אוניברסיטת אריאל

2024

Deep Learning Course Final Project

Sentiment Analysis of Social Media's Messages

Authors:

Nikita Breslavsky 332363498

Barak Finkel 206491714

Gilad Fisher 318471109

Abstract:

This project [1] is about finding out if posts on social media (specifically Twitter / Reddit) are either positive, negative, or neutral. People use social media a lot to express their thoughts and feelings, and our goal is to make a tool that can understand these sentiments. Building upon our initial objective, we developed a model with a promising 93% accuracy in classifying sentiments of social media posts as positive, negative, or neutral. This achievement, facilitated by the inclusion of Google's embedding techniques, highlights a significant step towards our goal. Yet, real-world application tests reveal opportunities for further refinement to better capture more general nuances of human sentiment, setting a goal for future improvement.

Introduction:

Sentiment analysis has emerged as a key technique in understanding the vast amounts of data generated on social media platforms daily. By analyzing the sentiment of messages, businesses and individuals can gain insights into public opinion, customer satisfaction, filtering cyber bullying, etc. This project focuses on the sentiment analysis of social media messages, aiming to classify them as either positive, negative, and neutral categories using deep learning models.

Despite the advancements in natural language processing, accurately determining the sentiment of short, informal texts remains a challenge due to the use of slang, irony, and emojis that are irrelevant on many occasions. This project seeks to address these challenges by implementing and comparing different deep learning approaches to identify effective models for sentiment analysis over a set of approx. 200k posts in total.
Our goal is to research those approaches utilizing what has been learned in the lectures and research acclaimed approaches via Kaggle.com to present motivation to a practical usage of them via sentiment analysis. The subsequent sections will detail the background, methodology, results, and conclusions of this work.

Related Work and Required Background

Practical Foundation

Implementation of Neural Networks: A key resource that informed our approach to this project in addition to the theoretical knowledge gained throughout the course was the YouTube video course titled "Python TensorFlow for Machine Learning – Neural Network Text Classification Tutorial" by Kylie Ying [2]. This comprehensive course helped with implementing neural networks using Python and TensorFlow, with a focus on text classification.

Pre-Project Exercises [3]: Such as diabetes prediction and wine reviews classification, were particularly beneficial, as they provided a hands-on understanding of data preprocessing and model building, directly influencing the methodologies applied in our sentiment analysis project.

Insights from Kritanjali Jain's Twitter Sentiment Analysis LSTM [4]

Overview: The project by Kritanjali Jain, notable for its high engagement on Kaggle, provided valuable methodologies for data management and LSTM model utilization in sentiment analysis.

Data Preprocessing:

- Text Cleaning: Removed non-letter characters to simplify the dataset.
- Stop Words Removal: Eliminated common words that add noise to data analysis.
- Stemming: Applied to reduce words to their root form, decreasing the dataset's complexity.

Feature Engineering:

- CountVectorizer: Used to create a limited vocabulary, merging frequently occurring adjacent words to reduce tokenizer complexity.
- Data Tokenization: Employed the refined vocabulary for tokenizing the data, preparing it for model input.

Model Architecture:

Implemented a TensorFlow model incorporating a bidirectional LSTM layer with additional CNN filters for enhanced sentiment classification performance.

Impact: Jain's methodology informed our approach to data preprocessing and model design, demonstrating the effectiveness of combining LSTM with CNN layers for sentiment analysis.

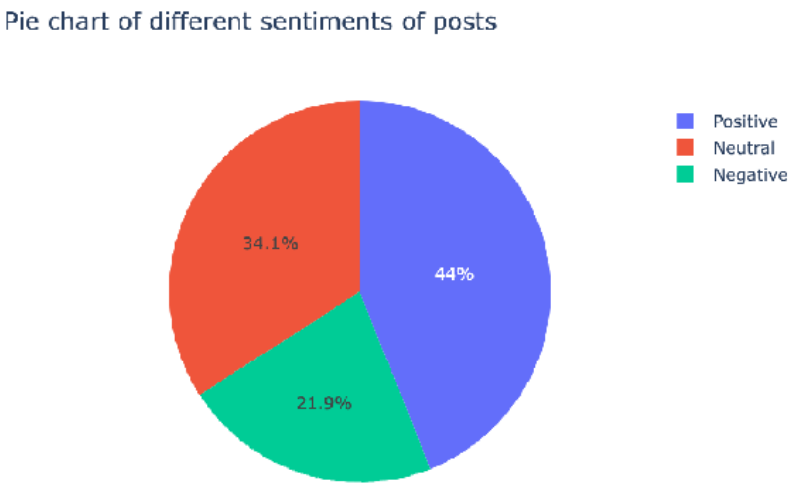
Project Description

In our project, the final model of our choice utilizes Google's embedding tool (which partakes in all the pre-processing of the data) to predict sentiments from our dataset. Even though this choice might be perceived as a shortcut, this strategy leveraged the available resources to output the best possible outcomes with our dataset based on the following factors:

