Sentiment Analysis to Identify Offensive Tweets

Divya Sharma (ds655), Revanth Chowdary Ganga (rg361), Udyan Sachdev (us26)

Introduction

Online conversations on social media platforms like Twitter (Now X) allow people to freely express ideas. However, some comments involve offensive, toxic, or abusive language which negatively impacts users. Detecting and filtering out such content can enable healthier discussions. But manual moderation does not scale while automated solutions struggle with limited data.
This project develops generative and discriminative machine learning models for enhanced offensive tweet classification performance. A Naive Bayes model will generate synthetic offensive tweet data to augment limited real examples. Meanwhile, Long Short-Term Memory (LSTM) neural networks will provide discriminative power. Comparative assessment on real and synthetic test data will evaluate these approaches and advance automated moderation capabilities through robust models and data generation.

Data

Source

The data was obtained from the following Github Repository, and contains 24,783 tweets which were classified into the following 3 categories (0- Hate Speech, 1- Offensive Language and 2- Neither). These classifications were performed by aggregating the category into which users classified the tweets into and taking the category with the maximum number of votes. The dataset also contains other information such as the number of users who voted on a particular tweet and count of votes under the individual categories.

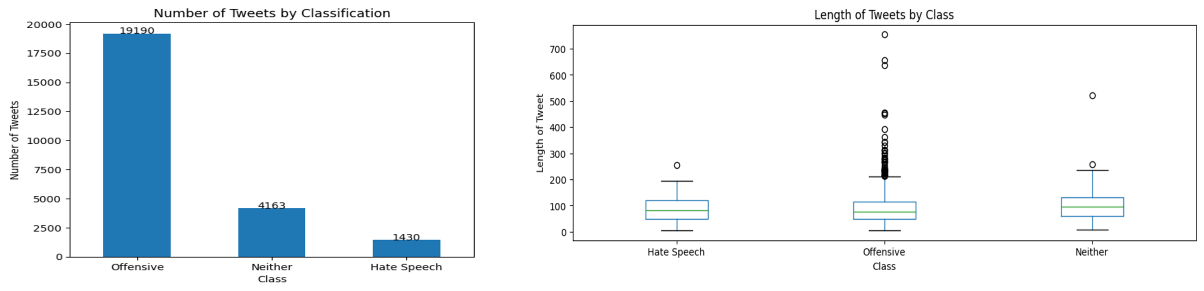
Exploratory Data Analysis

For the purpose of this analysis and project, only two fields from the original dataset were considered, the raw tweets and the final classification.

One viewing the distribution of the number of tweets based on the classification, it was discovered that the Dataset was Imbalanced and there were very few observations for "Hate Speech" category, hence this category was dropped from our analysis and we limited our scope to the remaining two categories.

The overall length of the tweets was similar across different categories with a few outliers in the offensive class, which maybe a result of the large number of observations for this class

Fig 1. Distribution of Number and Length of Tweets by Classification



Due to the highly unstructured nature of the dataset, further analysis was done post cleaning and processing of the data as explained in the next section.

Cleaning and Preprocessing:

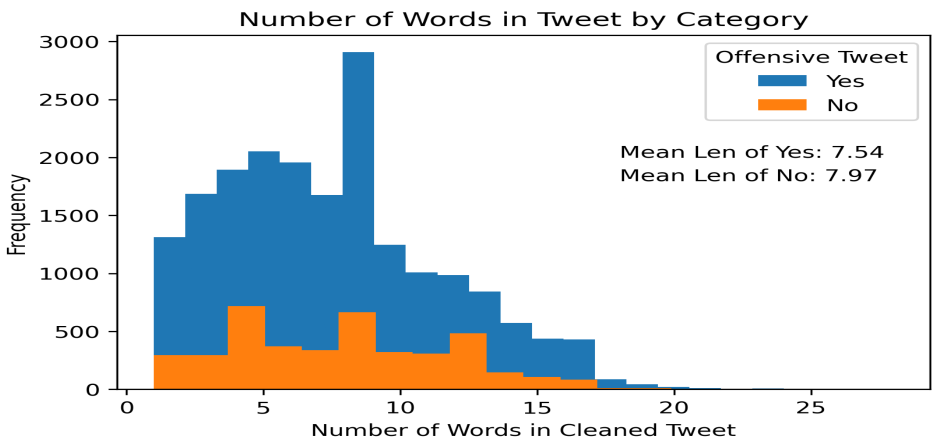
Since tweets are generally unstructured and contain information which is not useful for the modeling purpose, the following data cleaning and preprocessing steps were performed so as to improve the model performance.

