Thomas Courtney
Artificial Intelligence
Professor Blossom
20 October 2021

Predicting March Madness

Summary

The assignment called to develop code using a neural network function that would successfully predict what round a men's division one college basketball team will exit the infamous NCAA Tournament. This paper will explain what a neural network is along with how it was applied in our use case.

Introduction

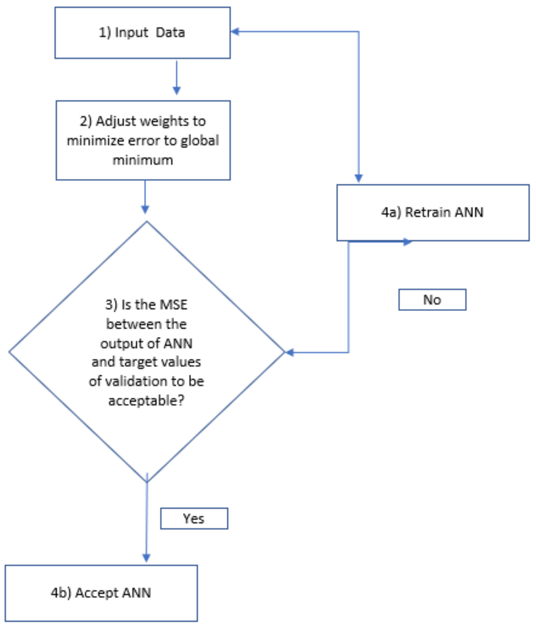
Artificial Neural Network (ANN) is one of the most common artificial intelligence coding designs in the world. Regression ANNs predict output variables by the function of the inputs. For regression ANNs, they require numeric dependent variables. ANNs are used throughout many applications, mostly involving predictive classification. Some ways that they are using a regressor are stock market and weather forecasting.

The code attached predicts the finish of college basketball teams in the NCAA tournament. The data set had multiple columns in the data set to help us create the prediction. The original dataset contains the columns in the table below. There is also a description of each column in the adjacent column.

Column Header	**Description**
TEAM	Division 1 basketball school
CONF	Athletic conference that the school participates in
G	Number of games the team played
W	Number of wins the team had that year prior to the NCAA tournament
ADJOE	Adjusted Offensive Efficiency, the estimate of the offensive efficiency (points scored per 100 possessions) that a team would get against an average NCAA Tournament team
ADJDE	Adjusted Defensive Efficiency, the estimate of the defensive efficiency (points allowed per 100 possessions) that a team would get against an average NCAA Tournament team
BARTHAG	Power Rating (chance of beating an average division 1 team)
EFG_O	Effective field goal percentage shot
EFG_D	Effective field goal percentage allowed
TOR	Offensive turnover rate
TORD	Defensive turnover rate
ORB	Offensive rebound rate
DRB	Offensive rebound rate allowed
FTR	Free throw rate (how often this team shoots free throws)
FTRD	Free throw rate allowed
2P_O	Two point shooting percentage
2P_D	Two point shooting percentage allowed
3P_O	Three point shooting percentage
3P_D	Three point shooting percentage allowed
ADJ_T	Adjusted Tempo: Number of possessions per 40 minutes, the team would have against an average NCAA team
WAB	Wins over bubble teams
POSTSEASON	Round where the team was eliminated
SEED	Seed in the march madness tournament
YEAR	Year that the data is from

Body

Artificial Neural Network Flowchart


1) Input Data
2) Adjust weights to minimize error to global minimum
3) Is the MSE between the output of ANN and target values of validation to be acceptable?
4a) Retrain ANN
No
Yes
4b) Accept ANN

—

Artificial Neural Network Code

To help understand the Artificial Neural Network process, the flow chart is above. In the on the pages to follow will see this process come to life. The first step is to input the data. The team then pretreated the data. The first thing the team did was change the values in the "Postseason" column. Originally, the values were listed as string variables. ANN models work best with numeric variables. We changed the values to be numeric in the table below.