- Performance Plateau: Having achieved similar results in most of the previously experimented models, we have sought to find an overall more efficient model.
- Simplicity: Usage of a third-party tool to reduce complexity of understanding the model.
- Speed: Utilizing an efficient pre-trained embedding model. This strategy significantly reduced computational resources and runtime complexity.
- Approx. Equal Power: Achieving the same results as the best of previous attempts.

Model Implementation:

1. **Data Collection and Preparation:**
   The foundation of our sentiment analysis model is constructed on a dataset amalgamated from Twitter and Reddit posts. This dataset, encompassing over 200,000 lines, is classified into negative, neutral, and positive sentiments. Initial examination revealed a measly 105 undefined data lines that were deleted accordingly. Despite a slight imbalance in sentiment distribution, a decision was made to retain the full



breadth of data, maximizing the model's exposure to diverse linguistic expressions, even if there is a slight lean towards positive sentiments.

2. **Preprocessing**

A pivotal step in refining our model's input data involved the utilization of the 'post_to_words' function. This function cleans the textual data by eradicating irrelevant symbols, converting text to lowercase, filtering out stop words, and applying stemming techniques. The cleaning method above sufficed due to the relatively clean state of our dataset, contributing to the model's accuracy and operational speed. Subsequently, the data was split, allocating 80% for training, with the remainder equally divided for validation and testing purposes, ensuring a comprehensive evaluation framework.

```
pattern  = re.compile(r"[^a-zA-Z0-9]")
def post_to_words(post):
    ''' Convert tweet text into a sequence of words '''
    # convert to lower case
    text = post.lower()
    # remove non letters
    text = re.sub(pattern, " ", text)
    # tokenize
    words = text.split()
    # remove stopwords
    words = [w for w in words if w not in stopwords.words("english")]
    # apply stemming
    words = [PorterStemmer().stem(w) for w in words]
    # return list
    return words
```

3. **Model**

Our model leverages Google's embedding model for word vectorization, harnessing an efficient method of representing textual data. This strategic choice did not only amplify the model's accuracy and speed but also simplified the understanding of the underlying code.

- A **pre-trained Google embedding layer** as the input layer, enhancing the model's comprehension of natural language.
- **Dense layers** with ReLU activation to learn complex patterns in the data.
- **Dropout layers** to prevent overfitting by randomly omitting neurons during training, thereby increasing the model's generalization ability.
- A **softmax layer** at the output, categorizing the sentiments into three distinct classes.

```
# Create the model
model = tf.keras.Sequential([
   hub_layer,
   tf.keras.layers.Dense(64, activation='relu'),
   tf.keras.layers.Dropout(0.4),
   tf.keras.layers.Dense(32, activation='relu'),
   tf.keras.layers.Dropout(0.2),
   tf.keras.layers.Dense(3, activation='softmax')
])


epochs=5
learning_rate = 0.01

# Compile the model
model.compile(
   optimizer = tf.keras.optimizers.legacy.Adam(learning_rate),
   loss =  tf.keras.losses.SparseCategoricalCrossentropy(), # loss function
   metrics = ['accuracy']
)
```

- **Optimizer and Loss Function**: The model was compiled with Adam as the optimizer and aimed to minimize the Sparse Categorical Cross-Entropy loss while tracking accuracy as the performance metric.
- **Epochs and Learning Rate**: The training was configured over 5 epochs with a standard learning rate of 0.01. This setup was chosen to test Adam's efficiency in rapidly approaching optimal model parameters.
- **Learning Rate Adjustment**: The learning rate was not explicitly set to decay over epochs in this configuration, relying instead on Adam's inherent mechanism to adaptively adjust the learning rates of individual parameters.

