In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
import datetime
from kaggle.competitions import nflrush
import tqdm
import re
from string import punctuation
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
import keras
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
from keras.utils import plot_model
import keras.backend as K
import tensorflow as tf

sns.set_style('darkgrid')
mpl.rcParams['figure.figsize'] = [15,10]
```
```
Using TensorFlow backend.
```

In [2]:
```python
env = nflrush.make_env()
```

In [3]:
```python
train = pd.read_csv('../input/nfl-big-data-bowl-2020/train.csv', dtype={'WindSpeed': 'object'})
```

Overall analysis

In [4]:
```python
train.head()
```

Out[4]:

	GameId	PlayId	Team	X	Y	S	A	Dis	Orientation	Dir	...	Week	Stadium	Location	StadiumType
0	2017090700	20170907000118	away	73.91	34.84	1.69	1.13	0.40	81.99	177.18	...	1	Gillette Stadium	Foxborough, MA	Outdoor
1	2017090700	20170907000118	away	74.67	32.64	0.42	1.35	0.01	27.61	198.70	...	1	Gillette Stadium	Foxborough, MA	Outdoor
2	2017090700	20170907000118	away	74.00	33.20	1.22	0.59	0.31	3.01	202.73	...	1	Gillette Stadium	Foxborough, MA	Outdoor
3	2017090700	20170907000118	away	71.46	27.70	0.42	0.54	0.02	359.77	105.64	...	1	Gillette Stadium	Foxborough, MA	Outdoor
4	2017090700	20170907000118	away	69.32	35.42	1.82	2.43	0.16	12.63	164.31	...	1	Gillette Stadium	Foxborough, MA	Outdoor

5 rows × 49 columns

Feature Engineering

In [5]:
```python
#from https://www.kaggle.com/prashantkikani/nfl-starter-lgb-feature-engg
train['DefendersInTheBox_vs_Distance'] = train['DefendersInTheBox'] / train['Distance']
```

Categorical features

In [6]:
```python
cat_features = []
for col in train.columns:
    if train[col].dtype =='object':
        cat_features.append((col, len(train[col].unique())))
```

In [7]:
```python
cat_features
```

Out[7]:
```
[('Team', 2),
 ('DisplayName', 2230),
 ('GameClock', 901),
 ('PossessionTeam', 32),
 ('FieldPosition', 33),
 ('OffenseFormation', 9),
 ('OffensePersonnel', 56),
 ('DefensePersonnel', 38),
 ('PlayDirection', 2),
 ('TimeHandoff', 22935),
 ('TimeSnap', 22943),
 ('PlayerHeight', 16),
 ('PlayerBirthDate', 1688),
 ('PlayerCollegeName', 301),
 ('Position', 25),
 ('HomeTeamAbbr', 32),
 ('VisitorTeamAbbr', 32),
 ('Stadium', 55),
 ('Location', 60),
 ('StadiumType', 30),
 ('Turf', 20),
 ('GameWeather', 62),
 ('WindSpeed', 41),
 ('WindDirection', 54)]
```

Let's preprocess some of those features.

Stadium Type

In [8]:
```python
train['StadiumType'].value_counts()
```

Out[8]:
```
Outdoor                   267696
Outdoors                   67474
Indoors                    40854
Dome                       17336
Indoor                     16148
Retractable Roof           15884
Open                        9614
Retr. Roof-Closed           7172
Retr. Roof - Closed         6446
Domed, closed               5918
Domed, open                 2684
Closed Dome                 2134
Domed                       1826
Dome, closed                1826
Oudoor                      1188
Retr. Roof Closed           1056
Indoor, Roof Closed         1056
Retr. Roof-Open              990
Bowl                         968
Outddors                     968
Heinz Field                  902
Retr. Roof - Open            880
Outdoor Retr Roof-Open       880
Indoor, Open Roof            858
Outdor                       858
Ourdoor                      858
Outside                      814
Domed, Open                  770
Cloudy                       770
Name: StadiumType, dtype: int64
```

We already can see some typos, let's fix them.

In [9]:
```python
def clean_StadiumType(txt):
    if pd.isna(txt):
        return np.nan
    txt = txt.lower()
    txt = ''.join([c for c in txt if c not in punctuation])
    txt = re.sub(' +', ' ', txt)
    txt = txt.strip()
    txt = txt.replace('outside', 'outdoor')
    txt = txt.replace('outdor', 'outdoor')
    txt = txt.replace('outddors', 'outdoor')
    txt = txt.replace('outdoors', 'outdoor')
    txt = txt.replace('oudoor', 'outdoor')
    txt = txt.replace('indoors', 'indoor')
    txt = txt.replace('ourdoor', 'outdoor')
    txt = txt.replace('retractable', 'rtr.')
    return txt
```

In [10]:
```python
train['StadiumType'] = train['StadiumType'].apply(clean_StadiumType)
```

By pareto's principle we are just going to focus on the words: outdoor, indoor, closed and open.

In [11]:
```python
def transform_StadiumType(txt):
    if pd.isna(txt):
        return np.nan
    if 'outdoor' in txt or 'open' in txt:
        return 1
    if 'indoor' in txt or 'closed' in txt:
        return 0

    return np.nan
```

In [12]:
```python
train['StadiumType'] = train['StadiumType'].apply(transform_StadiumType)
```

Turf

In [13]:
```python
#from https://www.kaggle.com/c/nfl-big-data-bowl-2020/discussion/112681#latest-649087
Turf = {'Field Turf':'Artificial', 'A-Turf Titan':'Artificial', 'Grass':'Natural', 'UBU Sports Speed S5-M':'Artificial',
        'Artificial':'Artificial', 'DD GrassMaster':'Artificial', 'Natural Grass':'Natural',
        'UBU Speed Series-S5-M':'Artificial', 'FieldTurf':'Artificial', 'FieldTurf 360':'Artificial', 'Natural grass':'Natural', 'grass':'Natural',
        'Natural':'Natural', 'Artifical':'Artificial', 'FieldTurf360':'Artificial', 'Naturall Grass':'Natural', 'Field turf':'Artificial',
        'SISGrass':'Artificial', 'Twenty-Four/Seven Turf':'Artificial', 'natural grass':'Natural'}

