GitHub-Project-ML02-Deception_Detectors

Demo-Video of our project.

Text Classification for Sexual Harassment Detection

By Team Deception Detectors

Introduction

This script implements a machine learning pipeline for text classification to detect different forms of sexual harassment, including commenting, ogling/facial expressions/staring, and touching/groping. The pipeline includes data preprocessing, model training, evaluation, and real-time text classification.

Data Preparation

- Data Loading: The script loads the training, development, and test datasets from CSV files.
- Data Cleaning: Various cleaning steps are performed to preprocess the text data, including lowercasing, removing special characters, and standardizing contractions.

```
In [5]: print('train set', train_set.head(10))
        print('test set', test_set.head(10))
```

```
train set                                          Description  Commenting  \
3784                                         harassment           0
2111  a man was tranculized to oversleep then the wi...           0
4076  ONE DAY I SAW A GROUP OF BOYS ARE TRY TO TOUCH...           0
3509                                  it was really bad.           0
418   The station and itself is very unsafe. The lig...           0
5033                            Taking pictures in. Mundka           0
1934  one night when my friend was sleeping, a man b...           0
3396  I was in the car with my uncle and my sister a...           0
5741  A lot of comments are endured by me while I am...           1
5381                                       Slapped a boy           1

      Ogling/Facial Expressions/Staring  Touching /Groping
3784                                  1                  1
2111                                  0                  0
4076                                  0                  1
3509                                  0                  1
418                                   0                  0
5033                                  0                  0
1934                                  0                  0
3396                                  0                  1
5741                                  0                  0
5381                                  0                  1
```

Exploratory Data Analysis

- Label Distribution: The script visualizes the distribution of labeled comments across different categories to understand the prevalence of each form of harassment.

[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]

```
labels = ['Commenting', 'Ogling/Facial Expressions/Staring', 'Touching /Groping']
```

```
test_features = test_set["Description"]
train_features = train_set["Description"]
```

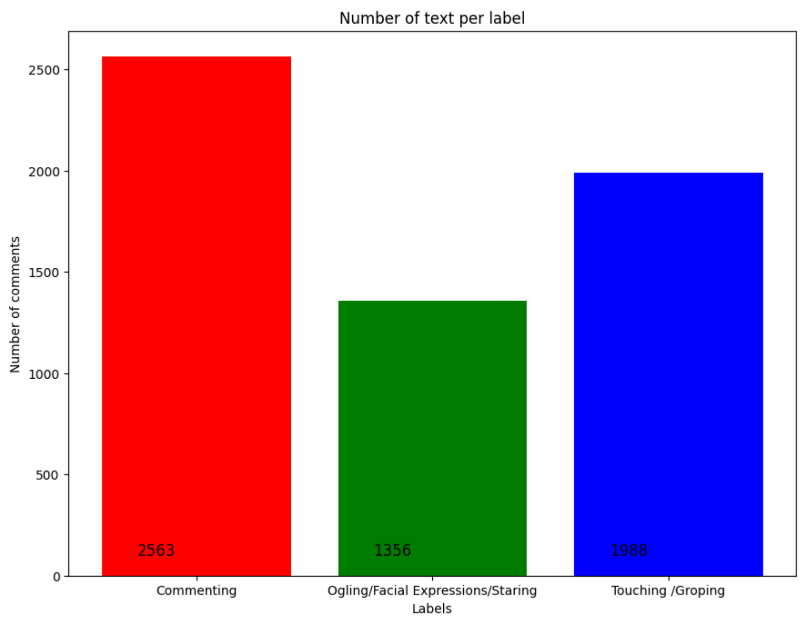
```
train_labels = train_set.drop(['Description'], axis = 1)
test_labels = test_set.drop(['Description'], axis = 1)
```

- ```
  def clean_text(text):
      text = text.lower()
      text = re.sub(r"what's", "what is ", text)
      text = re.sub(r"\'s", " ", text)
      text = re.sub(r"\'ve", " have ", text)
      text = re.sub(r"can't", "can not ", text)
      text = re.sub(r"n't", " not ", text)
      text = re.sub(r"i'm", "i am ", text)
      text = re.sub(r"\'re", " are ", text)
      text = re.sub(r"\'d", " would ", text)
      text = re.sub(r"\'ll", " will ", text)
      text = re.sub(r"\'scuse", " excuse ", text)
      text = re.sub('\W', ' ', text)
      text = re.sub('\s+', ' ', text)
      text = text.strip(' ')
      return text

  test_features_cleaned = test_features.map(lambda com : clean_text(com))
  train_features_cleaned = train_features.map(lambda com : clean_text(com))
  ```

Model Training and Evaluation

- Model Selection: Three machine learning algorithms, namely Naive Bayes, Logistic Regression, and Linear SVM, are employed for text classification.
- Pipeline Construction: Pipelines are built for each algorithm, incorporating TF-IDF vectorization and One-vs-Rest strategy for multi-label classification.
- Model Evaluation: The trained models are evaluated using various metrics such as ROC-AUC score, accuracy, hamming loss, confusion matrices, and classification reports.


  ```

[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]

Naive Bayesian pipelining

```
In [14]: run_pipeline(NB_pipeline, train_features, train_labels, test_features, test_labels)
```

```
roc_auc:  0.8208103565487298
HAMMING LOSS: 0.2087
accuracy:  0.5179987797437462
confusion matrices:
[[[ 877   94]
  [ 306  362]]

 [[1252    4]
  [ 345   38]]

 [[1166    9]
  [ 268  196]]]
classification_report:
                                 precision    recall  f1-score   support

