**REPORT ON NEWS TWEET SANITIZED**

**Project Overview**

In this project, we will explore the basics of natural language processing (NLP), the project's goal is to create a system that cleans up news tweets. Social media platforms aim to screen out material that could be relevant or irrelevant tweets, reduce the spread of erroneous information, and encourage the distribution of factual news.

Throughout this project, the following abilities will be exercised in NLP.

* How to rapidly analyze and clean up big text datasets with pandas.
* How to preprocess text data using fundamental NLP techniques and produce features.
* How to train, assemble, fit, and assess deep learning models in Keras.
* Ways to classify unlabeled text data.

**STEP 1: PROBLEM IDENTIFICATION**

* 1. **Problem statement**

To show the tweeted text that could be relatable or irrelevant newsposted.

* 1. **Context**

Certain conditions and factors must be considered to filter and sanitize news tweets. This is referred to as the context for news tweet sanitization. In contrast to the filtered news offered by media outlets, Twitter is an outstanding repository of uncorrected discussions, views, and news events that are directly tweeted by the persons themselves.

* 1. **Success Indicators**

Identify the themes of the tweets with at least 75% accuracy.

* 1. **Solution Approach**

Using the appropriate keywords, gather information from Twitter.

The data will be cleaned and preprocessed using Natural Language Processing (NLP) methods.

Distributed computing will be utilized to do this, and logistic regression and naive Bayes LSTM models will be trained and tested.

* 1. **Constraints**

For now, we observed the following points are constraints

* A limited dataset of news tweets can be considered a constraint.
* When dealing with the casual language, sarcasm, acronyms, or accents frequently employed in tweets, NLP approaches may encounter difficulties.
* It might be difficult to ensure appropriate comprehension and interpretation of such subtleties.
  1. **Stakeholders**

Professor and my team members

* 1. **Deliverables**
* Final Report
* Final Presentation
* code in Jupyter Notebooks for every phase of the process
* Model Building

**STEP 2: DATA COLLECTION**



The dataset contains a mixture of tweets that are labeled as either "Relevant" or "Not Relevant." The dataset appears to be related to various events, accidents, disasters, and incidents, as indicated by the content of the tweets and the relevance labels. The relevant tweets likely contain information about accidents, natural disasters, fires, traffic incidents, and other noteworthy events, while the irrelevant tweets appear to cover a wide range of topics that are not related to these events.

**STEP 3: DATA WRANGLING:**

The dataset needed to be cleaned up and converted into an effectively visualizable format before the data could be displayed. Finding a productive method of reading and cleaning a dataset containing 10324 rows is necessary. To do that, data from a CSV file might be read into a pandas data frame. This is how the data was read and cleaned in our project.

**Reading Data and Dropping Duplicates:**

Reading data from an Excel file into a pandas DataFrame.

Duplicate tweets are removed from the DataFrame using the drop\_duplicates function.

**Text Cleaning:**

The 'Tweet' column is cleaned to remove unwanted characters, URLs, mentions, and hashtags using regular expressions (re.sub).

The cleaned text is converted to lowercase.

A screenshot of a computer

Description automatically generated

**Text Normalization and Preprocessing:**

The code defines a function normalizepreprocessdata to normalize and preprocess the text data.

Stop words (common words like 'and', 'the', 'is', etc.) are removed from the text.

Emojis are normalized using the demojize function from the emoji library.

Repeated characters in words are handled by reducing them to two consecutive occurrences.

The cleaned and normalized text is joined back into a single sentence.



**Applying Text Preprocessing:**

The normalizepreprocessdata function is applied to the 'Cleaned\_Tweet' column using the apply function.

**Replacing Categories with Integers:**

A dictionary is created to map the string values 'Relevant' and 'Not Relevant' to integer values 1 and 0, respectively.

The 'Category' column is updated using the map function with the created dictionary, converting string labels to integers.

**Dropping Rows with Missing Values:**

Rows containing missing (NaN) values are removed from the DataFrame using the dropna function.

**Checking Unique Categories:**

The unique values in the transformed 'Category' column are displayed using the unique function.

A computer code with colorful text

Description automatically generatedTop of Form

**STEP 4: EXPLORATORY DATA ANALYSIS**

For an understanding of the structure of the dataset, I investigated and visualized the following attributes of the data with a tableau in this part.

1.Number of relevant tweets

2. Number of non-relevant tweets

3. Number of tweets that are under can’t decide

4. Word cloud

**A screenshot of a computer

Description automatically generated**

**WORD CLOUD**

Creating Word Clouds:

The relevant and not relevant tweets are concatenated into two separate strings (relevant\_tweets and not\_relevant\_tweets).

The WordCloud class is used to generate word frequency representations for both relevant and not relevant tweets.