train['Turf'] = train['Turf'].map(Turf)
train['Turf'] = train['Turf'] == 'Natural'
```

Possession Team

In [14]:
```python
train[(train['PossessionTeam']!=train['HomeTeamAbbr']) & (train['PossessionTeam']!=train['VisitorTeamAbbr'])][['PossessionTeam', 'HomeTeamAbbr', 'VisitorTeamAbbr']]
```

Out[14]:

	PossessionTeam	HomeTeamAbbr	VisitorTeamAbbr
2992	BLT	CIN	BAL
2993	BLT	CIN	BAL
2994	BLT	CIN	BAL
2995	BLT	CIN	BAL
2996	BLT	CIN	BAL
...	...	...	...
509669	ARZ	SEA	ARI
509670	ARZ	SEA	ARI
509671	ARZ	SEA	ARI
509672	ARZ	SEA	ARI
509673	ARZ	SEA	ARI

63822 rows × 3 columns

We have some problem with the enconding of the teams such as BLT and BAL or ARZ and ARI.

Let's try to fix them manually.

In [15]:
```python
sorted(train['HomeTeamAbbr'].unique()) == sorted(train['VisitorTeamAbbr'].unique())
```

Out[15]:
```
True
```

In [16]:
```python
diff_abbr = []
for x,y  in zip(sorted(train['HomeTeamAbbr'].unique()), sorted(train['PossessionTeam'].unique())):
    if x!=y:
        print(x + " " + y)
```
```
ARI ARZ
BAL BLT
CLE CLV
HOU HST
```

Apparently these are the only three problems, let's fix it.

In [17]:
```python
map_abbr = {'ARI': 'ARZ', 'BAL': 'BLT', 'CLE': 'CLV', 'HOU': 'HST'}
for abb in train['PossessionTeam'].unique():
    map_abbr[abb] = abb
```

In [18]:
```python
train['PossessionTeam'] = train['PossessionTeam'].map(map_abbr)
train['HomeTeamAbbr'] = train['HomeTeamAbbr'].map(map_abbr)
train['VisitorTeamAbbr'] = train['VisitorTeamAbbr'].map(map_abbr)
```

In [19]:
```python
train['HomePossesion'] = train['PossessionTeam'] == train['HomeTeamAbbr']
```

In [20]:
```python
train['Field_eq_Possession'] = train['FieldPosition'] == train['PossessionTeam']
train['HomeField'] = train['FieldPosition'] == train['HomeTeamAbbr']
```

Offense formation

In [21]:
```python
off_form = train['OffenseFormation'].unique()
train['OffenseFormation'].value_counts()
```

Out[21]:
```
SINGLEBACK    225434
SHOTGUN       150964
I_FORM        106062
PISTOL         13420
JUMBO          11462
WILDCAT         1782
EMPTY            506
ACE               22
Name: OffenseFormation, dtype: int64
```

Since I don't have any knowledge about formations, I am just goig to one-hot encode this feature

In [22]:
```python
train = pd.concat([train.drop(['OffenseFormation'], axis=1), pd.get_dummies(train['OffenseFormation'], prefix='Formation')], axis=1)
dummy_col = train.columns
```

Game Clock

Game clock is supposed to be a numerical feature.

In [23]:
```python
train['GameClock'].value_counts()
```

Out[23]:
```
15:00:00    14476
02:00:00     5236
14:54:00     2156
14:55:00     1958
14:56:00     1276
            ...
00:01:00      110
14:58:00       88
14:59:00       66
14:39:00       44
00:00:00       22
Name: GameClock, Length: 901, dtype: int64
```

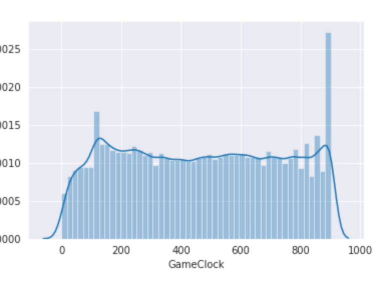
Since we already have the quarter feature, we can just divide the Game Clock by 15 minutes so we can get the normalized time left in the quarter.

In [24]:
```python
def strtoseconds(txt):
    txt = txt.split(':')
    ans = int(txt[0])*60 + int(txt[1]) + int(txt[2])/60
    return ans
```

In [25]:
```python
train['GameClock'] = train['GameClock'].apply(strtoseconds)
```

In [26]:
```python
sns.distplot(train['GameClock'])
```

Out[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f109ffc80f0>
```



Player height

In [27]:
```python
train['PlayerHeight']
```

Out[27]:
```
0          6-0
1          6-3
2          6-3
3          6-3
4          6-0
          ...
509757     6-6
509758     6-5
509759     6-5
509760     6-6
509761    5-11
Name: PlayerHeight, Length: 509762, dtype: object
```

We know that 1ft=12in, thus:

In [28]:
```python
train['PlayerHeight'] = train['PlayerHeight'].apply(lambda x: 12*int(x.split('-')[0])+int(x.split('-')[1]))
```

In [29]:
```python
train['PlayerBMI'] = 703*(train['PlayerWeight']/(train['PlayerHeight'])**2)
```

Time handoff and snap and Player BirthDate

In [30]:
```python
train['TimeHandoff']
```

Out[30]:
```
0         2017-09-08T00:44:06.000Z
1         2017-09-08T00:44:06.000Z
2         2017-09-08T00:44:06.000Z
3         2017-09-08T00:44:06.000Z
4         2017-09-08T00:44:06.000Z
                    ...
509757    2018-12-31T00:24:51.000Z
509758    2018-12-31T00:24:51.000Z
509759    2018-12-31T00:24:51.000Z
509760    2018-12-31T00:24:51.000Z
509761    2018-12-31T00:24:51.000Z
Name: TimeHandoff, Length: 509762, dtype: object
```

In [31]:
```python
train['TimeHandoff'] = train['TimeHandoff'].apply(lambda x: datetime.datetime.strptime(x, "%Y-%m-%dT%H:%M:%S.%fZ"))
train['TimeSnap'] = train['TimeSnap'].apply(lambda x: datetime.datetime.strptime(x, "%Y-%m-%dT%H:%M:%S.%fZ"))
```

In [32]:
```python
train['TimeDelta'] = train.apply(lambda row: (row['TimeHandoff'] - row['TimeSnap']).total_seconds(), axis=1)
```

In [33]:
```python
train['PlayerBirthDate'] = train['PlayerBirthDate'].apply(lambda x: datetime.datetime.strptime(x, "%m/%d/%Y"))
```

Let's use the time handoff to calculate the players age

In [34]:
```python
seconds_in_year = 60*60*24*365.25
train['PlayerAge'] = train.apply(lambda row: (row['TimeHandoff']-row['PlayerBirthDate']).total_seconds()/seconds_in_year, axis=1)
```

In [35]:
```python
train = train.drop(['TimeHandoff', 'TimeSnap', 'PlayerBirthDate'], axis=1)
```

Wind Speed and Direction

In [36]:
```python
train['WindSpeed'].value_counts()
```

Out[36]:
```
5                     53284
6                     41580
7                     39578
4                     34584
9                     31328
10                    29788
8                     29370
3                     26862
2                     24112
12                    23584
11                    17116
15                    13926
0                     13772
1                     12078
16                     9878
13                     8404
14                     6094
17                     3872
18                     1980
13 MPH                 1804
23                     1166
E                      1144
SE                     1122
10-20                  1100
7 MPH                  1100
Calm                   1100
12-22                  1056
6 mph                  1034
20                     1012
12mph                   968
24                      968
14-23                   968
10MPH                   902
4 MPh                   902
10mph                   902
15 gusts up to 25       836
SSW                     836
22                      836
11-17                   726
19                      660
Name: WindSpeed, dtype: int64
```

We can see there are some values that are not standardized(e.g. 12mph), we are going to remove mph from all our values.

In [37]:
```python
train['WindSpeed'] = train['WindSpeed'].apply(lambda x: x.lower().replace('mph', '').strip() if not pd.isna(x) else x)
```

In [38]:
```python
train['WindSpeed'].value_counts()
```

Out[38]:
```
5                     53284
6                     42614
7                     40678
4                     35486
10                    31592
9                     31328
8                     29370
3                     26862
12                    24552
2                     24112
11                    17116
15                    13926
0                     13772
1                     12078
13                    10208
16                     9878
14                     6094
17                     3872
18                     1980
23                     1166
e                      1144
se                     1122
calm                   1100
10-20                  1100
12-22                  1056
20                     1012
24                      968
14-23                   968
15 gusts up to 25       836
22                      836
ssw                     836
11-17                   726
19                      660
Name: WindSpeed, dtype: int64
```

In [39]:
```python
#let's replace the ones that has x-y by (x+y)/2
# and also the ones with x gusts up to y
train['WindSpeed'] = train['WindSpeed'].apply(lambda x: (int(x.split('-')[0])+int(x.split('-')[1]))/2 if not pd.isna(x) and '-' in x else x)
train['WindSpeed'] = train['WindSpeed'].apply(lambda x: (int(x.split()[0])+int(x.split()[-1]))/2 if not pd.isna(x) and type(x)!=float and 'gusts up to' in x else x)
```

In [40]:
```python
def str_to_float(txt):
    try:
        return float(txt)
    except:
        return -1
```

In [41]:
```python
train['WindSpeed'] = train['WindSpeed'].apply(str_to_float)
```

In [42]:
```python
train['WindDirection'].value_counts()
```

Out[42]:
```
NE                   30250
NW                   27236
SW                   25828
SE                   25784
WSW                  24222
N                    23188
W                    22198
S                    21384
NNE                  20394
South                20328
SSW                  19910
WNW                  19118
North                17182
NNW                  14036
West                 13618
SSE                  13376
E                    12826
ENE                  10802
ESE                   9878
East                  7348
Northwest             4070
From SW               3872
Northeast             3652
NorthEast             3212
From S                3146
s                     2728
SouthWest             2134
Southeast             1936
W-NW                  1804
Southwest             1804
West-Southwest        1386
8                     1144
1                     1122
North East            1100
East Southeast        1078
West Northwest        1056
Calm                  1056
N-NE                  1012
From W                 990
W-SW                   968
From NNE               968
North/Northwest        968
From SSW               968
From WSW               946
South Southeast        946
From NNW               924
EAST                   902
From SSE               880
South Southwest        858
13                     836
East North East        748
from W                 726
From ESE               682
Name: WindDirection, dtype: int64
```