                     Commenting       0.79      0.54      0.64       668
Ogling/Facial Expressions/Staring     0.90      0.10      0.18       383
                Touching /Groping     0.96      0.42      0.59       464

                      micro avg       0.85      0.39      0.54      1515
                      macro avg       0.88      0.35      0.47      1515
                   weighted avg       0.87      0.39      0.51      1515
                    samples avg       0.34      0.30      0.31      1515
```

Logistic regression pipelining

```
In [15]: run_pipeline(LR_pipeline, train_features, train_labels, test_features, test_labels)
```

```
roc_auc:  0.8475914049590177
HAMMING LOSS: 0.1714
accuracy:  0.6064673581452105
confusion matrices:
[[[ 890   81]
  [ 245  423]]

 [[1238   18]
  [ 280  103]]

 [[1139   36]
  [ 183  281]]]
classification_report:
                                 precision    recall  f1-score   support

                     Commenting       0.84      0.63      0.72       668
Ogling/Facial Expressions/Staring     0.85      0.27      0.41       383
                Touching /Groping     0.89      0.61      0.72       464

                      micro avg       0.86      0.53      0.66      1515
                      macro avg       0.86      0.50      0.62      1515
                   weighted avg       0.86      0.53      0.64      1515
                    samples avg       0.45      0.40      0.41      1515
```

SVM pipelining

[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]

```
In [16]: # run_pipeline(SVM_pipeline, train_features, train_labels, test_features, test_labels)
         run_SVM_pipeline(SVM_pipeline, train_features, train_labels, test_features, test_labels)
```

```
HAMMING LOSS: 0.1757
accuracy:  0.5930445393532642
confusion matrices:
[[[ 823  148]
  [ 206  462]]

 [[1200   56]
  [ 249  134]]

 [[1113   62]
  [ 143  321]]]
classification_report:
                                   precision    recall  f1-score   support

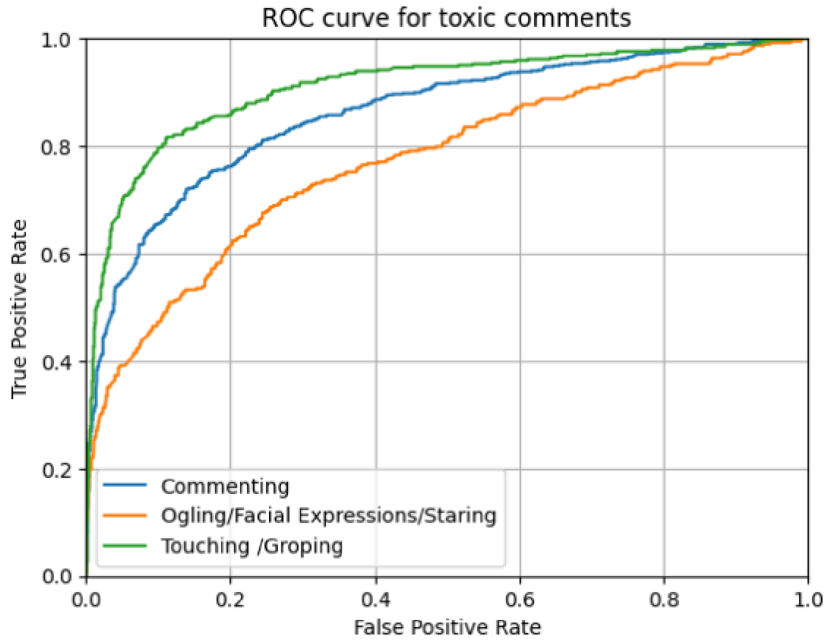
                       Commenting       0.76      0.69      0.72       668
Ogling/Facial Expressions/Staring       0.71      0.35      0.47       383
                 Touching /Groping      0.84      0.69      0.76       464

                        micro avg       0.78      0.61      0.68      1515
```

ROC Curve

Plot ROC curve for LogisticRegression model

```
In [17]: plot_pipeline_roc_curve(LR_pipeline, train_features, train_labels, test_features, test_labels)
```

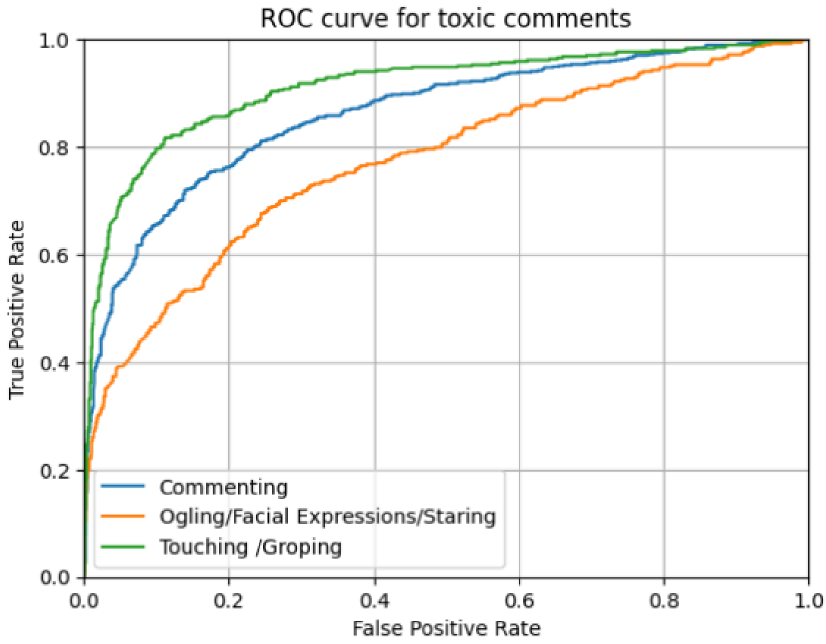


[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]

Plot ROC curve for LogisticRegression model

In [17]:
```
plot_pipeline_roc_curve(LR_pipeline, train_features, train_labels, test_features, test_labels)
```



Real-Time Text Classification

- Prediction Function: A function is defined to predict the probabilities of each harassment category for a given text input.
- Example Usage: An example demonstrates how to use the prediction function to classify real-time text inputs.

Now real time text classification (Sexual harassment categorization)

In [24]:
```
# Now let's define a function to predict the probabilities of each class for a given text input
def predict_probabilities(text, model):
    probabilities = model.predict_proba([text])[0]
    return probabilities

