Independent Study Report
A Natural Language Processing framework for Helios-Web

Name: Sagar Srinivas Prabhu (saprabh@iu.edu)
Course: CSCI - Y 790 (class number: 5914)
Spring 2024/Final Sem (MS in Computer Science)

Introduction

To get a handle on social media, we need to dive into what folks are posting and how they're linked up. By leveraging tools that seamlessly integrate these components, we can unlock invaluable insights into user behavior, sentiment analysis, narrative polarization, and topic summarization. To truly grasp what's happening on social media, we need to look at both what people are saying and how they're connected. By using tools that combine these aspects, we can better understand user behavior, like if people are feeling positive or negative about something, and sort through reviews to see if they're good, bad, or somewhere in between. The feelings expressed in Social media posts are really important. It shows why we need to look at both what's being said and how people are connected online.

In this report, I have goaled towards development of a Natural Language Processing platform, which is a Sentiment Analysis framework for the embedded data associated with the network nodes. The server would process texts in the form of JSON and perform polarization, Named Entity Recognition using Natural Language Processing techniques and packages like NLTK, VADER, CNN, BERT, LSTM through fine-tuned models. The server data would be communicated to the UI side and communicated to the users side through sentiment scores. Post this project, I would be integrating this framework with the Helios web application of OSoMe organization. Helios Web [10] is a web-based library that aims to visualize dynamic networks in real-time. In this report, I have detailed project methodology, technology stack, code with the results and the project's future work.

Throughout the development, I had faced some challenges where I had to incorporate larger chunks of data and calculate sentiment chunks seamlessly. Each of the sentiment technique packages had to be processed and optimized to take lesser execution time without back-end lags. The main goal of the project is to provide real-time response of sentiment scores to users for getting the emotions of the text sentences.

Methodology

My focus has been on advancing a Natural Language Processing platform, specifically a Sentiment Analysis framework tailored for text content associated with nodes in a network, e.g., posts in a social media network . The methodology is to process JSON-formatted texts utilizing various Natural Language Processing tools and techniques, integrated into fine-tuned sentiment models. The resulting sentiment scores are then relayed from the server to the user interface, facilitating seamless interaction. Following the completion of this endeavor, my plan involves integrating this framework into the Helios web application developed by the OSoMe organization.

1. Server Side:
    - Designed a Python-Flask server that processes text content using POST API method.
    - There are different sentiment analysis techniques and packages used: NLTK [1], VADER [2], Textblob [3], CNN [4], LSTM [5], BERT [6] , XLNET [7].
    - The Sentiment analysis API route takes input as JSON data in the form of key-value pairs. The keys are JSON ID's while values are pieces of texts or sentences.
    - The output is in the form of JSON with sentence ID and the aligned sentiment score.

2. Frontend Side:
    - The front end prototype in Javascript, HTML/CSS is developed for testing Sentiment Analysis API calls and checking whether correct output is derived.
    - The user has functionality to upload a csv file with sentences which can be targeted to extract the sentiment scores.
    - If the user wants to manually add the sentences, they can find the scores for the respective sentences too.
    - The user could be able to check the respective score coded in color format. If the score is in green, it's positive, red if negative and gray when neutral.

3. Frontend integration with Backend:
    - Integrated frontend development seamlessly with Backend server side with proper API end-to-end testing.

Technology Stack

Front-End (UI): HTML, CSS, Javascript, D3.js [8].

Back-end (Server): Python, Flask [9], Machine Learning Algorithms, NLP Packages - NLTK, VADER, TextBlob, TensorFlow, Transformers, LLM- BERT, XLNet, LSTM, CNN.

Project Timeline

In this section, I will be demonstrating the project work developed in parts around each week.

19th January, 2024:

- Prepared an open-source licensed Github repository for this project.
- Configured virtual python environment for the project files.
- I implemented the server side to calculate sentiment scores from JSON text inputs from a csv file in the given format:
  Input - {id : sentence};
  e.g - {0: 'The product is good', 1: 'The product design is bad'}

  Output - {id:{neg: score, pos: score}}
  e.g - {0: {neg:0, pos: 0.74}}

28th January, 2024:

- I developed the code to replace the csv file functionality and include API POST method with JSON input and calculate sentiment scores as output through NLTK.
- I have performed Natural Language Processing techniques like Named Entity Recognition (NER), which would help us to provide the more weighted scores as per entity of words and POS Tagging which involved tokenizing words based on parts of speech. For all these I utilized NLTK libraries.
- I also wrote independent methods to calculate sentiment scores using different NLP packages for me to find the most efficient one among all. I used VADER and Textblob sentiment packages.

