***PROJECT:03***

***IPL SCORE PREDICTION USING DEEP LEARNING***

**Why Deep Learning?**

We humans can’t easily identify patterns from huge data and thus here, machine learning and deep learning comes into play. It learns how the players and teams have performed against the opposite team previously and trains the model accordingly. Using only machine learning algorithm gives a moderate accuracy therefore we used deep learning which gives much better performance than our previous model and considers the attributes which can give accurate results.

**Tools used:**

1.) Jupyter Notebook / Google colab

2.) Visual Studio

**Technology used:**

Machine Learning.

Deep Learning

Well, for the smooth running of the project we’ve used few libraries like NumPy, Pandas, Scikit-learn, TensorFlow, and Matplotlib.

**CODE:**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from sklearn import preprocessing**

**import keras**

**import tensorflow as tf**

**Step 1: Loading the dataset!**

When dealing with cricket data, it contains data from the year 2008 to 2017. The dataset contains features like venue, date, batting and bowling team, names of batsman and bowler, wickets and more.

We imported both the datasets using. read\_csv() method into a dataframe using pandas and displayed the first 5 rows of each dataset.

**Step 2:** we use the head() method returns a specified number of rows, string from the top.

**CODE:**

**ipl = pd.read\_csv('ipl\_dataset.csv')**

**ipl.head()**

**Step 3: Data Pre-processing**

* **Dropping unimportant features**

We have created a new dataframe by dropping several columns from the original DataFrame.

The new DataFrame contains the remaining columns that we are going to train the predictive model.

**CODE:**

**df = ipl.drop(['date', 'runs', 'wickets', 'overs', 'runs\_last\_5', 'wickets\_last\_5','mid', 'striker', 'non-striker'], axis =1)**

* **Further Pre-Processing**

We have split the data frame into input features (X) and target features (y). Our target features is the total score.

We have created a new dataframe by dropping several columns from the original DataFrame.

The new DataFrame contains the remaining columns that we are going to train the predictive model.

**CODE:**

**X = df.drop(['total'], axis =1)**

**y = df['total']**

* **Label Encoding**

We have applied label encoding to your categorical features in X.

We have created separate LabelEncoder objects for each categorical feature and encoded their values.

We have created mappings to convert the encoded labels back to their original values, which can be helpful for interpreting the results.

**CODE:**

**#Label Encoding**

**from sklearn.preprocessing import LabelEncoder**

**# Create a LabelEncoder object for each categorical feature**

**venue\_encoder = LabelEncoder()**

**batting\_team\_encoder = LabelEncoder()**

**bowling\_team\_encoder = LabelEncoder()**

**striker\_encoder = LabelEncoder()**

**bowler\_encoder = LabelEncoder()**

**# Fit and transform the categorical features with label encoding**

**X['venue'] = venue\_encoder.fit\_transform(X['venue'])**

**X['bat\_team'] = batting\_team\_encoder.fit\_transform(X['bat\_team'])**

**X['bowl\_team'] = bowling\_team\_encoder.fit\_transform(X['bowl\_team'])**

**X['batsman'] = striker\_encoder.fit\_transform(X['batsman'])**

**X['bowler'] = bowler\_encoder.fit\_transform(X['bowler'])**

* **Train Test Split**

We have split the data into training and testing sets. The training set contains 70 percent of the dataset and rest 30 percent is in test set.

X\_train contains the training data for your input features.

X\_test contains the testing data for your input features.

y\_train contains the training data for your target variable.

y\_test contains the testing data for your target variable.

**CODE:**

**# Train test Split**

**from sklearn.model\_selection import train\_test\_split**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)**

* **Feature Scaling**

1.We have performed Min-Max scaling on our input features to ensure all the features are on the same scale

2.Scaling is performed to ensure consistent scale to improve model performance.

3.Scaling has transformed both training and testing data using the scaling parameters.

**CODE:**

**from sklearn.preprocessing import MinMaxScaler**

**scaler = MinMaxScaler()**

**# Fit the scaler on the training data and transform both training and testing data**

**X\_train\_scaled = scaler.fit\_transform(X\_train)**

**X\_test\_scaled = scaler.transform(X\_test)**

**Step 4: Define the Neural Network**

**Step 4: Define the Neural Network**

1.We have defined a neural network using TensorFlow and Keras for regression.

2.After defining the model, we have compiled the model using the Huber Loss.

