UFC Predictor App Project Plan

UFC Predictor App Project Plan

Overview

This project aims to build a UFC predictor app/website leveraging PyTorch for AI-based predictions and incorporating data on fighters, weight classes, and betting options. The app will allow users to select fighters, specify betting criteria, and receive probability-based recommendations.

Two-Week Development Plan

Week 1: Backend and AI Model

Day 1: Project Setup and Data Collection

1. Environment Setup:

   - Create a project directory:
     mkdir ufc_predictor_app
     cd ufc_predictor_app
   - Set up a Python virtual environment:

python -m venv venv

source venv/bin/activate # On Windows, use venv\Scripts\activate

pip install flask fastapi torch torchvision pandas scikit-learn matplotlib seaborn

2. Data Collection:

- Download datasets from Kaggle or scrape data from UFC Stats.

- Collect information on fighters: name, weight class, height, reach, record, fight style, etc.

- Save as a CSV file: fighter_stats.csv.

Day 2: Data Preprocessing

1. Load the dataset using pandas.

```
import pandas as pd
data = pd.read_csv('fighter_stats.csv')
```

2. Clean the data:

- Handle missing values (data.fillna() or drop rows).

- Normalize numeric fields like height and reach.

- Encode categorical fields (e.g., weight class, fight style) using one-hot encoding or label encoding.

3. Split data into training and testing sets:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2)
```

Day 3-4: Build and Train AI Model

1. Design the Neural Network:

```
import torch.nn as nn

class FightPredictor(nn.Module):
    def __init__(self):
        super(FightPredictor, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(input_size, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.fc(x)
```

2. Train the Model:

```
model = FightPredictor()
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

Train the model using X_train and evaluate on X_test.
```

3. Save the Trained Model:

```
torch.save(model.state_dict(), 'fight_predictor.pth')
```

Day 5: Backend API

1. Create a Flask or FastAPI app:

```
pip install flask
```

2. Set up an endpoint for predictions:

```
from flask import Flask, request, jsonify
import torch

app = Flask(__name__)
model = FightPredictor()
model.load_state_dict(torch.load('fight_predictor.pth'))
model.eval()
```

```
@app.route('/predict', methods=['POST'])
def predict():
    data = request.json
    input_data = torch.tensor(data['stats']).float()
    prediction = model(input_data).item()
    return jsonify({'win_probability': prediction})

if __name__ == '__main__':
    app.run(debug=True)
```

Week 2: Frontend and Integration

Day 6: Frontend Design

1. Use HTML/CSS/Bootstrap to create the interface:
   - Dropdown menus for weight class and fighter selection.
   - A form for inputting betting options.
2. Include a results section for displaying predictions.

Day 7-8: Connect Frontend to Backend

1. Use JavaScript or AJAX to send user inputs to the backend API:

```
fetch('/predict', {
    method: 'POST',
```

```
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ stats: userSelectedStats })
  })
  .then(response => response.json())
  .then(data => console.log(data.win_probability));
```

2. Display the prediction results dynamically on the frontend.

Day 9: Database Integration

1. Use SQLite or PostgreSQL to store fighter stats and match history:

```
  import sqlite3

  conn = sqlite3.connect('ufc_data.db')
  cursor = conn.cursor()
    cursor.execute("CREATE TABLE IF NOT EXISTS fighters (name TEXT, weight_class TEXT, ...)")
  conn.commit()
```

2. Fetch stats dynamically from the database instead of static CSV files.

Day 10: Deployment

1. Install gunicorn and nginx or use a platform like Heroku for deployment:

pip install gunicorn

gunicorn -w 4 app:app

2. Test the app on a live server.

Optional Enhancements (Cherry on Top)

1. Dockerize the Application:

- Create a Dockerfile:

```
FROM python:3.9
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:5000", "app:app"]
```

- Build and run the Docker container:

```
docker build -t ufc_predictor .
docker run -p 5000:5000 ufc_predictor
```

2. Web Scraping for Automatic Updates:

- Use BeautifulSoup and requests to periodically fetch and update fighter stats.

3. Visual Enhancements:

- Add graphs (e.g., radar charts comparing fighters) using matplotlib or JavaScript libraries like Chart.js.

4. Betting Simulator:

- Calculate payout odds and recommend bets based on prediction probabilities.

GitHub Repository Steps

1. Create a Git Repository:

- Navigate to your project directory.
- Initialize a Git repository:
git init
- Create a .gitignore file to exclude unnecessary files (e.g., virtual environment, model weights):
venv/
***.pyc**
__pycache__/
***.pth**

2. Document the Project:

- Create a README.md with:
  - Project overview.
  - Setup instructions.
  - Usage details.
  - Future enhancements.

3. Commit and Push Changes:

- Add files and commit:
  git add .
  git commit -m "Initial commit"
- Push to GitHub:
  git branch -M main
  git remote add origin <repository-url>
  git push -u origin main

4. Keep Documentation Updated:

- Update the README.md as new features are added.
- Add comments in code for clarity.