The TfidfVectorizer processes the raw text data and transforms it into a format suitable for machine learning algorithms.

The second step is Classification with the “**MultinomialNB”** classifier. This step uses a Multinomial Naive Bayes classifier from the sklearn.naive_bayes module. Multinomial Naive Bayes is a simple and effective probabilistic classification algorithm that is particularly well-suited for text classification tasks. It works with discrete features like word counts or TF-IDF values.

The pipeline is stored in the variable text_clf. Once the pipeline is defined, it can be used to fit the model on the training data and make predictions on new, unseen data. The advantage of using a pipeline is that it simplifies the process of applying multiple transformation steps and an estimator in a sequential manner.

```
text_clf = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', MultinomialNB())
])
```

7.7 Training the Model

Next, we train the previously defined **text_clf** pipeline. The fit method applies the following steps in the pipeline. First, it transforms the input text data into a matrix of TF-IDF features using the TfidfVectorizer. Then, it trains the MultinomialNB classifier on the transformed TF-IDF feature matrix and the corresponding target variable. After the fit method is called, the text_clf pipeline will be trained on the given data and can be used to make predictions on new, unseen data using the predict method

```
text_clf.fit(twenty_train.data, twenty_train.target)
```

7.8 Prediction

The predict method is called on the trained text_clf pipeline object with the test dataset (twenty_test.data). The test dataset is a list of text documents, where each document is a string. The pipeline first transforms the input text data into a matrix of TF-IDF features using the TfidfVectorizer and then uses the trained MultinomialNB classifier to predict the class labels. The predicted class labels are stored in the variable predicted_test.

```
predicted_test = text_clf.predict(twenty_test.data)
```

7.9 Model Accuracy

The accuracy_score function from the sklearn.metrics module is used to compute the accuracy of the classifier on the test dataset. The function takes two parameters: the true class labels (twenty_test.target) and the predicted class labels (predicted_test). The test accuracy is calculated as the proportion of correctly classified documents in the test dataset and is stored in the variable test_accuracy

```
test_accuracy = accuracy_score(twenty_test.target, predicted_test)
```

```
print(f'Test accuracy: {test_accuracy:.2f}')
```

7.10 Joblib

Lastly, we save two objects to disk using the joblib.dump function from the joblib module. The joblib.dump function is used to serialize Python objects to disk, which can later be reloaded (deserialized) and used. When our back-end file is loaded, we load this saved model in our back-end script using joblib.load function.

```
# train_and_save_model.py
joblib.dump(text_clf, 'text_classifier.joblib')
joblib.dump(twenty_train, 'twenty_train.joblib')
```

```
# classifier.py
text_clf = joblib.load('text_classifier.joblib')
twenty_train = joblib.load('twenty_train.joblib')
```

8 Lessons Learned

8.1 Technical Lessons

We found lessons to be learned from both the technical and non-technical aspects of the project. From the technical aspect of the project, we found issues concerning the operating system when trying to run the project. We found that the best practice was to ensure that the operating system is up to date when trying to run the project. Additionally, we struggled to find a concise method to download all of the dependencies needed to run the project. We managed to figure out the simplest command to install all the dependencies at once. “npm run install –all" is a short and concise command to install the dependencies in one go. Lastly, the documentation in our code was injected in the final stages of implementation. This led to a very confusing code structure. As a result, we learned to document more during development to avoid a convoluted code base.

8.2 Machine Learning Model Implementation

The machine learning model is a huge part of the technical function of the project. While we are still in the process of fully implementing the machine learning model, we have encountered and solved a few issues during our time in development. The most notable problem we found was our implementation of the machine learning model. In our original implementation, a new model was created every time the classify button was pushed. This caused our application to be very slow. We remedied this issue by separating the classifier script into two scripts. One script handles the model training. The other script handles the prediction. This new implementation allows for a single instance of the model after a file is uploaded. The classify button no longer creates a new model and instead just runs the prediction script.

8.3 Non-Technical Lessons/Team Structure

From the non-technical aspect of the project, we had to learn a lot due to the learning curve of the project. Right out of the gate, we were faced with many unfamiliar tools and technology needed to develop this project. Some of these include machine learning, AI, React.js, Scikit-learn, MongoDB, and Flask. Additionally, we learned a lot about how our team should function. To overcome challenges, we found that the best way is to schedule an online meeting to work on deliverables rather than assigning individual work. The bulk of the work on deliverables is done in these meetings. We only resort to individual work when we run out of time for our meetings, and we all have a firm grasp of how we should proceed. Overall, the meetings promote healthy communication between members of the group more than back-to-back communication via messages. Furthermore, these meetings allowed us to bounce ideas and features off of each other. Eventually, these features are implemented in their own Git branch and merged at the end.

9 Future Work

9.1 User-Selected Model

There are several potential avenues for expanding the project in future semesters. Firstly, a significant enhancement would be to permit the user to provide their own data set for training the classifier. This would enable the user to determine the classification category in which the classifier will produce precise predictions.

9.2 Additional File Type Support

Furthermore, integrating support for a more extensive range of file types would be a valuable addition to the application. Currently, our app supports text and zip files. Additional file types like HTML will prove beneficial to the user by providing more options.

9.3 Technical Lessons

Another potential expansion could be enhanced interactivity features. Users can already interact with the application through highlighting and text input. These interactivity features could be enhanced in various ways that would be beneficial to the user. There may even be new interactivity features that we have not thought of so far.

9.4 Account Registration/Authorization (non-google)

Currently, our application supports account authorization. The only caveat is that users can only log in using a Google account. This feature has the potential for future modifications. Specifically, we want users to be able to register their own unique accounts within the application. Users should be able to log in with their newly created account without the need for a pre-registered Google account.

Acknowledgments

Dr. Mohamad Farag
Client • Professor at Virginia Tech
Email: mmagdy@vt.edu

Ansh Gwash
UTA at Virginia Tech
Email: anshgwash@vt.edu

References

[1] Text classification: What it is and why it matters. MonkeyLearn. (n.d.). Retrieved February 15, 2023, from https://monkeylearn.com/text-classification/#:~:text=Tutorial-,What%20is%20Text%20Classification%3F,and%20all%20over%20the%20web

[2] V. Rani, "NLP tutorial for text classification in Python," *Medium*, 26-Jan-2023. [Online]. Available: https://medium.com/analytics-vidhya/nlp-tutorial-for-text-classification-in-python-8f19cd17b49e. [Accessed: 15-Feb-2023].

[3] What is text classification? - hugging face. What is Text Classification? - Hugging Face. (2022, May 17). Retrieved February 15, 2023, from https://huggingface.co/tasks/text-classification

[4] "TF–IDF," *Wikipedia*, 06-Mar-2023. [Online]. Available: https://en.wikipedia.org/wiki/Tf%E2%80%93idf. [Accessed: 21-Mar-2023].

[5] "Naive Bayes classifier," *Wikipedia*, 03-Mar-2023. [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier. [Accessed: 21-Mar-2023].