1. Tweets were converted to lowercase
2. All the URLs were removed from the tweets
3. All mentions(“@..”) of twitter handles were removed
4. All hashtags (“#..”) were removed
5. Special Characters and numbers were removed
6. Stop words were removed using NLTK
7. Character “RT” which signifies that that the tweet is a retweet was removed
8. Tokenization was performed (normal string form was also retained)
9. Stemming was performed using NLTK
10. Lemmatization was performed using NLTK

Post these cleaning steps the data was analyzed again to study the new distributions and to ensure that the data is balanced.

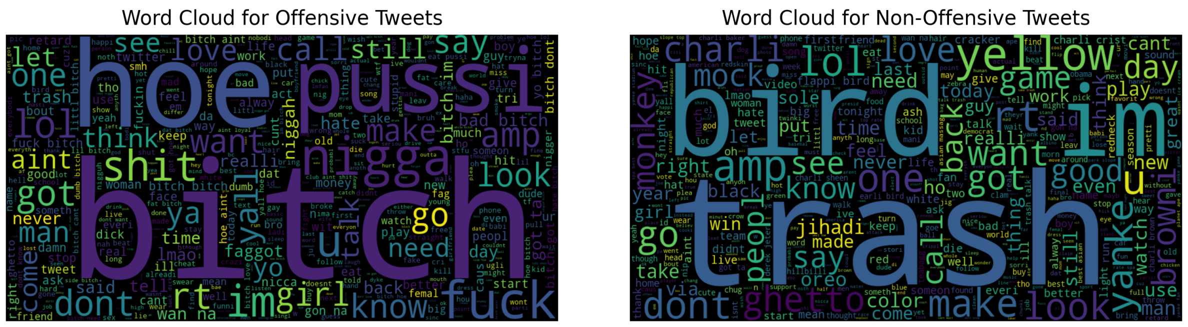
Instead of considering the raw length of tweets (number of characters) we analyzed the number of words in the tweet and found that both classes had a similar distribution and mean

Fig 2: Number of Words in Tweet by Category



A world cloud was generated to see if there were any common words which appeared with high frequency in the two classes

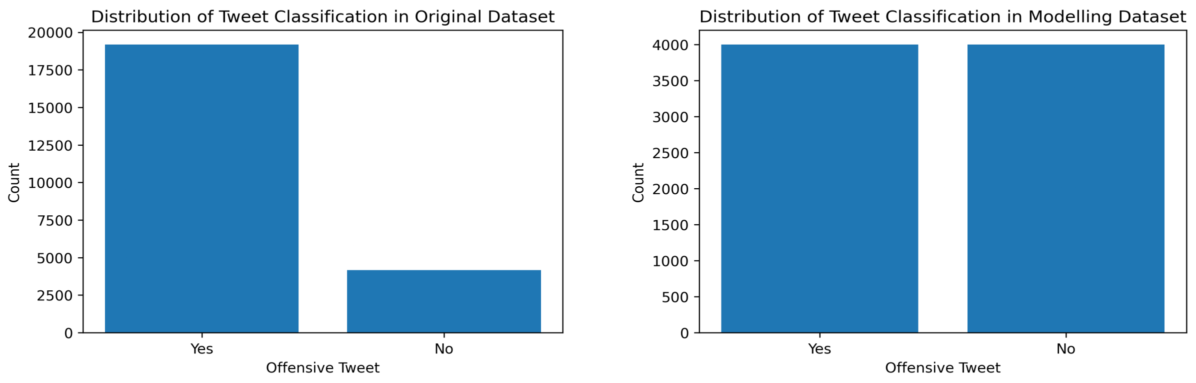
Fig 3: Word Cloud of Tweets based on Classification



Data Selection and Splitting

Since the Data was imbalanced, we first balanced it by taking equal number (4000) of samples from each category before splitting it into Test and Train Dataset

Fig 4: Distribution of Tweet Classification Counts in Original and Model Dataset



Once the balancing was done, the data was split into Train and Test dataset at a 80:20 ratio (with stratification) using the train_test_split function from scikit-learn.

Solutions:

Generative Solution

Naive Bayes stands out as the go-to generative model for expanding offensive language datasets due to its straightforwardness and scalability in mimicking language patterns. Unlike intricate discriminative models, Naive Bayes relies on a simplifying assumption: words are independent given a class, making it efficient for estimating probabilities. This computational efficiency suits generative tasks like creating additional offensive tweets, crucial for supplementing limited real-world examples.