In [43]:
```python
def clean_WindDirection(txt):
    if pd.isna(txt):
        return np.nan
    txt = txt.lower()
    txt = ''.join([c for c in txt if c not in punctuation])
    txt = txt.replace('from', '')
    txt = txt.replace(' ', '')
    txt = txt.replace('north', 'n')
    txt = txt.replace('south', 's')
    txt = txt.replace('west', 'w')
    txt = txt.replace('east', 'e')
    return txt
```

In [44]:
```python
train['WindDirection'] = train['WindDirection'].apply(clean_WindDirection)
```

In [45]:
```python
train['WindDirection'].value_counts()
```

Out[45]:
```
s       47586
n       40370
ne      38214
w       37532
sw      33638
nw      31306
se      27720
wsw     27522
nne     22374
wnw     21978
ssw     21736
e       21076
nnw     15928
sse     15202
ese     11638
ene     11550
8        1144
1        1122
calm     1056
13        836
Name: WindDirection, dtype: int64
```

In [46]:
```python
def transform_WindDirection(txt):
    if pd.isna(txt):
        return np.nan

    if txt=='n':
        return 0
    if txt=='nne' or txt=='nen':
        return 1/8
    if txt=='ne':
        return 2/8
    if txt=='ene' or txt=='nee':
        return 3/8
    if txt=='e':
        return 4/8
    if txt=='ese' or txt=='see':
        return 5/8
    if txt=='se':
        return 6/8
    if txt=='ses' or txt=='sse':
        return 7/8
    if txt=='s':
        return 8/8
    if txt=='ssw' or txt=='sws':
        return 9/8
    if txt=='sw':
        return 10/8
    if txt=='sww' or txt=='wsw':
        return 11/8
    if txt=='w':
        return 12/8
    if txt=='wnw' or txt=='nww':
        return 13/8
    if txt=='nw':
        return 14/8
    if txt=='nwn' or txt=='nnw':
        return 15/8
    return np.nan
```

In [47]:
```python
train['WindDirection'] = train['WindDirection'].apply(transform_WindDirection)
```

PlayDirection

In [48]:
```python
train['PlayDirection'].value_counts()
```

Out[48]:
```
left     256454
right    253308
Name: PlayDirection, dtype: int64
```

In [49]:
```python
train['PlayDirection'] = train['PlayDirection'].apply(lambda x: x.strip() == 'right')
```

Team

In [50]:
```python
train['Team'] = train['Team'].apply(lambda x: x.strip()=='home')
```

Game Weather

In [51]:
```python
train['GameWeather'].unique()
```

Out[51]:
```
array(['Clear and warm', 'Sun & clouds', 'Sunny', 'Controlled Climate',
       'Mostly Sunny', 'Clear', nan, 'Indoor', 'Mostly Cloudy',
       'Mostly Coudy', 'Partly sunny', 'Partly Cloudy', 'Cloudy',
       'Sunny, highs to upper 80s', 'Indoors', 'Light Rain', 'Showers',
       'Partly cloudy', 'Partly Sunny', '30% Chance of Rain',
       'Cloudy with periods of rain, thunder possible. Winds shifting to WNW, 10-20 mph.',
       'Rain', 'Cloudy, fog started developing in 2nd quarter', 'Coudy',
       'Rain likely, temps in low 40s.', 'Cold', 'N/A (Indoors)',
       'Clear skies', 'cloudy', 'Fair', 'Mostly cloudy',
       'Cloudy, chance of rain', 'Heavy lake effect snow', 'Party Cloudy',
       'Cloudy, light snow accumulating 1-3"', 'Cloudy and cold', 'Snow',
       'Hazy', 'Scattered Showers', 'Cloudy and Cool', 'N/A Indoor',
       'Rain Chance 40%', 'Clear and sunny', 'Mostly sunny',
       'Sunny and warm', 'Partly clear', 'Cloudy, 50% change of rain',
       'Clear and Sunny', 'Sunny, Windy', 'Clear and Cool',
       'Sunny and clear', 'Mostly Sunny Skies', 'Partly Clouidy',
       'Clear Skies', 'Sunny Skies', 'Overcast',
       'T: 51; H: 55; W: NW 10 mph', 'Cloudy, Rain', 'Rain shower',
       'Clear and cold', 'Rainy', 'Sunny and cold'], dtype=object)
```

We are going to apply the following preprocessing:

- Lower case
- N/A Indoor, N/A (Indoors) and Indoor => indoor Let's try to cluster those together.
- coudy and clouidy => cloudy
- party => partly
- sunny and clear => clear and sunny
- skies and mostly => ""

In [52]:
```python
train['GameWeather'] = train['GameWeather'].str.lower()
indoor = "indoor"
train['GameWeather'] = train['GameWeather'].apply(lambda x: indoor if not pd.isna(x) and indoor in x else x)
train['GameWeather'] = train['GameWeather'].apply(lambda x: x.replace('coudy', 'cloudy').replace('clouidy', 'cloudy').replace('party', 'partly') if not pd.isna(x) else x)
train['GameWeather'] = train['GameWeather'].apply(lambda x: x.replace('clear and sunny', 'sunny and clear') if not pd.isna(x) else x)
train['GameWeather'] = train['GameWeather'].apply(lambda x: x.replace('skies', '').replace("mostly", "").strip() if not pd.isna(x) else x)
```

In [53]:
```python
train['GameWeather'].unique()
```

Out[53]:
```
array(['clear and warm', 'sun & clouds', 'sunny', 'controlled climate',
       'clear', nan, 'indoor', 'cloudy', 'partly sunny', 'partly cloudy',
       'sunny, highs to upper 80s', 'light rain', 'showers',
       '30% chance of rain',
       'cloudy with periods of rain, thunder possible. winds shifting to wnw, 10-20 mph.',
       'rain', 'cloudy, fog started developing in 2nd quarter',
       'rain likely, temps in low 40s.', 'cold', 'fair',
       'cloudy, chance of rain', 'heavy lake effect snow',
       'cloudy, light snow accumulating 1-3"', 'cloudy and cold', 'snow',
       'hazy', 'scattered showers', 'cloudy and cool', 'rain chance 40%',
       'sunny and clear', 'sunny and warm', 'partly clear',
       'cloudy, 50% change of rain', 'sunny, windy', 'clear and cool',
       'overcast', 't: 51; h: 55; w: nw 10 mph', 'cloudy, rain',
       'rain shower', 'clear and cold', 'rainy', 'sunny and cold'],
      dtype=object)
```

Let's now look at the most common words we have in the weather description

In [54]:
```python
from collections import Counter
weather_count = Counter()
for weather in train['GameWeather']:
    if pd.isna(weather):
        continue
    for word in weather.split():
        weather_count[word]+=1

weather_count.most_common()[:15]
```

Out[54]:
```
[('cloudy', 193952),
 ('sunny', 127468),
 ('partly', 58256),
 ('clear', 55594),
 ('rain', 28952),
 ('indoor', 26950),
 ('controlled', 12540),
 ('climate', 12540),
 ('and', 10956),
 ('cloudy,', 4972),
 ('fair', 4972),
 ('snow', 4708),
 ('cold', 4510),
 ('of', 4026),
 ('light', 3608)]
```

To encode our weather we are going to do the following map:

- climate controlled or indoor => 3, sunny or sun => 2, clear => 1, cloudy => -1, rain => -2, snow => -3, others => 0
- partly => multiply by 0.5

I don't have any expercience with american football so I don't know if playing in a climate controlled or indoor stadium is good or not, if someone has a good idea on how to encode this it would be nice to leave it in the comments :)

In [55]:
```python
def map_weather(txt):
    ans = 1
    if pd.isna(txt):
        return 0
    if 'partly' in txt:
        ans*=0.5
    if 'climate controlled' in txt or 'indoor' in txt:
        return ans*3
    if 'sunny' in txt or 'sun' in txt:
        return ans*2
    if 'clear' in txt:
        return ans
    if 'cloudy' in txt:
        return -ans
    if 'rain' in txt or 'rainy' in txt:
        return -2*ans
    if 'snow' in txt:
        return -3*ans
    return 0
```