Original Value	**Numeric Value**
Champions	1
2nd	2
F4	3
E8	4
S16	5
R32	6
R64	7
R68	8
NA	9

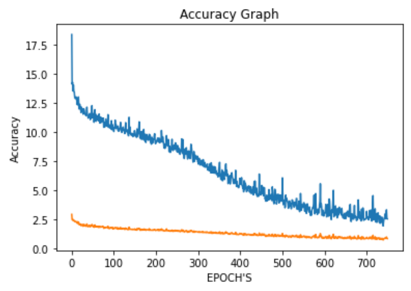
From there we ran a ridge model on all the variables that help predict which variables had the greatest influence on the final result, and which variables had a negative impact. Reading a ridge regression, the best possible coefficient has a score of 1.0, with the extremes having a greater overall effect on predicting the model. In order to run the model, we had to normalize the data contained in the WAB from 0 to 1. You cannot perform ridge models with negative values. This is the table from our results.

Column	**Ridge Result Value**
W	-.028
ADJOE	-.23
ADJDE	-.173
BARTHAG	8.412
EFG_O	.204
EFG_D	.107
TOR	-.097
TORD	.018
ORB	.032
DRB	-.028
FTR	.022
FTRD	.002
2P_O	-.07
2P_D	-.083
3P_O	-.05
3P_D	-.081
ADJ_T	.012
WAB	-2.248
SEED	-.148

We looked at these results and considered them when building our model. Below are the results with the description of what we did each time we ran the model. The goal of the code was to predict an estimate what round teams in the NCAA tournament were eliminated. We will use the Mean Squared Error and Mean Absolute Error as our performance metric. These are common measures for regression problems. It measures the performance of models by measuring the total number of errors. We want this value to be as close to zero as possible.

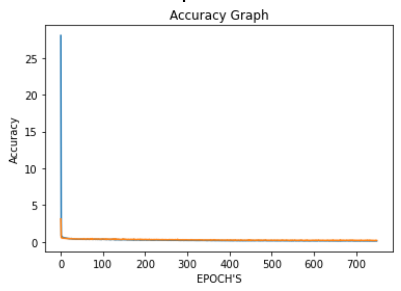
Below are the results when we changed which columns we included in the model. All factors that would change the mse or mae are kept the same.

No Factors Removed



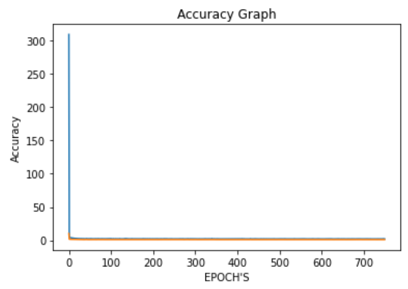
MSE: 2.5754
MAE: .8855
Time to Train: 57.828

All factors with a zero in the tenths place removed



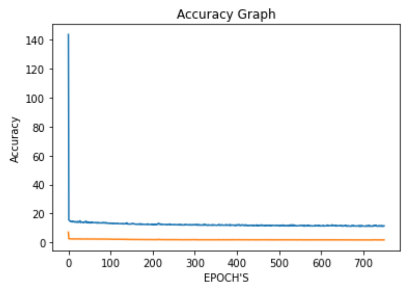
MSE: .1134
MAE: .1811
Time to Train: 57.88

3 Factors with the lowest absolute value removed



MSE: 2.1947
MAE: 1.511
Time to Train: 1:27.96

6 Factors with the lowest absolute value removed



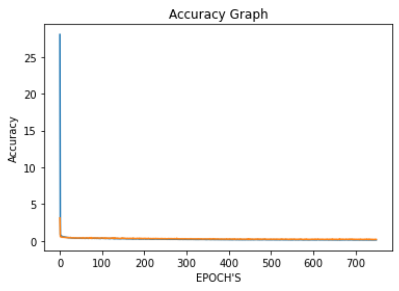
MSE: .11
MAE: .1626
Time to Train 56.838

How long each test took to train is above. Based off these results, we decided to remove all factors that had zeros in the tenths place based off the ridge model. I feared that our dataset had too many columns that it caused overfitting and would focus on the values that had the greatest impact on overall success in the tournament. The following columns we ended up containing in our model are in the table below, with their descriptions below. They are listed from what the model predicts to be most important to least important. It makes sense that these columns have the greatest impact on models, as it combines factors from the team against other teams or the average. Teams with a high chance of beating division one teams, wins over teams in the tournament are important factors to consider when building the model. It also places high values against how the team in question does against the average, and how well they score and how much scores they give up. I think it will be good to focus on these values when building the model.

