Final Project by Tom Adoni and Ryan Chen

Introduction

The National Basketball Association (NBA) is a premier professional basketball league renowned for its competitive nature and high-level athleticism. One key aspect of team success in the NBA is the ability to perform consistently across various statistical categories throughout the season. In this final project, we propose to build a neural network model that predicts NBA team wins in a season based on historical NBA seasonal team statistics from 2000 to 2018.

Dataset Description

We will utilize the NBA Team Statistics dataset, which spans from the 2000-2001 NBA season to the 2017-2018 NBA season. The dataset, available on Kaggle at the following link: NBA Team Statistics Dataset, includes a comprehensive collection of team-level statistics such as points per game, rebounds, assists, steals, blocks, turnovers, and more. These statistics encapsulate various aspects of team performance and are indicative of team success throughout the regular season.

Objectives

● Develop a neural network model that accurately predicts the number of wins for NBA teams in a given season based on their seasonal statistics.

● Evaluate the performance of the model using appropriate metrics such as mean squared error, mean absolute error, and R-squared.

● Identify the most influential statistical features contributing to team wins and gain insights into the underlying factors driving team success in the NBA.

Methodology

● Data Preprocessing: Clean and preprocess the dataset to handle missing values, normalize the features, and prepare the data for model training.

● Model Development: Construct and train a neural network regression model using libraries such as TensorFlow or PyTorch. Experiment with different architectures, activation functions, and hyperparameters to optimize model performance.

● Model Evaluation: Evaluate the trained model using cross-validation techniques and performance metrics to assess its predictive accuracy and generalization capabilities.

● Feature Importance Analysis: Employ techniques such as feature importance ranking or SHAP (SHapley Additive exPlanations) values to identify the most significant statistical features influencing team wins.

Deliverables

● Final Report: A comprehensive report detailing the project objectives, methodology, experimental results, and insights gained from the analysis.

● Neural Network Model: A trained neural network regression model capable of predicting NBA team wins based on seasonal statistics.

● Code Repository: A GitHub repository containing the codebase, including data preprocessing, model development, and evaluation scripts, along with documentation for reproducibility.

Data Cleaning and Model Training

In [10]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the dataset
dataset = pd.read_csv('nba_team_stats_00_to_21.csv')  # Replace 'nba_season_stats.csv' with your dataset filename

# Extract features (excluding non-numeric columns) and target variable
X = dataset.drop(columns=['TEAM', 'SEASON', 'W', 'WIN%', 'L', 'GP']).values
y = dataset['W'].values

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the neural network architecture
class NBA_NN(nn.Module):
    def __init__(self, input_size):
        super(NBA_NN, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# Instantiate the model
input_size = X_train.shape[1]
model = NBA_NN(input_size)

# Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Convert data to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

# Train the model
num_epochs = 100
batch_size = 32
for epoch in range(num_epochs):
    for i in range(0, len(X_train_tensor), batch_size):
        inputs = X_train_tensor[i:i+batch_size]
        targets = y_train_tensor[i:i+batch_size]

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs.squeeze(), targets)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Evaluate the model
with torch.no_grad():
    predicted = model(X_test_tensor).squeeze().numpy()

# You can perform further evaluation, such as calculating metrics like MSE, RMSE, etc.
```

```
Epoch [10/100], Loss: 147.8477
Epoch [20/100], Loss: 63.7202
Epoch [30/100], Loss: 50.0046
Epoch [40/100], Loss: 40.1241
Epoch [50/100], Loss: 31.7222
Epoch [60/100], Loss: 24.8131
Epoch [70/100], Loss: 20.0774
Epoch [80/100], Loss: 16.2276
Epoch [90/100], Loss: 13.2242
Epoch [100/100], Loss: 10.8201
```

Statistical Metrics of the Neural Network Model

In [11]:

```python
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Convert the predicted values to NumPy array
predicted = predicted.flatten()

# Compute evaluation metrics
mse = mean_squared_error(y_test, predicted)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, predicted)

print(f'Mean Squared Error (MSE): {mse:.4f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.4f}')
print(f'R-squared (R^2) Score: {r2:.4f}')
```

```
Mean Squared Error (MSE): 27.1671
Root Mean Squared Error (RMSE): 5.2122
R-squared (R^2) Score: 0.8112
```

Feature Selection using Lasso

In [13]:

```python
from sklearn.linear_model import LassoCV

# Initialize LassoCV model
lasso = LassoCV(cv=5)

# Fit LassoCV model to training data
lasso.fit(X_train, y_train)

# Get selected features based on non-zero coefficients
selected_features = dataset.drop(columns=['TEAM', 'SEASON', 'W', 'WIN%', 'L', 'GP']).columns[lasso.coef_ != 0]

