Detecting ADHD-Related Text Using Classical Machine Learning and Deep Learning Models

Introduction

❑ What is **ADHD**?

ADHD (Attention-Deficit/Hyperactivity Disorder) is a neurodevelopmental disorder that affects attention, impulsivity, and executive functioning.

❑ Why Early Detection?

- Early intervention improves long-term outcomes.
- Many adults remain undiagnosed due to lack of access or awareness.

❑ Motivation:

- Reddit has active ADHD communities.
- Users openly self-identify and describe their daily struggles.
- Can online behavior reflect underlying cognitive traits?

Problem Statement

Diagnosing ADHD often requires clinical access and self-awareness, leading many individuals to remain undiagnosed or misdiagnosed. Traditional screening methods are not easily scalable and are further limited by mental health stigma. This project explores whether machine learning and deep learning can detect ADHD-related patterns in Reddit posts, offering a potential pathway for accessible, large-scale pre-screening solutions.

The Challenge:

- Clinical diagnosis of ADHD requires time, medical access, and self-awareness.
- Many people remain undiagnosed or misdiagnosed.

Barriers:

- Traditional screening methods are not scalable.
- Mental health stigma deters people from seeking clinical help.

Objective

Primary Goal:
Classify Reddit users as ADHD or non-ADHD based on their post content.

Sub-Goals:

- Explore both traditional ML models and deep learning (LSTM).
- Analyze textual features that may be unique to ADHD users.
- Evaluate model performance using standard metrics.
- Address data quality and model generalizability challenges

Literature Review

Past Research Highlights:

- ML methods like SVM, Logistic Regression, and Naive Bayes used for mental health detection.
- Studies have explored BERT and LSTM for depression/suicide prediction.
- Reddit has been used for PTSD, anxiety, and autism detection.

Common Gaps:

- Limited ADHD-focused research using unstructured text.
- Most studies use structured questionnaires or survey data.
- Lack of comparison between classical ML and DL on real-world noisy data.

Our Contribution:

- Evaluate multiple classical ML and deep learning approaches.
- Use user-generated Reddit posts with minimal manual labeling.
- Focus on interpretability and real-world application.

Dataset

Source:

- Reddit posts collected from r/ADHD (self-declared ADHD users) and non-mental-health subreddits as control.

Dataset Details:

- Total: ~20,000 samples
- ADHD: 10,000 posts
- Non-ADHD: 10,000 posts
- Balance ensured to avoid model bias.

Post Nature:

- Long-form posts, mostly self-reflective or rant-style.
- Informal language, typos, emojis, and abbreviations common.

```
[7]: print(df.head())
```

	text	label
0	luscious i now have good thoughts	0
1	folk with jobs that require extended concentra...	1
2	fits the description of having sct perfectly e...	1
3	any artists with add need some advice deleted	1
4	different medications for different tasks removed	1

Data Preprocessing

1. Text Cleaning:

- Lowercasing, removing punctuation, links, emojis, stopwords, and non-alphabetic characters.

2. Vectorization:

- Classical ML: TF-IDF vectorization (unigrams & bigrams).
- LSTM: Tokenized using Keras tokenizer → padded to uniform length.