The word clouds are then plotted side by side using Matplotlib, displaying the most frequent words in the text data.

A screen shot of a computer code

Description automatically generated

A collage of words

Description automatically generated

**STEP 5: MODELLING**

For modeling news tweet sanitization, the following tasks should be done

1. The dataset should be tokenized, padded, and sequenced.

2. Separate the dataset into test and training sets.

3. prepared sequential model.

4. Evaluating models

**5.1 Tokenization and Padding:**

A Tokenizer object is initialized to tokenize the text data.

The tokenizer is fitted on the training data (X\_train), and then the text data in both X\_train and X\_test is converted to sequences of integers using the tokenizer.

Sequences are padded to a maximum sequence length of 100 using the pad\_sequences function. Padding ensures that all input sequences have the same length.

A screenshot of a computer code

Description automatically generated

**5.2 Model Architecture:**

A sequential model is created using the Sequential class from Keras.

An Embedding layer is added to convert integer-encoded words into dense vectors of fixed size (32) while preserving the order of words in sentences.

Two LSTM layers are added. The first LSTM layer returns sequences (return\_sequences=True), while the second LSTM layer doesn't return sequences, only the final state.

A final Dense layer with a sigmoid activation function is added for binary classification.

A screenshot of a computer program

Description automatically generated

**5.3 Model Compilation and Training:**

The model is compiled using binary cross-entropy as the loss function, 'adam' optimizer, and accuracy as the evaluation metric.

The model is trained using the training data (X\_train\_padded and y\_train) with validation data provided using the testing data (X\_test\_padded and y\_test).

The training is performed for 5 epochs with a batch size of 32.

A close-up of a computer code

Description automatically generated

**5.4 Model Evaluation:**

Predicted probabilities for the positive class are obtained using the trained model and testing data.

The ROC AUC score is calculated using the roc\_auc\_score function.

**Confusion Matrix and Misclassified Tweets:**

The confusion matrix is computed using the predicted binary labels and the true labels.

The confusion matrix is visualized using a heatmap.

The indices of misclassified tweets are identified based on the difference between true and predicted labels.

Misclassified tweets along with their actual and predicted labels are printed.

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

**ROC Curve:**

The ROC curve is computed using the roc\_curve function and plotted using Matplotlib.

The area under the ROC curve (AUC) is displayed on the plot.

A graph of a curve

Description automatically generated

A screenshot of a computer code

Description automatically generated

**Hyperparameter Tuning using Bayesian Optimization:**

The bayesian\_optimization method defines a hyperparameter search space using pbounds.

Inside the method, there's an evaluate\_model function that builds, trains, and evaluates a model with given hyperparameters.

The Bayesian Optimization library is used to search for optimal hyperparameters that maximize validation accuracy.

The best hyperparameters are printed, and the best model is trained and evaluated.

**OPTIMIZERS**

We used different optimization models, changing the weights of epochs to minimize the loss function. An optimizer is a procedure or method that modifies an artificial neural network's properties, such as its weights and learning rates. As a result, it contributes to cutting down on total loss and raising accuracy. Given that deep learning models typically contain millions of parameters, selecting the appropriate weights for the model is a difficult issue.

A screenshot of a computer code

Description automatically generated

**Optimizer Comparisons and Model Training:**

* The script executes different optimizer scenarios, where we provide an optimizer’s name.
* It creates an instance of TweetClassificationModel, preprocesses data, builds, trains, and evaluates a model using the specified optimizer.
* The ROC curve and learning curves are plotted.

**a) ADAM OPTIMIZER:**

An effective technique for increasing the accuracy and speed of deep learning models is the Adam optimizer. Adam may assist the neural network converge more quickly and precisely during training by examining the previous gradients and modifying the learning rate for each parameter in real time.

By using the Adam optimizer, we got an accuracy of 0.8387334484859172

**b)RMS prop OPTIMIZER:**

For minimizing the loss, RMSprop computes the gradient of the loss function with respect to the model's parameters and updates those parameters in a different direction from the gradient. To enhance the effectiveness of the method of optimization, RMSProp provides a few more strategies.

One distinguishing trait is the scaling of the learning rate for each parameter using a moving average of the squared gradients. As a result, the learning process is stabilized and oscillations in the optimization trajectory are avoided.

With this RMS prop optimizer, we can achieve the accuracy of AUC: 0.86357675

**c)SGD OPTIMIZER:**

With stochastic gradient descent (SGD), instead of using the entire dataset to

update the parameter values, just a small, randomly chosen part of the data is used.

