KBO Prediction Model Using Deep Neural Network

Jay Ryu

July 25, 2017

1 Introduction

This project focuses on examining the capability of deep learning in predicting the result of an event that involves many different factors. One good example is a baseball game, where team's members, the order of pitchers, etc affects the end result greatly. The neural network is expected to learn such hidden factors, access the individual team's strength, and provide an prediction of a game. The basic architecture of the deep neural network used in the KBO Prediction Model, is as shown in our github repository.

The data was obtained from the baseball page of South Korea's biggest portal site. I want to thank Nate Namgun Kim for helping me throughout the project, providing the dataset scraper, and mentoring on the making the code scalable.

2 Data Scraping

2.1 Note

All the scraping code is provided in scrape.py. In order to use the scrape.py, make sure to run it with python 3.6.

2.2 Usage

In order to obtain the entire 2017 KBO baseball data, simply run the following.

```
from scrape import *
year = '2015'
months = ['03', '04', '05', '06', '07', '08', '09', '10']
summaries = []
for month in months:
    summaries += MatchSummaryParser(year, month).parse()
```

```
matches = []
for summary in summaries:
    try:
        matches.append(
            MatchDetailParser(
                summary.year,
                summary.month,
                summary.day,
                summary.get_away_team_name(),
                summary.get_home_team_name()
            ).parse()
        )
    except DetailDataNotFoundException:
        # this is most likely a double header game.
        pass
```

2.3 Terms

The values used from the scraped JSON data are shown below.

summary.r : Final score

summary.b : Balls

summary.e : Errors

summary.h : Hits

team−standing.draws : Team draws

team−standing.era : Earned runs average

team−standing.hra : Home runs allowed

team−standing.wra : Win rate

team−standing.wins : Wins

team−standing.loses : Loses

team−standing.rank : Team ranking

3 Prediction Model

The scraped data are parsed in the formatter.py to follow the below format:

```
hometeam's data = [The summary] + [team-standing previous] for _ in k games
awayteam's data = [The summary] + [team-standing previous] for _ in k games

result of the game = [x, y]
```

Based on the user's input k, the formatter will add previous k games of the team to the input dataset. The user also has freedom to choose the proportion of the training dataset and test dataset from the raw JSON data.

The builder.py constructs the neural network that takes in the above format of data to be trained and predict the result of the game. When taking the individual team data, as the model does not distinguish the team specific statistics, auto encoder was used to generalize (or normalize) team specific factors then passed on to the drop out SeLU layers. The number of SeLU layers is yet to be provided as an user input. Note that each layer of neural network will perform dropout according to the user's designated drop out rate and SeLU/AdamOptimzer combination was used in this model. The SELU implementation was taken from here.

3.1 Getting Started

Python 3 is required to run the model. I suggest following the below instruction to run the code

```
virtualenv -p python3 .env
source .env/bin/activate
pip install -r requirements.txt
# Run the code
deactivate
```

3.2 Details

After activating the virtual envelop, run

```
python trainer.py -h

# Result
KBO Score Prediction Trainer

positional arguments:
  year             The year of data to train the model with
  train_size       The proportion of the training set to the test set
  model_name       The name of the model
  learn_rate       The learning rate
  sequence_length  Sequence length
  epoch            Training epoch
  drop_rate        Drop rate
```

```
optional arguments:
  -h, --help            show this help message and exit
```

The model takes in 7 inputs from the user.

1. The year of the game to train the Deep Learning model with.
2. The proportion of the training set to the test set in the raw dataset.
3. The name of the model. (For saving purpose)
4. The learning rate of the model.
5. The number of previous games the model should look at for each game.
6. The iteration of training.
7. The drop rate.

For example, run below command to obtain a model that is trained with the first 80 percent of the 2017 dataset with 0.0005 learning rate when back propagated, taking team's previous 7 game results as inputs, iterated 5000 times with the same data with drop out rate of 0.3. The fine tuning is entirely up to the user, as the details of model architecture relies on the user's decision.

```
python trainer.py 2017 0.8 TEST_MODEL 0.0005 7 5000 0.3
```

The trained model will be saved in /saved_graphs of the head directory as .ckpt extended file format. When the user interface is added in the future, best performing pre-trained model will be called to predict the outcome of a game between teams the user indicated.

3.3 Performance

Unfortunately, the extensive search for the best performance has not been conducted at this point. However, I was able to conduct a test that yielded the following results. This experiment uses 2017 data.

```
The default parameters:
0.8 train_size
0.0005 learning_rate
7 sequence_length
3000 epoch
0.2 drop_rate

The best performance by parameters:
Best epoch: 2000 yields 0.661971830986% Accuracy
Best drop rate: 0.2 yields 0.591549295775% Accuracy
Best seg length: 3 yields 0.586666666667% Accuracy
Best learn rate: 0.01 yields 0.605633802817% Accuracy
Best train size 0.8 yields 0.549295774648% Accuracy
```

3.4 Discussion

Given that the model uses a small dataset and small number of statistics (11 values used from 500 games), I consider 66% accuracy to be very promising. Alternative structures for the prediction model is a plausible solution, but the quality of the dataset and identifying meaningful statistical values must be the priority.

4 Acknowledgment

First of all, I thank Nate Namgun Kim for providing the data scraping code. Also, I wouldn't have been able to complete the project without the generous support from NSF, Two Sigma, and the Provost/VP. I thank Professor Anshumali Shrivastava for advising on the model structure.