Human Stress Detection Using ML

PRESENTED BY

LAIBA ATIQ - 372187

ABDULLAH TAHIR - 393056

Agenda

1. Project Overview
2. Methodology
3. Results & Metrics
4. Challenges
5. Way Forward
6. Conclusion

About 75%

Suffer From Stress

American Psychological Association, 2019

PROJECT OVERVIEW

01 Project Overview

This project aims to develop a stress detection system to identify stressed employees and promote well-being within the organization. By leveraging machine learning and employee self-reported data, the system predicts stress levels and can be used to offer targeted support mechanisms.

01 Stress dataset with numerous features

02 Interactive Interface to ensure usability

03 Improves employee well-being

APPLICATIONS

01. Proactive Workplace Support

02. Early Intervention for At-Risk Employees

03. Work Environment Analysis

PROJECT USERS

Lay Man/ Patients

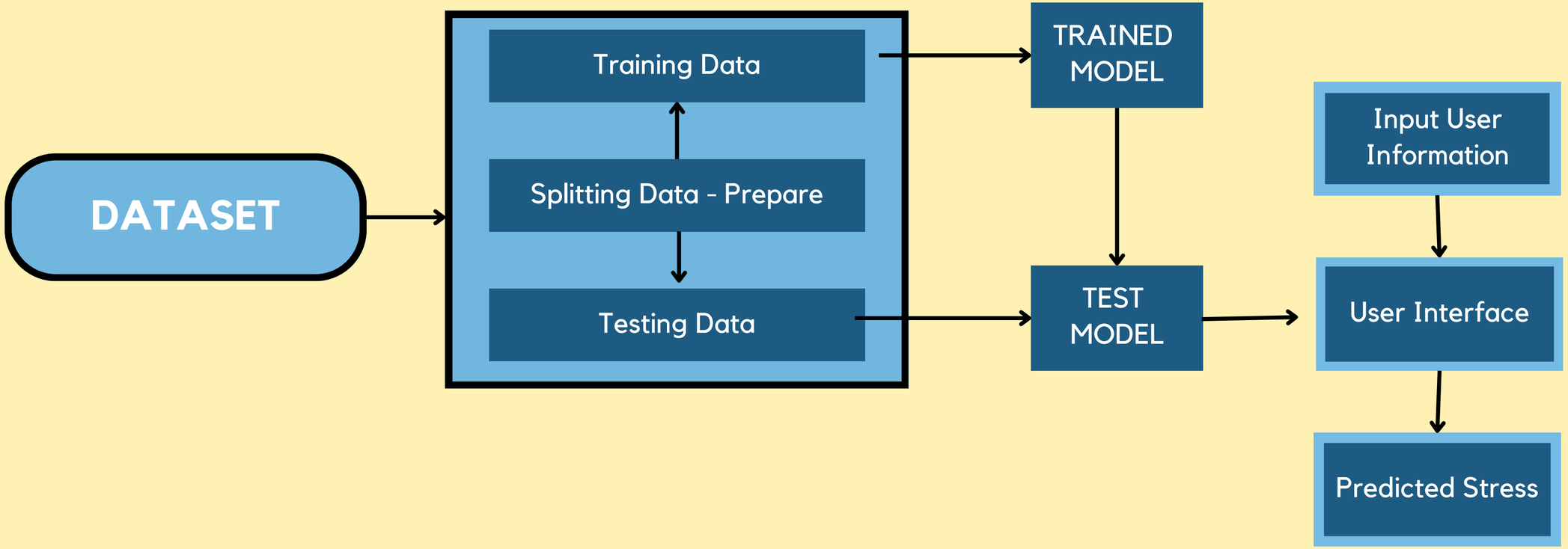
Mental Health Professionals

Offices

Research Institutions

02
METHODOLOGY

ML Pipeline


DATASET
Training Data
Splitting Data - Prepare
Testing Data
TRAINED MODEL
TEST MODEL
Input User Information
User Interface
Predicted Stress

CLASSIFYING MODELS

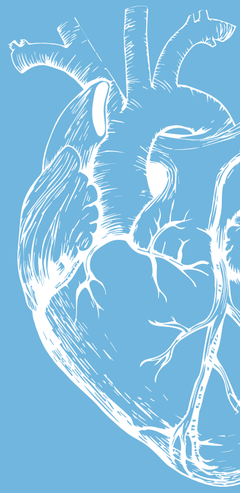
Bernoulli Naive Bayes

Logistic Regression

Multinominal Naive Bayes

SVM

Random Forest



PROGRAM WORKING

Input

ML Pipeline

Output

Text Input - Email/Survey etc

Humam Stress Prediction

OUR EQUIPEMNTS?

