**Word Warden: A Toxicity Classifier**

**Mini Project report submitted to**

Text, logo

Description automatically generated with medium confidence

**Group members:**

* Kartik Mittal (215890116)
* Ananya Gupta (215890030)

Under the supervision of

**Dr. Raguru Jaya krishna**

**Assistant Professor (Senior Scale)**

Department of Computer Science and Engineering

Manipal Institute of Technology, Bengaluru

**Manipal Institute of Technology**

**Bengaluru Campus-560064, Karnataka, India.**

|  |  |  |
| --- | --- | --- |
| **Contents** | | **Page. No.** |
| **1** | Introduction………………………………………………….. | **3** |
| **2** | Literature Review…………………………………………… | **6** |
| **3** | Packages Used………………………………………………... | **9** |
| **4** | Proposed Methodology……….…………………………….... | **11** |
| **5** | Implementation Details…………………………………….... | **14** |
| **6** | Results and Accuracy……………………………………….. | **19** |
| **7** | Conclusion…………………………………………………... | **23** |
| References……………………………………………………….…... | | **24** |

**Chapter 1**

**Introduction**

Conversational toxicity is an issue that can lead people both to stop genuinely expressing themselves and to stop seeking others' opinions out of fear of abuse/harassment. As a response to this pressing issue, the present study introduces Word Warden, an innovative natural language processing (NLP) project designed to mitigate the spread of toxic language in online interactions. The goal of this project will be to use machine learning to identify toxicity in text, which could be used to help deter users from posting potentially hurtful messages, craft more civil arguments when engaging in discourse with others, and to gauge the toxicity of other users' comments.

This project aims to implement various machine learning models – specifically Support Vector Machines (SVM), Logistic Regression, KNN Classification, Naïve Bayes Classification & Random Forest methods to tackle the above task, assessing these models' performances on both binary and multi-label classification tasks. Notably, this project has harnessed the power of the Naive Bayes method to achieve the highest accuracy in the classification of textual data, underscoring the efficacy of this particular approach in the context of identifying and moderating toxic content.

Moreover, in order to enhance user accessibility and interaction, the Word Warden project has seamlessly integrated a user-friendly interface through the utilization of the Stream lit framework in Python. This graphical interface enables users to conveniently input textual content and promptly receive categorization results, thereby facilitating a streamlined and intuitive user experience.

**1.1 Problem statement:**

The proliferation of toxic language in online communication poses a significant threat to the well-being and safety of internet users. Existing content moderation methods often lack accuracy and efficiency, leading to the dissemination of harmful content. Therefore, there is an urgent need to develop a robust and precise text classification system capable of accurately identifying and categorizing toxic language in online messages. This system should provide effective mitigation measures to foster a healthier and more secure online environment for all users.

**1.2 Objectives:**

* **To Develop a Text Classification System**: Create a text classification system that can accurately classify messages into one of four categories: severely toxic, toxic, mildly toxic, or non-toxic.
* **Enhance Online Safety**: Contribute to the ongoing effort to combat toxic language in online interactions, fostering a more inclusive and secure digital environment for internet users.
* **Scalability and Adaptability**: Design the Word Warden system with scalability and adaptability in mind, enabling its integration into a wide range of online platforms and environments.
* **User Education**: Educate users on responsible online communication and the potential consequences of toxic language, promoting a culture of respect and civility.

**1.3 Individual Contribution:**

In the development of the Word Warden project, Kartik focused on the application of machine learning algorithms, demonstrating expertise in Support Vector Machines (SVM), Naive Bayes, and logistic regression. He expertly employed the Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer, enhancing the model's understanding of the textual data.

Ananya, on the other hand, took charge of the user interface (UI) development, leveraging Stream lit in Python to create an intuitive and user-friendly platform for the Word Warden application. Ananya also implemented pre-processing techniques, such as the removal of special characters using regular expressions, contributing to the overall data cleaning and refinement process.

We both worked upon documentation in a collaborative manner, we did a lot of research regarding the proper documentation for the project.

This synergistically resulted in the successful realization of the Word Warden project.

**Chapter 2**

**Literature review**

* 1. **NLP Concepts:**
* **Text Representation**: The use of TF-IDF vectorization is a fundamental NLP concept in the Word Warden project. This technique captures the importance of words in a document corpus, enabling the algorithms to understand the context and significance of words within messages.
* **Regular Expression for Text Preprocessing:** The application of regular expressions to remove special characters is a crucial step in text preprocessing. This NLP technique enhances the quality of the input data by eliminating noise, ensuring that the machine learning models can focus on the semantic content of the messages.
* **Machine Learning Algorithms**: The incorporation of Support Vector Machines (SVM), Naive Bayes, and Logistic Regression, etc demonstrates the versatility of machine learning techniques in message classification. Each algorithm has its strengths, and the selection process is crucial to achieving the optimal balance between accuracy and efficiency.

