Introduction:

See how stats like PA, AB, H, 2B, 3B, HR, RBI, SB, BA, OBP, SLG, etc influence postseason birth

Methods:

Gathering MLB regular season team stats from 2012-2019, including stats like PA, AB, H, 2B, 3B, HR, RBI, SB, BA, OBP, SLG, etc,

Using SQL Server and Python(Spyder)

Building classification model

Try to build perfect model by comparing different learning rate, Model validation, early stopping, experiment different number of nodes in each layer, different number of layers

In [8]:
```
import pandas as pd
import pyodbc

#import regular season stats from MLB teams who got into postseason dur
ing 2012-2019
#items include Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, S
O, BA, OBP, SLG, OPS, TB, GDP
#total rows are 8(years)*10(teams each year)=80

sql_conn = pyodbc.connect('''DRIVER={ODBC Driver 13 for SQL Server};
                            SERVER=ALLENHO\MSSQLSERVER002;
                            DATABASE=Playoffbound;
                            Trusted_Connection=yes''')
query = '''
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['19B$']
where Tm in ('WSN','LAD','MIL','ATL','STL','HOU','NYY','MIN','TBR','OA
```

```
K')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['18B$']
where Tm in ('BOS','LAD','MIL','ATL','CHC','HOU','NYY','CLE','COL','OA
K')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['17B$']
where Tm in ('BOS','LAD','COL','WSN','CHC','HOU','NYY','CLE','ARI','MI
N')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['16B$']
where Tm in ('TOR','CLE','BOS','BAL','TEX','NYM','CHC','LAD','WSN','SF
G')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['15B$']
where Tm in ('TOR','KCR','HOU','NYY','TEX','NYM','CHC','LAD','STL','PI
T')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['14B$']
where Tm in ('BAL','KCR','OAK','LAA','DET','WSN','STL','LAD','PIT','SF
G')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['13B$']
where Tm in ('BOS','TBR','OAK','CLE','DET','ATL','STL','LAD','PIT','CI
N')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
```

```
BP, SLG, OPS, TB, GDP
from [dbo].['12B$']
where Tm in ('TEX','BAL','OAK','NYY','DET','ATL','STL','SFG','WSN','CI
N')
'''
df = pd.read_sql(query, sql_conn)

#stored as df_post
df_post = df

#import regular season stats from MLB teams who DIDN'T get into postsea
son during 2012-2019
#items are the same as above
#total rows are 8(years)*20(teams each year)=160
sql_conn = pyodbc.connect('''DRIVER={ODBC Driver 13 for SQL Server};
                            SERVER=ALLENHO\MSSQLSERVER002;
                            DATABASE=Playoffbound;
                            Trusted_Connection=yes''')
query = '''
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['19B$']
where Tm is not null and Tm not in ('WSN','LAD','MIL','ATL','STL','HO
U','NYY','MIN','TBR','OAK', 'LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['18B$']
where Tm is not null and Tm not in ('BOS','LAD','MIL','ATL','CHC','HO
U','NYY','CLE','COL','OAK', 'LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['17B$']
where Tm is not null and Tm not in ('BOS','LAD','COL','WSN','CHC','HO
U','NYY','CLE','ARI','MIN', 'LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
```

```
from [dbo].['16B$']
where Tm is not null and Tm not in ('TOR','CLE','BOS','BAL','TEX','NY
M','CHC','LAD','WSN','SFG', 'LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['15B$']
where Tm is not null and Tm not in ('TOR','KCR','HOU','NYY','TEX','NY
M','CHC','LAD','STL','PIT', 'LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['14B$']
where Tm is not null and Tm not in ('BAL','KCR','OAK','LAA','DET','WS
N','STL','LAD','PIT','SFG', 'LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['13B$']
where Tm is not null and Tm not in ('BOS','TBR','OAK','CLE','DET','AT
L','STL','LAD','PIT','CIN', 'LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, O
BP, SLG, OPS, TB, GDP
from [dbo].['12B$']
where Tm is not null and Tm not in ('TEX','BAL','OAK','NYY','DET','AT
L','STL','SFG','WSN','CIN', 'LgAvg')
'''
df = pd.read_sql(query, sql_conn)

#stored as df_npost
df_npost = df

#add each dataframe a new column named POST, which imply whether the te
am made the postseason
df_post['POST']= 1
df_npost['POST']= 0

#append two dataframes together
```

```
df_com=df_post.append(df_npost)


#------neural network, classification model--------------------------
------

# Import necessary modules

from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical

# Save the number of columns in predictors: n_cols
predictors=df_com.loc[:,'BatAge':'GDP'].to_numpy()
n_cols = predictors.shape[1]

# Convert the target to categorical: target
target = to_categorical(df_com['POST'])

# Set up the model
model = Sequential()

# Add the first and second layer
model.add(Dense(100, activation='relu', input_shape = (n_cols,)))
model.add(Dense(100, activation='relu'))

# Add the output layer
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fit the model
model.fit(predictors, target)
```

```
Epoch 1/1
240/240 [==============================] - 0s 2ms/step - loss: 1426996
9.1915 - accuracy: 0.5917
```

Out[8]: <keras.callbacks.callbacks.History at 0x16e97c27748>

In [3]:
```
#-----compare different learning rate-----------------------------------
-----
# Import the SGD optimizer
from keras.optimizers import SGD
input_shape = (n_cols,)

def get_new_model(input_shape = input_shape):
    model = Sequential()
    model.add(Dense(100, activation='relu', input_shape = input_shape))

    model.add(Dense(100, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    return(model)

# Create list of learning rates: lr_to_test
lr_to_test = [.000001, 0.01, 1]

# Loop over learning rates
for lr in lr_to_test:
    print('\n\nTesting model with learning rate: %f\n'%lr )
    # Build new model to test, unaffected by previous models
    model = get_new_model()
    # Create SGD optimizer with specified learning rate: my_optimizer
    my_optimizer = SGD(lr=lr)
    # Compile the model
    model.compile(optimizer=my_optimizer, loss='categorical_crossentrop
y')
    # Fit the model
    model.fit(predictors, target)
```

```
Testing model with learning rate: 0.000001

Epoch 1/1
240/240 [==============================] - 0s 420us/step - loss: 1313.9
385
```

```
Testing model with learning rate: 0.010000

Epoch 1/1
240/240 [==============================] - 0s 591us/step - loss: 231853
58.0425


Testing model with learning rate: 1.000000

Epoch 1/1
240/240 [==============================] - 0s 558us/step - loss: 524418
37020127.7109
```

In [4]:

```
#-----validating data--------------------------------------------------
------
# Save the number of columns in predictors: n_cols
n_cols = predictors.shape[1]
input_shape = (n_cols,)
# Specify the model
model = Sequential()
model.add(Dense(100, activation='relu', input_shape = input_shape))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metric
s=['accuracy'])
# Fit the model
hist = model.fit(predictors, target, validation_split=0.3)
```

```
Train on 168 samples, validate on 72 samples
Epoch 1/1
168/168 [==============================] - 1s 3ms/step - loss: 81.6105
- accuracy: 0.5893 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

In [5]:

```
#-----early stopping---------------------------------------------------
------
# Import EarlyStopping
from keras.callbacks import EarlyStopping
```

```python
# Save the number of columns in predictors: n_cols
n_cols = predictors.shape[1]
input_shape = (n_cols,)

# Specify the model
model = Sequential()
model.add(Dense(100, activation='relu', input_shape = input_shape))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metric
s=['accuracy'])

# Define early_stopping_monitor
early_stopping_monitor = EarlyStopping(patience=2)

# Fit the model
model.fit(predictors, target, epochs=30, validation_split=0.3, callback
s=[early_stopping_monitor])
```

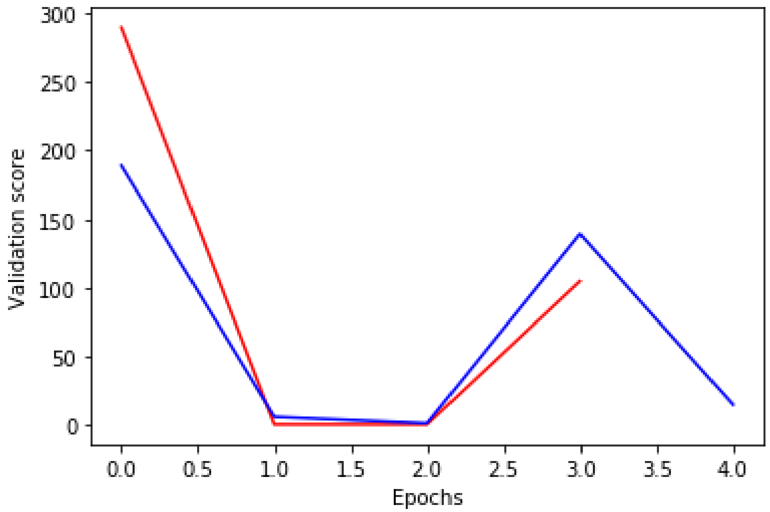
```
Train on 168 samples, validate on 72 samples
Epoch 1/30
168/168 [==============================] - 1s 7ms/step - loss: 235.9913
- accuracy: 0.5179 - val_loss: 461.1284 - val_accuracy: 0.0000e+00
Epoch 2/30
168/168 [==============================] - 0s 178us/step - loss: 157.83
76 - accuracy: 0.4524 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 3/30
168/168 [==============================] - 0s 167us/step - loss: 80.477
7 - accuracy: 0.5000 - val_loss: 1.1590e-08 - val_accuracy: 1.0000
Epoch 4/30
168/168 [==============================] - 0s 178us/step - loss: 21.843
1 - accuracy: 0.4405 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