Naive Bayes excels by estimating individual term probabilities instead of entire sequence likelihoods, enabling rapid synthetic data generation. Its processing scales linearly with features, making it suitable for the extensive vocabulary found on platforms like Twitter. While simplistic, Naive Bayes generates believable synthetic tweets containing common offensive terms in grammatically sensible structures, eliminating the need for manual crafting. Although not perfect, these synthetic tweets offer valuable augmentation for training more complex models like LSTMs.

In essence, Naive Bayes presents an efficient and scalable method for producing extra offensive training data by modeling independent features, distinct from the intricate sequence discrimination of complex models. Its computational efficiency provides vital generative capabilities for data augmentation, complementing discriminative approaches in classifying offensive tweets, especially when actual labeled data is scarce.

The synthetic data generated from a pre-trained Naive Bayes model have yielded lower accuracy due to assumptions that don't fully align with real-world complexities. Naive Bayes relies on independence assumptions between features, which might not accurately represent intricate relationships in actual data. Additionally, lack of diversity, potential overfitting of the model, and the impact of threshold selection could collectively contribute to the synthetic data's lower accuracy. To enhance synthetic data quality, validation against the original dataset, refining probability estimations, and exploring alternative generative approaches might be beneficial.

Discriminative Solution

Unlike generative models that assume independence between words, LSTMs can model interdependent sequences via gated memory cells that remember long-range dependencies. This avoids oversimplifying assumptions and enables robust context-based discrimination of offensive versus non-offensive language in tweets. Specifically, LSTMs can encode offensive terms differently based on the surrounding tweet text, overcoming challenges like implied offenses without explicit terminology. Their sequence modeling of entire tweets provides appropriate contextual understanding.
Additionally, LSTMs avoid restrictions on input sizes, accommodating Twitter’s variable content. Their memory prevents degradation over long inputs. Pre-trained LSTMs reduce data dependence, mitigating limited labeled examples. In summary, LSTMs offer optimal discrimination capabilities to address the context-dependent and length-variable challenges of identifying offensive content within noisy tweets more appropriately than other discriminative approaches. Their sequential memory suits this complex linguistic classification task.

Implementation:

Generative

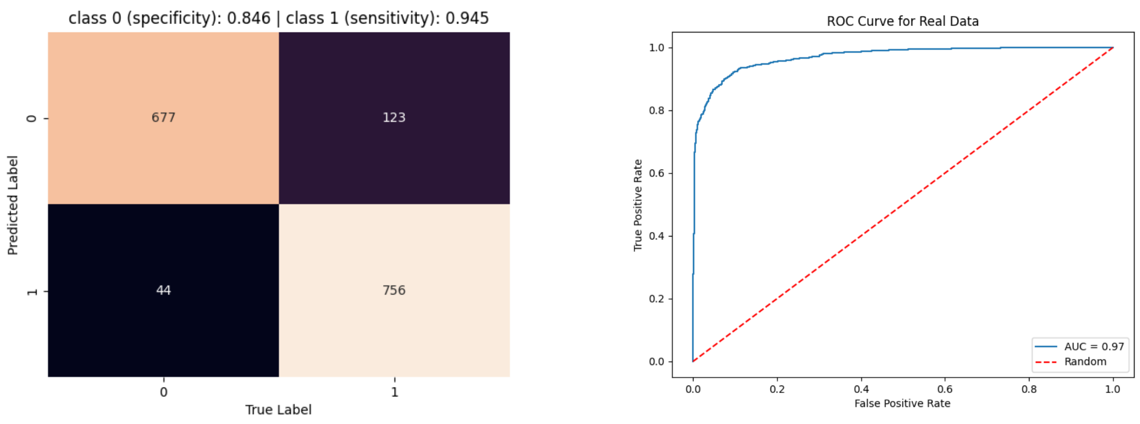
Naive Bayes is a probabilistic model that assumes independence between features. For sentence classification, it calculates the likelihood of a sentence belonging to a particular class based on the occurrence of words in the sentence. It operates by multiplying individual word probabilities to determine the overall probability of a sentence being in a given class. Finally, it assigns the sentence to the class with the highest calculated probability.

Workflow Description for Real World Data:

- **Data Splitting:** Split data into train/test.
- **TF-IDF Vectorization:** Initialize and fit a TF-IDF vectorizer for text-to-numeric conversion.
- **Feature Selection:** Collect, count, and choose top frequent words.
- **Naive Bayes Training:** Initialize and train a Multinomial Naive Bayes classifier.
- **Model Evaluation:** Calculate and print the accuracy of the Naive Bayes model.

