CRISPDM DOCUMENTATION

Authors:

- **Patrick Maina (Group Leader)**
- **Christine Ndungu**
- **George Nyandusi**
- **Teresia Njoki**

PROJECT TITLE: ***UNDERSTANDING PUBLIC SENTIMENT THROUGH TWITTER DATA***

- CRISP DM (Cross-Industry Standard Process for Data Mining) refers to a popular methodology used in Data Analytics and Data Science Projects. It provides a detailed framework and iterative workflow for tackling Data Science projects.

- Some of the fundamental stages involved in CRISP DM include:
    - Business Understanding
    - Data Understanding
    - Data Preparation
    - Modeling
    - Evaluation
    - Deployment

- We will use the CRISP DM framework to analyze the different types of sentiment (positive, negative and neutral) from user data, and provide actionable insights and recommendations on how to leverage this information in future productions.

Step 1: Business Understanding

Overview

Customer sentiment plays a significant role in shaping public perception, and guiding product development for tech giants like Apple and Google. By analyzing public opinions through tweets on Twitter, entities can identify trends, gauge satisfaction, and respond to customer needs more effectively. In light of this, this project aims to build a sentiment analysis model that classifies Tweets about Apple and Google products into sentiment categories (positive, negative and neutral sentiments)

Problem Statement

Apple and Google, two of the world's leading tech companies, heavily depend on public perception to maintain their market positions and customer trust. As consumers increasingly voice their opinions on platforms like Twitter, understanding these sentiments has become very paramount for brand management, product development, and customer engagement strategies.

Objectives

1. To apply text preprocessing and vectorization techniques in preparing the Twitter data for effective model training.

2. To develop a binary sentiment classifier to distinguish between positive and negative tweets, as a baseline

3. To extend the model to a multi-class classifier, that includes the neutral class.

4. To evaluate the classifier performances using appropriate metrics such as F1-score, precision and recall, particularly for imbalanced classes.

5. To provide actionable, data-driven insights and recommendations that will guide these tech companies in leveraging sentiment analysis for future product developments.

Challenges

Some of the challenges that were encountered in analyzing the Twitter sentiments include:

- **Imbalance in the dataset**: The dataset was heavily biased towards the neutral tweets, with the negative tweets being majorly underrepresented (about 6.27% of the total dataset)

- **Missing values in data:** The dataset had a huge number of missing entries in one of the columns.

- **Multi-class model bias:** The multi-class model was heavily biased towards the neutral class, and this reduced the instances in which positive and negative tweets were predicted accurately.

Proposed Solution

This project aims to analyze Twitter data, and build a robust predictive model that can classify sentiments based on tweet data. The solution entails a number of crucial steps—from data exploration, Exploratory Data Analysis (EDA), data cleaning, preprocessing and vectorization, model training and evaluation, model interpretability, and extracting data-driven insights and recommendations that will support robust decision-making.

Conclusion

These analysis and modeling techniques will provide these tech companies with strategic recommendations on how to analyze user sentiments, enhancing their approach towards product development and service provision, which will in turn enhance long-term profitability.

Step 2: Data Understanding

- The Brand and product emotions dataset was collected from data.world. It contains about 9093 entries, and 3 columns, which are all of object data type.

- It contains two features, and one target variable. They include:
    - **Tweet_text:** The actual content of the tweet. It is the main text input used for sentiment classification
    - **Emotion_in_tweet_is_directed_at:** This specifies the target of the emotion expressed in the tweet (e.g., Apple, Google)
    - **Is_there_an_emotion_directed_at_a_brand_or_product:** This indicates whether the tweet expresses an emotion directed at a brand or product. It serves as the label for classification

- Data checks:
    - Checked for missing values in the dataset
    - Checked for duplicate rows and inconsistency in the dataset
    - Checked the data type of each column in the dataset
    - Checked for uniformity of data in the dataset

Step 3: Data Preparation

- In the data preparation phase, we aimed to perform EDA to understand the relationship between the features and the target, and to clean, preprocess and vectorize the data for preparation for the modeling phase.

- Some of the key data preparation steps that were taken include:
    - Renaming the columns for readability and better understanding of the data. The columns were renamed to: **tweet, tweet_directed_at, and sentiment**
    - **Created a new sentiment class:** In order to have distinct sentiment classes, we combined the **'No emotion toward the**

product' and **'I can't tell'** classes to create a **'Neutral sentiment'** class.

- **Handling missing values**: In the **tweet_directed_at** column, which had the highest number of missing values, we filled the missing values with the string **'Not directed',** implying that the tweet was not intended for any of the entities.
- **Handling duplicate rows:** The dataset had about 22 duplicate rows, which we dropped entirely.
- **Plotting the distributions:** we performed EDA by visualizing the relationships between the features and the target variable. Some of the crucial visualizations we did include:
  - **Sentiment Distribution Analysis:** This plot illustrates the distribution of the sentiments across the dataset. From this plot, the neutral emotion class had the highest number of tweets, with about **5531** tweets, while the positive emotion class had about **2970** tweets, and the negative emotion class had about **569** tweets. This shows a huge imbalance in the classes.
  - **Tweet Destination Distribution:** This plot illustrates the distribution of the entities to which the tweets were aimed at. From the visualization, we can note that **5788** tweets were not directed towards a specific entity, while Apple and Google had about **659** and **428** tweets directed towards them.
  - **Sentiment by Tweet Destination:** This plot illustrates the distribution of the sentiments for the top 5 tweet destinations: **Apple, Google, 'Not Directed', iPad,** and **'iPad or iPhone'**. From the plot, we can note the following:
    - The tweets that were 'Not Directed' had the highest number of neutral tweets (about **5431** tweets)