# Example usage:
text_input = input("Enter the text: ")
probabilities = predict_probabilities(text_input, trained_model)

# Output the probabilities for each class
print("Probabilities:")
print("Commenting:", probabilities[0])
print("Ogling/Facial Expressions/Staring:", probabilities[1])
print("Touching /Groping:", probabilities[2])
```

```
Enter the text: he grab my breasts
Probabilities:
Commenting: 0.035849811603050674
Ogling/Facial Expressions/Staring: 0.07129851240718962
Touching /Groping: 0.6871592080442173
```

[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]

Applying the Model to New Data

- Reading New Data: The script reads a CSV file containing new text data to be classified.
- Classification: Each text entry is classified using the trained model, and the corresponding probabilities for each category are computed.
- Thresholding: Based on a probability threshold (0.3 in this case), the script assigns binary labels to each category.

LIME Implementation

LIME Implementation

In [28]:
```
from lime import lime_text
from sklearn.pipeline import make_pipeline
```

In [29]:
```
from lime.lime_text import LimeTextExplainer
explainer = LimeTextExplainer(class_names=class_names)
```

In [30]:
```
import keras
import nltk
import pandas as pd
import numpy as np
import re
import io
import matplotlib
from IPython.display import display
```

In [31]:
```
dev_set=pd.read_csv('dev.csv')
train_set = pd.read_csv('train.csv')
test_set = pd.read_csv('test.csv')
# Concatenate dev_set and test_set into train_set
train_set = pd.concat([train_set, dev_set, test_set], ignore_index=True)
```

In [32]:
```
from nltk.tokenize import RegexpTokenizer

# tokenizer for words, i.e. any length word (alphanumeric) characters separated by a whitespace
tokenizer = RegexpTokenizer(r'\w+')

train_set["tokens"] = train_set["Description"].apply(tokenizer.tokenize)
train_set.head()
```

Out[32]:

	Description	Commenting	Ogling/Facial Expressions/Staring	Touching /Groping	tokens
0	Was walking along crowded street, holding mums...	0	0	1	[Was, walking, along, crowded, street, holding...
1	This incident took place in the evening.I was ...	0	1	0	[This, incident, took, place, in, the, evening...
2	I WAS WAITING FOR THE BUS. A MAN CAME ON A BIK...	1	0	0	[I, WAS, WAITING, FOR, THE, BUS, A, MAN, CAME...

Hamming Scores

[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]

```
    # true positives + true negatives/ total
    accuracy = accuracy_score(y_test, y_predicted)

    # Hamming loss
    hamming = hamming_loss(y_test, y_predicted)

    return accuracy, precision, recall, f1, hamming

accuracy, precision, recall, f1, hamming = get_metrics(y_test, y_predicted_counts)
print(f'accuracy = {accuracy*100:.3f}%, precision = {precision*100:.3f}%, recall = {recall*100:.3f}%, f1 = {f1*100:.3f}%')
print(f'\033[1mHamming loss = {hamming*100:.3f}%\033[0m')
```

```
accuracy = 75.189%, precision = 75.817%, recall = 75.189%, f1 = 75.384%
Hamming loss = 24.811%
```

In [43]:
```
# define model again for reproducilibity and sanity
clf_w2v = LogisticRegression(
    C=30.0,
    class_weight='balanced',
    solver='newton-cg',
    random_state=69
)

# fit model on word2vec embeddings
clf_w2v.fit(X_train_word2vec, y_train_word2vec)

# get predictions on word2vec embeddings
y_predicted_word2vec = clf_w2v.predict(X_test_word2vec)
```

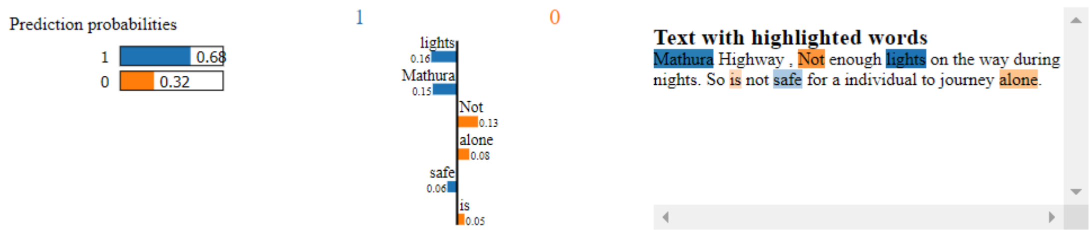
In [44]:
```
accuracy_word2vec, precision_word2vec, recall_word2vec, f1_word2vec, hamming_word2vec = get_metrics(y_test_word2vec, y_predicted_
print(f'accuracy = {accuracy_word2vec:.3f}, precision = {precision_word2vec:.3f}, recall = {recall_word2vec:.3f}, f1 = {f1_word2v
print(f'\033[1mHamming loss = {hamming_word2vec:.3f}\033[0m')
```

