NFL Big Data Bowl 2020 Official Starter Notebook

Introduction

In this competition you will predict how many yards a team will gain on a rushing play in an NFL regular season game. You will loop through a series of rushing plays; for each play, you'll receive the position, velocity, orientation, and more for all 22 players on the field at the moment of handing the ball off to the rusher, along with many other features such as teams, stadium, weather conditions, etc. You'll use this information to predict how many yards the team will gain on the play as a cumulative probability distribution. Once you make that prediction, you can move on to the next rushing play.

This competition is different from most Kaggle Competitions in that:

- You can only submit from Kaggle Notebooks, and you may not use other data sources, GPU, or internet access.
- This is a **two-stage competition**. In Stage One you can edit your Notebooks and improve your model, where Public Leaderboard scores are based on your predictions on rushing plays from the first few weeks of the 2019 regular season. At the beginning of Stage Two, your Notebooks are locked, and we will re-run your Notebooks over the following several weeks, scoring them based on their predictions relative to live data as the 2019 regular season unfolds.
- You must use our custom `kaggle.competitions.nflrush` Python module. The purpose of this module is to control the flow of information to ensure that you are not using future data to make predictions for the current rushing play. If you do not use this module properly, your code may fail when it is re-run in Stage Two.

In this Starter Notebook, we'll show how to use the `nflrush` module to get the training data, get test features and make predictions, and write the submission file.

TL;DR: End-to-End Usage Example

```
from kaggle.competitions import nflrush
env = nflrush.make_env()

# Training data is in the competition dataset as usual
train_df = pd.read_csv('/kaggle/input/nfl-big-data-bowl-2020/train.csv', low_memory=False)
train_my_model(train_df)

for (test_df, sample_prediction_df) in env.iter_test():
  predictions_df = make_my_predictions(test_df, sample_prediction_df)
  env.predict(predictions_df)

env.write_submission_file()
```

Note that `train_my_model` and `make_my_predictions` are functions you need to write for the above example to work.

In-depth Introduction

First let's import the module and create an environment.

In [1]:
```
from kaggle.competitions import nflrush
import pandas as pd

# You can only call make_env() once, so don't lose it!
env = nflrush.make_env()
```

Training data is in the competition dataset as usual

In [2]:
```
train_df = pd.read_csv('/kaggle/input/nfl-big-data-bowl-2020/train.csv', low_memory=False)
train_df
```

Out[2]:

	GameId	PlayId	Team	X	Y	S	A	Dis	Orientation	Dir	...	Week	Stadium	Location	Stadi
0	2017090700	20170907000118	away	73.91	34.84	1.69	1.13	0.40	81.99	177.18	...	1	Gillette Stadium	Foxborough, MA	
1	2017090700	20170907000118	away	74.67	32.64	0.42	1.35	0.01	27.61	198.70	...	1	Gillette Stadium	Foxborough, MA	
2	2017090700	20170907000118	away	74.00	33.20	1.22	0.59	0.31	3.01	202.73	...	1	Gillette Stadium	Foxborough, MA	
3	2017090700	20170907000118	away	71.46	27.70	0.42	0.54	0.02	359.77	105.64	...	1	Gillette Stadium	Foxborough, MA	
4	2017090700	20170907000118	away	69.32	35.42	1.82	2.43	0.16	12.63	164.31	...	1	Gillette Stadium	Foxborough, MA	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
509757	2018123015	20181230154157	home	86.77	24.20	2.14	2.12	0.22	44.55	15.31	...	17	CenturyLink Field	Seattle, WA	
509758	2018123015	20181230154157	home	86.76	27.18	1.16	0.66	0.11	53.63	42.80	...	17	CenturyLink Field	Seattle, WA	
509759	2018123015	20181230154157	home	87.26	27.05	2.59	1.18	0.26	3.96	21.12	...	17	CenturyLink Field	Seattle, WA	
509760	2018123015	20181230154157	home	84.57	24.37	4.36	1.79	0.47	148.08	183.34	...	17	CenturyLink Field	Seattle, WA	
509761	2018123015	20181230154157	home	80.80	26.35	4.87	4.10	0.45	135.44	118.24	...	17	CenturyLink Field	Seattle, WA	

509762 rows × 49 columns

`iter_test` function

Generator which loops through each rushing play in the test set and provides the observations at `TimeHandoff` just like the training set. Once you call **`predict`** to make your yardage prediction, you can continue on to the next play.