        - The iPad had the highest number of Positive and negative emotion tweets (about **792** tweets for positive, and about **125** tweets for negative)
- **Data cleaning, preprocessing and vectorization:** We created a class called **TweetProcessor()** that contained all the functions for all these steps. Some of the key steps we took include:
    - Data cleaning:
        - Removal of URLs, user mentions, hashtags, special characters, and repeated characters
        - Expansion of common English contractions (e.g., “can’t” -> “cannot”)
        - Normalization of whitespace and punctuation
    - Text Preprocessing:
        - Conversion to lowercase for consistency
        - Tokenization of the texts into words
        - Removal of stop words and short words (words with a length less than 2)
        - Lemmatization to reduce each token to its base form
    - Feature Extraction and Vectorization:
        - Integrated both TF-IDF and Count Vectorization techniques for transforming the cleaned and preprocessed texts into numerical feature vectors.
        - Obtained the feature names of the transformed output using ‘**get_feature_names_out()**’
        - Obtained the vocabulary dictionary using ‘**get_vocabulary()**’
- We encoded the ‘sentiment’ column as follows:
    - Negative: 0
    - Positive: 1
    - Neutral: 2
- In addition, we defined the X and y variables as follows:

        - For binary classification, we filtered out the neutral class to only work with the positive and negative classes by creating a new dataframe **'binary_df'.**
        - For multi-class classification, we implemented all the classes, thereby implementing the original **'tweet_df'** dataframe.
    - The dataset was split into X_train, X_test, y_train, and y_test for each task using **train_test_split()**, with a test size of 20%, and a train size of 80%.

Step 4: Modeling

- In the modeling phase, we performed several key steps to ensure the modeling process was successful.

- These steps include:
    - Created pipelines for each classifier model. The classifiers model implemented include:
        - Binary: **Logistic Regression, Random Forest, XGBoost, Naive Bayes**
        - Multi-class: **Logistic Regression, Random Forest, Support Vector Machine, Naive Bayes**
    - In each pipeline, we defined the following:
        - The **TweetPreprocessor()** class, implementing the Count Vectorizer module
        - SMOTE (Synthesis Over-Sampling Technique), which was used to resample the data to handle class imbalance.
        - The classifier algorithm, with **'class_weight=balanced'** for some of them, to further handle the class imbalance.
    - Saved the models in one dictionary, and created a training loop for each of the model pipelines. The training loop included:
        - The fit method
        - Prediction on the test data

        - Computing evaluation metrics such as accuracy, F1-score, precision and recall
    - Created plots for the model performances to get further insight into the performance of each model against the outlined metrics, including a confusion matrix for the best performing model, to understand the rate of positives and negatives.

Step 5: Evaluation

- In the evaluation phase, we mainly used the **F1-score** to measure the performance of the four models in each classification task.

- This was done by computing the F1 score for all the four models across each classification task using **f1_score(),** and comparing the scores of each model.

- From the evaluation, the top 3 models based on the F1 score in the binary classification task were:
    - Random Forest - 86.87%
    - XGBoost - 84.76%
    - Logistic Regression (base model) - 84.67%

- The top 3 models based on the F1 score in the multi-class classification task were:
    - Support Vector Machine - 65.05%
    - Logistic Regression (base model) - 64.63%
    - Naive Bayes - 61.51%

- In addition, the following steps were taken to test the best performing models against test data:
    - Saved and loaded the models using **joblib**

    - Used the best models to generate predictions from samples in the test set, highlighting the tweet, predicted sentiment, confidence score, and whether the prediction was correct
    - Identified and displayed examples of misclassified tweets to help understand the model's weaknesses
    - Created a function that obtained the last five tweets from the test set, and used the loaded models to make predictions.

- From the evaluation in the binary classification task, the Random Forest model misclassified a total of 82 out of 708 tweets in the test data, which is about 11.6% of the total test dataset.

- In the evaluation of the multi-class classification task, the Support Vector Machine model misclassified a total of 623 out of 1814 tweets in the test dataset, which is about 34.3% of the total test dataset.

- From this evaluation, we can safely conclude that the binary classification task is suitable in this use case. However, with the implementation of other techniques such as deep learning models, transfer learning (distilBERT or roBERTa), word2vec, etc., multi-class classification can still be implemented in this sentiment analysis, and would generate superb results.

Step 6: Deployment

- The analysis and modeling steps were carried out in a Jupyter Notebook. The model was then saved as a pkl file, which is ideally ready for production.

- The notebook and pkl files were then saved and uploaded to the GitHub cloud repository.

- Next steps in deployment would be:

- To wrap the model in a lightweight web framework such as FastAPI
- Create a Docker container that contains all the python and ML dependencies, and use tools such as **Docker Compose** or **Kubernetes** to define the operation of the container
- Implement model deployment hubs such as **Streamlit, Gradio** and **Hugging Face** for deploying our prediction model
- Integrate a **Continuous Integration/Continuous Deployment(CI/CD)** pipeline