**2.2 Related Works:**

* Toxicity Detection in Text: Previous research has explored toxicity detection in various forms of online content. The work of Jigsaw's Perspective API and the Kaggle Toxic Comment Classification Challenge has laid the foundation for identifying and mitigating toxic language.
* Comparative Analysis of ML Models: Research studies comparing the performance of machine learning models in text classification,
* such as the work by Zhang et al. (2018), provide valuable insights into the strengths and weaknesses of different algorithms. These insights can inform the decision-making process in the Word Warden project.
* Feature Engineering and Text Preprocessing Techniques: The literature also encompasses studies on feature engineering and advanced text preprocessing methods. Techniques such as word embeddings (Mikolov et al., 2013) and deep learning architectures (Hochreiter & Schmidhuber, 1997) may offer avenues for enhancing the Word Warden's classification accuracy.

|  |  |
| --- | --- |
| Research Articles/Projects | Link |
| Systematic Literature Review: Toxic Comment Classification | [Systematic Literature Review: Toxic Comment Classification | IEEE Conference Publication | IEEE Xplore](https://ieeexplore.ieee.org/document/9971338/references#references) |
| CS224N: Detecting and Classifying Toxic Comments | [6837517.pdf (stanford.edu)](https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/reports/6837517.pdf) |
| Classification of social media Toxic comments using Machine learning models | [Impact of SMOTE on Imbalanced Text Features for Toxic Comments Classification Using RVVC Model (arxiv.org)](https://arxiv.org/ftp/arxiv/papers/2304/2304.06934.pdf) |

* 1. **Literature Gap:**

The existing literature on toxic content classification predominantly explores the application of machine learning algorithms, including SVM, Naive Bayes, and logistic regression, etc, in combination with techniques such as TF-IDF vectorization and the removal of special characters using regular expressions, there remains a noticeable gap in the literature concerning the following aspects:

* **Deep Learning Approaches**: The current body of research primarily relies on traditional machine learning algorithms. Investigating the application of deep learning models, such as recurrent neural networks (RNNs) or transformers like BERT, might offer improved performance in capturing nuanced contextual dependencies and understanding the semantic intricacies of toxic language.
* **Dynamic and Adaptive Models**: The majority of existing studies focus on static models trained on fixed datasets. Investigating the development of dynamic and adaptive models capable of learning and evolving with the emergence of new linguistic patterns and shifts in online communication norms can contribute to the sustainability and relevance of the Word Warden system over time.
* **Cross-Cultural Sensitivity**: The current literature often assumes a universal understanding of toxicity, neglecting the nuances of language and cultural variations. Investigating the development of models that are sensitive to cross-cultural differences in communication styles and expressions of toxicity can broaden the applicability of the Word Warden system across diverse user bases.

**Chapter 3**

**Packages Used**

* **Nltk:** NLTK stands for Natural Language Toolkit. It is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources, such as WordNet. Additionally, NLTK includes a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, among other tasks in natural language processing.
* **Sklearn:** Scikit-learn, often abbreviated as sklearn, is a popular open-source machine learning library for the Python programming language. It is built on NumPy, SciPy, and matplotlib and provides a range of supervised and unsupervised learning algorithms through a consistent interface.
* **Wordcloud:** The wordcloud package is a Python library that allows for the creation of word clouds from text data. It provides a simple interface to generate word clouds, where the size of each word is proportional to its frequency in the text.
* **Re:** In Python, re is a built-in module that provides support for regular expressions (also known as regex). Regular expressions are a powerful tool used for pattern matching in strings. The re module allows you to perform various operations on strings, such as searching for specific patterns, replacing substrings, and splitting strings based on a specified pattern.
* **Seaborn:**  Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn is built on top of Matplotlib and integrates closely with Pandas data structures, making it particularly suited for data visualization from DataFrames.
* **Pandas:** Pandas is a popular open-source Python library that provides high-performance, easy-to-use data structures and data analysis tools. It is designed to facilitate data manipulation and analysis tasks, especially for tabular and time-series data. Pandas' primary data structures, Series and DataFrame, are built on top of NumPy and offer powerful data manipulation and analysis capabilities.

**Chapter 4**

**Proposed Methodology**

**4.1 Data Collection and Pre-processing:**

The dataset we used came from Jigsaw/Google's Toxic Comment Classification Challenge on Kaggle. The dataset contains 159,571 labelled examples of Wikipedia comments that have been labelled by human ratters for toxic behaviour. The data comes in the schema of <id.commentText. toxic, severe Toxic, obscene, threat, insult, identityHate>, where the labels for toxic, severeToxic, obscene, threat, insult, and identityHate are all Boolean labels.

Data pre-processing steps include:

* Removal of special characters and non-alphanumeric characters.
* Conversion of text to lowercase.
* Removal of non-ASCII characters.
* Replacing newline characters with spaces.

The code creates separate DataFrames for different types of toxic comments (e.g., toxic, severe\_toxic, obscene, etc.).

The datasets are balanced to ensure an equal representation of both positive and negative labels for each category.

**4.2 Exploratory Data Analysis (EDA):**

Basic EDA is performed with commands like data.sample(5), data.isnull().sum(), and data['toxic'].value\_counts() to understand the data structure.

Visualizations are used to understand the distribution of labels across different categories, using bar plots.

**4.3 Feature Engineering:**

1. Feature Extraction: Transforming data into numerical features. For text data, techniques like TF-IDF are used.
2. Feature Selection: Choosing relevant features while discarding less important ones to improve efficiency and reduce overfitting.
3. Feature Creation: Developing new features based on domain knowledge.
4. Scaling and Normalization: Ensuring features are on a similar scale.
5. Handling Missing Data: Imputing missing values or creating indicators for missing data.
6. Text Processing: Tokenization, punctuation removal, and text pre-processing in NLP.

**4.4 Model Selection and Evaluation:**

The code selects a diverse set of machine learning models, which include:

* Logistic Regression
* K-Nearest Neighbours
* Naive Bayes (both Bernoulli and Multinomial variants)
* Support Vector Machine
* Random Forest

Each of these models has its unique characteristics and is chosen to assess how well they perform in classifying toxic comments.

**Evaluation Framework**:To systematically evaluate the model performance, the code defines a function called ‘cv\_tf\_train\_test’. This function is designed to train and test the selected models using TF-IDF vectorization of the text data.The code calculates two critical evaluation metrics for each model**: F1 score** and **accuracy score**. These metrics provide insights into the model's ability to classify toxic comments and its overall accuracy.