# Filter the dataset to keep only selected features
X_train_selected = X_train[:, lasso.coef_ != 0]
X_test_selected = X_test[:, lasso.coef_ != 0]

print("Selected Features:", selected_features)
```

```
Selected Features: Index(['teamstatspk', 'MIN', '3PA', '3P%', 'FTA', 'FT%', 'OREB', 'TOV', 'BLKA',
       'PF', '+/-'],
      dtype='object')
```

Reduced Model (Feature Selected)

In [14]:

```python
# Normalize the selected features
scaler_selected = StandardScaler()
X_train_selected = scaler_selected.fit_transform(X_train_selected)
X_test_selected = scaler_selected.transform(X_test_selected)

# Define the neural network architecture
class NBA_NN_Selected(nn.Module):
    def __init__(self, input_size):
        super(NBA_NN_Selected, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# Instantiate the model
input_size_selected = X_train_selected.shape[1]
model_selected = NBA_NN_Selected(input_size_selected)

# Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model_selected.parameters(), lr=0.001)

# Convert data to PyTorch tensors
X_train_selected_tensor = torch.tensor(X_train_selected, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_selected_tensor = torch.tensor(X_test_selected, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

# Train the model
num_epochs = 100
batch_size = 32
for epoch in range(num_epochs):
    for i in range(0, len(X_train_selected_tensor), batch_size):
        inputs = X_train_selected_tensor[i:i+batch_size]
        targets = y_train_tensor[i:i+batch_size]

        # Forward pass
        outputs = model_selected(inputs)
        loss = criterion(outputs.squeeze(), targets)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Evaluate the model
with torch.no_grad():
    predicted_selected = model_selected(X_test_selected_tensor).squeeze().numpy()

# Compute evaluation metrics
mse_selected = mean_squared_error(y_test, predicted_selected)
rmse_selected = np.sqrt(mse_selected)
r2_selected = r2_score(y_test, predicted_selected)

print(f'Mean Squared Error (MSE) with selected features: {mse_selected:.4f}')
print(f'Root Mean Squared Error (RMSE) with selected features: {rmse_selected:.4f}')
print(f'R-squared (R^2) Score with selected features: {r2_selected:.4f}')
```

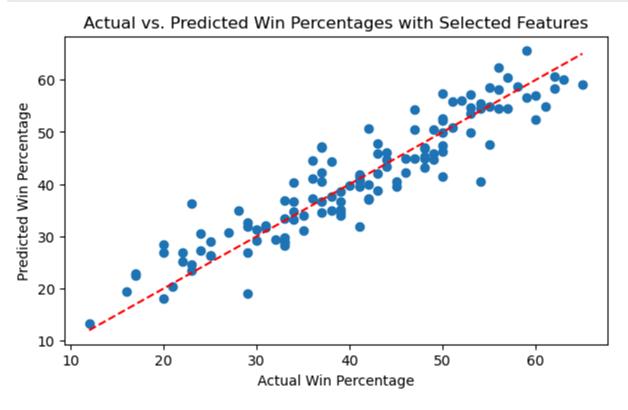
```
Epoch [10/100], Loss: 279.7950
Epoch [20/100], Loss: 56.7598
Epoch [30/100], Loss: 42.3286
Epoch [40/100], Loss: 35.6142
Epoch [50/100], Loss: 29.9797
Epoch [60/100], Loss: 25.0607
Epoch [70/100], Loss: 20.9255
Epoch [80/100], Loss: 17.1132
Epoch [90/100], Loss: 14.3831
Epoch [100/100], Loss: 12.3639
Mean Squared Error (MSE) with selected features: 20.1292
Root Mean Squared Error (RMSE) with selected features: 4.4866
R-squared (R^2) Score with selected features: 0.8601
```

As we can see, the reduced model improved in R-squared from 0.81 to 0.86

Actual vs Predicted

In [16]:

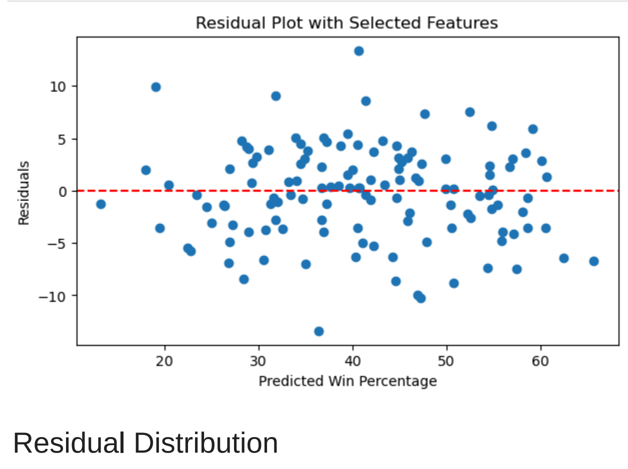
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(7, 4))
plt.scatter(y_test, predicted_selected)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', color='red')
plt.xlabel('Actual Win Percentage')
plt.ylabel('Predicted Win Percentage')
plt.title('Actual vs. Predicted Win Percentages with Selected Features')
plt.show()
```



