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 experimenting different number of neurons in each layer, different number of layers, different learning rate, model validation, dropout and early stopping

---

import regular season stats from MLB teams who got into postseason during 2012-2019

items include Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP

total rows are 8(years)*10(teams each year)=80

In [1]:

```
import pandas as pd
import pyodbc



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, OBP, SLG, OPS, TB, GDP
from [dbo].['19B$']
where Tm in ('WSN','LAD','MIL','ATL','STL','HOU','NYY','MIN','TBR','OAK')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['18B$']
where Tm in ('BOS','LAD','MIL','ATL','CHC','HOU','NYY','CLE','COL','OAK')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['17B$']
where Tm in ('BOS','LAD','COL','WSN','CHC','HOU','NYY','CLE','ARI','MIN')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['16B$']
where Tm in ('TOR','CLE','BOS','BAL','TEX','NYM','CHC','LAD','WSN','SFG')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['15B$']
where Tm in ('TOR','KCR','HOU','NYY','TEX','NYM','CHC','LAD','STL','PIT')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['14B$']
where Tm in ('BAL','KCR','OAK','LAA','DET','WSN','STL','LAD','PIT','SFG')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['13B$']
where Tm in ('BOS','TBR','OAK','CLE','DET','ATL','STL','LAD','PIT','CIN')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['12B$']
where Tm in ('TEX','BAL','OAK','NYY','DET','ATL','STL','SFG','WSN','CIN')
'''
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 postseason during 2012-2019 items are the same as above total rows are 8(years)*20(teams each year)=160

In [2]:

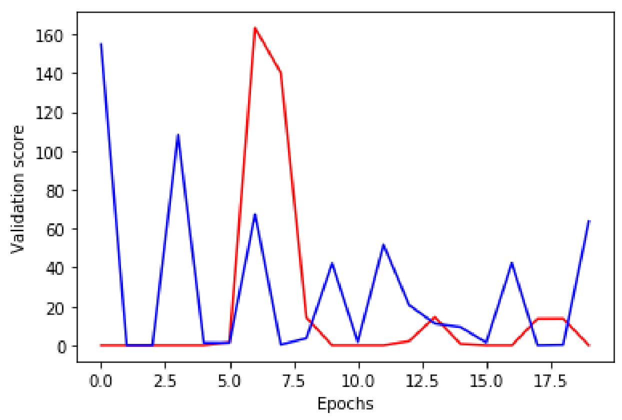
```
sql_conn = pyodbc.connect('''DRIVER={ODBC Driver 13 for SQL Server};
                            SERVER=ALLENHO\MSSQLSERVER002;
                            DATABASE=Playoffbound;
                            Trusted_Connection=yes''')
query = '''
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['19B$']
where Tm is not null and Tm not in ('WSN','LAD','MIL','ATL','STL','HOU','NYY','MIN','TBR','OAK', '
LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['18B$']
where Tm is not null and Tm not in ('BOS','LAD','MIL','ATL','CHC','HOU','NYY','CLE','COL','OAK', '
LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['17B$']
where Tm is not null and Tm not in ('BOS','LAD','COL','WSN','CHC','HOU','NYY','CLE','ARI','MIN', '
LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['16B$']
where Tm is not null and Tm not in ('TOR','CLE','BOS','BAL','TEX','NYM','CHC','LAD','WSN','SFG', '
LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['15B$']
where Tm is not null and Tm not in ('TOR','KCR','HOU','NYY','TEX','NYM','CHC','LAD','STL','PIT', '
LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['14B$']
where Tm is not null and Tm not in ('BAL','KCR','OAK','LAA','DET','WSN','STL','LAD','PIT','SFG', '
LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['13B$']
where Tm is not null and Tm not in ('BOS','TBR','OAK','CLE','DET','ATL','STL','LAD','PIT','CIN', '
LgAvg')
UNION ALL
select Tm, BatAge, PA, AB, R, H, 2B, 3B, HR, RBI, SB, CS, BB, SO, BA, OBP, SLG, OPS, TB, GDP
from [dbo].['12B$']
where Tm is not null and Tm not in ('TEX','BAL','OAK','NYY','DET','ATL','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 team made the postseason
df_post['POST']= 1
df_npost['POST']= 0

#append two dataframes together
df_com=df_post.append(df_npost)
```

Step 1: Build the deep learning classification model and experiment different number of neurons in each layer

In [5]:

```
# Import necessary modules

from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
import matplotlib.pyplot as plt

# Save the number of columns in predictors: n_cols
```

```
predictors=df_com.loc[:,'BatAge':'GDP'].to_numpy()
n_cols = predictors.shape[1]
input_shape = (n_cols,)

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

# Set up the model_1
model_1 = Sequential()

# Add the first and second layer
model_1.add(Dense(25, activation='relu', input_shape=input_shape))
model_1.add(Dense(25, 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(50, activation='relu', input_shape=input_shape))
model_2.add(Dense(50, 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=20, batch_size=100, validation_split=0.15
, verbose=False)

# Fit model_2
model_2_training = model_2.fit(predictors, target, epochs=20, batch_size=100, validation_split=0.15
, verbose=False)

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

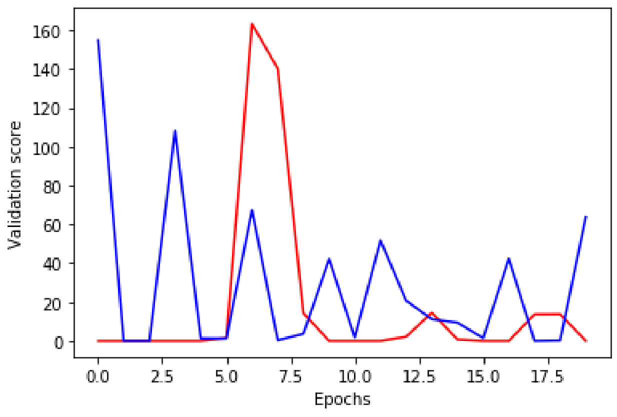


Result: 25 neurons in both hidden layers(red one) had a better performance than 50 neurons in both hidden layers(blue one)

Step 2: experiment different number of layers

In [13]:

```
# Create the new model: model_3
model_3 = Sequential()

# Add five hidden layers
model_3.add(Dense(25, activation='relu', input_shape=input_shape))
model_3.add(Dense(25, activation='relu'))
model_3.add(Dense(25, activation='relu'))
model_3.add(Dense(25, activation='relu'))
model_3.add(Dense(25, activation='relu'))

# Add the output layer
model_3.add(Dense(2, activation='softmax'))

# Compile model_3
model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Create the new model: model_4
model_4 = Sequential()

# Add ten hidden layers
model_4.add(Dense(25, activation='relu', input_shape=input_shape))
for i in range(9):
    model_4.add(Dense(25, activation='relu'))

# Add the output layer
model_4.add(Dense(2, activation='softmax'))

# Compile model_2
model_4.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fit model 1
model_3_training = model_3.fit(predictors, target, epochs=20, batch_size=100, validation_split=0.15
, verbose=False)

# Fit model 2
model_4_training = model_4.fit(predictors, target, epochs=20, batch_size=100, validation_split=0.15
, verbose=False)

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



Result: 5 hidden layers(red one) appeared to have better performance than 10 hidden layers(blue one)

Step 3: Try different learning rate

In [15]:

```
from keras.optimizers import adam

input_shape = (n_cols,)

def get_new_model(input_shape = input_shape):
    model_5 = Sequential()
    model_5.add(Dense(25, activation='relu', input_shape = input_shape))
    model_5.add(Dense(25, activation='relu'))
    model_5.add(Dense(25, activation='relu'))
    model_5.add(Dense(25, activation='relu'))
    model_5.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_5 = get_new_model()
    # Create adam optimizer with specified learning rate: my_optimizer
    my_optimizer = adam(lr=lr)
    # Compile the model
    model_5.compile(optimizer=my_optimizer, loss='categorical_crossentropy')
    # Fit the model
    model_5.fit(predictors, target, batch_size=100, epochs=10)
```

```
Testing model with learning rate: 0.000001

Epoch 1/10
240/240 [==============================] - 1s 4ms/step - loss: 0.6685
Epoch 2/10
240/240 [==============================] - 0s 79us/step - loss: 0.6685
Epoch 3/10
240/240 [==============================] - 0s 233us/step - loss: 0.6685
Epoch 4/10
240/240 [==============================] - 0s 79us/step - loss: 0.6685
Epoch 5/10
240/240 [==============================] - 0s 125us/step - loss: 0.6685
Epoch 6/10
240/240 [==============================] - 0s 100us/step - loss: 0.6685
Epoch 7/10
240/240 [==============================] - 0s 108us/step - loss: 0.6685
Epoch 8/10
240/240 [==============================] - 0s 104us/step - loss: 0.6685
Epoch 9/10
240/240 [==============================] - 0s 83us/step - loss: 0.6685
Epoch 10/10
240/240 [==============================] - 0s 117us/step - loss: 0.6685