PYTHON

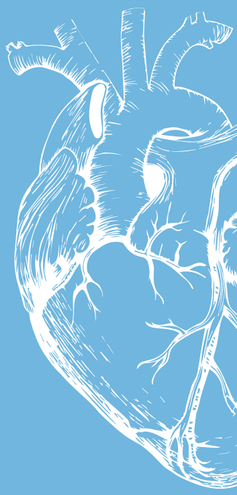
SCIKIT-LEARN

JOBLIB

Natural Language Toolkit (NLTK)

Regular Expressions (RE)

STREAMLIT



DATA SET

The dataset used for this stress prediction model consists of text excerpts sourced from Reddit posts across various subreddits e.g ptsd, relationships, etc.

KEY FEATURES

- subreddit: (Nominal)
- post_id: (Nominal)
- sentence_range: (Numerical)
- text: (Text)
- id: (Numerical)
- label: (Binary)
- confidence: (Numerical)
- social_timestamp: (Numerical)
- social_karma: (Numerical)
- syntax_ari: (Numerical)
- Lexical analysis columns: (Numerical)

DATA PREPARATION

```python
def clean(text):
    try:
        # Convert text to lowercase
        text = str(text).lower()
        # Remove brackets
        text = re.sub('[][)(]',' ', text)
        # Remove URLs
        text = [word for word in text.split() if not urlparse(word).scheme]
        text = ' '.join(text)
        # Remove escape characters
        text = re.sub(r'\@\w+','', text)
        # Remove HTML tags
        text = re.sub(re.compile("<.*?>"), '', text)
        # Remove non-alphanumeric characters
        text = re.sub("[^A-Za-z0-9]", ' ', text)
        # Tokenize text
        tokens = word_tokenize(text)
        # Remove stopwords
        tokens = [word for word in tokens if word not in stop_words]
        # Stem tokens
        tokens = [ps.stem(word) for word in tokens]

        # Join tokens back into text
        text = ' '.join(tokens)

        return text
    except Exception as ex:
        print(sent,"\n")
        print("Error ",ex)
```

```python
data["label"] = data["label"].map({0: "No Stress", 1: "Stress"})
data = data[["text", "label"]]
print(data.head())
```

MODEL DESIGN

Voting Classifier

A voting classifier was developed to aggregate the predictions from five different models.

1.

Model Fine-Tuning

Hyperparameters of all individual models were fine-tuned using grid search.

This process aimed to optimize each model for maximum predictive performance.

Accuracy Improvement

- **Initial accuracy**: 71%
- **Improved accuracy**: 77%

MODEL FINE TUNING

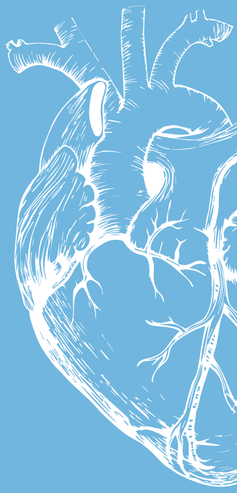
```python
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Define the parameter grid
param_grid = {
    'C': [0.1, 0.5, 1.0],  # Regularization parameter
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],  # Optimization algorithm
    'max_iter': [100, 200, 300]  # Maximum number of iterations
}

# Initialize the logistic regression classifier
logistic_reg = LogisticRegression(random_state=42)

# Perform grid search with 5-fold cross-validation
grid_search = GridSearchCV(logistic_reg, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(xtrain, ytrain)

# Get the best parameters and the best score
log_best_params = grid_search.best_params_
log_best_score = grid_search.best_score_

print("Best parameters:", log_best_params)
print("Best cross-validation score:", log_best_score)

# Get the best model
log_best_model = grid_search.best_estimator_

# Evaluate the best model on training and test sets
log_train_score = log_best_model.score(xtrain, ytrain)
log_test_score =log_best_model.score(xtest, ytest)

print("Train set accuracy with best parameters:", log_train_score)
print("Test set accuracy with best parameters:", log_test_score)
```