```
accuracy = 0.772, precision = 0.771, recall = 0.772, f1 = 0.771
Hamming loss = 0.228
```

Various Visualizations

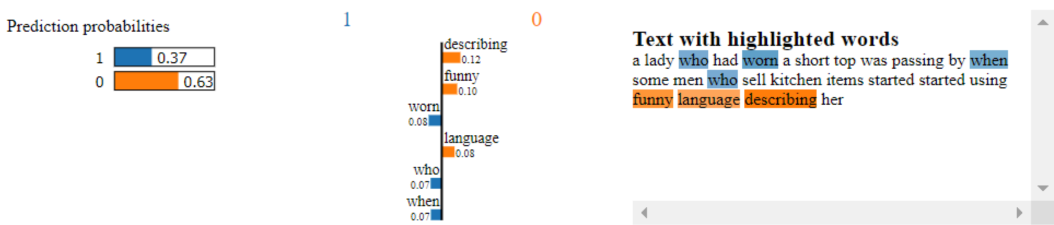
In [53]:
```
# interpret and visualize model's prediction on one tweet
visualize_one_exp(X_test_data, y_test_data, 100)
```

```
Index: 100
True class: 1
```



In [60]:
```
# interpret and visualize model's prediction on one tweet
visualize_one_exp(X_test_data, y_test_data, 75)
```

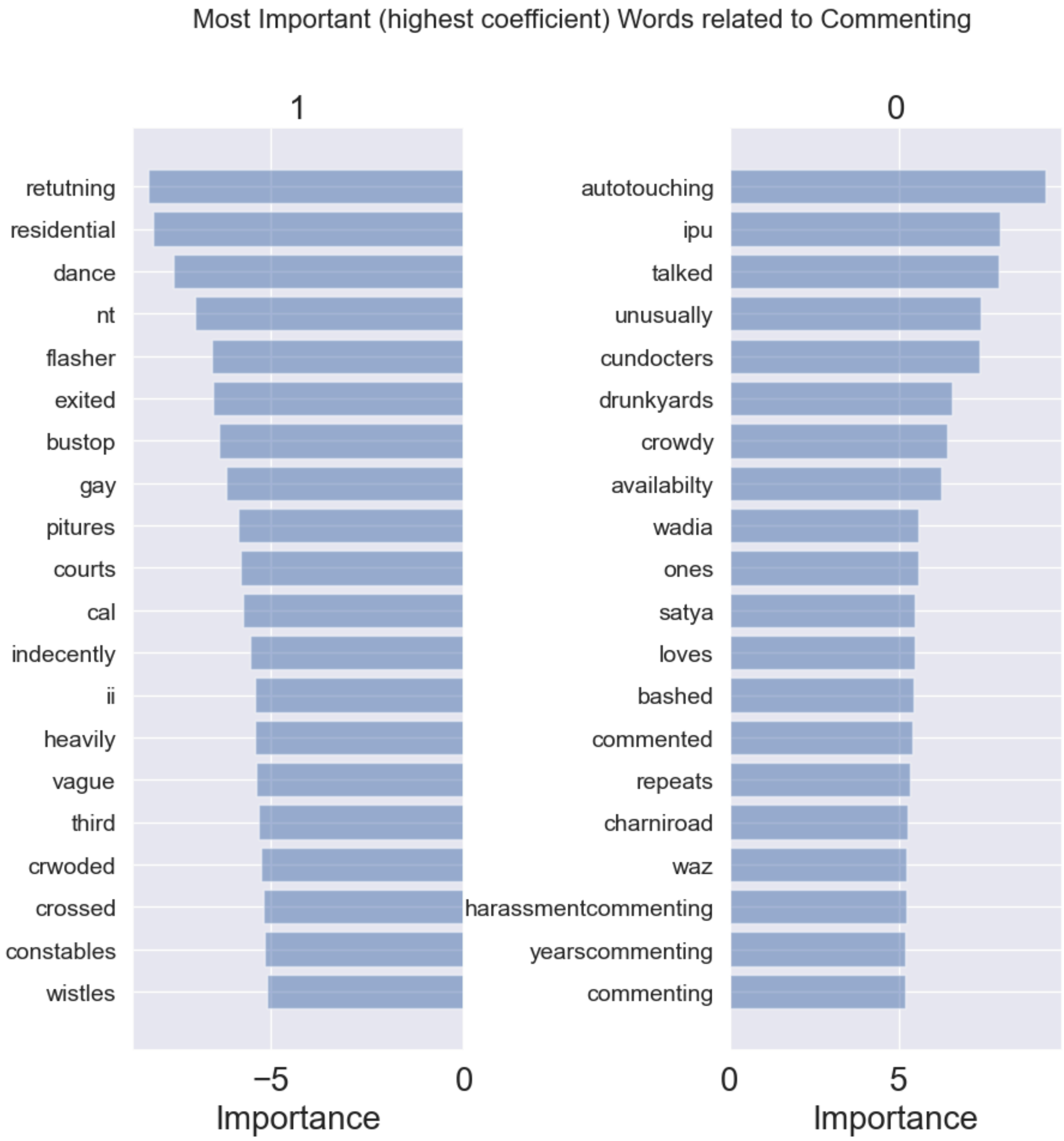
```
Index: 75
True class: 1
```



The most important words related to commenting