**Model Performance Assessment**:Model performance is assessed by testing the models on different balanced toxic comment datasets, such as toxic, severe\_toxic, obscene, etc. These datasets have been prepared to ensure an equal representation of both positive and negative labels.The results of model evaluations, including F1 scores and accuracy scores, are presented for each model.

**4.5 User Interface Development:**

In our model, we created a Stream lit-based user interface for a text toxicity detection system. Users can input a message, and the system assesses the toxicity of the message. It categorizes messages into different levels of toxicity and provides corresponding warnings or approval messages with icons based on the predicted toxicity level. Users are informed whether their message is safe to post on social media platforms.

**Chapter 5**

**Implementation Details**

**5.1 Importing libraries:**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import re

import nltk

import string

import nlputils

from wordcloud import WordCloud, STOPWORDS

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score

**5.2 Reading dataset**:

data = pd.read\_csv("train.csv")

data.sample(5)

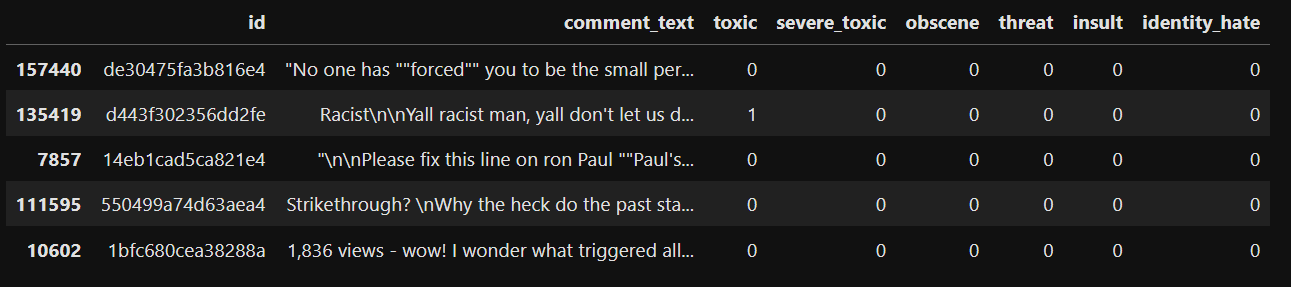


Figure Dataset

**5.3 Data preprocessing:**

alphanumeric = lambda x: re.sub('\w\*\d\w\*', ' ', x)

punc\_lower = lambda x: re.sub('[%s]' % re.escape(string.punctuation), ' ', x.lower())

remove\_n = lambda x: re.sub("\n", " ", x)

remove\_non\_ascii = lambda x: re.sub(r'[^\x00-\x7f]',r' ', x)

df['comment\_text'] = df['comment\_text'].map(alphanumeric).map(punc\_lower).map(remove\_n).map(remove\_non\_ascii)

Toxic\_comment\_df=df.loc[:,['id','comment\_text','toxic']]

Insulting\_comment\_df=df.loc[:,['id','comment\_text','insult']]

Threatening\_comment\_df=df.loc[:,['id','comment\_text','threat']]

IdentityHate\_comment\_df=df.loc[:,['id','comment\_text','identity\_hate']]

Obscene\_comment\_df=df.loc[:,['id','comment\_text','obscene']]

Severetoxic\_comment\_df=df.loc[:,['id','comment\_text','severe\_toxic']]

**5.4 Data visualization:**

import seaborn as sns

labels = ["toxic", "severe\_toxic", "obscene", "threat", "insult", "identity\_hate"]

targets = data[labels].values

import matplotlib.pyplot as plt

val\_counts = data[labels].sum()

plt.figure(figsize=(8,5))

ax = sns.barplot(x=val\_counts.index, y=val\_counts.values, alpha=0.8)

plt.title("Labels per Classes")

plt.xlabel("Various Label Type")

plt.ylabel("Counts of the Labels")

rects = ax.patches

labels = val\_counts.values

plt.show()

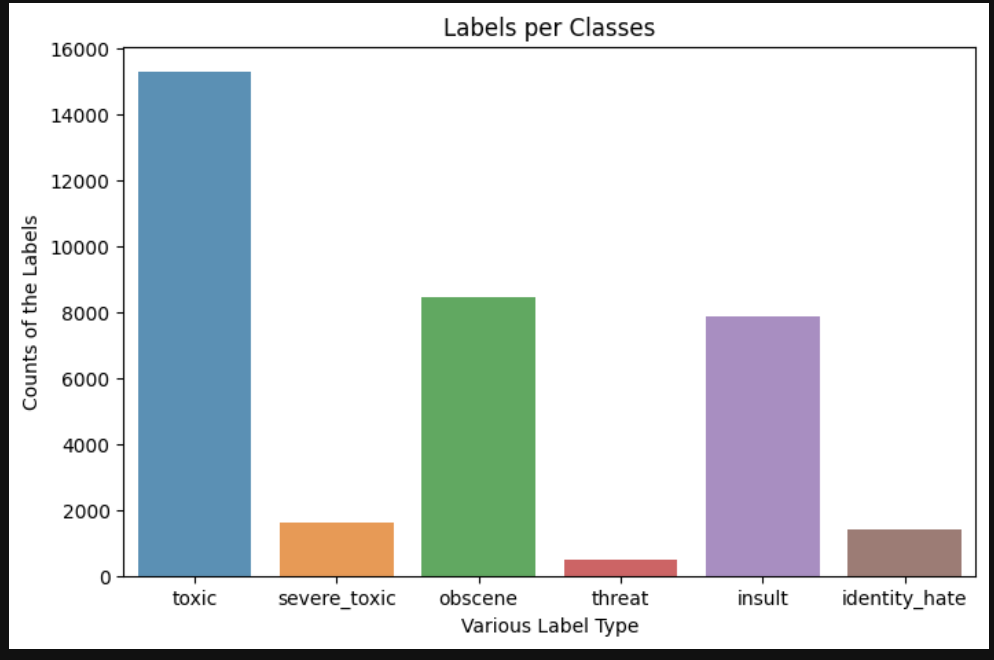


Figure Labels per class

def wordcloud(df, label):