Output plots for Real world data :

Fig 5: Confusion Matrix and ROC curve for Naive Bayes Model with Real Data

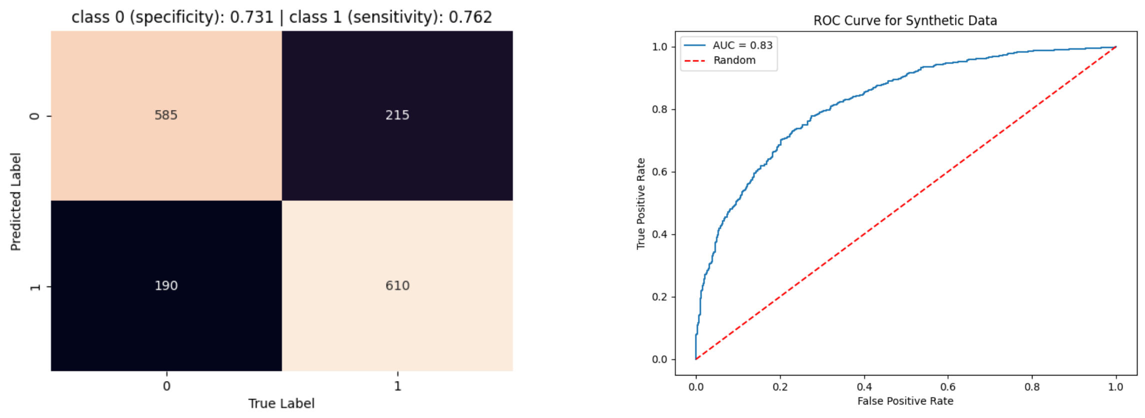


Workflow Description for Synthetic Data:

- **Probabilistic Vocabulary Generation:** Naive Bayes model used for word probabilities categorization.
- **Storage in Dictionaries:** Probabilities stored in 'offensive' and 'non-offensive' vocabularies.
- **Synthetic Data Generation:** Creates synthetic reviews guided by word probabilities.
- **DataFrame Construction:** Synthetic reviews and class labels organized into a simulated dataset.
- **Synthetic Data Processing:** Splits synthetic data and applies TF-IDF Vectorization.
- **Naive Bayes Training and Evaluation:** Trains and evaluates Naive Bayes on synthetic data.
- **Confusion Matrix & Metrics:** Generates confusion matrix, calculates Specificity and Sensitivity.

Output plots for Synthetic data :

Fig 6: Confusion Matrix and ROC curve for Naive Bayes Model with Synthetic Data



Discriminative

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) that is capable of learning long-term dependencies, which makes it highly effective for many sequence prediction problems. It is widely used in tasks such as text generation, sentiment analysis, and sequence prediction.

Key Features of LSTM:
- **Handling of Long-Term Dependencies**: LSTMs are designed to avoid the long-term dependency problem in traditional RNNs, making them effective for tasks that require learning from long sequences of data.
- **Memory Cells**: LSTMs have a 'memory cell' that can maintain information in memory for long periods of time, which is crucial for learning from sequences of data.
- **Gates**: LSTMs use gates to control the flow of information into and out of the memory cell.

Workflow Description for Real World Data:

- **Data Preparation**: The real-world tweet data is first preprocessed. The text data is tokenized, which involves converting the text into numerical representations that can be understood by the model. A vocabulary [size 5000] of the most frequent words is created, and each word in the text is replaced with its corresponding index in the vocabulary. The sentences are then padded with zeros to ensure they all have the same length. This is necessary because LSTM models require input data to have a consistent shape.

- **Conversion to Tensor**: After the data has been preprocessed, the features and labels are converted into Tensor datasets for compatibility with PyTorch. This is done using `torch.from_numpy()`.

- **Batching and Shuffling**: The data is loaded into a DataLoader, which allows for easy iteration over the dataset in batches. The data is shuffled at each epoch to ensure the model doesn't learn from the order of the samples.