[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]

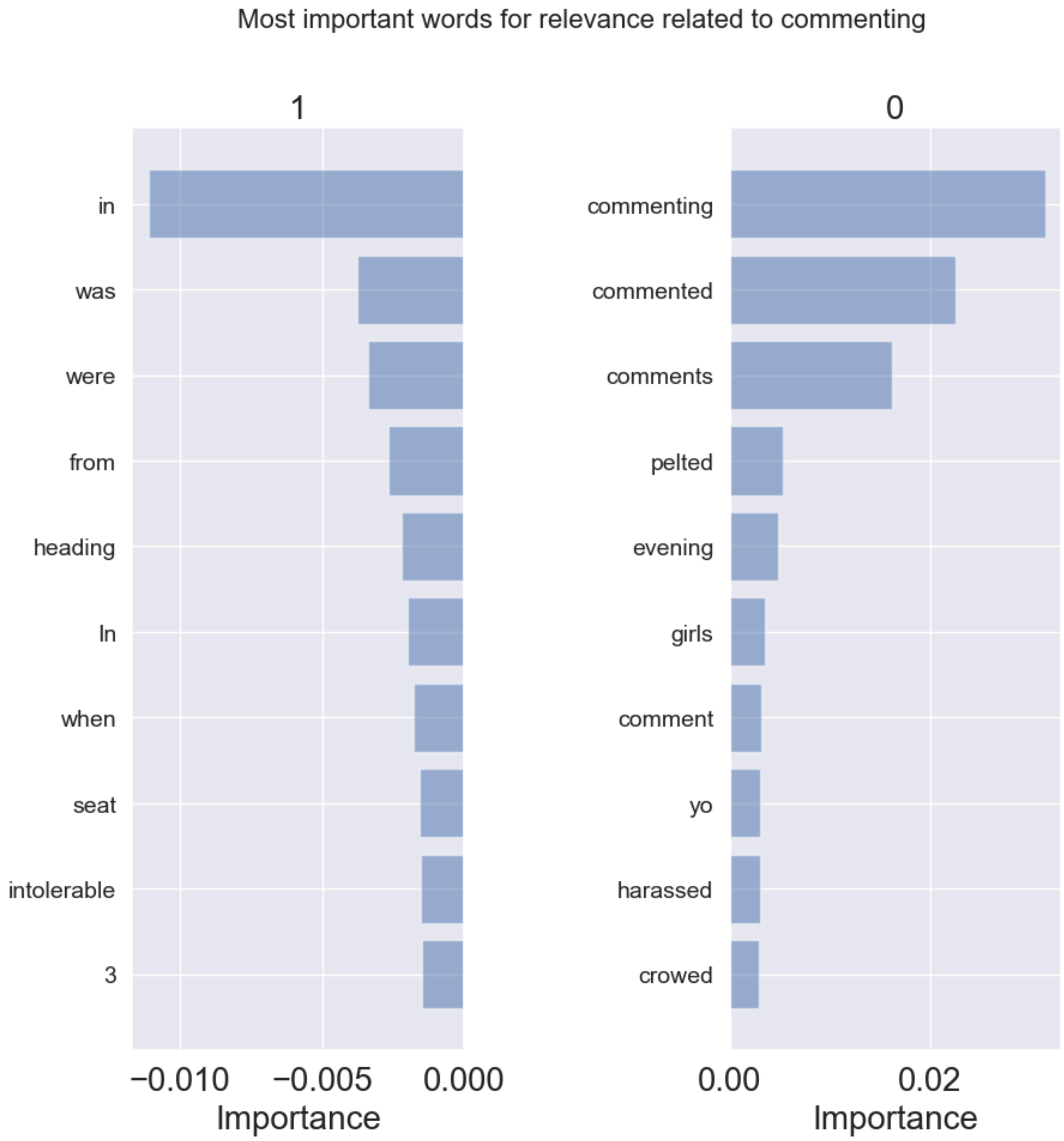


In [57]:
```
# Visualize the sorted contributions from all words
# First index is the class (Disaster)
# Second index is 0 for detractors, 1 for supporters
# Third is how many words we sample
top_words = sorted_contributions['Commenting']['supporters'][:10].index.tolist()
top_scores = sorted_contributions['Commenting']['supporters'][:10].tolist()
bottom_words = sorted_contributions['Commenting']['detractors'][:10].index.tolist()
bottom_scores = sorted_contributions['Commenting']['detractors'][:10].tolist()

