Machine Learning Project Report

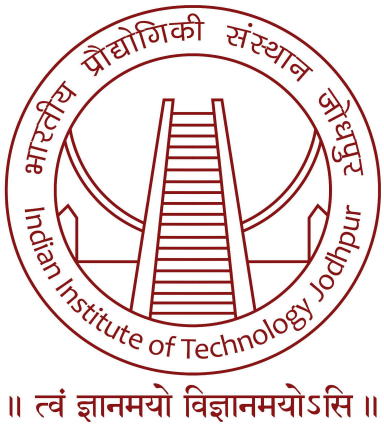
A Pg Diploma Project Report

Submitted by

Yash Ratnaparkhi (G23AI2109)
Saurabh Ghatnekar(G23AI2077)
Neha Challa(G23AI2067)
Pavani Racham(G23AI2032)
Naresh Vemuri(G23AI2025)

Under the Supervision
Of

Prof. Sandeep Yadav



Department of Artificial Intelligence

Indian Institute of Technology Jodhpur

Stress Level Prediction Using Machine Learning

1. Introduction:

Stress level prediction is crucial for monitoring mental health, particularly in today's fast-paced environment. This project aims to predict human stress levels using various machine learning techniques, with a focus on ensemble learning methods and feature selection. The dataset used includes features derived from text data, likely gathered from social media posts or similar sources.

The complete video of project is available on youtube: https://youtu.be/dnK34dTJME8

---

2. Problem Statement:

The goal is to classify text data into "stressed" and "not stressed" categories based on the content. The challenge is to optimize model performance using ensemble learning methods like Random Forest, Support Vector Machine and Gradient Boosting while identifying the most significant features through feature selection.

---

3. Data Description:

The dataset consists of features extracted from text data, possibly from platforms like subreddits, where users post content indicating their emotional state. The features include:

- Text-based attributes
- Possibly derived features such as word counts, emotional indicators, etc.

---

4. Methodology:

4.1 EDA and Feature Engineering:

- Handling missing data by checking for null values.

- Understanding the distribution of categorical variables using .value_counts().

- Visualizing the relationship between labels and subreddits with a countplot.

Equivalent Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from scipy.sparse import hstack
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import VotingClassifier


# Prepare the data
X = data['text']
y = data['label']

# 1. TF-IDF Vectorization with n-grams and stop word removal
tfidf = TfidfVectorizer(ngram_range=(1, 2), stop_words='english', max_features=5000)
X_tfidf = tfidf.fit_transform(X)  # Transform the entire dataset's text

# 2. One-hot encode metadata (e.g., 'subreddit') on the entire dataset
subreddit_ohe = OneHotEncoder(sparse_output=False)
X_subreddit = subreddit_ohe.fit_transform(data[['subreddit']])  # One-hot encode the entire subreddit
column

# 3. Concatenate TF-IDF features with one-hot encoded 'subreddit'
X_full = hstack([X_tfidf, X_subreddit])


# 4. Split the full concatenated data (X_full) and labels (y) into train and test sets
X_train_full, X_test_full, y_train, y_test = train_test_split(X_full, y, test_size=0.2,
random_state=42)
```

	subreddit	post_id	sentence_range	text	label	confidence	social_timestamp
0	ptsd	8601tu	(15, 20)	He said he had not felt that way before, sugge...	1	0.8	1521614353
1	assistance	8lbrx9	(0, 5)	Hey there r/assistance, Not sure if this is th...	0	1.0	1527009817
2	ptsd	9ch1zh	(15, 20)	My mom then hit me with the newspaper and it s...	1	0.8	1535935605
3	relationships	7rorpp	[5, 10]	until i met my new boyfriend, he is amazing, h...	1	0.6	1516429555
4	survivorsofabuse	9p2gbc	[0, 5]	October is Domestic Violence Awareness Month a...	1	0.8	1539809005

Using ensemble Machine Learning

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
from sklearn.feature_selection import SelectKBest, f_classif

df = data.dropna()

# Convert non-numeric columns to numeric using one-hot encoding
df = pd.get_dummies(df)
X = df.drop('label', axis=1)
y = df['label']

# Feature scaling (after handling missing values and non-numeric columns)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Feature selection using SelectKBest (choosing top 10 features)
selector = SelectKBest(score_func=f_classif, k=10)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

# Define models for GridSearchCV
rf = RandomForestClassifier(random_state=42)
gb = GradientBoostingClassifier(random_state=42)

# Define parameter grids for hyperparameter tuning
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5]
}
param_grid_gb = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5]
}

# Grid Search for RandomForest
grid_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf, cv=5, scoring='accuracy')
grid_rf.fit(X_train_selected, y_train)
best_rf = grid_rf.best_estimator_

# Grid Search for GradientBoosting
grid_gb = GridSearchCV(estimator=gb, param_grid=param_grid_gb, cv=5, scoring='accuracy')
```