subset=df[df[label]==1]

text=subset.comment\_text.values

wc= WordCloud(background\_color="black",max\_words=2000)

wc.generate(" ".join(text))

plt.figure(figsize=(20,20))

plt.subplot(221)

plt.axis("off")

plt.title("Words frequented in {}".format(label), fontsize=20)

plt.imshow(wc.recolor(colormap= 'gist\_earth' , random\_state=244), alpha=0.98)

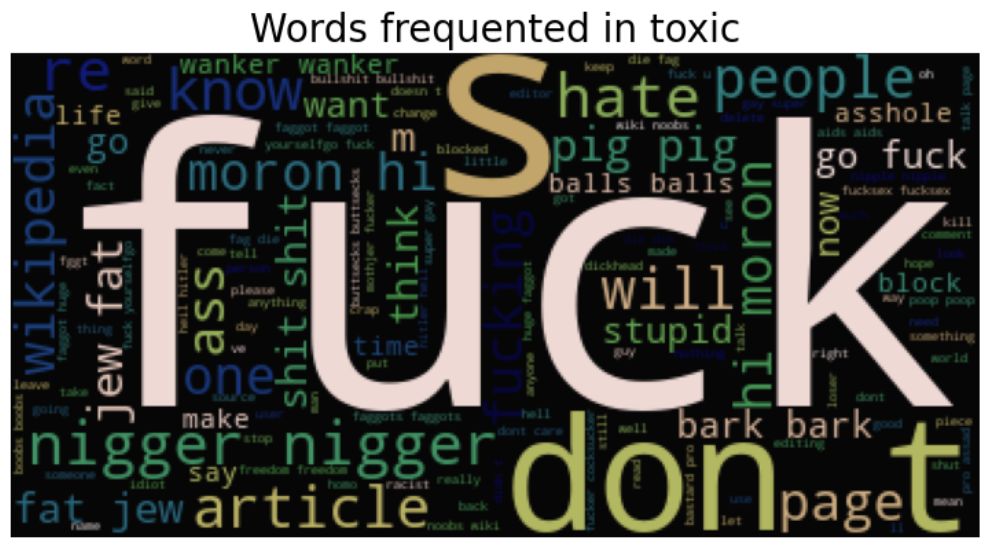


Figure Word Cloud

**5.5 Model Implementation:**

def cv\_tf\_train\_test(dataframe,label,vectorizer,ngram):

X = dataframe.comment\_text

y = dataframe[label]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=50)

cv1 = vectorizer(ngram\_range=(ngram), stop\_words='english')

X\_train\_cv1 = cv1.fit\_transform(X\_train)

X\_test\_cv1 = cv1.transform(X\_test)

lr = LogisticRegression()

lr.fit(X\_train\_cv1, y\_train)

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train\_cv1, y\_train)

bnb = BernoulliNB()

bnb.fit(X\_train\_cv1, y\_train)

mnb = MultinomialNB()

mnb.fit(X\_train\_cv1, y\_train)

svm\_model = LinearSVC()

svm\_model.fit(X\_train\_cv1, y\_train)

randomforest = RandomForestClassifier(n\_estimators=100,random\_state=50)

randomforest.fit(X\_train\_cv1, y\_train)

**5.7 Model Evaluation:**

f1\_score\_data = {'F1 Score':[f1\_score(lr.predict(X\_test\_cv1), y\_test), f1\_score(knn.predict(X\_test\_cv1), y\_test),f1\_score(bnb.predict(X\_test\_cv1), y\_test), f1\_score(mnb.predict(X\_test\_cv1), y\_test),f1\_score(svm\_model.predict(X\_test\_cv1), y\_test), f1\_score(randomforest.predict(X\_test\_cv1), y\_test)]}

accuracy\_score\_data = {'Accuracy Score':[accuracy\_score(lr.predict(X\_test\_cv1), y\_test), accuracy\_score(knn.predict(X\_test\_cv1), y\_test), accuracy\_score(bnb.predict(X\_test\_cv1), y\_test), accuracy\_score(mnb.predict(X\_test\_cv1), y\_test), accuracy\_score(svm\_model.predict(X\_test\_cv1), y\_test), accuracy\_score(randomforest.predict(X\_test\_cv1), y\_test)]}

df\_f1 = pd.DataFrame(f1\_score\_data, index=['Log Regression','KNN', 'BernoulliNB', 'MultinomialNB', 'SVM', 'Random Forest'])

df\_f2 = pd.DataFrame(accuracy\_score\_data, index=['Log Regression','KNN', 'BernoulliNB', 'MultinomialNB', 'SVM', 'Random Forest'])

df\_f=pd.concat([df\_f1,df\_f2],axis=1)

return df\_f

confusion\_matrix(mnb.predict(X\_test\_fit),y\_test)

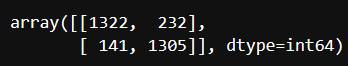


Figure Confusion Matrix of MNB

comment1 = ['Go kill yourself']

comment1\_vect = tfv.transform(comment1)

mnb.predict\_proba(comment1\_vect)[:,1]

comment2 = ['hello you are stupid']

comment2\_vect = tfv.transform(comment2)

mnb.predict\_proba(comment2\_vect)[:,1]