In [56]:
```python
train['GameWeather'] = train['GameWeather'].apply(map_weather)
```

NflId NflIdRusher

In [57]:
```python
train['IsRusher'] = train['NflId'] == train['NflIdRusher']
```

In [58]:
```python
train.drop(['NflId', 'NflIdRusher'], axis=1, inplace=True)
```

PlayDirection problems

As we can see, we have a problem if some features such as X and Y because of the play direction, let's fix those issues

X, orientation and direction

In [59]:
```python
train['X'] = train.apply(lambda row: row['X'] if row['PlayDirection'] else 120-row['X'], axis=1)
```

In [60]:
```python
#from https://www.kaggle.com/scirpus/hybrid-gp-and-nn
def new_orientation(angle, play_direction):
    if play_direction == 0:
        new_angle = 360.0 - angle
        if new_angle == 360.0:
            new_angle = 0.0
        return new_angle
    else:
        return angle

train['Orientation'] = train.apply(lambda row: new_orientation(row['Orientation'], row['PlayDirection']), axis=1)
train['Dir'] = train.apply(lambda row: new_orientation(row['Dir'], row['PlayDirection']), axis=1)
```

YardsLeft

Let's compute how many yards are left to the end-zone.

In [61]:
```python
train['YardsLeft'] = train.apply(lambda row: 100-row['YardLine'] if row['HomeField'] else row['YardLine'], axis=1)
train['YardsLeft'] = train.apply(lambda row: row['YardsLeft'] if row['PlayDirection'] else 100-row['YardsLeft'], axis=1)
```

In [62]:
```python
((train['YardsLeft']<train['Yards']) | (train['YardsLeft']-100>train['Yards'])).mean()
```

Out[62]:
```
0.009710413879418239
```

Clearly: Yards<=YardsLeft and YardsLeft-100<=Yards, thus we are going to drop those wrong lines.

In [63]:
```
train.drop(train.index[(train['YardsLeft']<train['Yards']) | (train['YardsLeft']-100>train['Yards'])], inplace=True)
```

Baseline model

Let's drop the categorical features and run a simple random forest in our model

In [64]:
```
train = train.sort_values(by=['PlayId', 'Team', 'IsRusher', 'JerseyNumber']).reset_index()
```

In [65]:
```
train.drop(['GameId', 'PlayId', 'index', 'IsRusher', 'Team'], axis=1, inplace=True)
```

In [66]:
```
cat_features = []
for col in train.columns:
    if train[col].dtype =='object':
        cat_features.append(col)

train = train.drop(cat_features, axis=1)
```

We are now going to make one big row for each play where the rusher is the last one

In [67]:
```
train.fillna(-999, inplace=True)
```

In [68]:
```
players_col = []
for col in train.columns:
    if train[col][:22].std()!=0:
        players_col.append(col)
```

In [69]:
```
X_train = np.array(train[players_col]).reshape(-1, len(players_col)*22)
```

In [70]:
```
play_col = train.drop(players_col+['Yards'], axis=1).columns
X_play_col = np.zeros(shape=(X_train.shape[0], len(play_col)))
for i, col in enumerate(play_col):
    X_play_col[:, i] = train[col][::22]
```

In [71]:
```
X_train = np.concatenate([X_train, X_play_col], axis=1)
y_train = np.zeros(shape=(X_train.shape[0], 199))
for i,yard in enumerate(train['Yards'][::22]):
    y_train[i, yard+99:] = np.ones(shape=(1, 100-yard))
```

In [72]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

In [73]:
```
batch_size=64
```

In [74]:
```
class RAdam(keras.optimizers.Optimizer):
    """RAdam optimizer.
    # Arguments
        learning_rate: float >= 0. Learning rate.
        beta_1: float, 0 < beta < 1. Generally close to 1.
        beta_2: float, 0 < beta < 1. Generally close to 1.
        epsilon: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
        decay: float >= 0. Learning rate decay over each update.
        weight_decay: float >= 0. Weight decay for each param.
        amsgrad: boolean. Whether to apply the AMSGrad variant of this
            algorithm from the paper "On the Convergence of Adam and
            Beyond".
        total_steps: int >= 0. Total number of training steps. Enable warmup by setting a positive valu
e.
        warmup_proportion: 0 < warmup_proportion < 1. The proportion of increasing steps.
        min_lr: float >= 0. Minimum learning rate after warmup.
    # References
        - [Adam - A Method for Stochastic Optimization](https://arxiv.org/abs/1412.6980v8)
        - [On the Convergence of Adam and Beyond](https://openreview.net/forum?id=ryQu7f-RZ)
        - [On The Variance Of The Adaptive Learning Rate And Beyond](https://arxiv.org/pdf/1908.03265v
1.pdf)
    """

    def __init__(self, learning_rate=0.001, beta_1=0.9, beta_2=0.999,
                 epsilon=None, decay=0., weight_decay=0., amsgrad=False,
                 total_steps=0, warmup_proportion=0.1, min_lr=0., **kwargs):
        learning_rate = kwargs.pop('lr', learning_rate)
        super(RAdam, self).__init__(**kwargs)
        with K.name_scope(self.__class__.__name__):
            self.iterations = K.variable(0, dtype='int64', name='iterations')
            self.learning_rate = K.variable(learning_rate, name='learning_rate')
            self.beta_1 = K.variable(beta_1, name='beta_1')
            self.beta_2 = K.variable(beta_2, name='beta_2')
            self.decay = K.variable(decay, name='decay')
            self.weight_decay = K.variable(weight_decay, name='weight_decay')
            self.total_steps = K.variable(total_steps, name='total_steps')
            self.warmup_proportion = K.variable(warmup_proportion, name='warmup_proportion')
            self.min_lr = K.variable(min_lr, name='min_lr')
        if epsilon is None:
            epsilon = K.epsilon()
        self.epsilon = epsilon
        self.initial_decay = decay
        self.initial_weight_decay = weight_decay
        self.initial_total_steps = total_steps
        self.amsgrad = amsgrad

    def get_updates(self, loss, params):
        grads = self.get_gradients(loss, params)
        self.updates = [K.update_add(self.iterations, 1)]

        lr = self.lr

        if self.initial_decay > 0:
            lr = lr * (1. / (1. + self.decay * K.cast(self.iterations, K.dtype(self.decay))))

        t = K.cast(self.iterations, K.floatx()) + 1

        if self.initial_total_steps > 0:
            warmup_steps = self.total_steps * self.warmup_proportion
            decay_steps = K.maximum(self.total_steps - warmup_steps, 1)
            decay_rate = (self.min_lr - lr) / decay_steps
            lr = K.switch(
                t <= warmup_steps,
                lr * (t / warmup_steps),
                lr + decay_rate * K.minimum(t - warmup_steps, decay_steps),
            )

        ms = [K.zeros(K.int_shape(p), dtype=K.dtype(p), name='m_' + str(i)) for (i, p) in enumerate(par
ams)]
        vs = [K.zeros(K.int_shape(p), dtype=K.dtype(p), name='v_' + str(i)) for (i, p) in enumerate(par
ams)]

        if self.amsgrad:
            vhats = [K.zeros(K.int_shape(p), dtype=K.dtype(p), name='vhat_' + str(i)) for (i, p) in enu
merate(params)]
        else:
            vhats = [K.zeros(1, name='vhat_' + str(i)) for i in range(len(params))]

        self.weights = [self.iterations] + ms + vs + vhats

        beta_1_t = K.pow(self.beta_1, t)
        beta_2_t = K.pow(self.beta_2, t)

        sma_inf = 2.0 / (1.0 - self.beta_2) - 1.0
        sma_t = sma_inf - 2.0 * t * beta_2_t / (1.0 - beta_2_t)

        for p, g, m, v, vhat in zip(params, grads, ms, vs, vhats):
            m_t = (self.beta_1 * m) + (1. - self.beta_1) * g
            v_t = (self.beta_2 * v) + (1. - self.beta_2) * K.square(g)

            m_corr_t = m_t / (1.0 - beta_1_t)
            if self.amsgrad:
                vhat_t = K.maximum(vhat, v_t)
                v_corr_t = K.sqrt(vhat_t / (1.0 - beta_2_t))
                self.updates.append(K.update(vhat, vhat_t))
            else:
                v_corr_t = K.sqrt(v_t / (1.0 - beta_2_t))

            r_t = K.sqrt((sma_t - 4.0) / (sma_inf - 4.0) *
                         (sma_t - 2.0) / (sma_inf - 2.0) *
                         sma_inf / sma_t)

            p_t = K.switch(sma_t >= 5, r_t * m_corr_t / (v_corr_t + self.epsilon), m_corr_t)

            if self.initial_weight_decay > 0:
                p_t += self.weight_decay * p

            p_t = p - lr * p_t

            self.updates.append(K.update(m, m_t))
            self.updates.append(K.update(v, v_t))
            new_p = p_t

            # Apply constraints.
            if getattr(p, 'constraint', None) is not None:
                new_p = p.constraint(new_p)

            self.updates.append(K.update(p, new_p))
        return self.updates

    @property
    def lr(self):
        return self.learning_rate

    @lr.setter
    def lr(self, learning_rate):
        self.learning_rate = learning_rate

    def get_config(self):
        config = {
            'learning_rate': float(K.get_value(self.learning_rate)),
            'beta_1': float(K.get_value(self.beta_1)),
            'beta_2': float(K.get_value(self.beta_2)),
            'decay': float(K.get_value(self.decay)),
            'weight_decay': float(K.get_value(self.weight_decay)),
            'epsilon': self.epsilon,
            'amsgrad': self.amsgrad,
            'total_steps': float(K.get_value(self.total_steps)),
            'warmup_proportion': float(K.get_value(self.warmup_proportion)),
            'min_lr': float(K.get_value(self.min_lr)),
        }
        base_config = super(RAdam, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))
```

In [75]:
```
#from https://www.kaggle.com/davidcairuz/nfl-neural-network-w-softmax
def crps(y_true, y_pred):
    return K.mean(K.square(y_true - K.cumsum(y_pred, axis=1)), axis=1)
```

In [76]:
```
def get_model():
    x = keras.layers.Input(shape=[X_train.shape[1]])
    fc1 = keras.layers.Dense(units=450, input_shape=[X_train.shape[1]])(x)
    act1 = keras.layers.PReLU()(fc1)
    bn1 = keras.layers.BatchNormalization()(act1)
    dp1 = keras.layers.Dropout(0.55)(bn1)
    gn1 = keras.layers.GaussianNoise(0.15)(dp1)
    concat1 = keras.layers.Concatenate()([x, gn1])
    fc2 = keras.layers.Dense(units=600)(concat1)
    act2 = keras.layers.PReLU()(fc2)
    bn2 = keras.layers.BatchNormalization()(act2)
    dp2 = keras.layers.Dropout(0.55)(bn2)
    gn2 = keras.layers.GaussianNoise(0.15)(dp2)
    concat2 = keras.layers.Concatenate()([concat1, gn2])
    fc3 = keras.layers.Dense(units=400)(concat2)
    act3 = keras.layers.PReLU()(fc3)
    bn3 = keras.layers.BatchNormalization()(act3)
    dp3 = keras.layers.Dropout(0.55)(bn3)
    gn3 = keras.layers.GaussianNoise(0.15)(dp3)
    concat3 = keras.layers.Concatenate([concat2, gn3])
    output = keras.layers.Dense(units=199, activation='softmax')(concat2)
    model = keras.models.Model(inputs=[x], outputs=[output])
    return model


def train_model(X_train, y_train, X_val, y_val):
    model = get_model()
    model.compile(optimizer=RAdam(warmup_proportion=0.1, min_lr=1e-7), loss=crps)
    er = EarlyStopping(patience=20, min_delta=1e-4, restore_best_weights=True, monitor='val_loss')
    model.fit(X_train, y_train, epochs=200, callbacks=[er], validation_data=[X_val, y_val], batch_size=
batch_size)
    return model
```

In [77]:
```
from sklearn.model_selection import RepeatedKFold

rkf = RepeatedKFold(n_splits=5, n_repeats=5)

models = []

for tr_idx, vl_idx in rkf.split(X_train, y_train):

    x_tr, y_tr = X_train[tr_idx], y_train[tr_idx]
    x_vl, y_vl = X_train[vl_idx], y_train[vl_idx]

    model = train_model(x_tr, y_tr, x_vl, y_vl)
    models.append(model)
```

```
Train on 18356 samples, validate on 4590 samples
Epoch 1/200
18356/18356 [==============================] - 7s 397us/step - loss: 0.0338 - val_loss: 0.0157
Epoch 2/200
18356/18356 [==============================] - 6s 341us/step - loss: 0.0157 - val_loss: 0.0148
Epoch 3/200
18356/18356 [==============================] - 6s 345us/step - loss: 0.0151 - val_loss: 0.0143
Epoch 4/200
18356/18356 [==============================] - 6s 333us/step - loss: 0.0146 - val_loss: 0.0142
Epoch 5/200
18356/18356 [==============================] - 6s 335us/step - loss: 0.0143 - val_loss: 0.0141
Epoch 6/200
18356/18356 [==============================] - 7s 359us/step - loss: 0.0141 - val_loss: 0.0140
Epoch 7/200
18356/18356 [==============================] - 6s 347us/step - loss: 0.0139 - val_loss: 0.0140
Epoch 8/200
18356/18356 [==============================] - 6s 334us/step - loss: 0.0137 - val_loss: 0.0140
Epoch 9/200
18356/18356 [==============================] - 6s 337us/step - loss: 0.0136 - val_loss: 0.0140
Epoch 10/200
18356/18356 [==============================] - 6s 336us/step - loss: 0.0135 - val_loss: 0.0138
Epoch 11/200
18356/18356 [==============================] - 6s 333us/step - loss: 0.0134 - val_loss: 0.0139
Epoch 12/200
18356/18356 [==============================] - 6s 333us/step - loss: 0.0132 - val_loss: 0.0139
Epoch 13/200
18356/18356 [==============================] - 6s 328us/step - loss: 0.0131 - val_loss: 0.0138
Epoch 14/200
18356/18356 [==============================] - 6s 328us/step - loss: 0.0130 - val_loss: 0.0139
Epoch 15/200
18356/18356 [==============================] - 6s 331us/step - loss: 0.0129 - val_loss: 0.0139
Epoch 16/200
18356/18356 [==============================] - 6s 333us/step - loss: 0.0127 - val_loss: 0.0139
Epoch 17/200
18356/18356 [==============================] - 6s 329us/step - loss: 0.0125 - val_loss: 0.0138
Epoch 18/200
18356/18356 [==============================] - 6s 329us/step - loss: 0.0124 - val_loss: 0.0139
Epoch 19/200
18356/18356 [==============================] - 6s 329us/step - loss: 0.0122 - val_loss: 0.0140
Epoch 20/200
18356/18356 [==============================] - 6s 331us/step - loss: 0.0120 - val_loss: 0.0139
Epoch 21/200
18356/18356 [==============================] - 6s 351us/step - loss: 0.0118 - val_loss: 0.0140
Epoch 22/200
18356/18356 [==============================] - 6s 333us/step - loss: 0.0116 - val_loss: 0.0140
Epoch 23/200
18356/18356 [==============================] - 6s 328us/step - loss: 0.0114 - val_loss: 0.0139
Epoch 24/200
18356/18356 [==============================] - 6s 330us/step - loss: 0.0112 - val_loss: 0.0140
Epoch 25/200
18356/18356 [==============================] - 6s 332us/step - loss: 0.0111 - val_loss: 0.0140
Epoch 26/200
18356/18356 [==============================] - 6s 334us/step - loss: 0.0109 - val_loss: 0.0141
Epoch 27/200
18356/18356 [==============================] - 6s 333us/step - loss: 0.0107 - val_loss: 0.0142
Epoch 28/200
18356/18356 [==============================] - 6s 328us/step - loss: 0.0105 - val_loss: 0.0142
Epoch 29/200
18356/18356 [==============================] - 6s 333us/step - loss: 0.0104 - val_loss: 0.0141
Epoch 30/200
18356/18356 [==============================] - 6s 338us/step - loss: 0.0102 - val_loss: 0.0142
Train on 18357 samples, validate on 4589 samples
Epoch 1/200
18357/18357 [==============================] - 8s 411us/step - loss: 0.0339 - val_loss: 0.0161
Epoch 2/200
18357/18357 [==============================] - 6s 351us/step - loss: 0.0163 - val_loss: 0.0144
Epoch 3/200
18357/18357 [==============================] - 6s 353us/step - loss: 0.0150 - val_loss: 0.0139
Epoch 4/200
18357/18357 [==============================] - 6s 354us/step - loss: 0.0145 - val_loss: 0.0138
Epoch 5/200
18357/18357 [==============================] - 6s 350us/step - loss: 0.0142 - val_loss: 0.0138
Epoch 6/200
18357/18357 [==============================] - 6s 349us/step - loss: 0.0141 - val_loss: 0.0136
Epoch 7/200
18357/18357 [==============================] - 6s 351us/step - loss: 0.0138 - val_loss: 0.0137
Epoch 8/200
18357/18357 [==============================] - 6s 351us/step - loss: 0.0136 - val_loss: 0.0136
Epoch 9/200
18357/18357 [==============================] - 7s 386us/step - loss: 0.0136 - val_loss: 0.0136
Epoch 10/200
18357/18357 [==============================] - 7s 370us/step - loss: 0.0134 - val_loss: 0.0137
Epoch 11/200
18357/18357 [==============================] - 7s 376us/step - loss: 0.0133 - val_loss: 0.0135
Epoch 12/200
18357/18357 [==============================] - 7s 364us/step - loss: 0.0132 - val_loss: 0.0136
Epoch 13/200
18357/18357 [==============================] - 7s 366us/step - loss: 0.0130 - val_loss: 0.0136
Epoch 14/200
18357/18357 [==============================] - 7s 369us/step - loss: 0.0129 - val_loss: 0.0137
Epoch 15/200
18357/18357 [==============================] - 7s 358us/step - loss: 0.0127 - val_loss: 0.0137
Epoch 16/200
18357/18357 [==============================] - 7s 361us/step - loss: 0.0126 - val_loss: 0.0135
Epoch 17/200
18357/18357 [==============================] - 7s 354us/step - loss: 0.0123 - val_loss: 0.0136
Epoch 18/200
18357/18357 [==============================] - 6s 353us/step - loss: 0.0122 - val_loss: 0.0137
Epoch 19/200
18357/18357 [==============================] - 7s 361us/step - loss: 0.0120 - val_loss: 0.0136
Epoch 20/200
18357/18357 [==============================] - 7s 359us/step - loss: 0.0118 - val_loss: 0.0137
Epoch 21/200
18357/18357 [==============================] - 7s 361us/step - loss: 0.0117 - val_loss: 0.0140
Epoch 22/200
18357/18357 [==============================] - 7s 362us/step - loss: 0.0114 - val_loss: 0.0138
Epoch 23/200
18357/18357 [==============================] - 7s 358us/step - loss: 0.0112 - val_loss: 0.0138
Epoch 24/200
18357/18357 [==============================] - 7s 380us/step - loss: 0.0111 - val_loss: 0.0139
Epoch 25/200
18357/18357 [==============================] - 7s 376us/step - loss: 0.0108 - val_loss: 0.0139
Epoch 26/200
18357/18357 [==============================] - 6s 352us/step - loss: 0.0106 - val_loss: 0.0140
Train on 18357 samples, validate on 4589 samples
Epoch 1/200
18357/18357 [==============================] - 8s 427us/step - loss: 0.0329 - val_loss: 0.0161
Epoch 2/200
18357/18357 [==============================] - 7s 370us/step - loss: 0.0160 - val_loss: 0.0142
Epoch 3/200
18357/18357 [==============================] - 7s 376us/step - loss: 0.0150 - val_loss: 0.0138
Epoch 4/200
18357/18357 [==============================] - 7s 375us/step - loss: 0.0147 - val_loss: 0.0137
Epoch 5/200
18357/18357 [==============================] - 7s 372us/step - loss: 0.0143 - val_loss: 0.0136
Epoch 6/200
18357/18357 [==============================] - 7s 373us/step - loss: 0.0140 - val_loss: 0.0133
Epoch 7/200
18357/18357 [==============================] - 7s 375us/step - loss: 0.0139 - val_loss: 0.0133
Epoch 8/200
18357/18357 [==============================] - 7s 383us/step - loss: 0.0137 - val_loss: 0.0132
Epoch 9/200
18357/18357 [==============================] - 7s 389us/step - loss: 0.0135 - val_loss: 0.0133
Epoch 10/200
18357/18357 [==============================] - 7s 406us/step - loss: 0.0135 - val_loss: 0.0133
Epoch 11/200
18357/18357 [==============================] - 7s 389us/step - loss: 0.0134 - val_loss: 0.0132
Epoch 12/200
18357/18357 [==============================] - 7s 374us/step - loss: 0.0132 - val_loss: 0.0134
Epoch 13/200
18357/18357 [==============================] - 7s 374us/step - loss: 0.0131 - val_loss: 0.0133
Epoch 14/200
18357/18357 [==============================] - 7s 379us/step - loss: 0.0130 - val_loss: 0.0132
Epoch 15/200
18357/18357 [==============================] - 7s 379us/step - loss: 0.0128 - val_loss: 0.0132
Epoch 16/200
18357/18357 [==============================] - 7s 378us/step - loss: 0.0126 - val_loss: 0.0133
Epoch 17/200
18357/18357 [==============================] - 7s 373us/step - loss: 0.0125 - val_loss: 0.0132
Epoch 18/200
18357/18357 [==============================] - 7s 367us/step - loss: 0.0123 - val_loss: 0.0132
Epoch 19/200
18357/18357 [==============================] - 7s 367us/step - loss: 0.0121 - val_loss: 0.0134
Epoch 20/200
18357/18357 [==============================] - 7s 370us/step - loss: 0.0119 - val_loss: 0.0134
Epoch 21/200
18357/18357 [==============================] - 7s 372us/step - loss: 0.0118 - val_loss: 0.0136
Epoch 22/200
18357/18357 [==============================] - 7s 366us/step - loss: 0.0116 - val_loss: 0.0135
Epoch 23/200
18357/18357 [==============================] - 7s 365us/step - loss: 0.0113 - val_loss: 0.0136
Epoch 24/200
18357/18357 [==============================] - 7s 380us/step - loss: 0.0113 - val_loss: 0.0137
Epoch 25/200
18357/18357 [==============================] - 7s 364us/step - loss: 0.0110 - val_loss: 0.0135
Epoch 26/200
18357/18357 [==============================] - 7s 368us/step - loss: 0.0107 - val_loss: 0.0136
Epoch 27/200
18357/18357 [==============================] - 7s 398us/step - loss: 0.0106 - val_loss: 0.0138
Epoch 28/200
18357/18357 [==============================] - 7s 372us/step - loss: 0.0105 - val_loss: 0.0137