Yields:

- While there are more rushing play(s) and `predict` was called successfully since the last yield, yields a tuple of:
  - `test_df`: DataFrame with player and game observations for the next rushing play.
  - `sample_prediction_df`: DataFrame with an example yardage prediction. Intended to be filled in and passed back to the `predict` function.
- If `predict` has not been called successfully since the last yield, prints an error and yields `None`.

In [3]:
```
# You can only iterate through a result from `env.iter_test()` once
# so be careful not to lose it once you start iterating.
iter_test = env.iter_test()
```

Let's get the data for the first test play and check it out.

In [4]:
```
(test_df, sample_prediction_df) = next(iter_test)
test_df
```

Out[4]:

	GameId	PlayId	Team	X	Y	S	A	Dis	Orientation	Dir	...	Week	Stadium	Location	StadiumType
0	2019090500	20190905000050	away	34.32	24.27	5.09	1.95	0.50	52.07	19.31	...	1	Soldier Field	Chicago	Outdoor
1	2019090500	20190905000050	away	33.13	30.92	3.59	2.06	0.36	20.13	347.96	...	1	Soldier Field	Chicago	Outdoor
2	2019090500	20190905000050	away	30.68	24.69	3.84	2.41	0.38	271.54	272.36	...	1	Soldier Field	Chicago	Outdoor
3	2019090500	20190905000050	away	34.82	30.70	4.65	1.25	0.48	46.80	5.17	...	1	Soldier Field	Chicago	Outdoor
4	2019090500	20190905000050	away	34.22	29.19	4.80	0.87	0.49	57.18	8.61	...	1	Soldier Field	Chicago	Outdoor
5	2019090500	20190905000050	away	34.65	34.91	4.23	1.53	0.41	77.40	21.34	...	1	Soldier Field	Chicago	Outdoor
6	2019090500	20190905000050	away	34.43	27.48	4.13	0.30	0.43	88.14	1.04	...	1	Soldier Field	Chicago	Outdoor
7	2019090500	20190905000050	away	34.18	25.22	4.45	2.57	0.43	74.58	3.06	...	1	Soldier Field	Chicago	Outdoor
8	2019090500	20190905000050	away	31.55	29.30	5.68	1.08	0.56	59.50	23.59	...	1	Soldier Field	Chicago	Outdoor
9	2019090500	20190905000050	away	28.76	26.93	5.96	3.23	0.58	75.49	24.50	...	1	Soldier Field	Chicago	Outdoor
10	2019090500	20190905000050	away	36.04	14.15	3.95	4.51	0.39	142.52	133.61	...	1	Soldier Field	Chicago	Outdoor
11	2019090500	20190905000050	home	35.36	35.36	3.80	4.18	0.36	246.95	1.70	...	1	Soldier Field	Chicago	Outdoor
12	2019090500	20190905000050	home	36.83	31.29	5.30	0.72	0.52	279.03	331.16	...	1	Soldier Field	Chicago	Outdoor
13	2019090500	20190905000050	home	35.09	25.51	4.61	0.62	0.48	278.11	2.19	...	1	Soldier Field	Chicago	Outdoor
14	2019090500	20190905000050	home	34.15	20.83	4.81	3.41	0.46	301.39	349.99	...	1	Soldier Field	Chicago	Outdoor
15	2019090500	20190905000050	home	47.19	20.10	3.29	2.48	0.32	319.37	38.27	...	1	Soldier Field	Chicago	Outdoor
16	2019090500	20190905000050	home	37.60	13.58	1.58	2.05	0.15	275.48	183.57	...	1	Soldier Field	Chicago	Outdoor
17	2019090500	20190905000050	home	35.17	27.28	3.99	1.62	0.39	268.30	358.37	...	1	Soldier Field	Chicago	Outdoor
18	2019090500	20190905000050	home	33.67	31.75	3.73	1.35	0.37	253.21	327.64	...	1	Soldier Field	Chicago	Outdoor
19	2019090500	20190905000050	home	43.03	30.40	4.29	3.66	0.41	265.35	298.95	...	1	Soldier Field	Chicago	Outdoor
20	2019090500	20190905000050	home	35.47	29.97	4.27	0.48	0.44	293.44	357.08	...	1	Soldier Field	Chicago	Outdoor
21	2019090500	20190905000050	home	36.62	27.74	5.98	1.37	0.59	298.55	330.13	...	1	Soldier Field	Chicago	Outdoor

22 rows × 48 columns

Note how our predictions need to take the form of a cumulative probability distribution over the range of possible yardages. Each column indicates the probability that the team gains <= that many yards on the play. For example, the value for `Yards-2` should be your prediction for the probability that the team gains at most -2 yards, and `Yard10` is the probability that the team gains at most 10 yards. Theoretically, `Yards99` should equal `1.0`.