BARTHAG	Power Rating (chance of beating an average division 1 team)
WAB	Wins over bubble teams
ADJOE	Adjusted Offensive Efficiency, the estimate of the offensive efficiency (points scored per 100 possessions) that a team would get against an average NCAA Tournament team
EFG_O	Effective field goal percentage shot
ADJDE	Adjusted Defensive Efficiency, the estimate of the defensive efficiency (points allowed per 100 possessions) that a team would get against an average NCAA Tournament team
SEED	Seed in the march madness tournament
EFG_D	Effective field goal percentage allowed

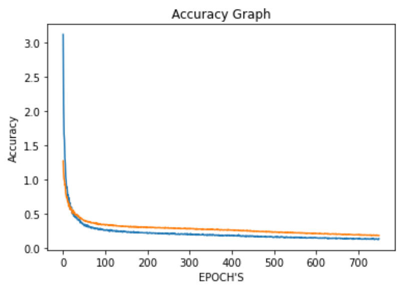
I then began testing the difference between which optimizer would be best, between Adam and RMSprop, with only the above seven variables in question. These different optimizers typically have a great effect on the outcome of the model. There are some key differences between the different models. Adam is straightforward to implement, computationally efficient and there are little memory requirements required. RMSprop is a gradient based optimization technique using the step size. Below are the accuracy graphs and the MSE and MAE along with each graph

Adam



MSE: .1134
MAE: .1811
Time to Train: 57.88

RMSprop

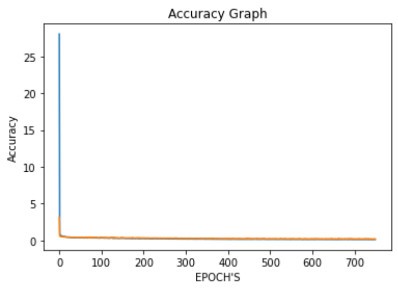


MSE: .1312
MAE: .01790
Time to Train:1:23.2

Based off the MSE value and the difference in how long it took to train, we chose to continue developing and testing our models with Adam because it had the lower MSE and length it took to train.

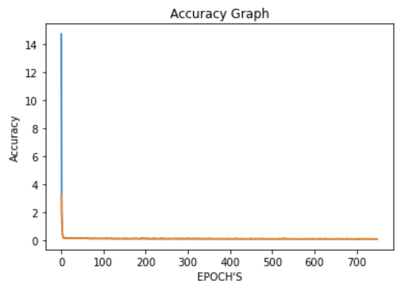
Next, we attempted to change the activation layer on each model. There are four different optimizer layers that we could use, including relu, tanh and sigmoid. In previous examples, we had been using relu. Below are the results when we compare the different activation types.

Relu



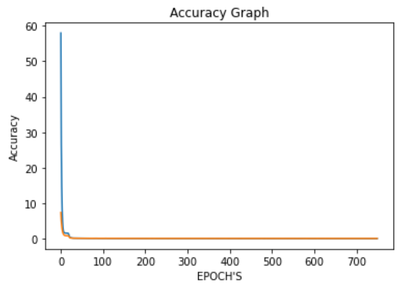
MSE: .1134
MAE: .1811
Time to Train: 57.88

Tanh



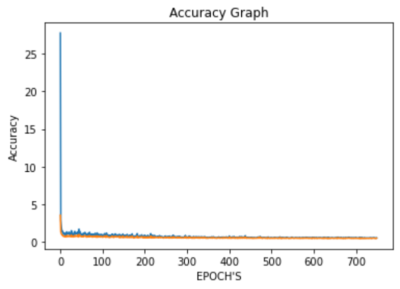
MSE: .1182
MAE: .1321
Time to Train: 1:11.41