MODEL TRAINING

```
estimators=[
        ("BNB", BernoulliNB(alpha=1)),
        ("Logistics Regression", log_best_model),
        ("SVM", svm_best_model),
        ("Random Forest", RFC_best_model),
        ("MultinomialNB", MNB_best_model)]

# Use voting classifier with top three models
voting_clf = VotingClassifier(estimators, voting='hard')
voting_clf.fit(xtrain, ytrain)

# Predict using voting classifier
voting_pred = voting_clf.predict(xtest)

# Evaluate accuracy of voting classifier
voting_accuracy = accuracy_score(ytest, voting_pred)
print("Accuracy of Voting Classifier :", voting_accuracy)
```

```
Accuracy of Voting Classifier : 0.7676056338028169
```

MODEL EVALUATION



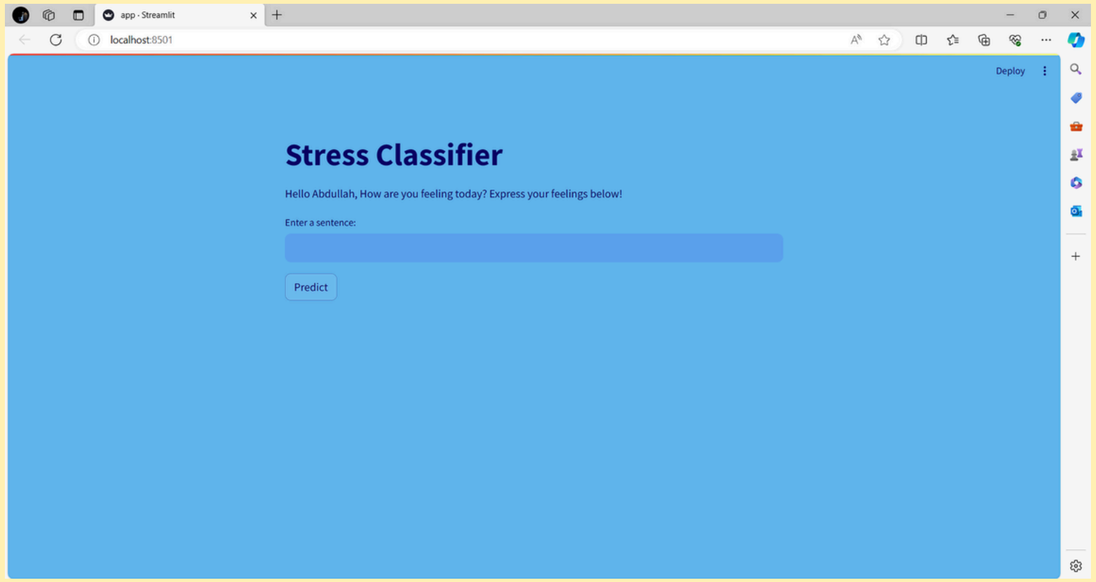
💡 Click here to ask Blackbox to help you code faster

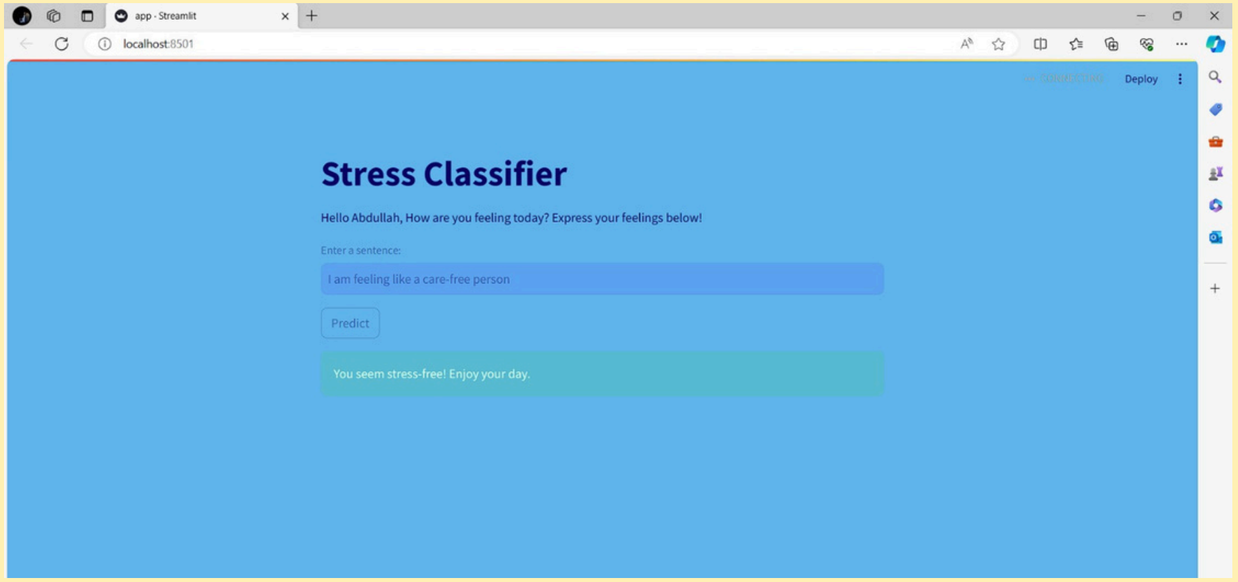
```
y_pred = voting_clf.predict(xtest)  # Predictions on the test set

cm = confusion_matrix(ytest, y_pred)
print(cm)

report = classification_report(ytest, y_pred)
print(report)
```