2nd February, 2024:

- Implement BERT Transformer technique and utilize pre-trained models to write the code to utilize JSON inputs and output sentiments.
- Getting some package installation issues, checked the alternate package solutions and fixed it.
- Worked towards optimizing sentiment technique methods through batch processing and processing data into small chunks.

- Updated Github Readme with project description, technology stack and steps on running it.

9th February, 2024:

- Worked towards optimizing sentiment technique methods through batch processing and processing data into small chunks.
- Implemented LSTM package to output sentiment scores based on LSTM as another method.

16th February, 2024:

- Worked towards optimizing sentiment technique methods through batch processing and processing data into small chunks.
- Implemented CNN package to output sentiment scores based on CNN as another method.

23th February, 2024:

- Worked towards optimizing sentiment technique methods through batch processing and processing data into small chunks.
- Implemented XLNET package to output sentiment scores based on LSTM as another method.

8th March, 2024:

- From the Backend server, I changed the output structure and returned a maximum score which can be a positive, negative or neutral.
- Implemented the Front-end/ UI Functionality as per the UI prototype, I and Filipi had discussed mutually.
- In my UI screen, the user has the option to upload a CSV file to fetch all statements and show sentiment scores to the user. All the users can also individually post sentence by sentence as the input to find the calculated score.
- Worked on making the Backend code more modular.

15th March, 2024:

- Updated the UI functionality to include color changes for sentiment scores.
- Implemented exquisite file functionality- server and UI code.
- Worked towards Front end and Backend server integration.
- Worked on Modularizing the Server code.

27th March, 2024:

- Optimized the API code such that only 1 request is processed for the whole chunks of JSON sentence data rather than sharing each API request for a sentence.
- Worked towards Front end and Backend server integration.
- Started project documentation as part of the Independent study report.
- Implemented the entire code modularization for Backend API calls.

5th April, 2024:

- Optimized the UI and Backend server code integration and tested the entire working functionality.
- Working towards the project documentation as part of the Independent study report.

9th April, 2024:

- Pushed the Code modularization changes- Backend and Frontend to Github repository.
- Completed the Project- Report documentation.

Project Results/ Code

The Frontend of the project is a standard template of css-js-html. The server is modularized where each NLP model is organized into a single python file and could utilize reusable components from different files.

A)Modular Structure

```
--Frontend
    --css
        -- styles.css
    -- js
        -- fetch_server.js
    -- screens
        -- upload.html
    -- index.html

-- Server
    -- sentiment_analysis.py
    -- nltk_server.py
```

-- textblob_sentiment.py
-- vader_sentiment.py
-- bert_sentiment.py
-- lstm_sentiment.py
-- cnn_sentiment.py

B) Backend Server

1. NLTK

Packages used- SentimentInternsityAnalyzer, pos_tag, ne_chunk. [11]

```python
def nlp_nltk_sentiment(self, text):
    scores = self.sia.polarity_scores(text)
    # Exclude the compound score from the result
    return {key: value for key, value in scores.items() if key != 'compound'}

# Performed Named Entity Recognition (NER) using NLTK and inbuilt methods:
def nlp_ner(self, text):
    words = word_tokenize(text)
    pos_tags = pos_tag(words)
    named_entities = ne_chunk(pos_tags)
    #return named_entities
    # Perform sentiment analysis for each named entity
    entities_sentiment = {}
    for entity in named_entities:
        if isinstance(entity, tuple):
            # If the entity is a word tuple, perform sentiment analysis on the word
            word, pos = entity
            entity_text = f"{word}/{pos}"
            entity_sentiment = self.nlp_nltk_sentiment(word)
            entities_sentiment[entity_text] = entity_sentiment

    return entities_sentiment
```

Figure: NLTK and Named Entity Recognition Implementation

The **'nlp_nltk_sentiment()'** function, takes a piece of text as input, likely a sentence or a document, and calculates sentiment scores using the SentimentIntensityAnalyzer (SIA) from NLTK. The method then returns a dictionary containing the sentiment scores for individual aspects such as positivity, negativity, and neutrality, excluding the compound score.