**Chapter 6**

**Results and Accuracy**

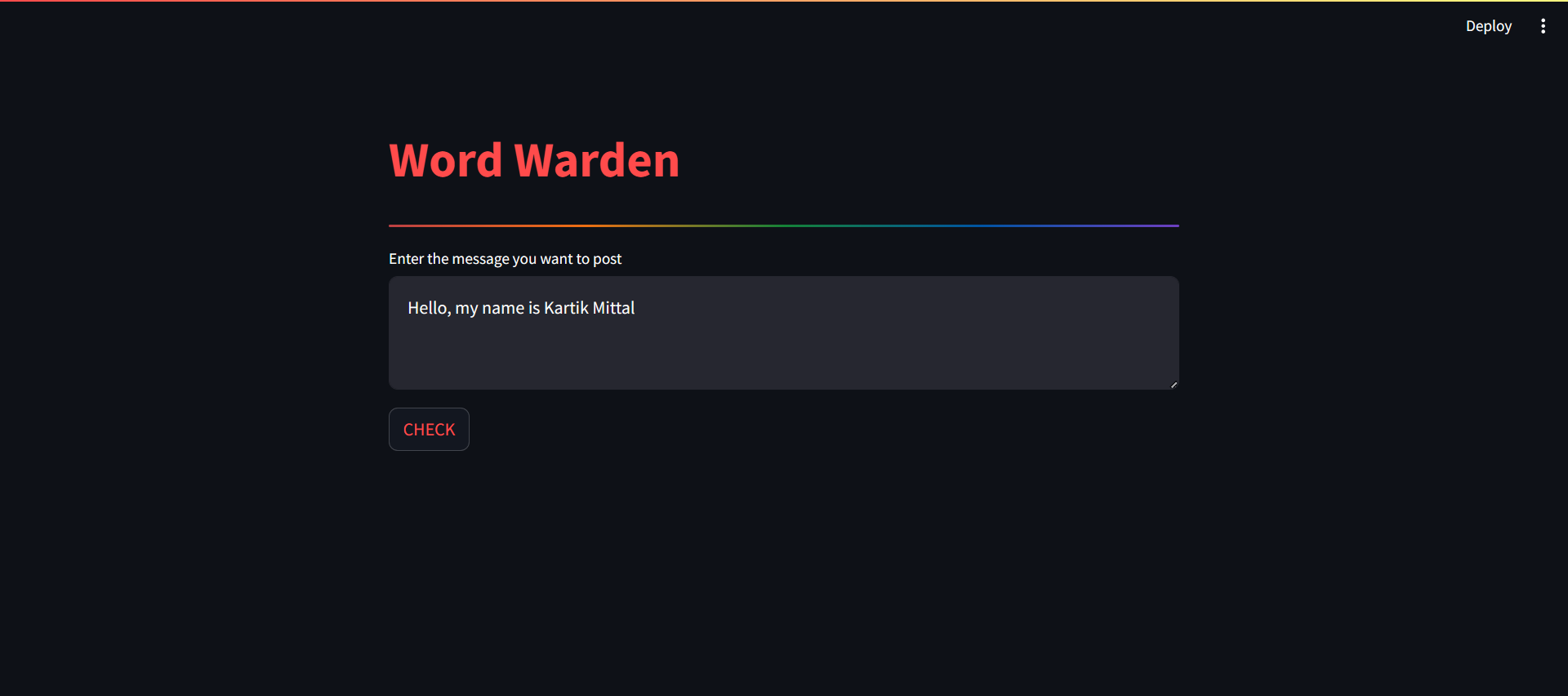


Figure Example

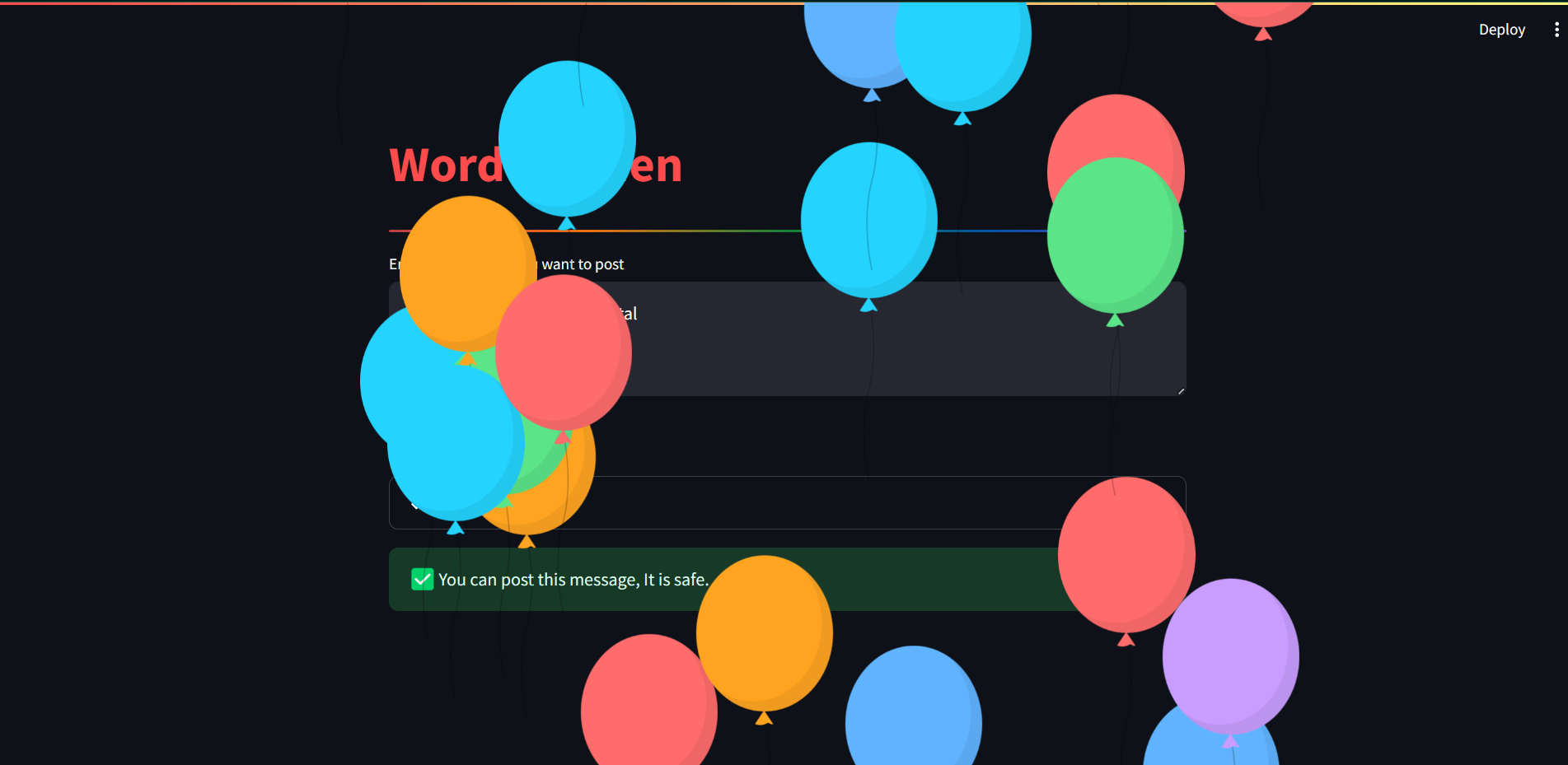


Figure Result

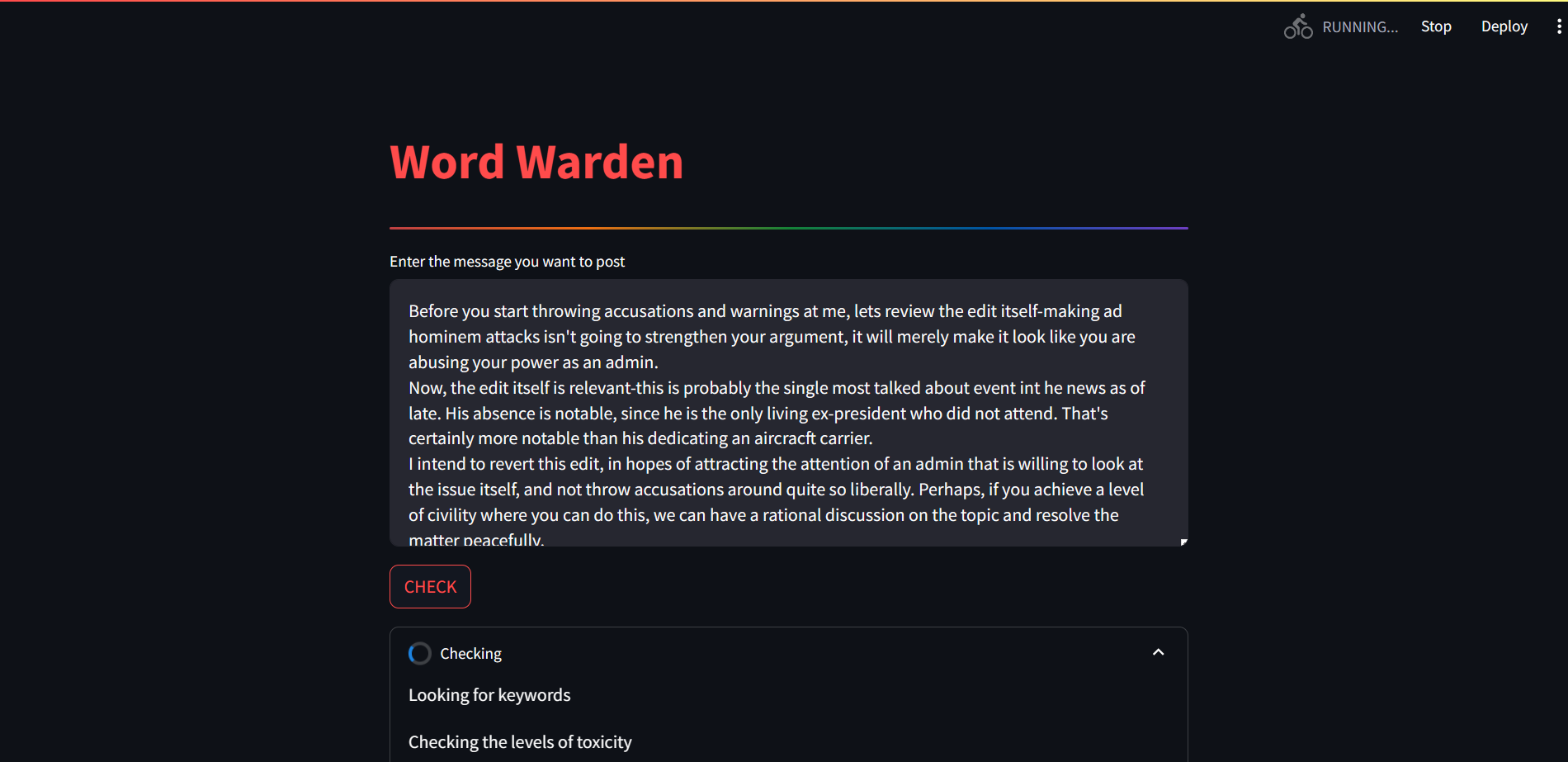


Figure Large message

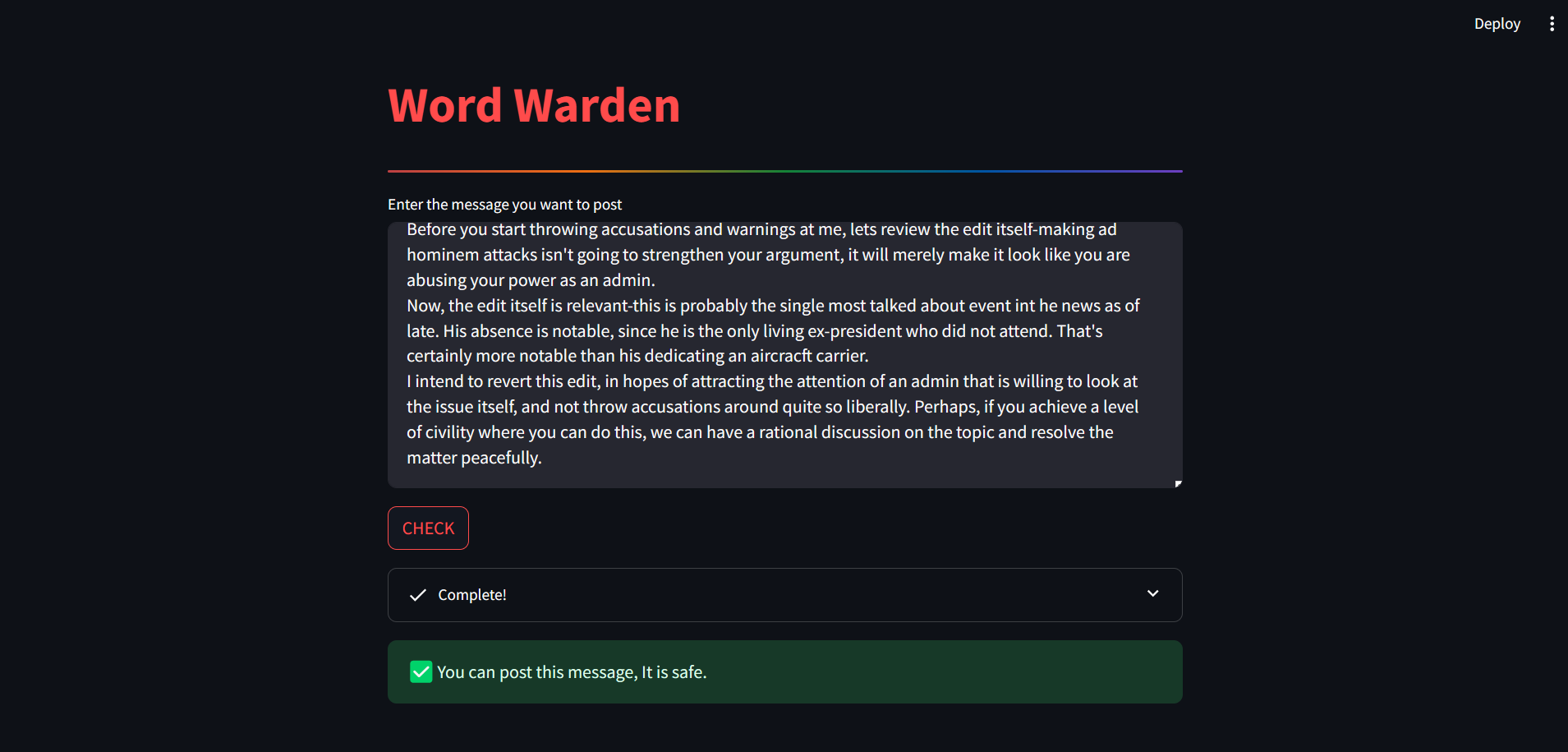


Figure Result

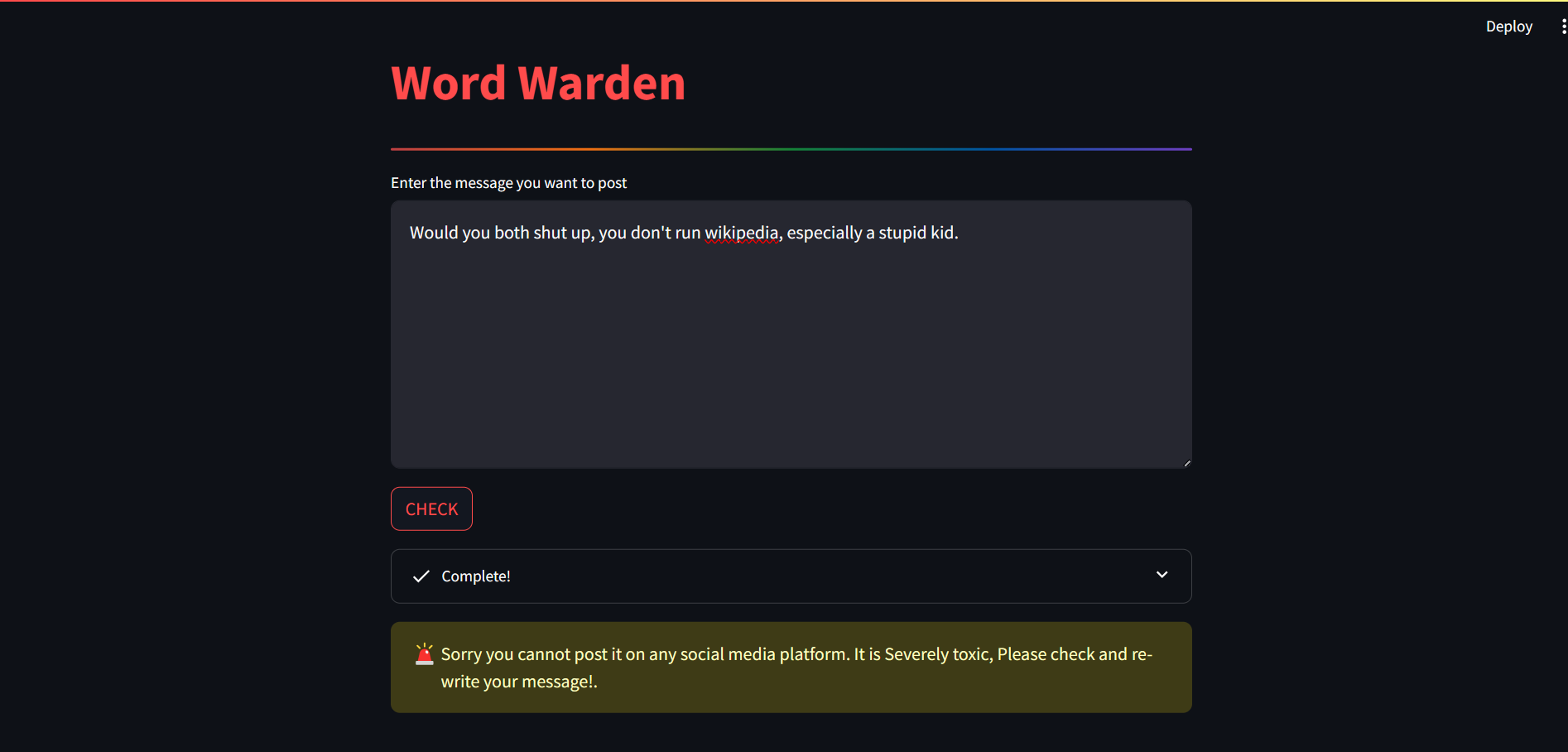


Figure Toxic message

Table Accuracy of Labels

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Accuracy Score (Toxic) | Accuracy Score (Severe\_Toxic) | Accuracy Score (Obscene) | Accuracy Score (Threat) | Accuracy Score (Insult) | Accuracy Score (Identity Hate) |
| Logistic regression | 87 | 94.2 | 90.5 | 90.5 | 90.6 | 90.9 |
| KNN | 53.6 | 86.4 | 53.3 | 84.3 | 60.8 | 81.2 |
| Bernoulli Naïve Bayes | 70.3 | 74.5 | 71.3 | 67.5 | 72.8 | 72.7 |
| MNB | 87.5 | 93.1 | 88.6 | 90.2 | 89.6 | 90.2 |
| SVM | 87.3 | 93.9 | 91.7 | 90.2 | 90.9 | 90 |
| Random Forest | 84.9 | 94.2 | 88.7 | 92.6 | 89.5 | 88.9 |