3. Tokenization & Embeddings (for LSTM):

- Max sequence length: 100
- Vocabulary size: 10,000

```
adhd_df['text'] = (adhd_df['title'].fillna('') + ' ' + adhd_df['selftext'].fillna(''))
adhd_df['label'] = 1

nonadhd_df['text'] = nonadhd_df['body'].fillna('')
nonadhd_df['label'] = 0

adhd_df = adhd_df[['text', 'label']]
nonadhd_df = nonadhd_df[['text', 'label']]

df = pd.concat([adhd_df, nonadhd_df], ignore_index=True)
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
```

```
def clean_text(text):
    if not isinstance(text, str):
        return ""
    text = text.lower()
    text = re.sub(r"http\S+|www.\S+", "", text)
    text = re.sub(r"\n", " ", text)
    text = re.sub(r"[^a-z\s]", "", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text

df['text'] = df['text'].apply(clean_text)
```

```
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X = vectorizer.fit_transform(df['text'])
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model Architecture – Classical ML

Algorithms Used:

- Support Vector Machine (SVM)
- Random Forest
- XGBoost
- AdaBoost

Input Features:

- TF-IDF vector (max 5000 features)
- N-grams: unigram and bigram used

Training Setup:

- 80:20 train-test split
- Hyperparameters tuned using GridSearchCV for SVM and XGBoost
- StandardScaler applied if needed (SVM)

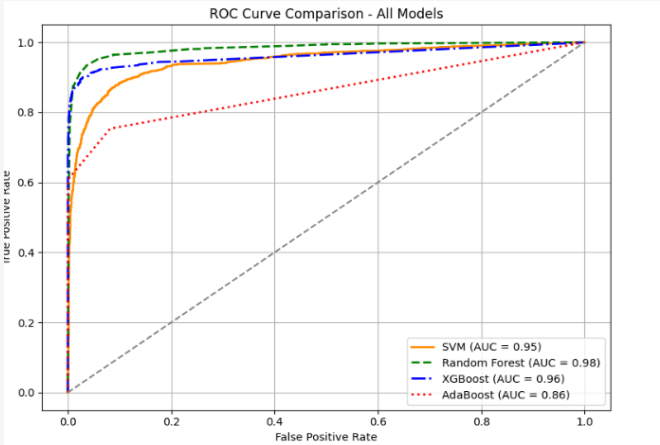
Advantages:

- Faster training
- Easier to interpret
- Performs well with small/medium data

