AROGO Ai ASSIGNMENT
OM THAKARE

Mental Health Prediction System Documentation

Overview

The Mental Health Prediction System is a machine learning-based application designed to predict mental health conditions, particularly suicidal tendencies, using user-provided data. The system employs two machine learning models—Random Forest and XGBoost—to classify individuals based on various features related to mental health.

The system includes data preprocessing, exploratory data analysis (EDA), feature selection, model training, evaluation, and deployment through a Streamlit-based user interface. Additionally, a language model (DistilGPT-2) generates explanations for the model's predictions, offering personalized mental health insights.

Repository Structure

- **src/**
    - predict_mental_health.py - Prepares data, trains models, and saves artifacts.
    - mental_health_ui.py - Streamlit UI for user interaction.
    - llm_explanations.py - Generates explanations using a language model.
- **models/** - Stores trained models, scaler, feature selection artifacts, and visualization plots.
- **data/raw/** - Contains the raw dataset (depression_anxiety_data.csv).
- **requirements.txt** - Lists dependencies required to run the project.

---

1. Data Preparation

Data Loading

The dataset is loaded from the data/raw/depression_anxiety_data.csv file using pandas.

data_path = os.path.join("data/raw/depression_anxiety_data.csv")

data = pd.read_csv(data_path)

Data Cleaning & Preprocessing

- **Duplicate Removal**: Duplicates are removed.
- **Handling Missing Values**:
    - Numeric columns are filled with the median value.
    - Categorical columns are filled with the mode value.
- **Encoding Categorical Variables**: LabelEncoder is applied to categorical columns.

```
data.drop_duplicates(inplace=True)
data.fillna(data.median(numeric_only=True), inplace=True)
for col in data.select_dtypes(include=['object']).columns:
    data[col].fillna(data[col].mode()[0], inplace=True)
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
```

---

2. Exploratory Data Analysis (EDA)

Distribution of Target Variable

```
plt.figure(figsize=(10, 6))
sns.countplot(x='suicidal', data=data)
plt.title("Distribution of Suicidal Cases")
plt.savefig("models/eda_countplot.png")
plt.close()
```

Feature Correlation Heatmap

```
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, cmap="coolwarm")
plt.title("Feature Correlation Matrix")
plt.savefig("models/eda_heatmap.png")
plt.close()
```

---

3. Feature Engineering & Selection

Feature Selection using Random Forest Importance

- Unnecessary columns (id, suicidal) are dropped.
- The top 10 features are selected based on feature importance.

```
rf = RandomForestClassifier(random_state=42)

rf.fit(X, y)

feature_importances = pd.DataFrame({'Feature': X.columns, 'Importance': rf.feature_importances_})

feature_importances.sort_values(by='Importance', ascending=False, inplace=True)

selected_features = feature_importances["Feature"].values[:10]

X = X[selected_features]
```

---

4. Model Development

Data Splitting & Standardization

- The dataset is split into training and testing sets.
- Features are standardized using StandardScaler.

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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

Hyperparameter Tuning & Model Training

- GridSearchCV is used to optimize hyperparameters for both Random Forest and XGBoost models.

```
rf_params = {'n_estimators': [100, 200], 'max_depth': [None, 10, 20]}

rf_grid = GridSearchCV(RandomForestClassifier(random_state=42), rf_params, cv=3, scoring='accuracy')

rf_grid.fit(X_train, y_train)
```

```
rf_best = rf_grid.best_estimator_


xgb_params = {'n_estimators': [100, 200], 'max_depth': [3, 6]}
xgb_grid = GridSearchCV(XGBClassifier(eval_metric='logloss'), xgb_params, cv=3, scoring='accuracy')
xgb_grid.fit(X_train, y_train)
xgb_best = xgb_grid.best_estimator_
```

---

5. Model Evaluation

The models are evaluated using Accuracy, Precision, Recall, F1 Score, and ROC-AUC.

```
def evaluate_model(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    try:
        roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:,1])
    except Exception:
        roc_auc = 'N/A'

    print(f"--- {model_name} ---")
    print(f"Accuracy:  {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall:    {recall:.4f}")
    print(f"F1 Score:  {f1:.4f}")
```

print(f"ROC-AUC: {roc_auc}")

print(classification_report(y_test, y_pred))

evaluate_model(rf_best, X_test, y_test, "Random Forest")

evaluate_model(xgb_best, X_test, y_test, "XGBoost")

---

6. Model Interpretation with SHAP

SHAP values are used to interpret feature importance and explain predictions.

explainer = shap.TreeExplainer(rf_best, X_train)

shap_values = explainer.shap_values(X_test, check_additivity=False)

shap.summary_plot(shap_values, X_test, show=False)

plt.savefig('models/shap_summary.png')

plt.close()

---

7. Deployment & UI (Streamlit)

- Users input symptom data through a web interface.
- Predictions are generated and displayed.

if st.button("Predict"):

predictions = predict(user_input_df)

st.subheader("Model Predictions")

st.write(predictions)

explanation = generate_explanation(user_input_df, predictions)

st.subheader("Explanation and Recommendations")

st.write(explanation)

---

8. Running the Application

Install dependencies:

pip install -r requirements.txt

Run prediction script:

python src/predict_mental_health.py

Launch Streamlit UI:

streamlit run src/mental_health_ui.py