Simulation Results

Machine setup: MacBook Pro M1-chip (ARM architecture). 16gb ram, 10 cores CPU.

Model Performance

Utilizing the configuration specified above, the final model required approximately 2 minutes to fit the data. Evaluation on the test dataset was remarkably quick, taking less than half a second, and provided an accuracy of 93%. While these results are commendable, it's important to acknowledge that the model's familiarity with the dataset’s characteristic behavior - stemming from the data's uniformity in training and test sets - may have contributed to this high level of accuracy.

Real-world Application and Testing

To assess the model's real-world applicability, a custom function was created to test the model against various texts, comparing the predicted sentiments with expected outcomes. Out of 15 examples, only 7 predictions aligned with the expected results. This discrepancy highlights potential areas for improvement, particularly in the model's ability to navigate nuanced or ambiguous expressions. It may also suggest as mentioned before that the data set's characteristic behavior is not as broad as we have hoped.

```
def predict_sentiment(model, text, expected):
    text = text.lower()  # Convert the text to lowercase

    # Convert the text to a pandas DataFrame
    df = pd.DataFrame([text], columns=['Tweet'])

    # Preprocess the text
    df['Tweet'] = df['Tweet'].apply(post_to_words)
    df['Tweet'] = df['Tweet'].apply(lambda x: ' '.join(x))

    # Predict sentiment
    prediction = model.predict([df['Tweet'].iloc[0]])
    predicted_label = tf.argmax(prediction, axis=1).numpy()[0]

    # Map the numerical label back to the original label
    sentiment_map = {0: 'Negative', 1: 'Neutral', 2: 'Positive'}
    predicted_sentiment = sentiment_map[predicted_label]

    # Check if prediction succeeded
    success = "Success" if predicted_sentiment.lower() == expected.lower() else "Failed"

    # Print the results
    print(f"Original text: '{text}'")
    print(f"Processed text: '{df['Tweet'][0]}'")
    print(f"Expected sentiment: {expected}")
    print(f"Predicted sentiment: {predicted_sentiment} - {success}")
```

Insights and Reflections

The mixed success rate in real-world testing scenarios suggests a limitation in the model's current capacity to generalize beyond the dataset it was trained on. This observation is not uncommon in machine learning projects and points towards the need for further model refinement.

1. Diverse Data Collection: Expanding the dataset to include a wider array of sources and sentiments could enhance the model's understanding and adaptability to varied expressions.
2. Advanced Preprocessing: Experimenting with more sophisticated text preprocessing techniques might yield significant improvements. Adjusting the current

post_to_words function to better capture the essence of the text or exploring alternative methods could be beneficial.

3. Model Enhancements: While the current model benefits from the efficiency and simplicity provided by Google's embedding, exploring additional layers or alternative architectures could uncover new pathways to accuracy improvements.

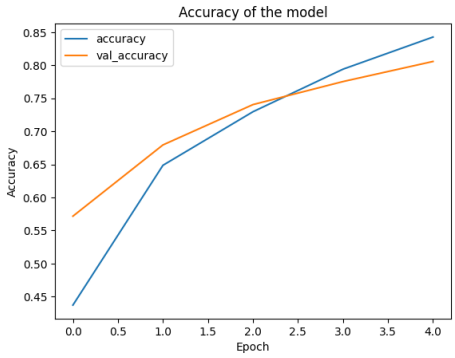
Previous Attempts

Initial Exploration and Model Building with NNLM

The project commenced with the objective of understanding the capabilities of TensorFlow in processing and analyzing sentiment from social media data. An initial dataset from Kaggle [5], specifically for Twitter entity sentiment analysis, served as the foundation for this exploration. This dataset was divided into training, validation, and testing (80%, 10%, 10%) segments to facilitate model evaluation.