plot_important_words(top_scores, top_words, bottom_scores, bottom_words, "Most important words for relevance related to commentir
```

[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]

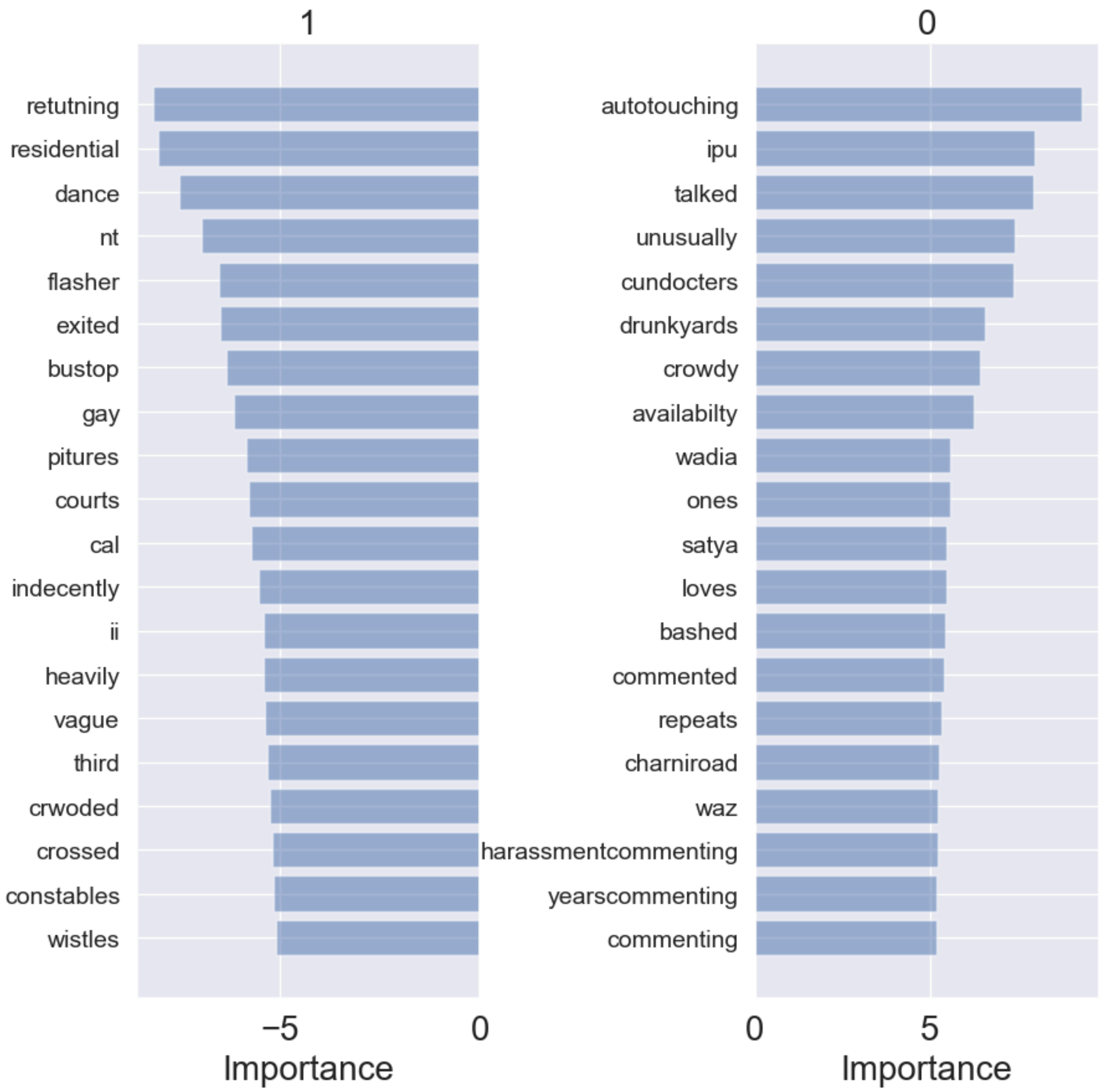


Most important words related to Ogling.

[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]

Most Important (highest coefficient) Words related to Ogling/Facial Expressions/Staring

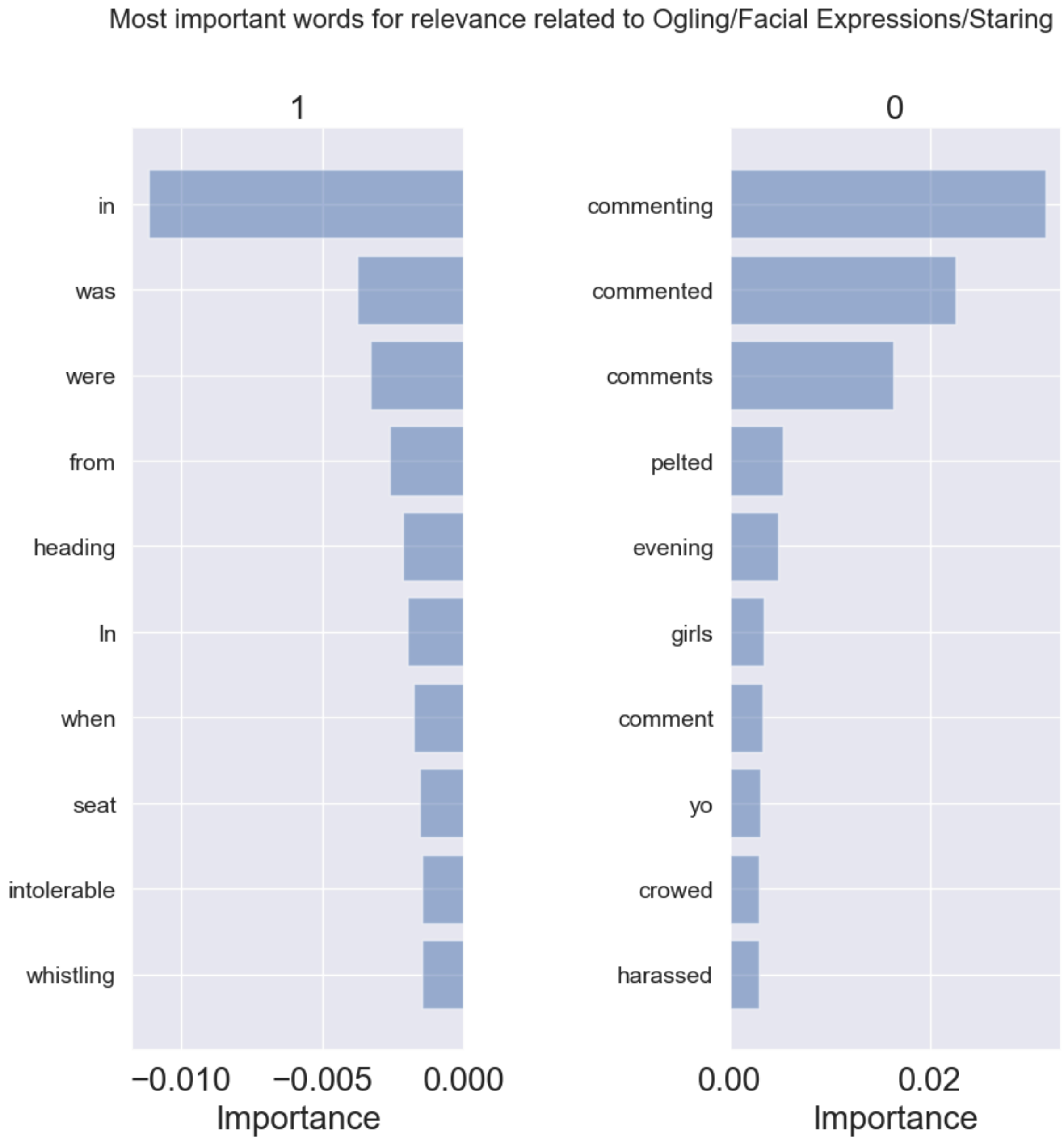


In [64]:
```
# Visualize the sorted contributions from all words
# First index is the class (Disaster)
# Second index is 0 for detractors, 1 for supporters
# Third is how many words we sample
top_words = sorted_contributions['Ogling/Facial Expressions/Staring']['supporters'][:10].index.tolist()
top_scores = sorted_contributions['Ogling/Facial Expressions/Staring']['supporters'][:10].tolist()
bottom_words = sorted_contributions['Ogling/Facial Expressions/Staring']['detractors'][:10].index.tolist()
bottom_scores = sorted_contributions['Ogling/Facial Expressions/Staring']['detractors'][:10].tolist()