The above NER code **‘nlp_ner()’ function** performs Named Entity Recognition using NLTK's built-in methods. It is tokenizing the input text, assigns part-of-speech tags to each token, and then identifies named entities using the ne_chunk function. After extracting named entities, it proceeds to perform sentiment analysis on each entity. If an entity is recognized as a tuple (a word and its part-of-speech tag), sentiment analysis is conducted on the word. Finally, it returns a dictionary where the keys are the named entities with their part-of-speech tags, and the values are sentiment scores for each entity.

Calculating Sentiment Scores using NLTK package

```
nlp_object = SentimentAnalyzer()

@app.route('/nlp_nltk_sentiment_bulk', methods=['POST'])
def analyze_sentiment_endpoint():
    try:
        if not request.is_json:
            return jsonify({"error": "No JSON data provided"}), 400

        json_data = request.get_json()

        results = {}
        for key, sentence in json_data.items():
            scores = nlp_object.nlp_nltk_sentiment(sentence)
            # Take the maximum score and determine its category
            max_score_key = max(scores, key=scores.get)
            if max_score_key == 'pos':
                max_score = scores['pos']
            elif max_score_key == 'neg':
                max_score = -scores['neg']
            else:
                max_score = 0  # Return 0 for 'neu'

            results[key] = max_score

        return jsonify(results)
```

Figure: NLTK- Sentiment Analysis API Endpoint

I have written this code that defines a Flask endpoint ’ /nlp_nltk_sentiment_bulk’ which accepts POST requests. It expects JSON data containing sentences to analyze for sentiment. The endpoint iterates through each sentence, calculates sentiment scores using an NLTK-based sentiment analysis function /nlp_nltk_sentiment_bulk, and returns the maximum absolute sentiment score for each sentence. The sentiment score is determined by selecting the maximum absolute score and then categorizing in the categories of 'pos'

(positive), 'neg' (negative), and 'neu' (neutral). The endpoint returns a JSON object containing the maximum sentiment score for each input sentence. If an error occurs during processing, it returns an error message with a 500 status code.

2. TextBlob and VADER

```python
def nlp_textblob_sentiment(self, text):
    blob = TextBlob(text)
    sentiment_scores = {
        'neg': blob.sentiment.polarity,
        'pos': blob.sentiment.polarity,
        'neu': 1.0 - abs(blob.sentiment.polarity)
    }
    return sentiment_scores

def nlp_vader_sentiment(self, text):
    vader_scores = self.vader_analyzer.polarity_scores(text)
    return {'neg': vader_scores['neg'], 'pos': vader_scores['pos'], 'neu': vader_scores['neu']}
```

Figure: Textblob and VADER Implementation

The first function, **'nlp_textblob_sentiment'**, utilizes TextBlob, a Python library for processing textual data. It calculates sentiment scores for the input text based on polarity. It assesses the negativity, positivity, and neutrality of the text by analyzing its polarity.

The second function, **'nlp_vader_sentiment'**, uses VADER (Valence Aware Dictionary and sEntiment Reasoner), a tool specifically designed for sentiment analysis of social media text. It provides sentiment scores for negativity, positivity, and neutrality based on the input text using pre-trained models.

```python
@app.route('/nlp_textblob', methods=['POST'])
def textblob_sentiments():
    try:
        if not request.is_json:
            return jsonify({"error": "No JSON data provided"}), 400

        json_data = request.get_json()

        results = {}
        for key, sentence in json_data.items():
            sentiment_score = nlp_object.nlp_textblob_sentiment(sentence)
            results[int(key)] = sentiment_score

        return jsonify(results)

    except Exception as e:
        return jsonify({"error": str(e)}), 500


@app.route('/nlp_vader', methods=['POST'])
def vader_sentiments():
    try:
        if not request.is_json:
            return jsonify({"error": "No JSON data provided"}), 400

        json_data = request.get_json()

        results = {}
        for key, sentence in json_data.items():
            sentiment_scores = nlp_object.nlp_vader_sentiment(sentence)
            results[int(key)] = sentiment_scores

        return jsonify(results)

    except Exception as e:
```

Figure: Textblob and VADER API Endpoints Implementation

The above flask route endpoints '/nlp_textblob' and '/nlp_vader' accept POST requests with JSON data containing sentences. It then iterates over each sentence and calculates sentiment scores.

3. BERT Transformer and LSTM

BERT and LSTM models are different techniques which are effectively used for sentiment analysis.