Train on 18357 samples, validate on 4589 samples
Epoch 1/200
18357/18357 [==============================] - 8s 459us/step - loss: 0.0329 - val_loss: 0.0158
Epoch 2/200
18357/18357 [==============================] - 7s 371us/step - loss: 0.0159 - val_loss: 0.0150
Epoch 3/200
18357/18357 [==============================] - 7s 365us/step - loss: 0.0152 - val_loss: 0.0147
Epoch 4/200
18357/18357 [==============================] - 7s 362us/step - loss: 0.0148 - val_loss: 0.0145
Epoch 5/200
18357/18357 [==============================] - 7s 365us/step - loss: 0.0145 - val_loss: 0.0144
Epoch 6/200
18357/18357 [==============================] - 7s 363us/step - loss: 0.0143 - val_loss: 0.0142
Epoch 7/200
18357/18357 [==============================] - 7s 364us/step - loss: 0.0140 - val_loss: 0.0142
Epoch 8/200
18357/18357 [==============================] - 7s 368us/step - loss: 0.0139 - val_loss: 0.0145
Epoch 9/200
18357/18357 [==============================] - 7s 370us/step - loss: 0.0138 - val_loss: 0.0143
Epoch 10/200
18357/18357 [==============================] - 7s 373us/step - loss: 0.0137 - val_loss: 0.0142
Epoch 11/200
18357/18357 [==============================] - 7s 370us/step - loss: 0.0135 - val_loss: 0.0143
Epoch 12/200
18357/18357 [==============================] - 7s 363us/step - loss: 0.0134 - val_loss: 0.0142
Epoch 13/200
18357/18357 [==============================] - 7s 369us/step - loss: 0.0133 - val_loss: 0.0141
Epoch 14/200
18357/18357 [==============================] - 7s 396us/step - loss: 0.0130 - val_loss: 0.0141
Epoch 15/200
18357/18357 [==============================] - 7s 394us/step - loss: 0.0129 - val_loss: 0.0142
Epoch 16/200
18357/18357 [==============================] - 7s 384us/step - loss: 0.0127 - val_loss: 0.0142
Epoch 17/200
18357/18357 [==============================] - 7s 378us/step - loss: 0.0127 - val_loss: 0.0142
Epoch 18/200
18357/18357 [==============================] - 7s 385us/step - loss: 0.0125 - val_loss: 0.0140
Epoch 19/200
18357/18357 [==============================] - 7s 392us/step - loss: 0.0122 - val_loss: 0.0143
Epoch 20/200
18357/18357 [==============================] - 7s 380us/step - loss: 0.0121 - val_loss: 0.0142
Epoch 21/200
18357/18357 [==============================] - 7s 373us/step - loss: 0.0119 - val_loss: 0.0141
Epoch 22/200
18357/18357 [==============================] - 7s 372us/step - loss: 0.0117 - val_loss: 0.0141
Epoch 23/200
18357/18357 [==============================] - 7s 384us/step - loss: 0.0115 - val_loss: 0.0142
Epoch 24/200
18357/18357 [==============================] - 8s 426us/step - loss: 0.0114 - val_loss: 0.0142
Epoch 25/200
18357/18357 [==============================] - 7s 366us/step - loss: 0.0111 - val_loss: 0.0142
Epoch 26/200
18357/18357 [==============================] - 7s 403us/step - loss: 0.0110 - val_loss: 0.0143
Epoch 27/200
18357/18357 [==============================] - 7s 360us/step - loss: 0.0108 - val_loss: 0.0142
Epoch 28/200
18357/18357 [==============================] - 7s 355us/step - loss: 0.0107 - val_loss: 0.0143
Epoch 29/200
18357/18357 [==============================] - 7s 355us/step - loss: 0.0104 - val_loss: 0.0144
Epoch 30/200
18357/18357 [==============================] - 7s 359us/step - loss: 0.0103 - val_loss: 0.0145
Epoch 31/200
18357/18357 [==============================] - 7s 365us/step - loss: 0.0101 - val_loss: 0.0145
Epoch 32/200
18357/18357 [==============================] - 7s 369us/step - loss: 0.0101 - val_loss: 0.0145
Epoch 33/200
18357/18357 [==============================] - 7s 366us/step - loss: 0.0100 - val_loss: 0.0145
Epoch 34/200
18357/18357 [==============================] - 7s 368us/step - loss: 0.0098 - val_loss: 0.0145
Epoch 35/200
18357/18357 [==============================] - 7s 369us/step - loss: 0.0096 - val_loss: 0.0146
Epoch 36/200
18357/18357 [==============================] - 7s 368us/step - loss: 0.0095 - val_loss: 0.0147
Epoch 37/200
18357/18357 [==============================] - 7s 371us/step - loss: 0.0094 - val_loss: 0.0146
Epoch 38/200
18357/18357 [==============================] - 7s 375us/step - loss: 0.0093 - val_loss: 0.0147
Train on 18357 samples, validate on 4589 samples
Epoch 1/200
18357/18357 [==============================] - 9s 469us/step - loss: 0.0320 - val_loss: 0.0168
Epoch 2/200
18357/18357 [==============================] - 7s 379us/step - loss: 0.0159 - val_loss: 0.0152
Epoch 3/200
18357/18357 [==============================] - 7s 406us/step - loss: 0.0151 - val_loss: 0.0150
Epoch 4/200
18357/18357 [==============================] - 7s 385us/step - loss: 0.0148 - val_loss: 0.0144
Epoch 5/200
18357/18357 [==============================] - 7s 378us/step - loss: 0.0143 - val_loss: 0.0142
Epoch 6/200
18357/18357 [==============================] - 7s 371us/step - loss: 0.0141 - val_loss: 0.0144
Epoch 7/200
18357/18357 [==============================] - 7s 368us/step - loss: 0.0139 - val_loss: 0.0143
Epoch 8/200
18357/18357 [==============================] - 7s 365us/step - loss: 0.0138 - val_loss: 0.0141
Epoch 9/200
18357/18357 [==============================] - 7s 361us/step - loss: 0.0136 - val_loss: 0.0142
Epoch 10/200
18357/18357 [==============================] - 7s 364us/step - loss: 0.0135 - val_loss: 0.0142
Epoch 11/200
18357/18357 [==============================] - 7s 359us/step - loss: 0.0133 - val_loss: 0.0143
Epoch 12/200
18356/18356 [==============================] - 7s 378us/step - loss: 0.0139 - val_loss: 0.0138
Epoch 9/200
18356/18356 [==============================] - 7s 380us/step - loss: 0.0139 - val_loss: 0.0139
Epoch 10/200
18356/18356 [==============================] - 7s 380us/step - loss: 0.0136 - val_loss: 0.0139
Epoch 11/200
18356/18356 [==============================] - 7s 379us/step - loss: 0.0135 - val_loss: 0.0139
Epoch 12/200
18356/18356 [==============================] - 8s 409us/step - loss: 0.0133 - val_loss: 0.0136
Epoch 13/200
18357/18357 [==============================] - 7s 388us/step - loss: 0.0124 - val_loss: 0.0138
Epoch 17/200
18357/18357 [==============================] - 7s 393us/step - loss: 0.0122 - val_loss: 0.0138
Epoch 18/200
18357/18357 [==============================] - 7s 390us/step - loss: 0.0121 - val_loss: 0.0137
Epoch 19/200
18357/18357 [==============================] - 8s 410us/step - loss: 0.0119 - val_loss: 0.0138
Epoch 20/200
18357/18357 [==============================] - 7s 401us/step - loss: 0.0118 - val_loss: 0.0137
Epoch 21/200
18357/18357 [==============================] - 8s 456us/step - loss: 0.0112 - val_loss: 0.0145
Epoch 24/200
18357/18357 [==============================] - 8s 450us/step - loss: 0.0110 - val_loss: 0.0145
Epoch 25/200
18357/18357 [==============================] - 8s 440us/step - loss: 0.0108 - val_loss: 0.0146
Epoch 26/200
18357/18357 [==============================] - 9s 471us/step - loss: 0.0107 - val_loss: 0.0145
Epoch 27/200
18357/18357 [==============================] - 8s 444us/step - loss: 0.0120 - val_loss: 0.0142
Epoch 21/200
18357/18357 [==============================] - 8s 432us/step - loss: 0.0117 - val_loss: 0.0143
Epoch 22/200
18357/18357 [==============================] - 9s 470us/step - loss: 0.0117 - val_loss: 0.0143
Epoch 23/200
18357/18357 [==============================] - 8s 462us/step - loss: 0.0114 - val_loss: 0.0143
Epoch 24/200
18357/18357 [==============================] - 8s 448us/step - loss: 0.0120 - val_loss: 0.0137
Epoch 20/200
18357/18357 [==============================] - 8s 445us/step - loss: 0.0119 - val_loss: 0.0139
Epoch 21/200
18357/18357 [==============================] - 8s 443us/step - loss: 0.0119 - val_loss: 0.0137
Epoch 22/200
18357/18357 [==============================] - 8s 432us/step - loss: 0.0116 - val_loss: 0.0138
Epoch 23/200
18357/18357 [==============================] - 8s 429us/step - loss: 0.0113 - val_loss: 0.0138
Epoch 24/200
18357/18357 [==============================] - 8s 443us/step - loss: 0.0112 - val_loss: 0.0139
Epoch 25/200
18357/18357 [==============================] - 8s 437us/step - loss: 0.0109 - val_loss: 0.0139
Epoch 26/200
18357/18357 [==============================] - 8s 434us/step - loss: 0.0108 - val_loss: 0.0138
Epoch 27/200
18357/18357 [==============================] - 8s 446us/step - loss: 0.0107 - val_loss: 0.0140
Train on 18357 samples, validate on 4589 samples
Epoch 1/200
18357/18357 [==============================] - 13s 690us/step - loss: 0.0340 - val_loss: 0.0170
Epoch 2/200
18357/18357 [==============================] - 8s 458us/step - loss: 0.0165 - val_loss: 0.0158
Epoch 3/200
18357/18357 [==============================] - 9s 477us/step - loss: 0.0155 - val_loss: 0.0153
Epoch 4/200
18357/18357 [==============================] - 8s 450us/step - loss: 0.0149 - val_loss: 0.0147
Epoch 5/200
18357/18357 [==============================] - 8s 442us/step - loss: 0.0144 - val_loss: 0.0148
Epoch 6/200
18357/18357 [==============================] - 8s 435us/step - loss: 0.0141 - val_loss: 0.0144
Epoch 7/200
18357/18357 [==============================] - 9s 468us/step - loss: 0.0128 - val_loss: 0.0147
Epoch 21/200
18357/18357 [==============================] - 9s 466us/step - loss: 0.0126 - val_loss: 0.0145
Epoch 22/200
18357/18357 [==============================] - 9s 470us/step - loss: 0.0124 - val_loss: 0.0146
Epoch 23/200
18357/18357 [==============================] - 9s 470us/step - loss: 0.0123 - val_loss: 0.0148
Epoch 24/200
 8256/18357 [============>.................] - ETA: 4s - loss: 0.0120
```