Model Outputs – Classical ML

```python
models = {
    "SVM": SVC(kernel="linear", probability=True),
    "Random Forest": RandomForestClassifier(n_estimators=10),
    "XGBoost": XGBClassifier(eval_metric='logloss', n_estimators=10),
    "AdaBoost": AdaBoostClassifier(n_estimators=10),
}

for name, model in models.items():
    print(f"\nFitting {name}...")
    try:
        t1 = time.time()
        model.fit(X_train_scaled, y_train)
        t2 = time.time()
        print(f"{name} trained in {round(t2 - t1, 2)}s.")
        models[name] = model
    except Exception as e:
        print(f"{name} failed: {e}")
```

```
Fitting SVM...
SVM trained in 100.63s.

Fitting Random Forest...
Random Forest trained in 1.76s.

Fitting XGBoost...
XGBoost trained in 1.28s.

Fitting AdaBoost...
AdaBoost trained in 0.84s.
```



Model Architecture – Deep Learning

Architecture:

- **Input:** Padded sequences of tokenized words
- **Embedding Layer:** Converts word indices into dense vectors (128-dim)
- **LSTM Layer:** 128 hidden units
- **Dropout:** 0.3 to prevent overfitting
- **Dense Output:** 1 neuron (sigmoid activation for binary classification)

Training Parameters:

- Loss: Binary Crossentropy
- Optimizer: Adam
- Epochs: 10
- Batch Size: 64

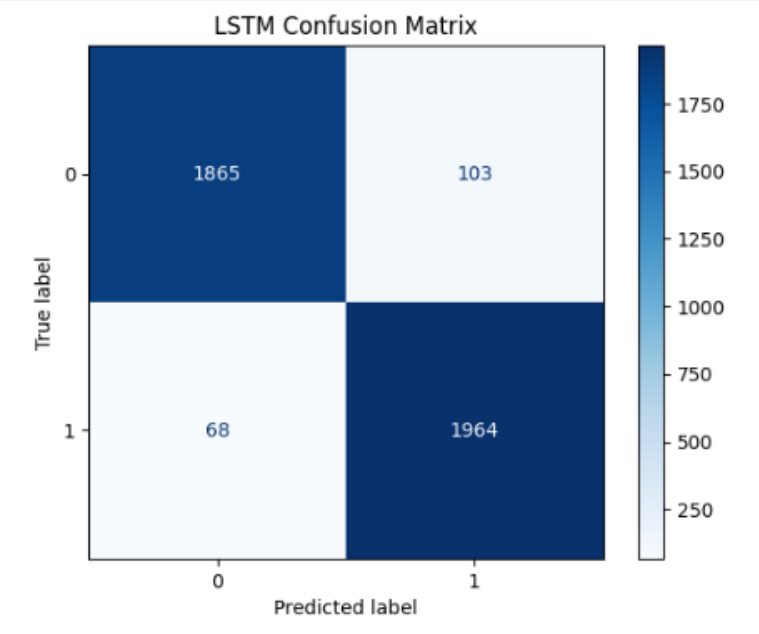
Rationale:

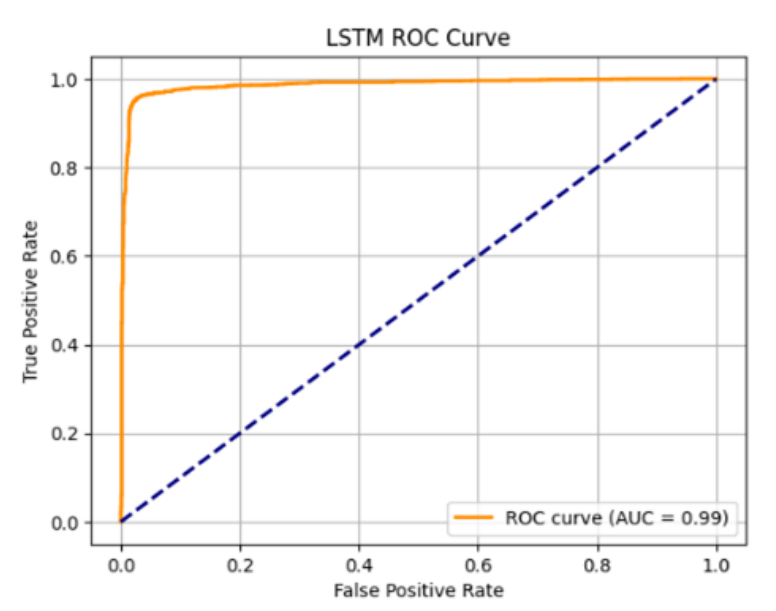
- LSTM can capture word order, context, and long-term dependencies in user writing style.

```
EMBEDDING_DIM = 64
BATCH_SIZE = 64
EPOCHS = 10

lstm_model = Sequential([
    Embedding(input_dim=MAX_NUM_WORDS, output_dim=EMBEDDING_DIM, input_length=300),
    SpatialDropout1D(0.2),
    Bidirectional(LSTM(32, dropout=0.2, recurrent_dropout=0.2)),
    Dense(1, activation='sigmoid')
])

lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


lstm_model.fit(X_train_dl, y_train_dl,
               batch_size=BATCH_SIZE,
               epochs=EPOCHS,
               validation_data=(X_test_dl, y_test_dl),
               verbose=1)
```

```
Epoch 1/10
250/250 [==============================] - 129s 494ms/step - loss: 0.3530 - accuracy: 0.8486 - val_loss: 0.1208 - val_accuracy: 0.9653
Epoch 2/10
250/250 [==============================] - 119s 477ms/step - loss: 0.0912 - accuracy: 0.9728 - val_loss: 0.1061 - val_accuracy: 0.9665
Epoch 3/10
250/250 [==============================] - 126s 505ms/step - loss: 0.0585 - accuracy: 0.9823 - val_loss: 0.1015 - val_accuracy: 0.9715
Epoch 4/10
250/250 [==============================] - 151s 604ms/step - loss: 0.0495 - accuracy: 0.9863 - val_loss: 0.1069 - val_accuracy: 0.9690
Epoch 5/10
250/250 [==============================] - 119s 475ms/step - loss: 0.0343 - accuracy: 0.9902 - val_loss: 0.1201 - val_accuracy: 0.9653
Epoch 6/10
250/250 [==============================] - 116s 464ms/step - loss: 0.0251 - accuracy: 0.9929 - val_loss: 0.1202 - val_accuracy: 0.9647
Epoch 7/10
250/250 [==============================] - 118s 473ms/step - loss: 0.0199 - accuracy: 0.9945 - val_loss: 0.1282 - val_accuracy: 0.9675
Epoch 8/10
250/250 [==============================] - 118s 472ms/step - loss: 0.0227 - accuracy: 0.9935 - val_loss: 0.1357 - val_accuracy: 0.9622
Epoch 9/10
250/250 [==============================] - 117s 466ms/step - loss: 0.0196 - accuracy: 0.9947 - val_loss: 0.1452 - val_accuracy: 0.9620
Epoch 10/10
250/250 [==============================] - 119s 476ms/step - loss: 0.0221 - accuracy: 0.9926 - val_loss: 0.1832 - val_accuracy: 0.9572
<keras.callbacks.History at 0x181a8435b20>
```

Model Outputs – Deep Learning





Experimental Setup

Train-Test Split:

- 80% training, 20% testing

Evaluation Metrics:

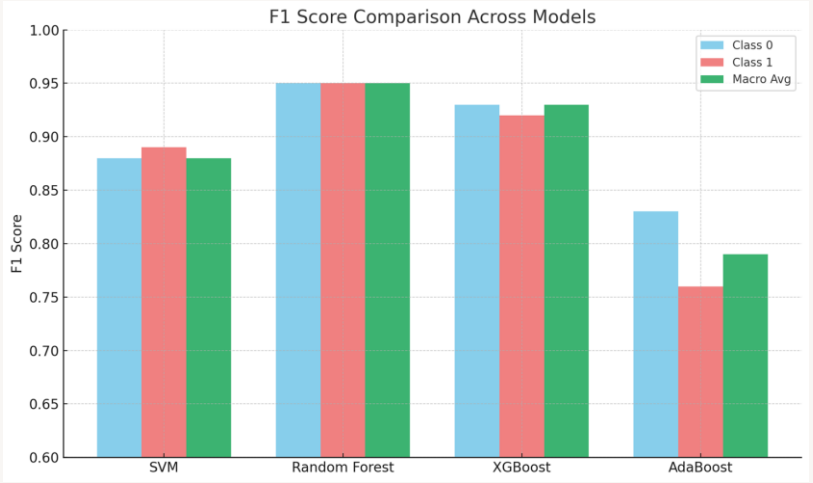
- Accuracy: Correct predictions / Total
- Precision: True Positives / Predicted Positives
- Recall: True Positives / Actual Positives
- F1-Score: Harmonic mean of Precision and Recall
- Confusion Matrix: Visualization of classification

Tools & Libraries:

- Python, Scikit-learn, TensorFlow, Keras, Pandas, Matplotlib, Seaborn

Results

Model	Accuracy	Precision (0)	Precision (1)	Recall (0)	Recall (1)	F1-Score (0)	F1-Score (1)
SVM	88%	0.90	0.87	0.86	0.91	0.88	0.89
Random Forest	95%	0.93	0.96	0.96	0.93	0.95	0.95
XGBoost	93%	0.88	0.99	0.99	0.87	0.93	0.92
AdaBoost	80%	0.71	1.00	1.00	0.61	0.83	0.76
LSTM	96%	0.98	0.97	0.97	0.98	0.98	0.97



Discussion

Why LSTM Performed Best:

- Captures contextual flow and long-term dependencies
- More robust to informal language patterns

Interesting Patterns Detected:

- ADHD posts: More personal pronouns ("I", "me"), emotional words, chaotic sentence structure
- Control posts: More informative, task-specific, neutral tone

Limitations:

- TF-IDF ignores word order/context → classical models limited
- Deep models need large data to avoid overfitting

Challenges & Limitations

- **Self-Reported ADHD:**
  Not clinically verified — introduces noise in labels.
- **Reddit-specific Language:**
  Informal, sarcastic, meme-like — may not generalize.
- **LSTM Needs More Data:**
  Risk of underfitting with small or short posts.
- **Computational Constraints:**
  DL training time longer, especially with tuning and large embeddings.

Future Work

Model Improvement:

- Try BERT or RoBERTa for contextual embedding.
- Add attention layers to improve interpretability.

Data Enhancement:

- Collect posts from other platforms (e.g., Twitter, Tumblr).
- Use clinician-verified or crowdsourced labeling.

Application:

- Build a real-time ADHD screening chatbot or plugin for forums.
- Integration with wellness apps or health platforms.

Multi-label Expansion:

Include related disorders like anxiety, OCD, or depression.

Conclusion

- **Reddit posts** contain rich linguistic cues that can help in identifying **ADHD-related behaviors**, offering a new avenue for passive digital screening.
- **Deep learning models like LSTM** significantly outperform traditional ML models in capturing context, sequential patterns, and subtle language use.
- This work demonstrates the **potential of AI in augmenting early mental health screening**, especially in **scalable and non-invasive ways**.
- It paves the way for future tools like **AI-based screening chatbots or mental health assistants**—making support more accessible, especially for underserved populations.

Thank you !