Sigmoid



MSE: .1193
MAE: .1274
Time to Train: 55.80

Linear

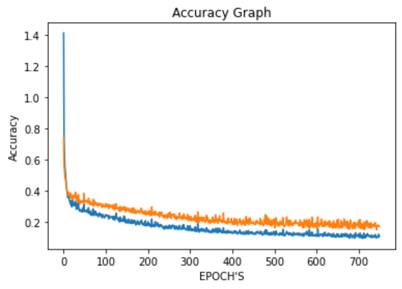


MSE: .5443
MAE: .5024
Time to Train: 1:00.87

Most of the results were relatively the same, except for linear, which is our outlier. All three of these options seem to be good options as they produce very similar results. We will stick with Relu moving forward because it is the suggested optimizer for RNN models.

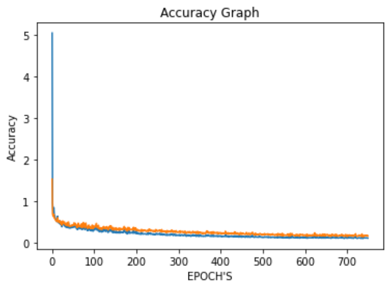
Next, I adjusted the unit number and layer size. Since our model already had a very good MSE and MAE, we did some light changes. We first adjusted the second layer by changing it to either 32, 64, 128, or 256 (all multiples of the original value) We then compared the MSE and MAE below. In previous examples, we set the first layer to 32 and the second layer to 256.

Layer 1: 32
Layer 2: 32



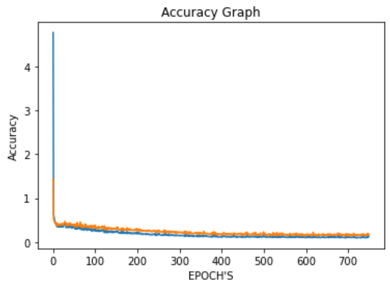
MSE: .1081
MAE: .1722
Time to Train: 55.287

Layer 1: 32
Layer 2: 64



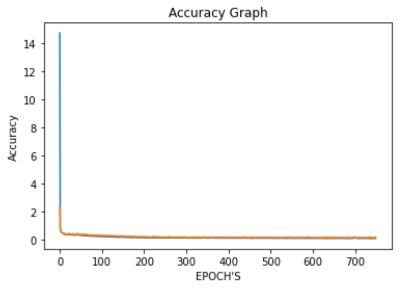
MSE: .1073
MAE: .1553
Time to Train:59.97

Layer 1: 32
Layer 2: 128



MSE: .1368
MAE: .1798
Time to Train: 1:28.43

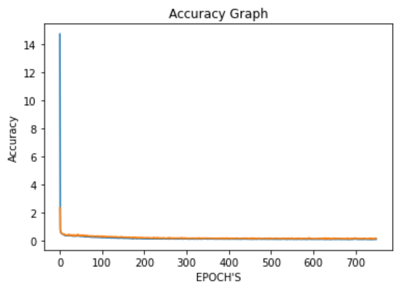
Layer 1: 32
Layer 2: 256



MSE: .0097
MAE: .1601
Time to train: 1:00.87

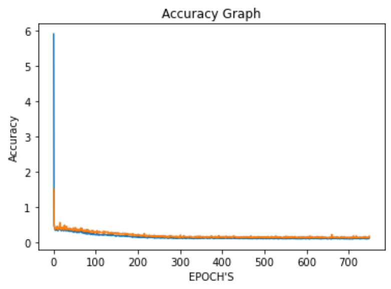
After adjusting the second layer, we confirmed lowest MSE to be when we have the layer with 256. Because of this, we continued to test with 256 and adjusting the first layer. The MSE and MAE scores along with the values in each layer are below.