In [78]:
```
def make_pred(df, sample, env, models):
    df['StadiumType'] = df['StadiumType'].apply(clean_StadiumType)
    df['StadiumType'] = df['StadiumType'].apply(transform_StadiumType)
    df['DefendersInTheBox_vs_Distance'] = df['DefendersInTheBox'] / df['Distance']
    df['OffenseFormation'] = df['OffenseFormation'].apply(lambda x: x if x in off_form else np.nan)
    df = pd.concat([df.drop(['OffenseFormation'], axis=1), pd.get_dummies(df['OffenseFormation'], prefi
x='Formation')], axis=1)
    missing_cols = set( dummy_col ) - set( df.columns )-set('Yards')
    for c in missing_cols:
        df[c] = 0
    df = df[dummy_col]
    df.drop(['Yards'], axis=1, inplace=True)
    df['Turf'] = df['Turf'].map(Turf)
    df['Turf'] = df['Turf'] == 'Natural'
    df['PossessionTeam'] = df['PossessionTeam'].map(map_abbr)
    df['HomeTeamAbbr'] = df['HomeTeamAbbr'].map(map_abbr)
    df['VisitorTeamAbbr'] = df['VisitorTeamAbbr'].map(map_abbr)
    df['HomePossesion'] = df['PossessionTeam'] == df['HomeTeamAbbr']
    df['Field_eq_Possession'] = df['FieldPosition'] == df['PossessionTeam']
    df['HomeField'] = df['FieldPosition'] == df['HomeTeamAbbr']
    df['GameClock'] = df['GameClock'].apply(strtoseconds)
    df['PlayerHeight'] = df['PlayerHeight'].apply(lambda x: 12*int(x.split('-')[0])+int(x.split('-')[1
]))
    df['PlayerBMI'] = 703*(df['PlayerWeight']/(df['PlayerHeight'])**2)
    df['TimeHandoff'] = df['TimeHandoff'].apply(lambda x: datetime.datetime.strptime(x, "%Y-%m-%dT%H:%
M:%S.%fZ"))
    df['TimeSnap'] = df['TimeSnap'].apply(lambda x: datetime.datetime.strptime(x, "%Y-%m-%dT%H:%M:%S.%f
Z"))
    df['TimeDelta'] = df.apply(lambda row: (row['TimeHandoff'] - row['TimeSnap']).total_seconds(), axis
=1)
    df['PlayerBirthDate'] = df['PlayerBirthDate'].apply(lambda x: datetime.datetime.strptime(x, "%m/%d
/%Y"))
    seconds_in_year = 60*60*24*365.25
    df['PlayerAge'] = df.apply(lambda row: (row['TimeHandoff']-row['PlayerBirthDate']).total_seconds()/
seconds_in_year, axis=1)
    df['WindSpeed'] = df['WindSpeed'].apply(lambda x: x.lower().replace('mph', '').strip() if not pd.is
na(x) else x)
    df['WindSpeed'] = df['WindSpeed'].apply(lambda x: (int(x.split('-')[0])+int(x.split('-')[1]))/2 if
not pd.isna(x) and '-' in x else x)
    df['WindSpeed'] = df['WindSpeed'].apply(lambda x: (int(x.split()[0])+int(x.split()[-1]))/2 if not p
d.isna(x) and type(x)!=float and 'gusts up to' in x else x)
    df['WindSpeed'] = df['WindSpeed'].apply(str_to_float)
    df['WindDirection'] = df['WindDirection'].apply(clean_WindDirection)
    df['WindDirection'] = df['WindDirection'].apply(transform_WindDirection)
    df['PlayDirection'] = df['PlayDirection'].apply(lambda x: x.strip() == 'right')
    df['Team'] = df['Team'].apply(lambda x: x.strip()=='home')
    indoor = "indoor"
    df['GameWeather'] = df['GameWeather'].apply(lambda x: indoor if not pd.isna(x) and indoor in x else
 x)
    df['GameWeather'] = df['GameWeather'].apply(lambda x: x.lower().replace('coudy', 'cloudy').replace(
'clouidy', 'cloudy').replace('party', 'partly').replace('clear and sunny', 'sunny and clear').replace(
'skies', '').replace("mostly", "").strip() if not pd.isna(x) else x)
    df['GameWeather'] = df['GameWeather'].apply(map_weather)
    df['IsRusher'] = df['NflId'] == df['NflIdRusher']
    df['X'] = df.apply(lambda row: row['X'] if row['PlayDirection'] else 120-row['X'], axis=1)
    df['Orientation'] = df.apply(lambda row: new_orientation(row['Orientation'], row['PlayDirection']),
 axis=1)
    df['Dir'] = df.apply(lambda row: new_orientation(row['Dir'], row['PlayDirection']), axis=1)
    df['YardsLeft'] = df.apply(lambda row: 100-row['YardLine'] if row['HomeField'] else row['YardLine'
], axis=1)
    df['YardsLeft'] = df.apply(lambda row: row['YardsLeft'] if row['PlayDirection'] else 100-row['Yards
Left'], axis=1)
    df = df.sort_values(by=['PlayId', 'Team', 'IsRusher', 'JerseyNumber']).reset_index()
    df = df.drop(['TimeHandoff', 'TimeSnap', 'PlayerBirthDate', 'NflId', 'NflIdRusher', 'GameId', 'Play
Id', 'index', 'IsRusher', 'Team'], axis=1)
    cat_features = []
    for col in df.columns:
        if df[col].dtype =='object':
            cat_features.append(col)

    df = df.drop(cat_features, axis=1)
    df.fillna(-999, inplace=True)
    X = np.array(df[players_col]).reshape(-1, len(players_col)*22)
    play_col = df.drop(players_col, axis=1).columns
    X_play_col = np.zeros(shape=(X.shape[0], len(play_col)))
    for i, col in enumerate(play_col):
        X_play_col[:, i] = df[col][::22]
    X = np.concatenate([X, X_play_col], axis=1)
    X = scaler.transform(X)
    y_pred = np.mean([np.cumsum(model.predict(X), axis=1) for model in models], axis=0)
    yardsleft = np.array(df['YardsLeft'][::22])

    for i in range(len(yardsleft)):
        y_pred[i, :yardsleft[i]-1] = 0
        y_pred[i, yardsleft[i]+100:] = 1
    env.predict(pd.DataFrame(data=y_pred.clip(0,1),columns=sample.columns))
    return y_pred
```

In [79]:
```
for test, sample in tqdm.tqdm(env.iter_test()):
    make_pred(test, sample, env, models)
```

```
3438it [16:52,  3.40it/s]
```

In [80]:
```
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.
```

End

If you reached this far please comment and upvote this kernel, feel free to make improvements on the kernel and please share if you found anything useful!