Residual Plot

In [17]:

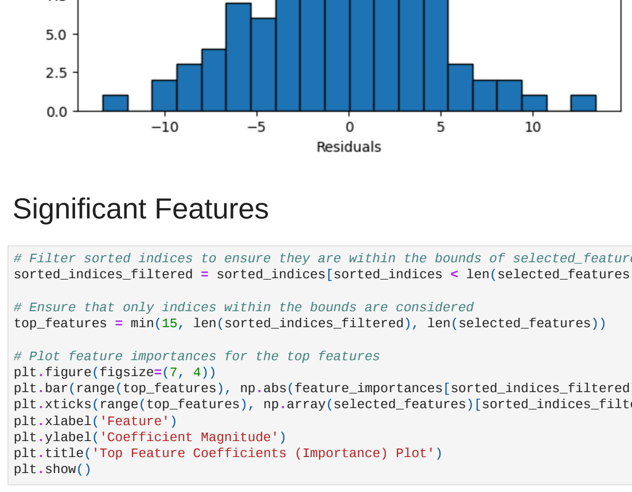
```python
residuals_selected = y_test - predicted_selected

plt.figure(figsize=(7, 4))
plt.scatter(predicted_selected, residuals_selected)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Win Percentage')
plt.ylabel('Residuals')
plt.title('Residual Plot with Selected Features')
plt.show()
```



Residual Distribution

In [18]:

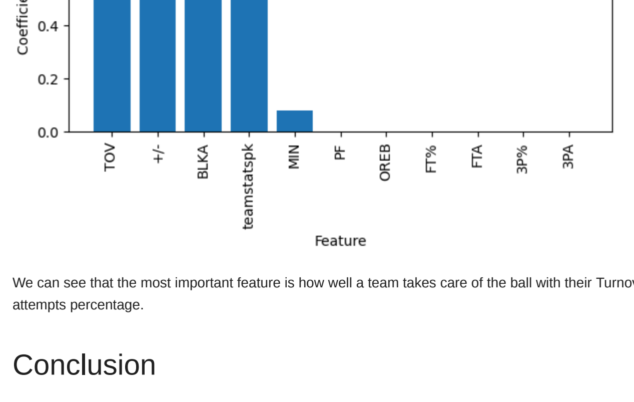
```python
plt.figure(figsize=(7, 4))
plt.hist(residuals_selected, bins=20, edgecolor='k')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Distribution of Residuals with Selected Features')
plt.show()
```



Significant Features

In [33]:

```python
# Filter sorted indices to ensure they are within the bounds of selected_features
sorted_indices_filtered = sorted_indices[sorted_indices < len(selected_features)]

# Ensure that only indices within the bounds are considered
top_features = min(15, len(sorted_indices_filtered), len(selected_features))

# Plot feature importances for the top features
plt.figure(figsize=(7, 4))
plt.bar(range(top_features), np.abs(feature_importances[sorted_indices_filtered][:top_features]), align='center')
plt.xticks(range(top_features), np.array(selected_features)[sorted_indices_filtered][:top_features], rotation=90)
plt.xlabel('Feature')
plt.ylabel('Coefficient Magnitude')
plt.title('Top Feature Coefficients (Importance) Plot')
plt.show()
```



We can see that the most important feature is how well a team takes care of the ball with their Turnover rating. Also, the +/- which is the average point differential and blocked field goal attempts percentage.

Conclusion

We have developed a great reduced Neural Network model predicting team wins in the NBA using feature selection from Lasso.

We learned the most important features were TOV, +/- and BLKA%

We can now predict team wins in the NBA using team statistics.

We should look for more data from the past few seasons to try and improve upon the model.

References

● Kaggle Dataset: NBA Team Statistics Dataset

● TensorFlow: https://www.tensorflow.org/

● PyTorch: https://pytorch.org/

In [ ]: