AI Smart Tic-Tac-Toe: Minimax SVM

Project Overview

This project implements a Tic-Tac-Toe game where the player competes against an AI powered by a Support Vector Machine (SVM) classifier. The AI uses a Minimax algorithm to determine optimal moves, ensuring it plays strategically to win or at least draw. The project is structured into several key components, including data loading, preprocessing, exploratory data analysis (EDA), model training, hyperparameter tuning, evaluation, and a graphical user interface (GUI).

Table of Contents

Introduction

Tic-Tac-Toe is a classic two-player game that involves a 3x3 grid where players take turns marking a square with their respective symbols (usually X and O). The objective is to get three of one’s symbols in a row, column, or diagonal. This project aims to create a smart AI that plays Tic-Tac-Toe optimally using a machine learning approach.

The AI's intelligence is derived from training on historical game data, which is used to build an SVM model. The Minimax algorithm complements the SVM by enabling the AI to evaluate potential game states to determine the best move.

Dataset

The dataset used in this project is named **Tic tac initial results.csv**, containing game results in terms of player moves and outcomes. The dataset comprises several columns:

- **MOVE1 to MOVE7**: These columns represent the moves made by the players in the game. Each move is a number corresponding to the grid position (0-8) where a player places their symbol.
- **CLASS**: This column represents the outcome of the game. It can have three possible values: 'loss', 'win', or 'draw'.

The dataset is preprocessed to handle missing values and convert categorical outcomes into numerical labels suitable for model training.

Environment Setup

To run this project, you need to have the following Python packages installed:

- `pandas`
- `numpy`
- `seaborn`
- `matplotlib`
- `sklearn`
- `tkinter` (for the GUI)

You can install these packages using pip:

```
bash
Copy code
pip install pandas numpy seaborn matplotlib scikit-learn
```

Code Explanation

1. Loading and Reading Dataset Header

```
python
Copy code
import pandas as pd
import numpy as np

# Load the dataset
data_path = r"D:\# DATA SCIENCE\# PROJECTS\- PROJECTS
INTERNSHIPS\CODECLAUSE -AI ENGINEERING\Tic-Tac-Toe AI\Data\Tic tac initial
results.csv"
df = pd.read_csv(data_path)

# Display the dataset header
print(df.head())
```

In this section, we import the necessary libraries and load the dataset into a Pandas DataFrame. The `print(df.head())` command displays the first few rows of the dataset, allowing us to get a quick overview of its structure.

2. Handling Missing Values

```
python
Copy code
```

```
# Replace '?' with NaN
df.replace('?', np.nan, inplace=True)

# Display the number of missing values in each column
print(df.isnull().sum())

# Fill missing values with 9 (out of range for Tic-Tac-Toe moves 0-8)
df.fillna(9, inplace=True)

# Verify that there are no more missing values
print(df.isnull().sum())
```

This section addresses missing values in the dataset. First, we replace any '?' values with `NaN` using `df.replace()`. Next, we display the number of missing values per column. Since Tic-Tac-Toe moves are represented by numbers from 0 to 8, we fill any missing values with `9`, which is out of the valid range for moves. Finally, we confirm that there are no remaining missing values.

3. Data Preparation and Preprocessing

```
python
Copy code
# Map the 'CLASS' column to numerical values
class_mapping = {'loss': 0, 'win': 1, 'draw': 2}
df['CLASS'] = df['CLASS'].map(class_mapping)

# Now, convert the 'CLASS' column to integer
df['CLASS'] = df['CLASS'].astype(int)

# Convert MOVE columns to integer types if not already
df[['MOVE1', 'MOVE2', 'MOVE3', 'MOVE4', 'MOVE5', 'MOVE6', 'MOVE7']] =
df[['MOVE1', 'MOVE2', 'MOVE3', 'MOVE4', 'MOVE5', 'MOVE6',
'MOVE7']].astype(int)

# Verify the data types
print(df.dtypes)
```

In this section, we preprocess the data for model training:

1. **Mapping Outcomes**: The `CLASS` column, which contains string outcomes, is mapped to numerical values: 'loss' to 0, 'win' to 1, and 'draw' to 2.
2. **Type Conversion**: We ensure that all MOVE columns are of integer type, suitable for model input.
3. **Data Type Verification**: Finally, we print the data types of each column to confirm the successful conversions.

4. Exploratory Data Analysis (EDA)

```
python
Copy code
import seaborn as sns
import matplotlib.pyplot as plt

# Plot the distribution of the target class
sns.countplot(x='CLASS', data=df)
plt.title('Distribution of Game Outcomes (CLASS)')
plt.show()

# Correlation matrix
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.show()
```

This section includes EDA to understand the data better. We create visualizations to analyze:

1. **Outcome Distribution**: The count of each game outcome (win, loss, draw) using a count plot.
2. **Correlation Matrix**: We compute the correlation matrix to identify relationships between features, visualized using a heatmap.

5. Splitting Data, Validation, and Cross-Validation

```
python
Copy code
from sklearn.model_selection import train_test_split, cross_val_score

# Split data into features (X) and target (y)
X = df.drop('CLASS', axis=1)
y = df['CLASS']

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

# Optional: Cross-validation
from sklearn.svm import SVC
svm = SVC()
cv_scores = cross_val_score(svm, X_train, y_train, cv=5)
print("Cross-validation scores:", cv_scores)
```

In this section, we prepare the data for training and evaluation:

1. **Feature and Target Split**: We separate the features (X) from the target variable (y).
2. **Train-Test Split**: We split the dataset into training and testing sets using a 70-30 ratio.
3. **Cross-Validation**: We optionally evaluate the SVM model's performance using 5-fold cross-validation, providing a more reliable estimate of its generalization ability.

6. Hyperparameter Tuning

```
python
Copy code
from sklearn.model_selection import GridSearchCV

# Define hyperparameter grid
param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf', 'poly'],
'gamma': ['scale', 'auto']}

# Grid search
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=2, cv=5)
grid.fit(X_train, y_train)
```

```
# Best hyperparameters
print("Best parameters:", grid.best_params_)
```

This part of the code focuses on optimizing the SVM model's hyperparameters:

1. **Parameter Grid Definition**: We define a grid of hyperparameters (`C`, `kernel`, `gamma`) to search for the best combination.
2. **Grid Search**: Using `GridSearchCV`, we perform an exhaustive search over the specified parameter grid, employing cross-validation for each parameter combination.
3. **Best Parameters Output**: After fitting the grid search, we output the best parameters found for our SVM model.

7. Modeling with SVM

```
python
Copy code
# Train the model using the best parameters
svm_model = SVC(C=grid.best_params_['C'],
kernel=grid.best_params_['kernel'], gamma=grid.best_params_['gamma'])
svm_model.fit(X_train, y_train)
```

After determining the best hyperparameters, we create and train the SVM model using the `SVC` class from `sklearn`.

8. Evaluation

```
python
Copy code
from sklearn.metrics import classification_report, accuracy_score

# Make predictions
y_pred = svm_model.predict(X_test)

# Evaluation metrics
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

In the evaluation phase, we:

1. **Predictions**: Use the trained SVM model to make predictions on the test set.
2. **Performance Metrics**: Print a classification report showing precision, recall, F1-score for each class and overall accuracy.

9. Game Interface Using GUI

```
python
Copy code
import tkinter as tk
from tkinter import messagebox

# Create the GUI window
root = tk.Tk()
root.title("Tic-Tac-Toe AI")
```

```
# Function for game logic
def play_move(position):
    # Implement game logic for player and AI moves
    pass  # Replace with actual logic

# UI components (buttons, etc.)
# ...

# Run the GUI main loop
root.mainloop()
```

This section introduces the GUI using `tkinter`. We create a window for the Tic-Tac-Toe game, where players can interact with the game board. The `play_move` function handles the game logic, allowing players to make moves and the AI to respond.

Conclusion

The AI Smart Tic-Tac-Toe project demonstrates the application of machine learning, specifically SVM and the Minimax algorithm, in a simple yet engaging game. Through a systematic approach involving data preprocessing, EDA, model training, and GUI development, we have created a functional and intelligent Tic-Tac-Toe game.

Future Improvements

- Implement additional game modes (e.g., two-player mode).
- Optimize the AI further with deep learning techniques.
- Improve the GUI for a more engaging user experience.
- Incorporate reinforcement learning to enhance AI learning from player strategies.

References

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron
- The Elements of Statistical Learning by Trevor Hastie, Robert Tibshirani, and Jerome Friedman