Testing model with learning rate: 0.010000

Epoch 1/10
240/240 [==============================] - 1s 4ms/step - loss: 0.6672
Epoch 2/10
240/240 [==============================] - 0s 75us/step - loss: 0.6608
Epoch 3/10
240/240 [==============================] - 0s 67us/step - loss: 0.6565
Epoch 4/10
240/240 [==============================] - 0s 67us/step - loss: 0.6516
Epoch 5/10
240/240 [==============================] - 0s 96us/step - loss: 0.6480
Epoch 6/10
240/240 [==============================] - 0s 92us/step - loss: 0.6449
Epoch 7/10
240/240 [==============================] - 0s 92us/step - loss: 0.6417
Epoch 8/10
240/240 [==============================] - 0s 87us/step - loss: 0.6402
Epoch 9/10
240/240 [==============================] - 0s 75us/step - loss: 0.6390
Epoch 10/10
240/240 [==============================] - 0s 87us/step - loss: 0.6376


Testing model with learning rate: 1.000000

Epoch 1/10
240/240 [==============================] - 1s 5ms/step - loss: 0.8614
Epoch 2/10
240/240 [==============================] - 0s 62us/step - loss: 0.7988
Epoch 3/10
```

```
240/240 [==============================] - 0s 67us/step - loss: 0.6581
Epoch 4/10
240/240 [==============================] - 0s 83us/step - loss: 0.7845
Epoch 5/10
240/240 [==============================] - 0s 108us/step - loss: 0.6650
Epoch 6/10
240/240 [==============================] - 0s 79us/step - loss: 0.7335
Epoch 7/10
240/240 [==============================] - 0s 117us/step - loss: 0.6450
Epoch 8/10
240/240 [==============================] - 0s 71us/step - loss: 0.6993
Epoch 9/10
240/240 [==============================] - 0s 75us/step - loss: 0.6695
Epoch 10/10
240/240 [==============================] - 0s 92us/step - loss: 0.6518
```

Result: learning rate as 0.01(default) had the best performance

Step 4: Try the model with only training data with the best combination of neurons(25), layers(5) and learning rate(0.01, default)

In [19]:

```
# 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_6 = Sequential()

# Add the first and second layer
model_6.add(Dense(25, activation='relu', input_shape = input_shape))
for i in range(4):
    model_6.add(Dense(25, activation='relu'))

# Add the output layer
model_6.add(Dense(2, activation='softmax'))

# Compile the model
model_6.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fit the model
model_6.fit(predictors, target, batch_size=100, epochs=5)
```

```
Epoch 1/5
240/240 [==============================] - 1s 3ms/step - loss: 31.3220 - accuracy: 0.6625
Epoch 2/5
240/240 [==============================] - 0s 92us/step - loss: 14.6264 - accuracy: 0.4042
Epoch 3/5
240/240 [==============================] - 0s 87us/step - loss: 5.3396 - accuracy: 0.6042
Epoch 4/5
240/240 [==============================] - 0s 79us/step - loss: 2.7311 - accuracy: 0.3708
Epoch 5/5
240/240 [==============================] - 0s 87us/step - loss: 2.0924 - accuracy: 0.6333
```

Out[19]:

```
<keras.callbacks.callbacks.History at 0x1de3fd862b0>
```

Step 5: Perform with validating

In [20]:

```
# 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_7 = Sequential()

# Add the first and second layer
model_7.add(Dense(25, activation='relu', input_shape = (n_cols,)))
for i in range(4):
    model_7.add(Dense(25, activation='relu'))

# Add the output layer
model_7.add(Dense(2, activation='softmax'))

# Compile the model
model_7.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fit the model
model_7.fit(predictors, target, validation_split=0.15, batch_size=100, epochs=5)
```