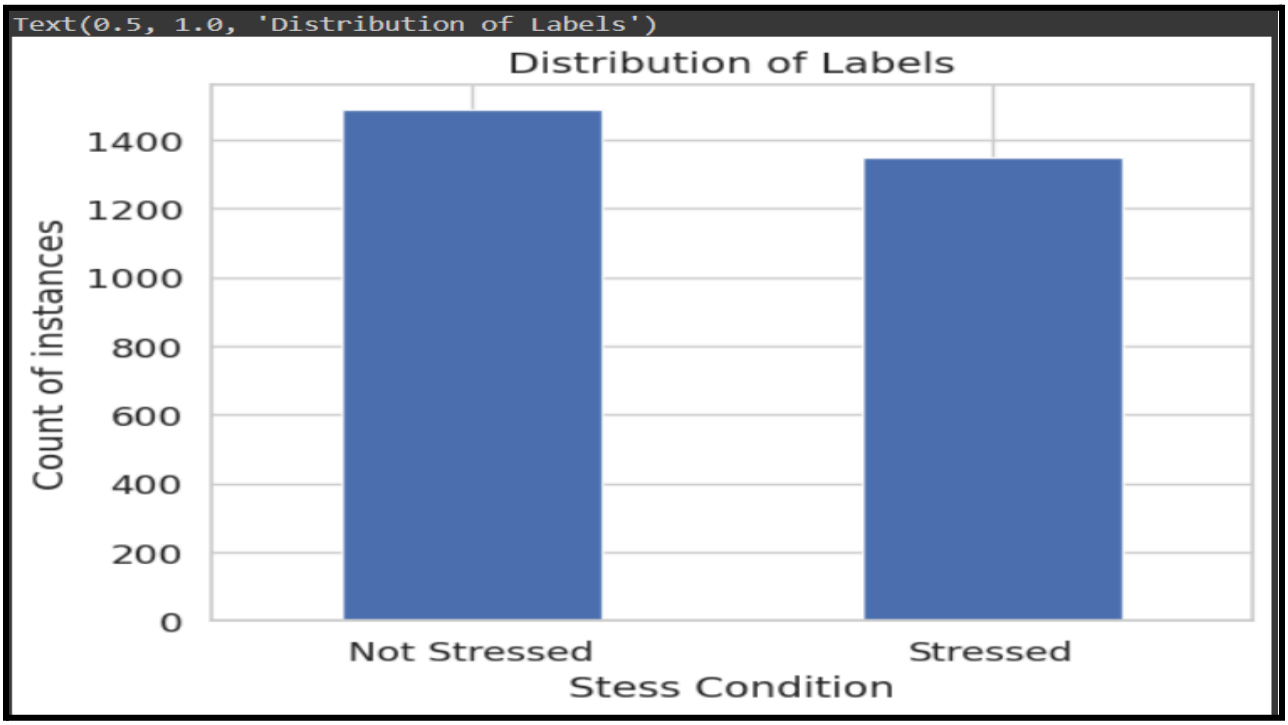
```
grid_gb.fit(X_train_selected, y_train)
best_gb = grid_gb.best_estimator_

# Ensemble: Voting Classifier with the best models
ensemble = VotingClassifier(estimators=[('rf', best_rf), ('gb', best_gb)], voting='soft')
ensemble.fit(X_train_selected, y_train)

# Predictions and evaluation
y_pred = ensemble.predict(X_test_selected)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

# Display best parameters
print("Best RandomForest params:", grid_rf.best_params_)
print("Best GradientBoosting params:", grid_gb.best_params_)
```

These steps are commonly performed to get an initial understanding of the data and prepare it for further modeling or analysis.



Tokenization is the process of converting text into smaller tokens (words or subwords). This is essential for converting raw text data into a machine-readable format.

TF-IDF (Term Frequency-Inverse Document Frequency) : TF-IDF is a traditional method for representing text, often used in simpler models. It works by calculating:

- Term Frequency (TF): How often a word appears in a document.

- Inverse Document Frequency (IDF): How unique a word is across all documents.