"BERT, which stands for Bidirectional Encoder Representations from Transformers, is based on transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection. Historically, language models could only read input text sequentially -- either left-to-right or right-to-left -- but couldn't do both at the same time. BERT is different because it's designed to read in both directions at once. The introduction of transformer models enabled this capability, which is known as bidirectionality. Long Short-Term Memory Networks is a deep learning, sequential neural network that allows information to persist. It is a special type of Recurrent Neural Network which is capable of handling the vanishing gradient problem faced by RNN. " [12] In my project case, BERT is bidirectional and is better than LSTM as it can take into account the complete context of the input, picking up on minute nuances in language.

Packages used- BertTokenizer, BertForSequenceClassification and

DistilBertForSequenceClassification.

```python
def bert_sentiment(self, text):
    inputs = self.tokenizer_bert(text, return_tensors='pt', padding=True, truncation=True)
    with torch.no_grad():
        outputs = self.model_bert(**inputs)
        logits = outputs.logits
        probabilities = torch.softmax(logits, dim=1).tolist()[0]
    return {'neg': probabilities[0], 'pos': probabilities[1]}
    # print("Probabilities:", probabilities)  # Add print statement for debugging
    # return probabilities

def lstm_sentiment(self, text):
    embeddings = self.model_lstm([text]).numpy()
    # Assuming a simple polarity classification based on embeddings
    polarity_score = embeddings[0][0]
    # Convert float32 to Python float for JSON serialization
    polarity_score = float(polarity_score)
    return {'neg': 1.0 - polarity_score, 'pos': polarity_score, 'neu': 0.0}  # Assuming a binary classification, 'neu' is set to 0
```

Figure: BERT and LSTM Implementation

In the above code, I have developed BERT and LSTM sentiment models.

In the BERT Model, the input text is tokenized using a BERT tokenizer, then passed through a pre-trained BERT model. The model outputs logits, which are converted into probabilities using softmax. The probabilities for negative and positive sentiments are returned as a dictionary.

In LSTM sentiment modeling, the input text is passed through a pre-trained LSTM (Long Short-Term Memory) model to generate embeddings. The polarity score is extracted from the embeddings and converted to a Python float. The polarity score is then used to determine the negative and positive sentiment probabilities, with neutral sentiment assumed to be 0.0.

C) Frontend (UI)