The precision of our SGD optimizer allows us to attain an AUC: 0.5327358290184512

**d) AdaDELTA OPTIMIZER**

Like RMSProp, AdaDelta is an optimization method, but it lacks a call for a hyperparameter learning rate. As an alternative, it calculates the most recent scale using a continuously declining average of the gradients and their squares.

Because of the AdaDelta optimizer's accuracy, we were able to achieve an

AUC of 0.5327358290184512.

**e) ADMAX OPTIMIZER:**

First-order gradient-based optimization is what Adamax, a variation of Adam built on the infinite norm, does. It is appropriate to train time-dependent process.

due to their capacity to alter the learning rate based on data properties.

We were successful in reaching an AUC:0.8565869415366858 due to the correctness of the Admax optimizer.

By observing all these optimizers, we conclude that SGD and AdaDelta optimizers performed poorly with relatively low training and validation accuracy, and their AUC values were not significantly better than random guessing.

RMSprop and AdaMax optimizers seem to have provided better outputs for this specific model and dataset, as they achieved higher validation accuracy and AUC compared to the other optimizers.

**4.4 Model Evaluation:**

Finally, the AUC value for the ROC curve is approximately 0.86. This indicates that the LSTM model has a reasonably good ability to discriminate between relevant and not relevant tweets. An AUC value of 0.5 represents random guessing, and the model's AUC being significantly higher than 0.5 indicates that it performs better than random chance.

**Performance Variation:** The performance of different optimizers varies significantly. RMSprop and AdaMax optimizers tend to perform better compared to Adam optimizer. SGD and

AdaDelta optimizers performed poorly with low accuracy and AUC values.

**Optimal Hyperparameters**: The Bayesian Optimization technique identified optimal hyperparameters for the model, resulting in a validation accuracy of 76.59%, which is competitive with the better-performing individual optimizers.

**Importance of Hyperparameter Tuning:** Hyperparameter tuning through Bayesian Optimization was able to yield better results than some individual optimizers. This highlights the importance of finding suitable hyperparameters for the specific dataset and model architecture.

**Best Optimizers:** RMSprop and AdaMax optimizers seem to be the most effective for this.

text classification task, achieving higher validation accuracy and AUC values compared to other optimizers.

**Poor Optimizers:** SGD and AdaDelta optimizers had poor performance in this context, indicating that certain optimizers might not be well-suited for the chosen model architecture and dataset.

Bayesian Optimization was used to search for the best hyperparameters for an LSTM-based text classification model. Best hyperparameters that were found through the Bayesian Optimization process:

**Dropout Rate: 0.3832**

Learning Rate: 0.0002156

**LSTM Units: 125.1**

These hyperparameters were found to yield a validation accuracy of 76.59% for the LSTM model on your dataset.

Top of Form

A screenshot of a computer

Description automatically generated

RMSprop appears to be the best optimizer.

**RMSprop Optimizer:**

Validation Accuracy: 80.84%

AUC: 0.8608

**Web Application:**

Flask is used to build user interface (UI) for web application.

Flask is a lightweight web framework for building web applications in Python. It is designed to be simple and flexible, allowing developers to quickly create web applications without imposing too many constraints. Flask provides the tools and libraries necessary to handle routing, templating, and other common web development tasks.

Flask web server and defines a route that renders an HTML template named 'index.html'. When a user visits the root URL ("/"), the index () function is triggered, which returns the rendered HTML template.

**A screenshot of a computer program

Description automatically generated**

**Text Classification GUI:**

The GUI allows users to enter a text snippet, and the trained classification model predicts whether the text is relevant or not relevant.

The trained model (dnn\_model.h5) is loaded using load\_model.

The Tokenizer is fitted on the cleaned tweet data in the DataFrame (df['Cleaned\_Tweet']).

The classify\_text function is defined to process the user-entered text, tokenize, and pad it, make a prediction using the model, and display the result (Relevant or Not Relevant) in the GUI.

The GUI includes a Text widget for inputting text, a "Classify" button that triggers the classification process, and another Text widget to display the classification result.

A screenshot of a computer program

Description automatically generated

The HTML code represents a basic UI for a Flask web application. This UI includes an input text box, a "Submit" button, and a space to display the classified result.

The JavaScript code captures the input value, processes it, stores a classification result in the session storage, and updates the UI to reflect the classification result and UI is a simple text tweet classifier where users can enter a tweet text, click "Submit," and receive a classification result.

Rendering simplifies the deployment of your application by enabling us to effortlessly update our code within your source control. When we link our Render account with our GitHub account, Render will autonomously construct and launch our services following each code update. This streamlined process ensures that our services are consistently built and deployed without manual intervention.

We have used default configuration of Render.

**UI:**

[A screenshot of a computer

Description automatically generated](https://sampledeploymentpython.onrender.com/)

Link: <https://sampledeploymentpython.onrender.com/>