```
Train on 204 samples, validate on 36 samples
Epoch 1/5
204/204 [==============================] - 1s 6ms/step - loss: 40.8806 - accuracy: 0.6078 - val_lo
ss: 11.8255 - val_accuracy: 0.0000e+00
Epoch 2/5
204/204 [==============================] - 0s 304us/step - loss: 11.1786 - accuracy: 0.3922 - val_
loss: 1.0926 - val_accuracy: 0.6389
Epoch 3/5
204/204 [==============================] - 0s 152us/step - loss: 3.9776 - accuracy: 0.6373 - val_l
oss: 0.2371 - val_accuracy: 0.8611
Epoch 4/5
204/204 [==============================] - 0s 142us/step - loss: 1.7780 - accuracy: 0.5931 - val_l
oss: 0.0606 - val_accuracy: 0.9722
Epoch 5/5
204/204 [==============================] - 0s 147us/step - loss: 1.5788 - accuracy: 0.6078 - val_l
oss: 0.6102 - val_accuracy: 0.7500
```

Out[20]:

```
<keras.callbacks.callbacks.History at 0x1de4058b4e0>
```

Step 6: Try dropout and see the change of accuracy of both training and validating data set

In [21]:

```
from keras.layers import Dropout

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

# Set up the model
model_8 = Sequential()

# Add the layers
model_8.add(Dense(25, activation='relu', input_shape = (n_cols,)))
model_8.add(Dropout(0.5))
for i in range(4):
    model_8.add(Dense(25, activation='relu'))
    model_8.add(Dropout(0.5))
```

```
# Add the output layer
model_8.add(Dense(2, activation='softmax'))

# Compile the model
model_8.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fit the model
model_8.fit(predictors, target, validation_split=0.15, batch_size=100, epochs=5)
```

```
Train on 204 samples, validate on 36 samples
Epoch 1/5
204/204 [==============================] - 1s 7ms/step - loss: 846.2524 - accuracy: 0.4902 - val_l
oss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 2/5
204/204 [==============================] - 0s 155us/step - loss: 913.0423 - accuracy: 0.5637 - val
_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 3/5
204/204 [==============================] - 0s 152us/step - loss: 676.3121 - accuracy: 0.5686 - val
_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 4/5
204/204 [==============================] - 0s 162us/step - loss: 645.4380 - accuracy: 0.5000 - val
_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/5
204/204 [==============================] - 0s 113us/step - loss: 650.7121 - accuracy: 0.5441 - val
_loss: 0.0000e+00 - val_accuracy: 1.0000
```

Out[21]:

```
<keras.callbacks.callbacks.History at 0x1de42e17400>
```

As expected, with dropout, training data set became less accurate than without dropout, however validating set became more accurate

Step 7: Try early stopping and see the change of accuracy of both training and validating data set

In [24]:

```
# Import EarlyStopping
from keras.callbacks import EarlyStopping

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

# Specify the model
model_9 = Sequential()
model_9.add(Dense(25, activation='relu', input_shape = input_shape))
for i in range(4):
    model_9.add(Dense(25, activation='relu'))
model_9.add(Dense(2, activation='softmax'))

# Compile the model
model_9.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Define early_stopping_monitor
early_stopping_monitor = EarlyStopping(patience=2)

# Fit the model
model_9.fit(predictors, target, validation_split=0.15, batch_size=100, epochs=5, callbacks=[early_s
topping_monitor])
```

```
Train on 204 samples, validate on 36 samples
Epoch 1/5
204/204 [==============================] - 1s 6ms/step - loss: 74.2591 - accuracy: 0.6078 - val_lo
ss: 95.3208 - val_accuracy: 0.0000e+00
Epoch 2/5
204/204 [==============================] - 0s 113us/step - loss: 72.6171 - accuracy: 0.3922 - val_
loss: 67.1380 - val_accuracy: 0.0000e+00
Epoch 3/5
204/204 [==============================] - 0s 113us/step - loss: 27.9019 - accuracy: 0.4167 - val_
loss: 0.0000e+00 - val_accuracy: 1.0000
```

```
Epoch 4/5
204/204 [==============================] - 0s 118us/step - loss: 45.9220 - accuracy: 0.6078 - val_
loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/5
204/204 [==============================] - 0s 108us/step - loss: 53.2466 - accuracy: 0.6078 - val_
loss: 0.0000e+00 - val_accuracy: 1.0000
```

Out[24]:

```
<keras.callbacks.callbacks.History at 0x1de47739ef0>
```

Result: two epochs to wait before early stop if no progress on the validation set as expected