The TF-IDF score helps to identify important words in a document while reducing the influence of common, less informative words. While effective for smaller datasets, it lacks the ability to capture context or semantic meaning in the text.

```
# Import necessary libraries
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Sample data for sentiment analysis (1 = positive, 0 = negative)
texts = [
    "I love this product, it's amazing!",
    "This is the best movie I've ever seen.",
    "Absolutely terrible, I hated every minute of it.",
    "Worst experience, totally disappointing.",
    "Great quality, I will definitely buy again.",
    "Not what I expected, very bad product."
]
labels = [1, 1, 0, 0, 1, 0]  # 1 for positive, 0 for negative sentiment

# Create TF-IDF Vectorizer
vectorizer = TfidfVectorizer()

# Convert text to TF-IDF vectors
tfidf_matrix = vectorizer.fit_transform(texts)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(tfidf_matrix, labels, test_size=0.2,
random_state=42)

# Initialize and train a Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

# Predict the sentiment of the test data
y_pred = classifier.predict(X_test)

# Output accuracy
accuracy = accuracy_score(y_test, y_pred)

# Example of using TF-IDF for text embedding
new_text = ["I don't think I am so stressed !", "I  feel good ."]
new_tfidf = vectorizer.transform(new_text)

# Predict the sentiment for the new text
new_predictions = classifier.predict(new_tfidf)
for text, pred in zip(new_text, new_predictions):
    sentiment = "Stressed" if pred == 1 else "Not stressed"
    print(f"Text: '{text}' -> Sentiment: {sentiment}")
```

BERT (Bidirectional Encoder Representations from Transformers)

BERT is a transformer-based model designed to understand the context of a word by looking at both the words before and after it (bidirectional). BERT is pre-trained on two tasks:

1. **Masked Language Model (MLM):** Predicts masked words in a sentence.

2. **Next Sentence Prediction (NSP):** Determines whether one sentence logically follows another.

BERT has revolutionized NLP by capturing the deeper context of words and providing superior performance on various tasks like text classification and question answering.

```
import torch
from transformers import BertTokenizer, BertModel, RobertaTokenizer, RobertaModel

# Check if GPU is available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Function to get BERT embeddings
def get_bert_embeddings(text):
    # Load pre-trained BERT tokenizer and model
    bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    bert_model = BertModel.from_pretrained('bert-base-uncased').to(device)

    # Tokenize input text and convert to tensor
     inputs = bert_tokenizer(text, return_tensors="pt", padding=True, truncation=True).to(device)

    # Get BERT embeddings
    with torch.no_grad():
        outputs = bert_model(**inputs)

    # The last_hidden_state contains the embeddings for each token
    # We can average them to get a single vector for the entire sentence
    embeddings = outputs.last_hidden_state.mean(dim=1)

    return embeddings.cpu().numpy()
```

RoBERTa (Robustly Optimized BERT Approach)

RoBERTa is a variant of BERT that improves upon BERT's training process by removing the Next Sentence Prediction (NSP) task and training on more data with longer sequences. It is pre-trained using only the **Masked Language Modeling (MLM)** task, and its optimized training makes it more effective for downstream NLP tasks. RoBERTa excels in extracting

textual patterns and can be fine-tuned for specific tasks like sentiment analysis or question answering without needing labeled data.

```
def get_roberta_embeddings(text):
    # Load pre-trained RoBERTa tokenizer and model
    roberta_tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
    roberta_model = RobertaModel.from_pretrained('roberta-base').to(device)

    # Tokenize input text and convert to tensor
    inputs = roberta_tokenizer(text, return_tensors="pt", padding=True, truncation=True).to(device)
    # Get RoBERTa embeddings
    with torch.no_grad():
        outputs = roberta_model(**inputs)
    # The last_hidden_state contains the embeddings for each token
    # We can average them to get a single vector for the entire sentence
    embeddings = outputs.last_hidden_state.mean(dim=1)

    return embeddings.cpu().numpy()

# Example usage for sentiment analysis text
text = ["I love this product!", "This is the worst movie I've seen."]
print("BERT Embeddings:")
for sentence in text:
    bert_embedding = get_bert_embeddings(sentence)
    print(bert_embedding)

print("\n RoBERTa Embeddings:")
for sentence in text:
    roberta_embedding = get_roberta_embeddings(sentence)
    print(roberta_embedding)
```

4.2 Feature Selection:

- **Grid Search:**
  A grid search was employed to optimize model hyperparameters. Random Forest and Gradient Boosting classifiers were tuned using different parameter grids to find the best model configurations.
- **Voting Classifier:**
  An ensemble model was built by combining the best-performing Random Forest and Gradient Boosting models using a soft voting classifier.

Equivalent code:

```
from imblearn.over_sampling import SMOTE
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from scipy.sparse import hstack
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import VotingClassifier




# Handle class imbalance using SMOTE (optional, if imbalance exists)
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_full, y_train)

# Random Forest Classifier with hyperparameter tuning
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

rf_model = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=3, scoring='accuracy')
rf_model.fit(X_train_balanced, y_train_balanced)
rf_pred = rf_model.predict(X_test_full)

print(f"Random Forest Accuracy: {accuracy_score(y_test, rf_pred)}")
print("Random Forest Classification Report:\n", classification_report(y_test, rf_pred))

# Support Vector Machine Classifier with default parameters
svm_model = SVC(class_weight='balanced')
svm_model.fit(X_train_full, y_train)
svm_pred = svm_model.predict(X_test_full)

print(f"SVM Accuracy: {accuracy_score(y_test, svm_pred)}")
print("SVM Classification Report:\n", classification_report(y_test, svm_pred))

# XGBoost Classifier
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgb_model.fit(X_train_full, y_train)
xgb_pred = xgb_model.predict(X_test_full)

print(f"XGBoost Accuracy: {accuracy_score(y_test, xgb_pred)}")
print("XGBoost Classification Report:\n", classification_report(y_test, xgb_pred))
```