To harness the power of pre-trained models and transfer learning, Google's embedding model - nnlm [6], was integrated into a straightforward TensorFlow model structure:

```
model = tf.keras.Sequential() # Create a sequential model
model.add(hub_layer) # Add the pre-trained layer
model.add(tf.keras.layers.Dense(16,
activation='relu',kernel_regularizer=tf.keras.regularizers.l2(0.01))) #
Add a hidden layer with 16 neurons
model.add(tf.keras.layers.Dense(3, activation='softmax'))
model.compile(
    optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=0.001), #
gradient descent algorithm
    loss =  tf.keras.losses.SparseCategoricalCrossentropy(), # loss
function
    metrics = ['accuracy']
)
```

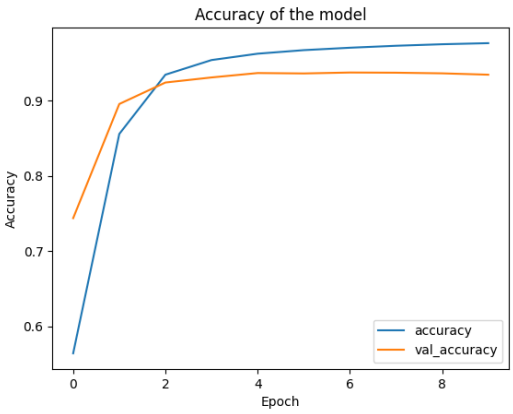


This approach, despite minimal preprocessing and model optimization (beyond removing null values), achieved an impressive 80% precision on previously unseen data, underscoring TensorFlow's potential in sentiment analysis tasks.

Refinements and Dataset Enhancement

Recognizing the importance of data quality and the need for further model refinement, several action points were identified:

- Enhancing data cleanliness by removing links, usernames, normalizing text to lower case, etc.
- Experimenting with different embedding models.
- Fitting model variables and layers
- Exploring advanced model architectures like RNN and CNN.



The initial dataset's limitations, particularly its noise and suboptimal classification, prompted a search for a more robust dataset. A subsequent discovery on Kaggle led to a well-classified and cleaner dataset of tweets and Reddit posts [7], which significantly improved model performance to 90% accuracy on unseen data. This transition underscored the critical role of high-quality, well-classified data in sentiment analysis.

Logistic Regression

We utilized the TfidfVectorizer from Scikit-learn to convert the text data into TF-IDF features, enabling the Logistic Regression model to process and learn from the text.

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
```

The model was trained on a 90/10 train-test split of the data, ensuring careful evaluation. Post-training, the model achieved 85.21% accuracy on the test set, highlighting its capability to effectively evaluate sentiment from textual data. Moreover, it took only 17sec to fit the train data and 35ms to predict unseen data.

The Logistic Regression section of our project highlights the model's robustness and speed, offering a compelling baseline for comparison with more complex deep learning models. Its

success on the dataset emphasizes the potential and validity of older machine learning techniques in sentiment analysis even today.

Experimenting with LSTM

Further exploration involved experimenting with an LSTM model on the new dataset, without extensive model fitting:

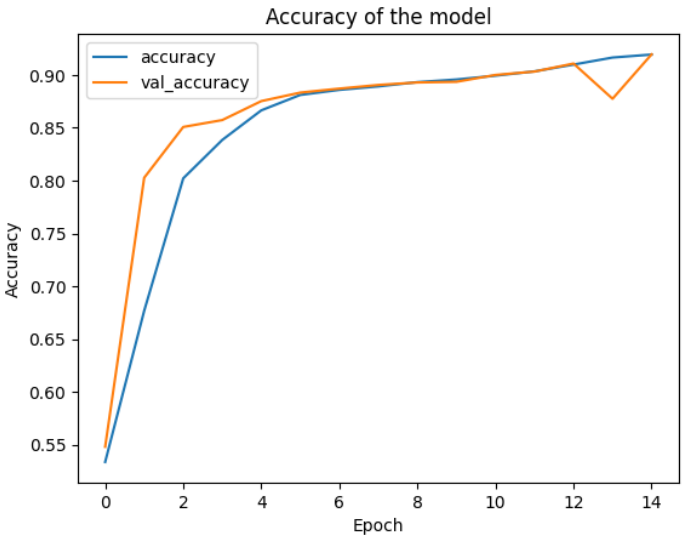
```
# LSTM MODEL
encoder = tf.keras.layers.TextVectorization(max_tokens=2000)
encoder.adapt(train_data.map(lambda text, label: text))
vocab = np.array(encoder.get_vocabulary())
vocab[:20]
model2 = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=32,
        # Use masking to handle the variable sequence lengths
        mask_zero=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(3, activation='softmax')
])
model2.compile(
    optimizer =
tf.keras.optimizers.legacy.Adam(learning_rate=0.001), # gradient
descent algorithm
    loss =  tf.keras.losses.SparseCategoricalCrossentropy(), # loss
function
    metrics = ['accuracy']
)
```

This model, even in its initial form, demonstrated a promising 89% success rate, indicating the effectiveness of LSTM architectures for sentiment analysis tasks. However, the computation was rather slow, as it took more than 20 minutes to fit the data (over 5 epochs) and 13 seconds to evaluate the test data.

Based on insights from Kritanjai Jain's Twitter Sentiment Analysis, we improved our model by enchasing data preprocessing, feature engineering and exploring model architecture.