Layer 1: 32
Layer 2: 256



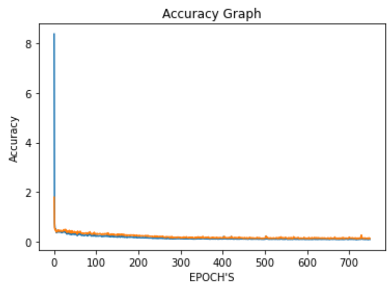
MSE: .0097
MAE: .1601
Time to Train: 1:00.87

Layer 1: 64
Layer 2: 256



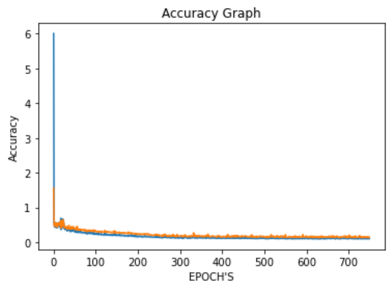
MSE: .1488
MAE: .1790
Time to Train: 1:41.63

Layer 1: 128
Layer 2: 256



MSE: .1017
MAE: .1397
Time to Train: 1:14.90

Layer 1: 256
Layer 2: 256

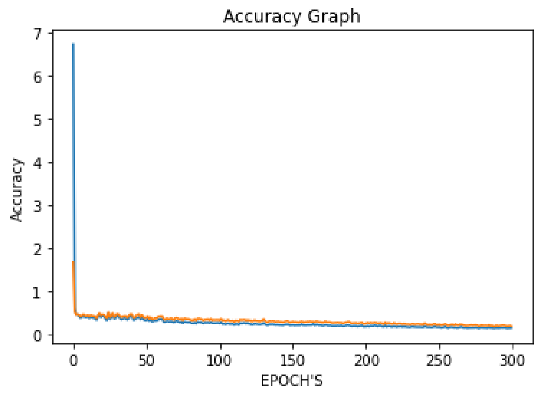


MSE: .1087
MAE: .1481
Time to Train: 1:17.43

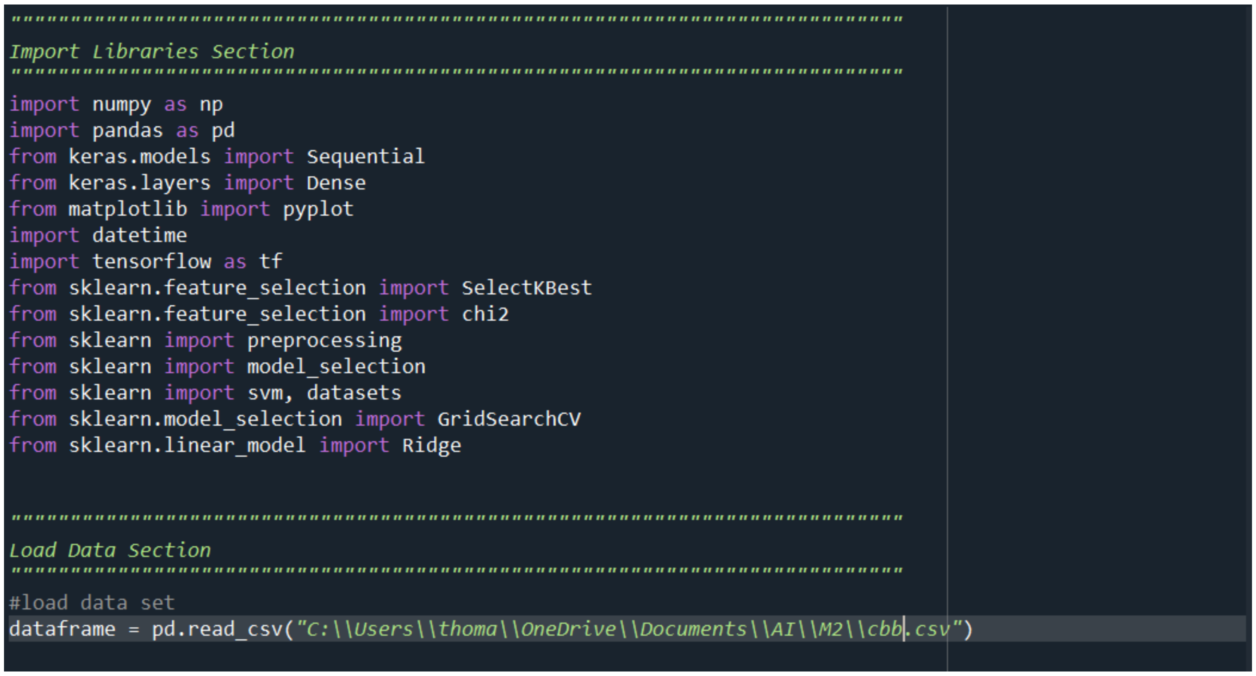
Based off the results, the lowest MSE was when we set the first layer at 32 and the second at 256. Due to these results, we completed testing the models and kept them at these scores.