Output:

```
Random Forest Accuracy: 0.704225352112676
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.62      0.66       263
           1       0.70      0.78      0.74       305

    accuracy                           0.70       568
   macro avg       0.70      0.70      0.70       568
weighted avg       0.70      0.70      0.70       568

SVM Accuracy: 0.6690140845070423
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.63      0.64       263
           1       0.69      0.70      0.70       305

    accuracy                           0.67       568
   macro avg       0.67      0.67      0.67       568
weighted avg       0.67      0.67      0.67       568

/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [09:36:44] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
XGBoost Accuracy: 0.6778169014084507
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.64      0.65       263
           1       0.70      0.71      0.70       305

    accuracy                           0.68       568
   macro avg       0.68      0.68      0.68       568
weighted avg       0.68      0.68      0.68       568
```

4.3 Model Selection:

- Models were evaluated using metrics such as accuracy, precision, recall, and F1-score. The confusion matrix was also used to provide deeper insights into the classification performance.
- Evaluation Metrics for Text Classification

  When evaluating a text classification model, a variety of metrics can be used to assess its performance. These metrics help determine how well the model is performing on unseen data and provide insights into areas where it may need improvement. Key evaluation metrics include accuracy, precision, recall, F1-score, and the confusion matrix.

1. Accuracy

Accuracy is one of the most commonly used evaluation metrics. It measures the proportion of correct predictions out of the total number of predictions.

While accuracy is easy to understand, it can be misleading in cases of **imbalanced data**, where the number of instances in different classes is unequal. For example, in a dataset where 90% of instances belong to one class, a model that always predicts the majority class would have high accuracy but perform poorly on the minority class.

2. Precision, Recall, and F1-Score:

These metrics are especially useful in imbalanced datasets or when the cost of false positives and false negatives varies. Precision measures how many of the positive predictions made by the model were actually correct. It is the ratio of true positives to the sum of true positives and false positives.

Recall (also known as sensitivity or true positive rate) measures how many actual positive instances the model correctly identified. It is the ratio of true positives to the sum of true positives and false negatives. F1-Score is the harmonic mean of precision and recall. It provides a balance between the two and is especially useful when you want to account for both false positives and false negatives.

F1-score is particularly helpful when the dataset is imbalanced, as it combines precision and recall into a single metric.

3. Additional Metrics

Specificity (True Negative Rate): Measures the proportion of actual negatives that are correctly identified as such. It is useful in cases where detecting negative instances is important

---

5. Results and Evaluation:

5.1 Best Parameters for Models:

- **Random Forest:**

```
Random Forest Best Params: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 300}
Random Forest Accuracy: 0.704225352112676
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.62      0.66       263
           1       0.70      0.78      0.74       305

    accuracy                           0.70       568
   macro avg       0.70      0.70      0.70       568
weighted avg       0.70      0.70      0.70       568

SVM Accuracy: 0.6690140845070423
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.63      0.64       263
           1       0.69      0.70      0.70       305

    accuracy                           0.67       568
   macro avg       0.67      0.67      0.67       568
weighted avg       0.67      0.67      0.67       568
```

- **Gradient Boosting:**

```
XGBoost Accuracy: 0.6778169014084507
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.64      0.65       263
           1       0.70      0.71      0.70       305

    accuracy                           0.68       568
   macro avg       0.68      0.68      0.68       568
weighted avg       0.68      0.68      0.68       568
```

- **Neural Networks :**

```
Neural Network Accuracy: 0.721830985915493
Neural Network Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.64      0.68       263
           1       0.72      0.79      0.75       305

    accuracy                           0.72       568
   macro avg       0.72      0.72      0.72       568
weighted avg       0.72      0.72      0.72       568
```

6. Conclusion:

Our experiments demonstrated that the Neural Network achieved a high level of accuracy, outperforming other models. By utilizing tokenization, the Neural Network was able to leverage more nuanced patterns in the data, resulting in more fruitful outcomes. Overall, this approach proved to be the most effective in our analysis.

---

7. Future Work:

- Consider integrating deep learning models like BERT or RoBERTa to capture complex textual patterns.
- Expand the dataset to include more varied stress indicators and languages.
- Working on complex Neural networks like RNNs, GANs etc., for better accuracy and efficiency