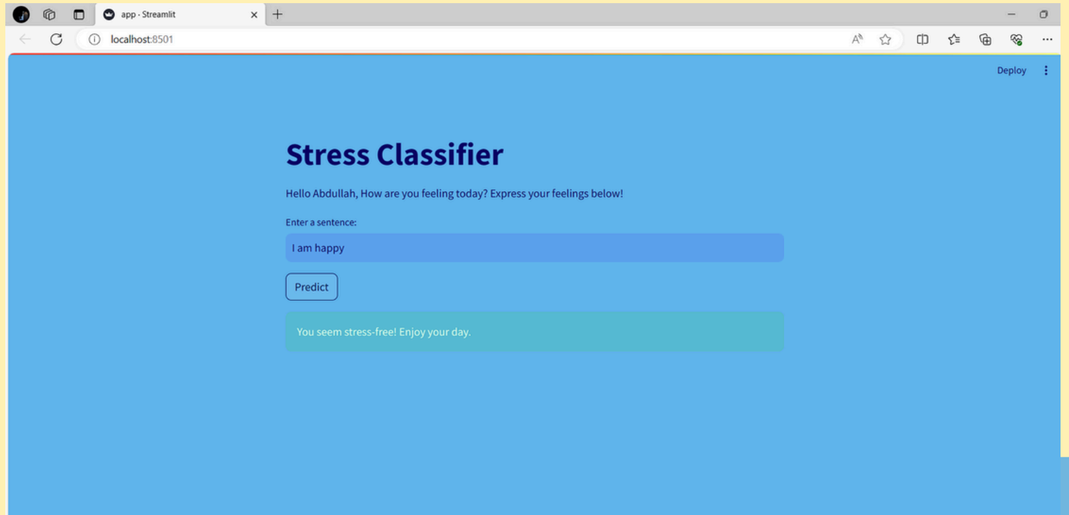
✓ 0.4s

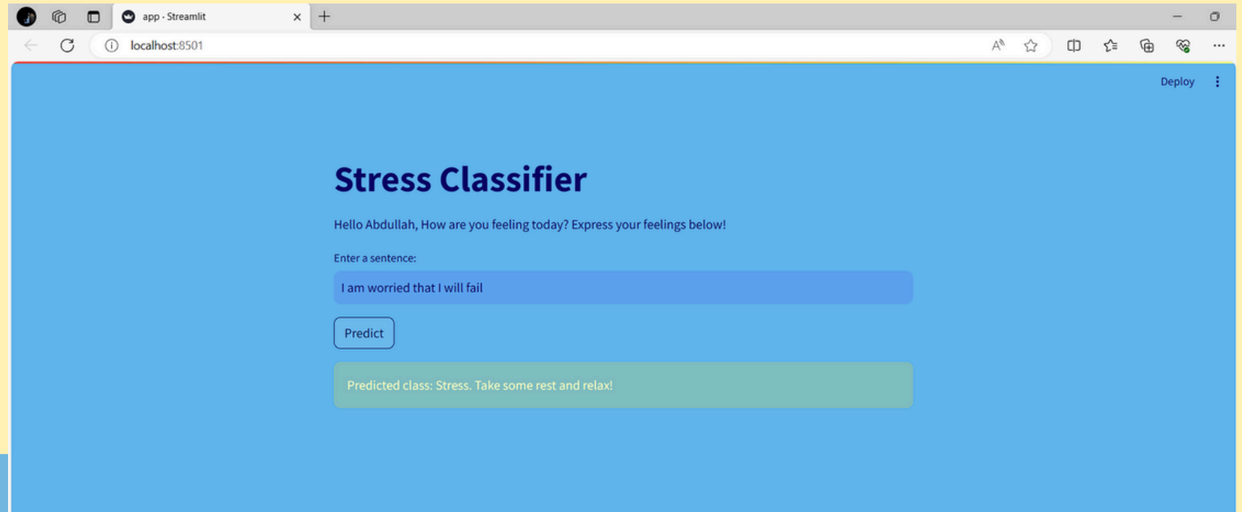
```
[[189  74]
 [ 58 247]]
              precision    recall  f1-score   support

   No Stress       0.77      0.72      0.74       263
      Stress       0.77      0.81      0.79       305

    accuracy                           0.77       568
   macro avg       0.77      0.76      0.77       568
weighted avg       0.77      0.77      0.77       568
```