Out[5]:
```
<keras.callbacks.callbacks.History at 0x16ea610ba90>
```

In [6]:
```python
#-----experiment different number of nodes in each layer---------------
```

```
----------
import matplotlib.pyplot as plt

# Set up the model_1
model_1 = Sequential()

# Add the first and second layer
model_1.add(Dense(50, activation='relu', input_shape=input_shape))
model_1.add(Dense(50, activation='relu'))

# Add the output layer
model_1.add(Dense(2, activation='softmax'))

# Compile the model
model_1.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Create the new model: model_2
model_2 = Sequential()

# Add the first and second layers
model_2.add(Dense(55, activation='relu', input_shape=input_shape))
model_2.add(Dense(55, activation='relu'))

# Add the output layer
model_2.add(Dense(2, activation='softmax'))

# Compile model_2
model_2.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

# Fit model_1
model_1_training = model_1.fit(predictors, target, epochs=15, validatio
n_split=0.2, callbacks=[early_stopping_monitor], verbose=False)

# Fit model_2
model_2_training = model_2.fit(predictors, target, epochs=15, validatio
```

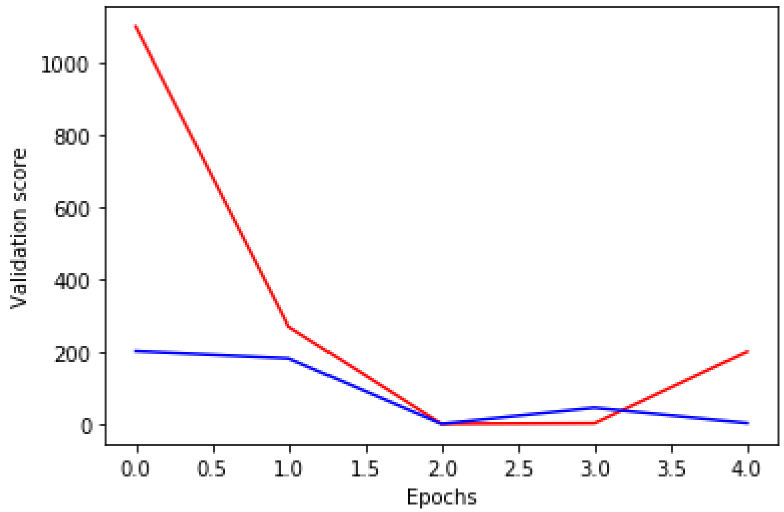
```
n_split=0.2, callbacks=[early_stopping_monitor], verbose=False)

# Create the plot
plt.plot(model_1_training.history['val_loss'], 'r', model_2_training.hi
story['val_loss'], 'b')
plt.xlabel('Epochs')
plt.ylabel('Validation score')
plt.show()
```



In [7]:

```
#-----experiment different number of layers----------------------------
-------
# Create the new model: model_1
model_1 = Sequential()

# Add one hidden layer
model_1.add(Dense(54, activation='relu', input_shape=input_shape))

# Add the output layer
model_1.add(Dense(2, activation='softmax'))

# Compile model_2
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
```

```
# Create the new model: model_2
model_2 = Sequential()

# Add the first, second, and third hidden layers
model_2.add(Dense(54, activation='relu', input_shape=input_shape))
model_2.add(Dense(54, activation='relu'))
model_2.add(Dense(54, activation='relu'))

# Add the output layer
model_2.add(Dense(2, activation='softmax'))

# Compile model_2
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])

# Fit model 1
model_1_training = model_1.fit(predictors, target, epochs=20, validatio
n_split=0.4, callbacks=[early_stopping_monitor], verbose=False)

# Fit model 2
model_2_training = model_2.fit(predictors, target, epochs=20, validatio
n_split=0.4, callbacks=[early_stopping_monitor], verbose=False)

# Create the plot
plt.plot(model_1_training.history['val_loss'], 'r', model_2_training.hi
story['val_loss'], 'b')
plt.xlabel('Epochs')
plt.ylabel('Validation score')
plt.show()
```



1000
800
600
400
200
0
Validation score
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epochs