- **Model Training**: An LSTM model is trained using the training data. The model's parameters are updated to minimize the loss function.The following parameters are used -

1. Number of Layers (n_layers = 2): The output of the first LSTM Layer is used as the input for the second LSTM Layer
2. Vocabulary Size (vocab_size = 5001): The total number of unique words in the text data plus one for padding
3. Embedding Dimension (embedding_dim = 30): Each word in the vocabulary is represented as a 30-dimensional vector in the embedding space
4. Output Dimension (output_dim = 1): This is because the LSTM is being used for binary classification
5. Hidden Dimension (hidden_dim = 256): This is the size of the hidden LSTM layers. This will help the model learn more complex patterns, at computational cost
6. Gradient Chopping (clip = 5): The gradients are clipped to prevent numerical instability and poor performance. The value of a gradient will range from -5 to 5
7. Number of Epochs (epochs = 5): The model is trained for 5 epochs, so 5 passes through the entire training dataset
8. Best Loss Logic: During training, the model's performance is evaluated on the validation set at the end of each epoch. This helps identify the best model across the different epochs

- **Model Evaluation**: The trained LSTM model's performance is evaluated using the validation data. The model's predictions are compared with the actual labels to calculate metrics such as accuracy or loss.

Fig 7: Accuracy and Loss Plots for LSTM over 5 epochs for Real Data

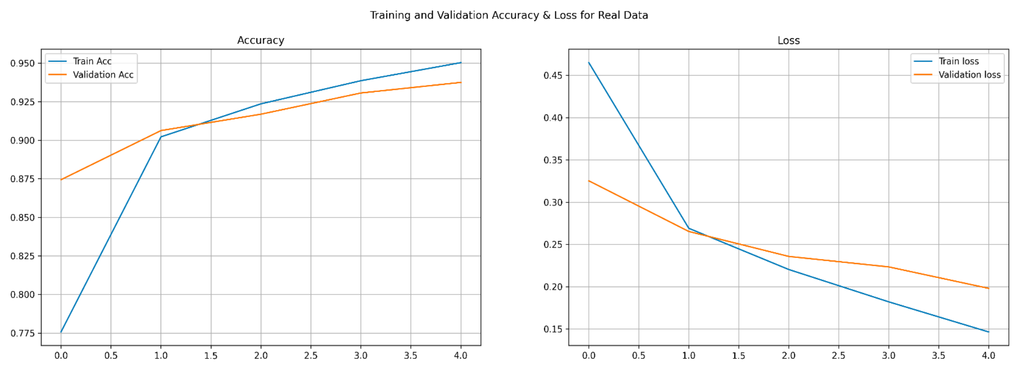
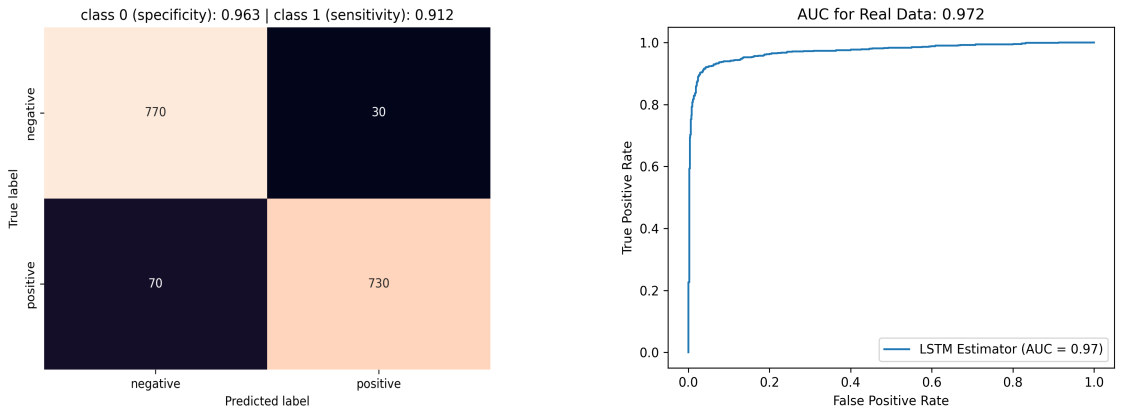


Fig 8: Confusion Matrix and ROC curve for LSTM Model with Real Data



Workflow Description for Synthetic Data:

The synthetic data used to train and validate the LSTM Model is the same data that was generated in the Naive Bayes model above. The process for training and evaluating the LSTM model on the synthetic data is identical to that used for real world data, with one key difference - the vocabulary size for the synthetic data is 3000, compared to 5000 in the real-world data. This difference is due to the unique characteristics of the generated synthetic data.