MODEL PREDICTIONS



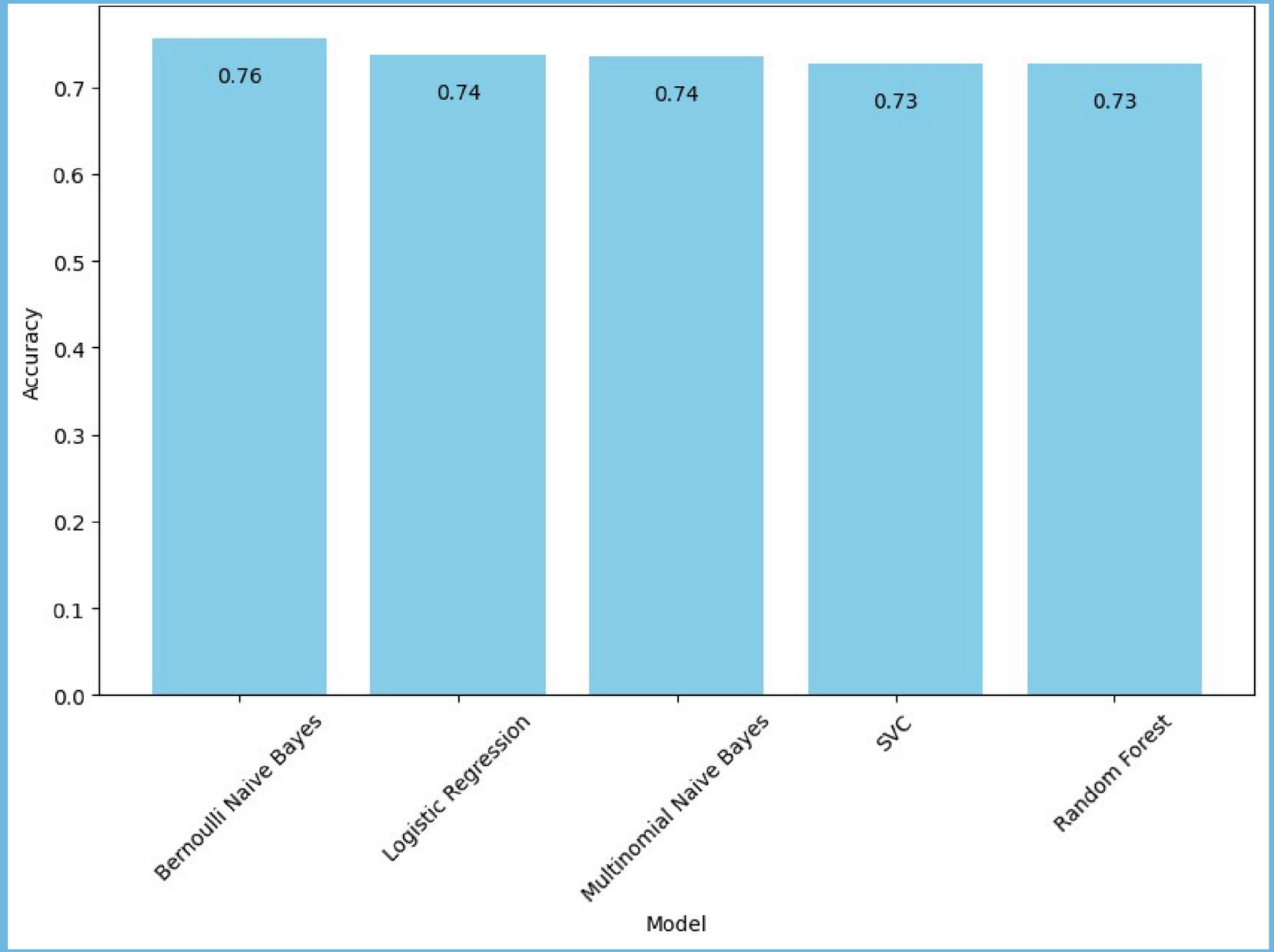


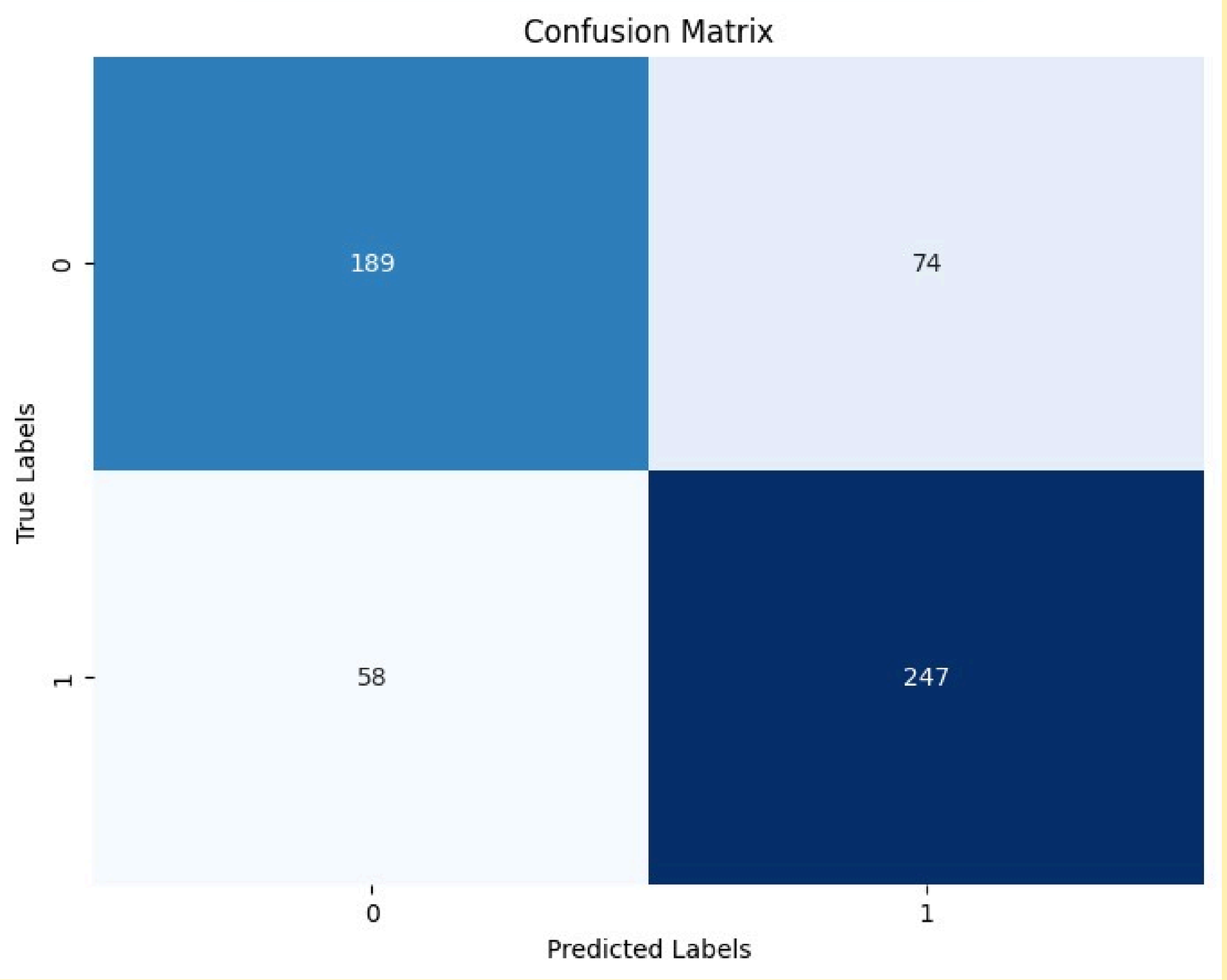




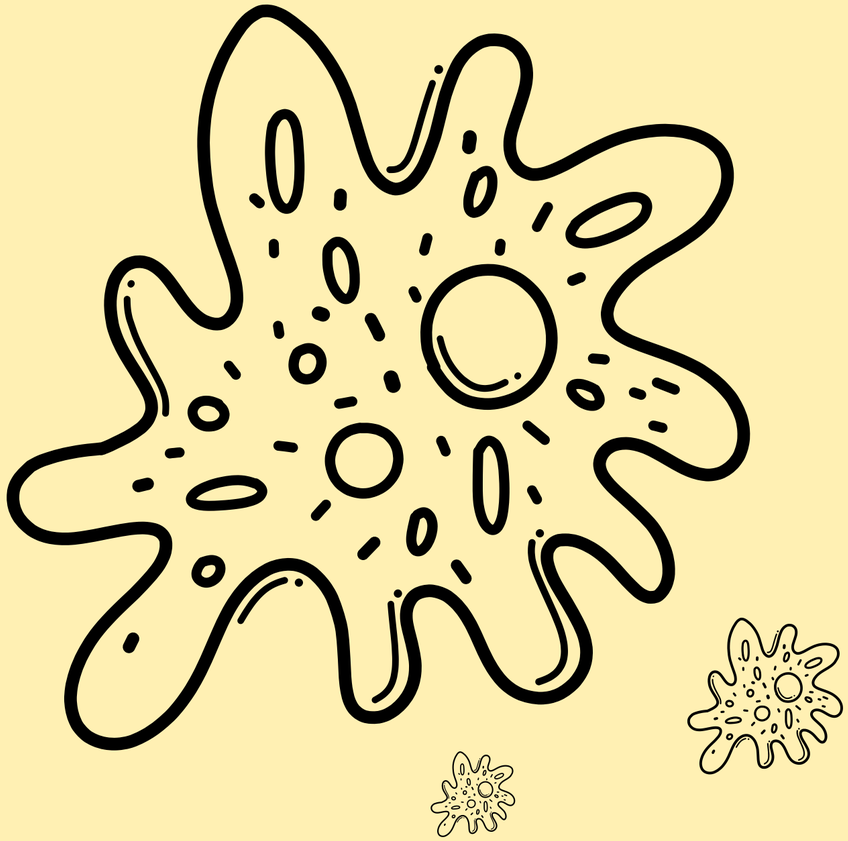
03
RESULTS & METRICS

Accuracy of classifiers



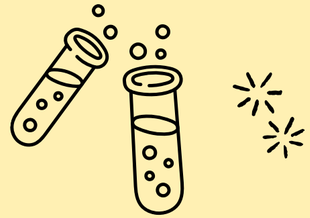

Confusion Matrix
189
74
58
247
0
1
True Labels
0
1
Predicted Labels

PRESENTED BY
OLIVIA WILSON



RESULT INTERPRETATION

- The implementation of a voting classifier, along with fine-tuning via grid search, resulted in a substantial improvement in accuracy **UPTO 77%.**
- This approach demonstrates the effectiveness of ensemble learning and hyperparameter optimization techniques in enhancing predictive models.



04
CHALLENGES

Preprocessing Refinement

Optimizing the text preprocessing stage proved challenging, particularly in deciding between stemming and lemmatization techniques.

Dataset Complexity

The dataset initially presented challenges due to its numerous columns. Understanding the nature and potential impact of each column on the stress prediction task required careful analysis.

Iterative Improvement

Our initial model achieved an accuracy of 71%. Through meticulous fine-tuning, voting classifier ensemble, we were able to significantly boost the accuracy to 77%.

05
WAY FORWARD

1. **Delve into RNNs:** Explore how architectures like LSTM networks can uncover subtle language nuances, potentially improving stress detection accuracy.

2. **Evaluate Resource Trade-offs:** Assess if the benefits of deep learning outweigh the increased computational demands and dataset sizes typically associated with these models.

3. **Enhance User Interaction:** Develop a more user-friendly interface for real-time stress assessment, enhancing system usability and engagement.

06
CONCLUSION

Our stress prediction tool, leveraging machine learning and text analysis techniques, offers a promising approach to proactively assess and address stress in individuals. With an accuracy of 77% through classification and meticulous hyperparameter tuning, we're poised to make meaningful strides in supporting mental well-being.

Thank You