```
vocab_size = 5000
embedding_size = 32
epochs=15
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.9
model2 = tf.keras.Sequential([
  tf.keras.layers.Embedding(input_dim=vocab_size,output_dim= embedding_size,
input_length=max_len, mask_zero=True),
  tf.keras.layers.Conv1D(filters=32,kernel_size= 3, activation='relu', padding='same'),
  tf.keras.layers.MaxPooling1D(2),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.GlobalAveragePooling1D(),
  tf.keras.layers.Dense(3, activation='softmax')
])

optimizer = tf.keras.optimizers.legacy.SGD(learning_rate=learning_rate, momentum=momentum,
decay=decay_rate, nesterov=False)
model2.compile(
  optimizer = optimizer, # gradient descent algorithm
  loss =  tf.keras.losses.SparseCategoricalCrossentropy(), # loss function
  metrics = ['accuracy']
)
```



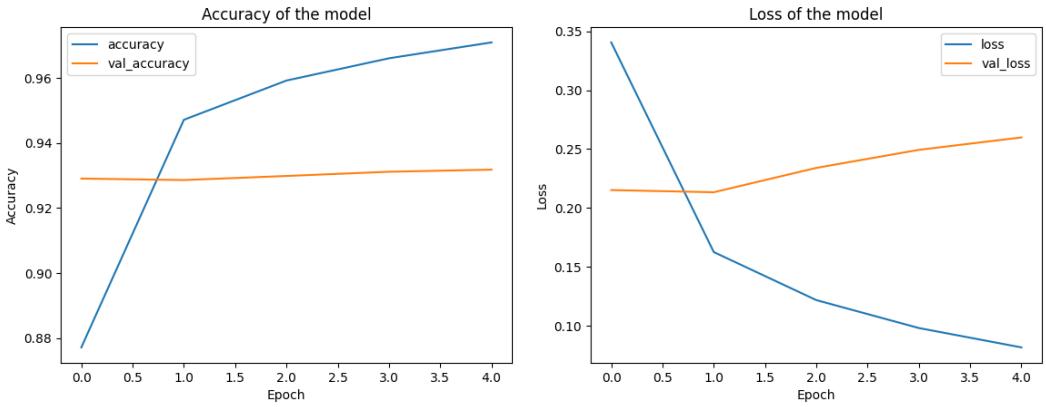
This version provided accelerated fitting speed, resulting in taking approx. 6min (over 15 epochs) and test evaluation of 3s. Notably, the preprocessing of the data partook in most of the impact, and the SGD algorithm as implemented above is simpler and faster than Adam used before. The model's accuracy increased to 92%.

Optimizers

While refining our sentiment analysis model, we evaluated two optimization algorithms: Adam and Stochastic Gradient Descent (SGD). Each optimizer brought its unique characteristics to the table, influencing the model's training dynamics and performance in distinct ways.

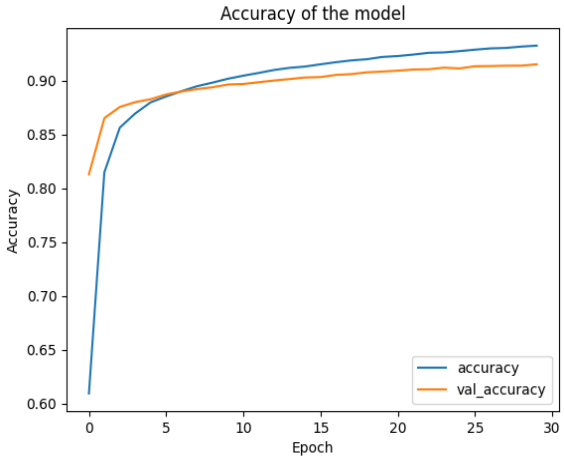
Adam Optimizer:

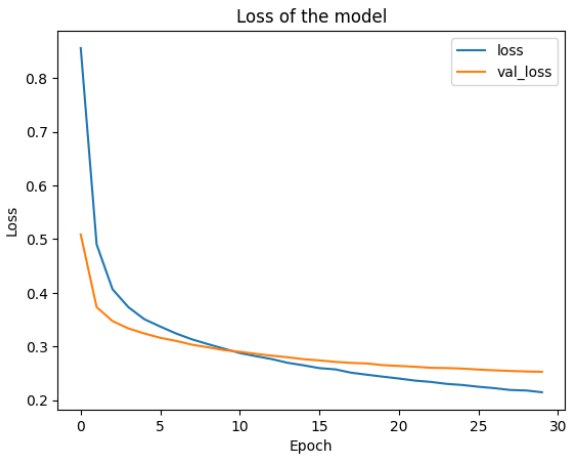
Adam, renowned for its adaptive learning rate capabilities, demonstrated a slower training speed in comparison to SGD. However, its ability to converge in fewer epochs highlighted its efficiency in improving results in less attempts. Despite this advantage, Adam's performance was less predictable. This was observed in the fluctuations in accuracy and loss metrics across training sessions, suggesting a sensitivity to initial conditions and hyperparameter settings. The adaptive nature of Adam, while beneficial in many scenarios, introduced a level of variance that necessitated careful tuning and evaluation.



SGD Optimizer:

Conversely, the SGD optimizer, with its straightforward and consistent update mechanism, showcased a faster training speed. However, it required more epochs to improve model fit. This characteristic of SGD, necessitating more iterations to converge, was counterbalanced by its predictability and consistency. The accuracy and error graphs for models trained with SGD exhibited smoother trends and less volatility, facilitating a more balanced performance across different training runs. This consistency is particularly valuable in scenarios where model stability and reliability are of high importance.





Conclusion

The exploration of various models for sentiment analysis highlighted key insights:

Logistic Regression (LogR) was identified as the fastest model, offering substantial accuracy. It stands out for its suitability on less powerful machines or under strict time constraints, despite not achieving the highest accuracy.

Long Short-Term Memory (LSTM) models were the most complex and flexible, providing extensive parameters for optimization. However, they required more computational resources, making them slower compared to other models. The LSTM's strength lies in its ability to avoid reliance on third-party models and its adaptability, making it preferable for more capable machines.

The final model utilizing Google's embedding positioned itself as a middle ground, balancing speed and readability. Even though it lacks the ultimate flexibility due to the fixed nature of the pre-trained embedding, it remains adaptable in other aspects.

The consistent accuracy plateau at 93% across different models suggests the maximization of potential insights from the dataset. This outcome highlights the importance of choosing the appropriate model based on the specific requirements of computational power, time constraints, and flexibility needs.

Future enhancements to our sentiment analysis approach can significantly benefit from selecting a broader dataset, in both sample amount and variation in structure. Expanding the data scope could address the observed limitations in generalizing beyond the specific contexts of the current dataset, increasing its accuracy in predicting sentiment in a more general and Real-World context.

Moreover, exploring the combination of different models presents a promising direction for leveraging the strengths of all models we experimented on. Careful integration between them might enhance predictive performance while mindful strategies will be required to prevent overfitting and maintain model simplicity.

References

[1] "Sentiment Analysis of Social Media's Messages," [Online]. Available: https://github.com/NickBres/SentimentAnalysis.

[2] K. Ying, "Python TensorFlow for Machine Learning – Neural Network Text Classification Tutorial," [Online]. Available: https://www.youtube.com/watch?v=VtRLrQ3Ev-U.

[3] N. Breslavsky, "Wine Review Classification and Diabetes Prediction," [Online]. Available: https://github.com/NickBres/TF_learning.

[4] K. Jain, "Twitter Sentiment Analysis LSTM," [Online]. Available: https://www.kaggle.com/code/kritanjalijain/twitter-sentiment-analysis-lstm.

[5] "Twitter Sentiment Analysis," [Online]. Available: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis?rvi=1.

[6] Google, "nnlm," [Online]. Available: https://www.kaggle.com/models/google/nnlm.

[7] C. KUMAR, "Twitter and Reddit Sentimental analysis Dataset," [Online]. Available: https://www.kaggle.com/datasets/cosmos98/twitter-and-reddit-sentimental-analysis-dataset.

[8] "SentimentAnalysis," [Online]. Available: https://github.com/NickBres/SentimentAnalysis.