plot_important_words(top_scores, top_words, bottom_scores, bottom_words, "Most important words for relevance related to Ogling/Fa
```

[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]

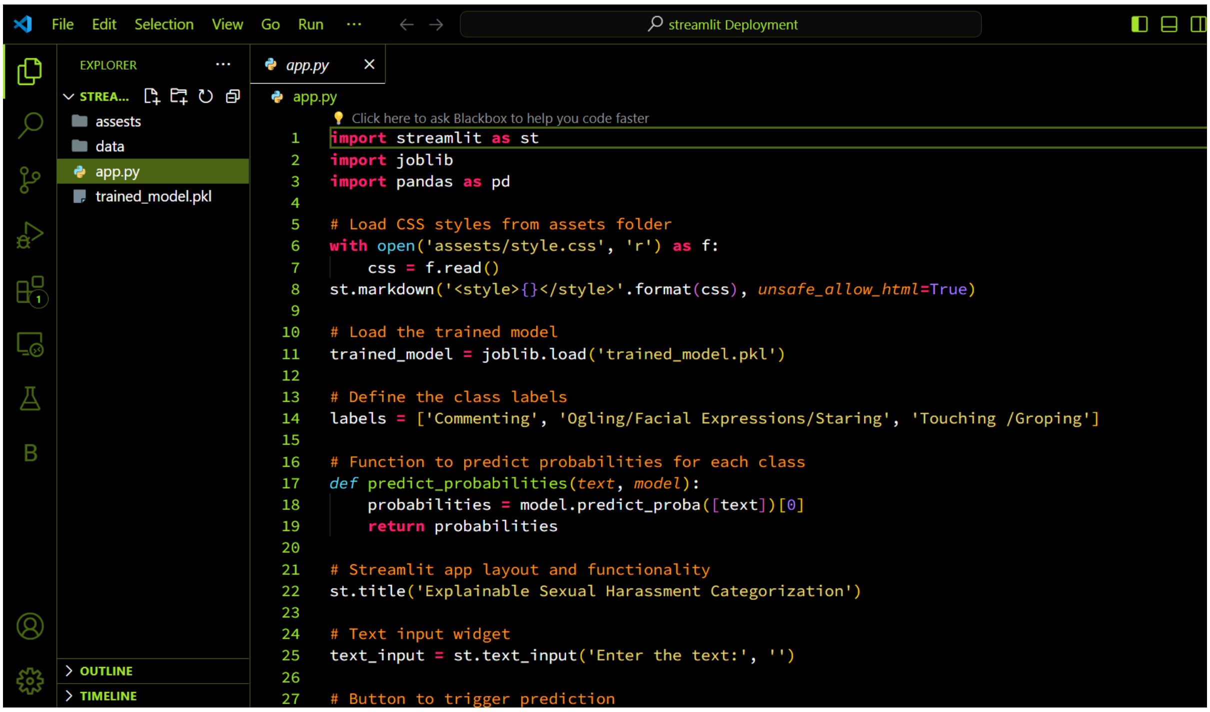


MODEL DEPLOYMENT

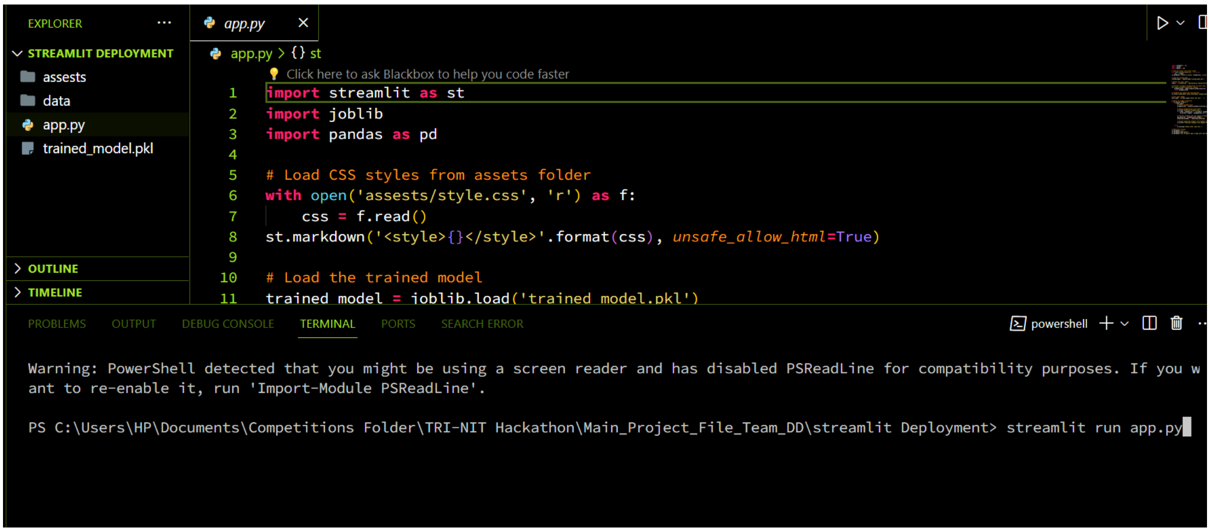
We have used **Streamlit** for the deployment of our application.
First, we converted our model into a pickle file and then loaded it into the **app.py** of streamlit application.

[GitHub-Project-ML02-Deception_Detectors]