We achieved a maximum of **87.5% accuracy in Toxic** label by implementing Multinomial Naïve Bayes, **94.2% accuracy in Severe\_Toxic** label by implementing Logistic Regression, **91.7% accuracy in Obscene** label by implementing SVM, **92.6% accuracy in Threat** label by implementing Random Forest, **90.9% accuracy in Insult** label by implementing SVM & **90.9% accuracy in Identity\_Hate** label by implementing Logistic Regression.

Table Visual Representation of accuracy

**Chapter 7**

**Conclusion**

The Word Warden project aimed to develop a robust system for classifying messages into categories of toxicity, ranging from severely toxic to non-toxic. Through the utilization of machine learning algorithms such as Support Vector Machines (SVM), Logistic Regression, and Naive Bayes, coupled with the implementation of a user-friendly interface using Streamlit in Python, the project has successfully achieved notable milestones.

Throughout the project, the significance of addressing toxic language in digital communication has been emphasized. The potential applications of Word Warden are vast, ranging from content moderation on online platforms to fostering healthier and more respectful online interactions.

While the achieved accuracy of 87.5% is commendable, it is important to acknowledge the ever-evolving nature of language and the challenges posed by the dynamic landscape of online communication. Ongoing efforts should be directed towards maintaining and improving the model's performance by incorporating additional data, exploring advanced feature engineering techniques, and staying abreast of emerging linguistic trends.

**References**

1. F. Museng, A. Jessica, N. Wijaya, A. Anderies and I. A. Iswanto, "Systematic Literature Review: Toxic Comment Classification," 2022 IEEE 7th International Conference on Information Technology and Digital Applications (ICITDA), Yogyakarta, Indonesia, 2022, pp. 1-7, doi: 10.1109/ICITDA55840.2022.9971338.
2. C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad, and Y. Chang, "Abusive language detection in online user content," in Proc. 25th Int. Conf World Wide Web. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, Apr. 2016, pp. 145- 153.
3. CS224N: Detecting and Classifying Toxic Comments [6837517.pdf (stanford.edu)](https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/reports/6837517.pdf)
4. [Toxic Comment Classification Challenge | Kaggle](https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge)
5. [Simple WordCloud Using nltk Library in Python | RP’s Blog on AI (nlpfy.com)](https://nlpfy.com/simple-wordcloud-using-nltk-library-in-python/)
6. [NLTK :: nltk package](https://www.nltk.org/api/nltk.html)