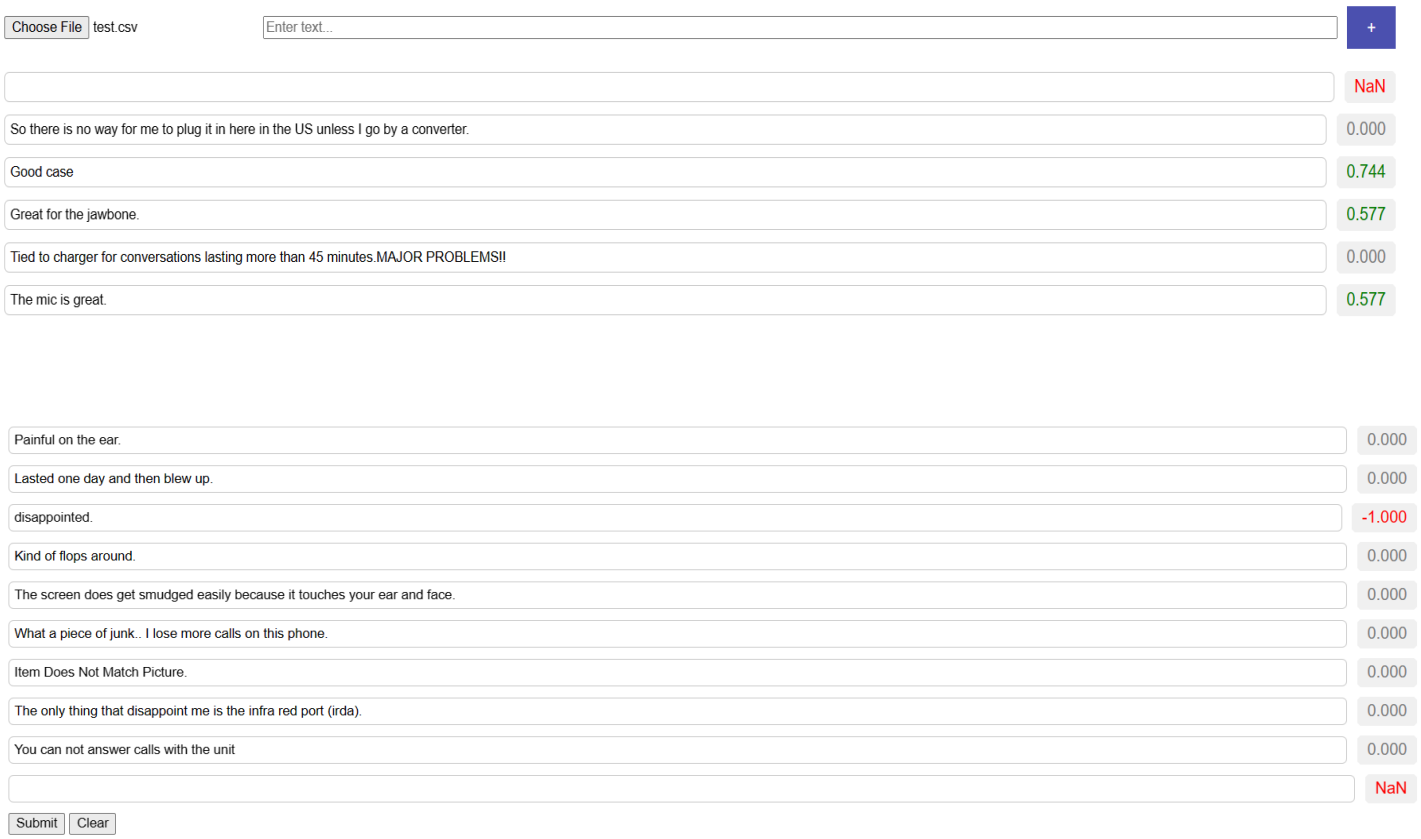

Figure: Frontend/ UI Screen Functionality

In the Frontend/ UI part, the user has functionality to upload a CSV file with various sentences for which they want to check the calculated sentiment scores. The user can also write individual sentences in the text input and check the respective score.

If the score is in green color and has no sign then it's positive sentiment while if the score has text in red color with a '-' sign then it's negative sentiment. If the score is 0.000 and in grey color, then it's neutral sentiment.

Future Work

I've engineered a novel Natural Language Processing platform tailored for sentiment analysis within network nodes' embedded data. Employing advanced NLP tools such as NLTK, VADER, CNN, BERT, and LSTM, the server swiftly processes JSON texts and distills sentiment analysis scores via meticulously tuned models. For the future work, I would be working more on enhancing server-side performance and adeptly managing voluminous data sets for sentiment score computations. I can potentially solve this issue using GPUs in the server-side, testing smaller BERT models or creating passthroughs to other online LLMs, such as ChatGPT, Gemini,etc. I would be then testing the full application functionality to possess low lags, working towards executing API endpoints in lower time and evaluating the models on datasets with data from social media. Finally, I would be integrating this NLP framework with the Helios web application of OSoMe organization, a tool aimed to visualize dynamic networks.

References

[1] https://realpython.com/python-nltk-sentiment-analysis/
https://www.guru99.com/pos-tagging-chunking-nltk.html

[2] https://www.analyticsvidhya.com/blog/2021/06/vader-for-sentiment-analysis/

[3] https://neptune.ai/blog/sentiment-analysis-python-textblob-vs-vader-vs-flair
https://www.analyticsvidhya.com/blog/2021/01/sentiment-analysis-vader-or-textblob/

[4]https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/

[5] https://www.analyticsvidhya.com/blog/2022/01/sentiment-analysis-with-lstm/
https://towardsdatascience.com/sentiment-analysis-using-lstm-step-by-step-50d074f09948

[6] https://www.tensorflow.org/text/tutorials/classify_text_with_bert
https://huggingface.co/blog/sentiment-analysis-python
https://wandb.ai/mukilan/BERT_Sentiment_Analysis/reports/An-Introduction-to-BERT-And-How-To-Use-It--VmlldzoyNTIyOTA1

[7] https://medium.com/swlh/using-xlnet-for-sentiment-classification-cfa948e65e85

[8] https://www.freecodecamp.org/news/d3js-tutorial-data-visualization-for-beginners/

[9] https://realpython.com/flask-javascript-frontend-for-rest-api/

[10] https://heliosweb.io

[11] https://www.geeksforgeeks.org/named-entity-recognition-in-nlp/

[12] https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model
https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

Project Source

NLP Framework Github:

https://github.com/sagarsp123/NLP_OpenSource_Framework_HeliosWeb