Fig 9: Accuracy and Loss Plots for LSTM over 5 epochs for Synthetic Data

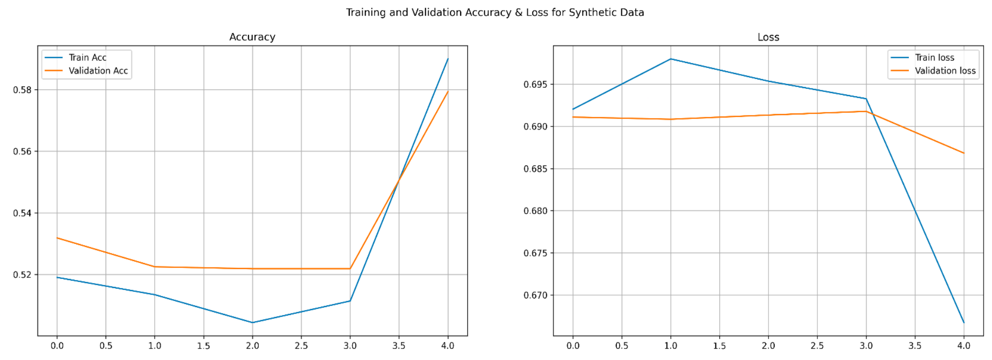
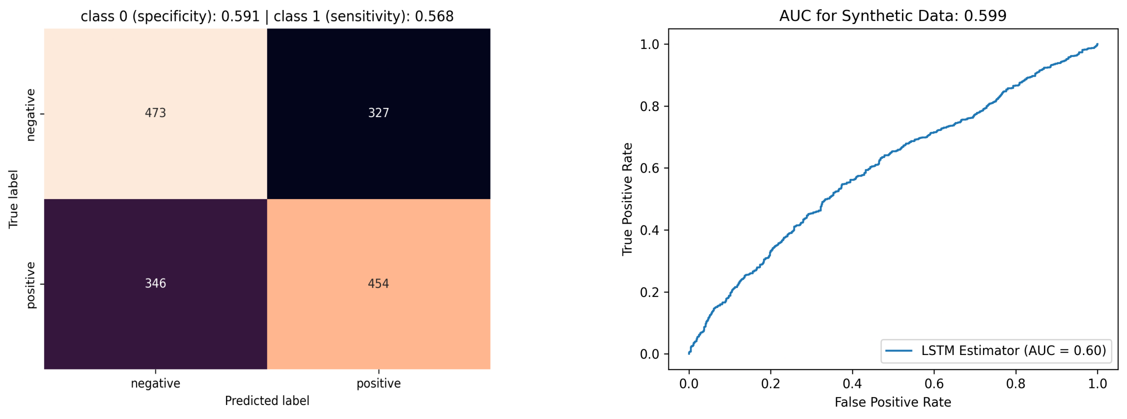


Fig 10: Confusion Matrix and ROC curve for LSTM Model with Synthetic Data



Comparison & Conclusion:

Comparison

Real World Data

Metric	Naive Bayes Model	LSTM
Accuracy	89.56%	93.75%
AUC	0.894	0.972

Synthetic Data

Metric	Naive Bayes Model	LSTM
Accuracy	74.69%	57.94%
AUC	0.826	0.599

Conclusion:

In this project, we explored the performance of various models on both real-world and synthetic data. The models included a generative model, and a discriminative model.

The LSTM model trained on real-world data achieved an accuracy of 94% and an AUC of 0.97, demonstrating a high level of performance. However, when the same LSTM model was trained on synthetic data generated by a Naive Bayes model, the performance dropped significantly, with an accuracy of 58% and an AUC of 0.6. This stark difference in performance could be attributed to the quality and distribution of the synthetic data, which may not have adequately captured the complexity and nuances of the real-world data.

The generative model, on the other hand, achieved an accuracy of 90% on the real-world data. This suggests that while the generative model was less accurate than the LSTM model on real-world data, it still performed reasonably well.

The discriminative model achieved an accuracy of 75% on the real-world data, which was lower than both the LSTM and generative models. However, it outperformed the LSTM model trained on synthetic data, achieving an accuracy of 75% compared to the LSTM's 60%.

In conclusion, while the LSTM model performed best on real-world data, its performance dropped significantly on synthetic data. The generative model performed reasonably well on real-world data, but not as well as the LSTM model. The discriminative model had the lowest performance on real-world data, but outperformed the LSTM model on synthetic data. These results highlight the importance of the quality and representativeness of the training data in model performance, as well as the strengths and weaknesses of different types of models. Future work could explore ways to improve the quality of synthetic data or adjust the complexity of the models to enhance performance.