[Demo-Video of our project.]



By the help of *"streamlit run app.py"* command, we have started our deployment server.



Our Deployed website(localhost)

[GitHub-Project-ML02-Deception_Detectors]

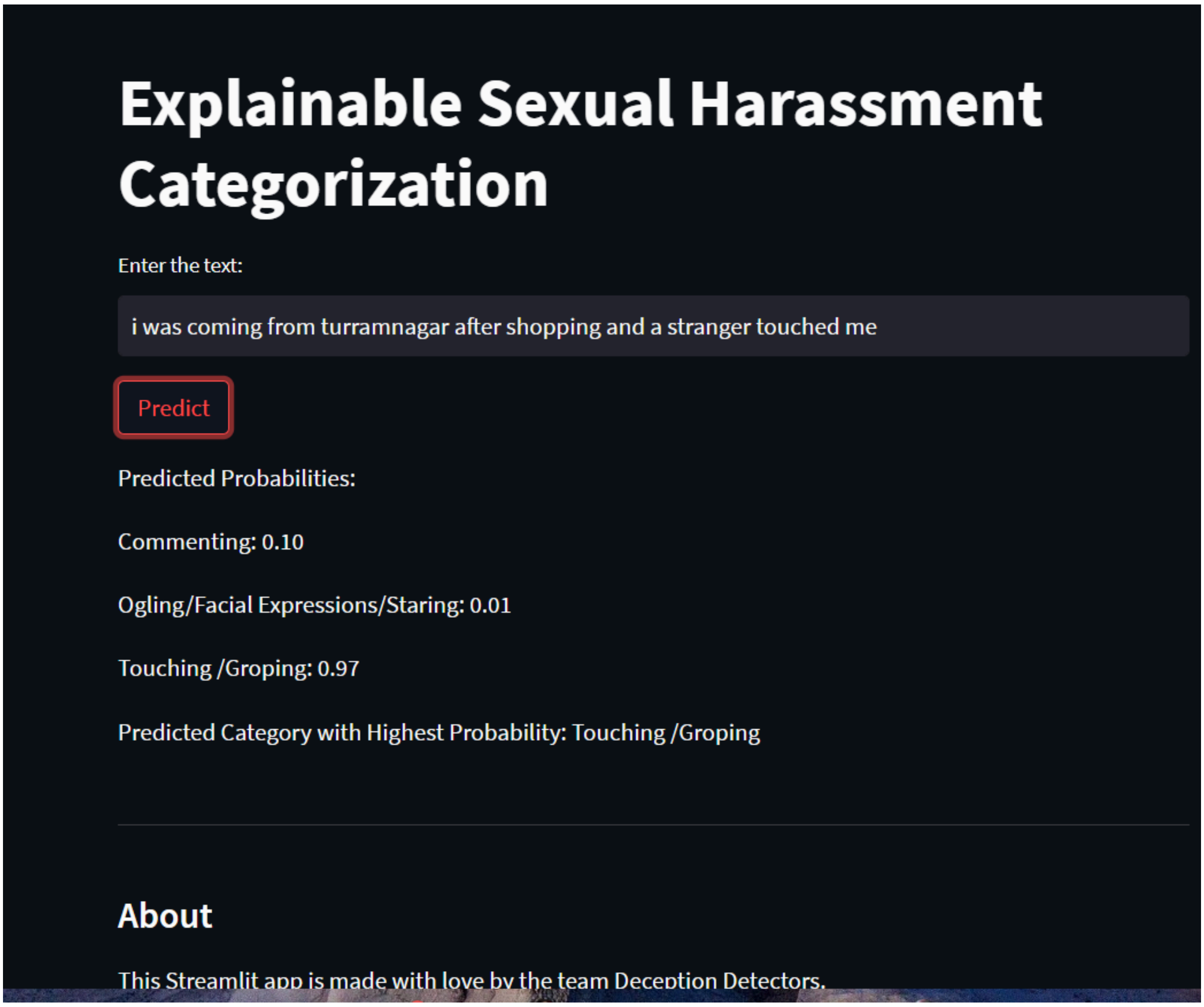
[Demo-Video of our project.]

Testing the model on user input

For other categories

GitHub-Project-ML02-Deception_Detectors

Demo-Video of our project.



We can see that our ML model is performing well on the testing data as well as the random user inputs.