Python Script Documentation: PyTorch DNN on Tom Brady's Superbowl Win Prediction

1 Introduction

This document provides detailed documentation of a Python script that uses a deep neural network (DNN) to predict Superbowl wins based on NFL data. The script trains and evaluates models using data from specified year ranges and outputs performance metrics. The following sections cover:

- Breakdown of the code.
- Layered explanation of loops and nested operations.
- Visualization of the structure using a flowchart.
- Libraries used and error handling.

2 Libraries Used

The script imports the following key libraries:

- `os`: Provides functions for creating and manipulating file directories.
- `pandas`: Used for loading, manipulating, and processing data.
- `sklearn.preprocessing.StandardScaler`: Standardizes the feature data.
- `sklearn.metrics`: Computes evaluation metrics such as accuracy, precision, recall, and F1 score, and creates confusion matrices.

- `matplotlib.pyplot`: Used to plot the confusion matrix.
- `torch`, `torch.nn`, and `torch.optim`: Libraries used for creating, training, and optimizing the deep neural network.

3 Main Functionality

The script's primary functionality includes:

- Loading NFL data from a CSV file.
- Preprocessing the data (standardization, handling missing values).
- Defining a deep neural network (DNN) model with three hidden layers.
- Training the model on a specified range of years.
- Evaluating model performance with accuracy, precision, recall, F1 score, and confusion matrix.
- Saving performance metrics and confusion matrix plots to a specified directory.

4 Code Breakdown

4.1 Class: `DNNModel`

This class defines the architecture of a deep neural network with three hidden layers.

- `fc1`, `fc2`, `fc3`: Fully connected layers with 64, 32, and 16 neurons respectively.
- `output`: Final layer producing a single output for binary classification.
- `forward`: Defines the forward pass through the network, applying ReLU activation to the hidden layers and sigmoid activation to the output.

4.2 Function: `dnn`

This function is responsible for training and evaluating the DNN model for a given range of years.

- **Input Parameters**:
  - `train_year_start`: Start year for the training data.
  - `train_year_end`: End year for the training data.
  - `features`: DataFrame of feature variables.
  - `target`: Series representing the target variable (Superbowl Win).
  - `save_directory`: Path for saving the outputs (plots, metrics).
- **Steps**:
  - Split data into training and testing sets.
  - Standardize the feature data using `StandardScaler`.
  - Convert the feature and target data to PyTorch tensors.
  - Initialize the DNN model, define a loss function (`BCELoss`), and set up the Adam optimizer.
  - Train the model for 100 epochs, performing forward and backward passes, and updating weights with gradient descent.
  - Evaluate the model on the test data and compute metrics (accuracy, precision, recall, F1 score).
  - Plot and save the confusion matrix as a PNG file.
  - Save the evaluation metrics in a text file.

4.3 Function: `main`

This function manages the workflow of data loading, preprocessing, and model training.

- Define the data and result directories.
- Load the NFL data into a pandas DataFrame from a CSV file.
- Separate the feature set and the target variable (Superbowl Win).

- Remove any columns containing missing values from the feature set.
- Call the `dnn` function twice, once for the year range 2002-2009 and once for 2002-2019.

5 Layered Explanation of Loops and Nested Operations

The `dnn` function contains a loop for training the DNN model:

- **For Loop (Training Epochs)**:
  - `for epoch in range(100)`: Trains the model for 100 epochs.
  - Inside the loop:
    * `optimizer.zero_grad()`: Clears the gradients of the previous step.
    * `outputs = model(features_train)`: Performs a forward pass through the model.
    * `loss = criterion(outputs, target_train)`: Computes the loss.
    * `loss.backward()`: Performs backpropagation to calculate gradients.
    * `optimizer.step()`: Updates the model's weights.

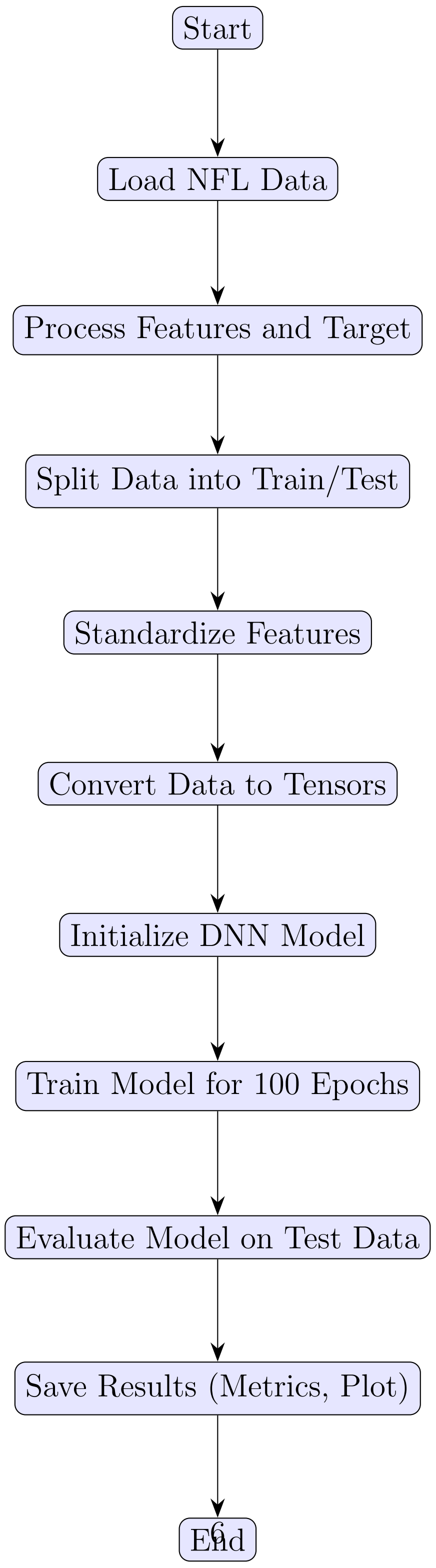
6 Error Handling

The script currently does not implement explicit error handling mechanisms. However, potential improvements include:

- Checking if the data file exists before loading it.
- Ensuring that directories for saving results exist and creating them if they do not.
- Handling any PyTorch or data conversion errors with try-except blocks.

7 Flowchart

Below is a flowchart that illustrates the overall structure of the Python script using the TikZ package:


Start
Load NFL Data
Process Features and Target
Split Data into Train/Test
Standardize Features
Convert Data to Tensors
Initialize DNN Model
Train Model for 100 Epochs
Evaluate Model on Test Data
Save Results (Metrics, Plot)
End

8 Conclusion

This Python script trains a deep neural network (DNN) to predict Superbowl wins based on NFL data. It processes data, trains a model for different time ranges, and evaluates the model's performance. This LaTeX documentation explains the script's structure, provides a detailed breakdown of its code, and includes a flowchart illustrating its execution.