**CODE:**

**# Define the neural network model**

**model = keras.Sequential([**

**keras.layers.Input( shape=(X\_train\_scaled.shape[1],)), # Input layer**

**keras.layers.Dense(512, activation='relu'), # Hidden layer with 512 units and ReLU activation**

**keras.layers.Dense(216, activation='relu'), # Hidden layer with 216 units and ReLU activation**

**keras.layers.Dense(1, activation='linear') # Output layer with linear activation for regression**

**])**

**# Compile the model with Huber loss**

**huber\_loss = tf.keras.losses.Huber(delta=1.0) # You can adjust the 'delta' parameter as needed**

**model.compile(optimizer='adam', loss=huber\_loss) # Use Huber loss for regression**

**Step 5: Model Training**

We have trained the neural network model using the scaled training data.

**CODE:**

**# Train the model**

**model.fit(X\_train\_scaled, y\_train, epochs=50, batch\_size=64, validation\_data=(X\_test\_scaled, y\_test))**

After the training, we have stored the training and validation loss values to our neural network during the training process.

**CODE:**

**model\_losses = pd.DataFrame(model.history.history)**

**model\_losses.plot()**

**Step 6: Model Evaluation**

We have predicted using the trained neural network on the testing data.

The variable predictions contain the predicted total run scores for the test set based on the model’s learned patterns.

**CODE:**

**# Make predictions**

**predictions = model.predict(X\_test\_scaled)**

**from sklearn.metrics import mean\_absolute\_error,mean\_squared\_error**

**mean\_absolute\_error(y\_test,predictions)**

**Output:**

**9.62950576317203**

**Step 7: Let’s create an Interactive Widget**

We have created an interactive widget using ipywidgets to predict the score based on user input for venue, batting team, bowling team, striker, and bowler.

We have created dropdown widgets to select values for venue, batting team, bowling team, striker, and bowler.

Then, we have added a “Predicted Score” button widget. Whenever, the button will be clicked, the predict\_score function will be called and then perform the following steps:

Decodes the user-selected values to their original categorical values.

Encodes and scales these values to match the format used in model training.

Uses the trained model to make a prediction based on the user’s input.

Displays the predicted score.

**CODE:**

**import ipywidgets as widgets**

**from IPython.display import display, clear\_output**

**import warnings**

**warnings.filterwarnings("ignore")**

**venue = widgets. Dropdown(options=df['venue']. unique (). tolist(),description='Select Venue:')**

**batting\_team = widgets.Dropdown(options =df['bat\_team'].unique().tolist(), description='Select Batting Team:')**

**bowling\_team = widgets.Dropdown(options=df['bowl\_team'].unique().tolist(), description='Select Batting Team:')**

**striker = widgets.Dropdown(options=df['batsman'].unique().tolist(), description='Select Striker:')**

**bowler = widgets.Dropdown(options=df['bowler'].unique().tolist(), description='Select Bowler:')**

**predict\_button = widgets.Button(description="Predict Score")**

**def predict\_score(b):**

**with output:**

**clear\_output() # Clear the previous output**

**# Decode the encoded values back to their original values**

**decoded\_venue = venue\_encoder.transform([venue.value])**

**decoded\_batting\_team = batting\_team\_encoder.transform([batting\_team.value])**

**decoded\_bowling\_team = bowling\_team\_encoder.transform([bowling\_team.value])**

**decoded\_striker = striker\_encoder.transform([striker.value])**

**decoded\_bowler = bowler\_encoder.transform([bowler.value])**

**input = np.array([decoded\_venue, decoded\_batting\_team, decoded\_bowling\_team,decoded\_striker, decoded\_bowler])**

**input = input.reshape(1,5)**

**input = scaler.transform(input)**

**#print(input)**

**predicted\_score = model.predict(input)**

**predicted\_score = int(predicted\_score[0,0])**

**print(predicted\_score)**

The widget-based interface allows you to interactively predict the score for specific match scenarios. Now, we have set up the button to trigger the predict\_score function when clicked and display the widgets for venue, batting team, bowling team, striker and bowler.

**CODE:**

**predict\_button.on\_click(predict\_score)**

**output = widgets.Output()**

**display (venue, batting\_team, bowling\_team, striker, bowler, predict\_button, output)**

We have predicted the score of the match between CSK and King XI Punjab in Punjab Cricket Stadium. The predicted score of the match is 183.