In [5]:
```
sample_prediction_df
```

Out[5]:

	Yards-99	Yards-98	Yards-97	Yards-96	Yards-95	Yards-94	Yards-93	Yards-92	Yards-91	Yards-90	...	Yards90	Yards91	Yards92	Yards93	Yards94	Yards
0	0	0	0	0	0	0	0	0	0	0	...	1	1	1	1	1	

1 rows × 199 columns

The sample prediction here just predicts that exactly 3 yards were gained on the play.

In [6]:
```
sample_prediction_df[sample_prediction_df.columns[98:108]]
```

Out[6]:

	Yards-1	Yards0	Yards1	Yards2	Yards3	Yards4	Yards5	Yards6	Yards7	Yards8
0	0	0	0	0	1	1	1	1	1	1

Note that we'll get an error if we try to continue on to the next test play without making our predictions for the current play.

In [7]:
```
next(iter_test)
```

```
ERROR: You must call `predict()` successfully before you can continue with `iter_test()`.
```

`predict` function

Stores your predictions for the current rushing play. Expects the same format as you saw in `sample_prediction_df` returned from the `iter_test` generator.

Args:

- `predictions_df`: DataFrame which must have the same format as `sample_prediction_df`.

This function will raise an Exception if not called after a successful iteration of the `iter_test` generator.

Let's make a dummy prediction using the sample provided by `iter_test`.

In [8]:
```
env.predict(sample_prediction_df)
```

Main Loop

Let's loop through all the remaining plays in the test set generator and make the default prediction for each. The `iter_test` generator will simply stop returning values once you've reached the end.

When writing your own Notebooks, be sure to write robust code that makes as few assumptions about the `iter_test`/`predict` loop as possible. For example, the number of iterations will change during Stage Two of the competition, since you'll be tested on rushing plays which hadn't even occurred when you wrote your code. There may also be players in the updated test set who never appeared in any Stage One training or test data.

You may assume that the structure of `sample_prediction_df` will not change in this competition.

In [9]:
```
for (test_df, sample_prediction_df) in iter_test:
    env.predict(sample_prediction_df)
```

`write_submission_file` function

Writes your predictions to a CSV file (`submission.csv`) in the Notebook's *output* directory.

You must call this function and not generate your own `submission.csv` file manually.

Can only be called once you've completed the entire `iter_test`/`predict` loop.

In [10]:
```
env.write_submission_file()
```

```
Your submission file has been saved!  Once you `Commit` your Notebook and it finishes running, you ca
n submit the file to the competition from the Notebook Viewer `Output` tab.
```

In [11]:
```
# We've got a submission file!
import os
print([filename for filename in os.listdir('/kaggle/working') if '.csv' in filename])
```

```
['submission.csv']
```

As indicated by the helper message, calling `write_submission_file` on its own does **not** make a submission to the competition. It merely tells the module to write the `submission.csv` file as part of the Notebook's output. To make a submission to the competition, you'll have to **Commit** your Notebook and find the generated `submission.csv` file in that Notebook Version's Output tab (note this is *outside* of the Notebook Editor), then click "Submit to Competition". When we re-run your Notebook during Stage Two, we will run the Notebook Version(s) (generated when you hit "Commit") linked to your chosen Submission(s).

Restart the Notebook to run your code again

In order to combat cheating, you are only allowed to call `make_env` or iterate through `iter_test` once per Notebook run. However, while you're iterating on your model it's reasonable to try something out, change the model a bit, and try it again. Unfortunately, if you try to simply re-run the code, or even refresh the browser page, you'll still be running on the same Notebook execution session you had been running before, and the `nflrush` module will still throw errors. To get around this, you need to explicitly restart your Notebook execution session, which you can do by **clicking "Run"->"Restart Session"** in the Notebook Editor's menu bar at the top.