Lastly, I settled on 300 Epochs. We wanted to test on 750 to see if there was any change, but there were no major changes in our final model. At 300 Epochs, there were no longer any changes in the model. The average test model took around one minute to test throughout our different tests and our final model now runs at 32.497 seconds and finishes with a mse of .1403 and an mae of .1833.
Our final model has the defining characteristics:

- Uses only variables that the Ridge model finds to be the most important on final scores
- Uses the adam optimizer
- Uses the relu activation function
- First layer: 32
- Second layer 256
- Epochs: 300

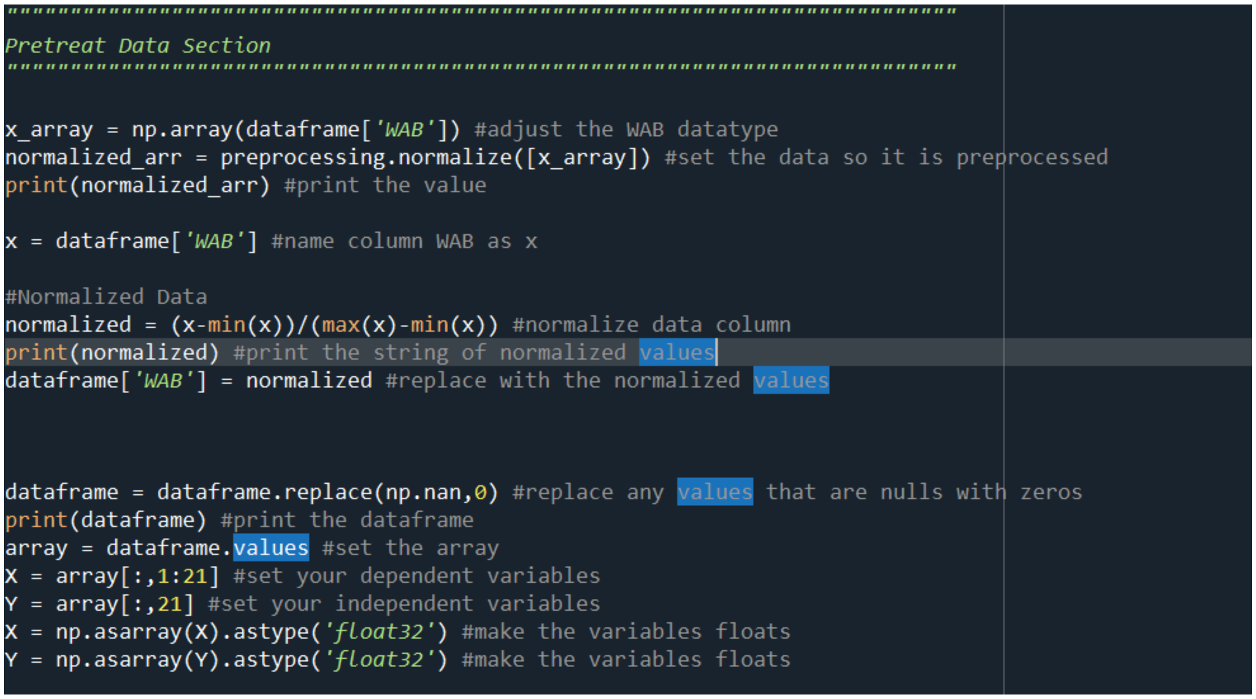


Final Code



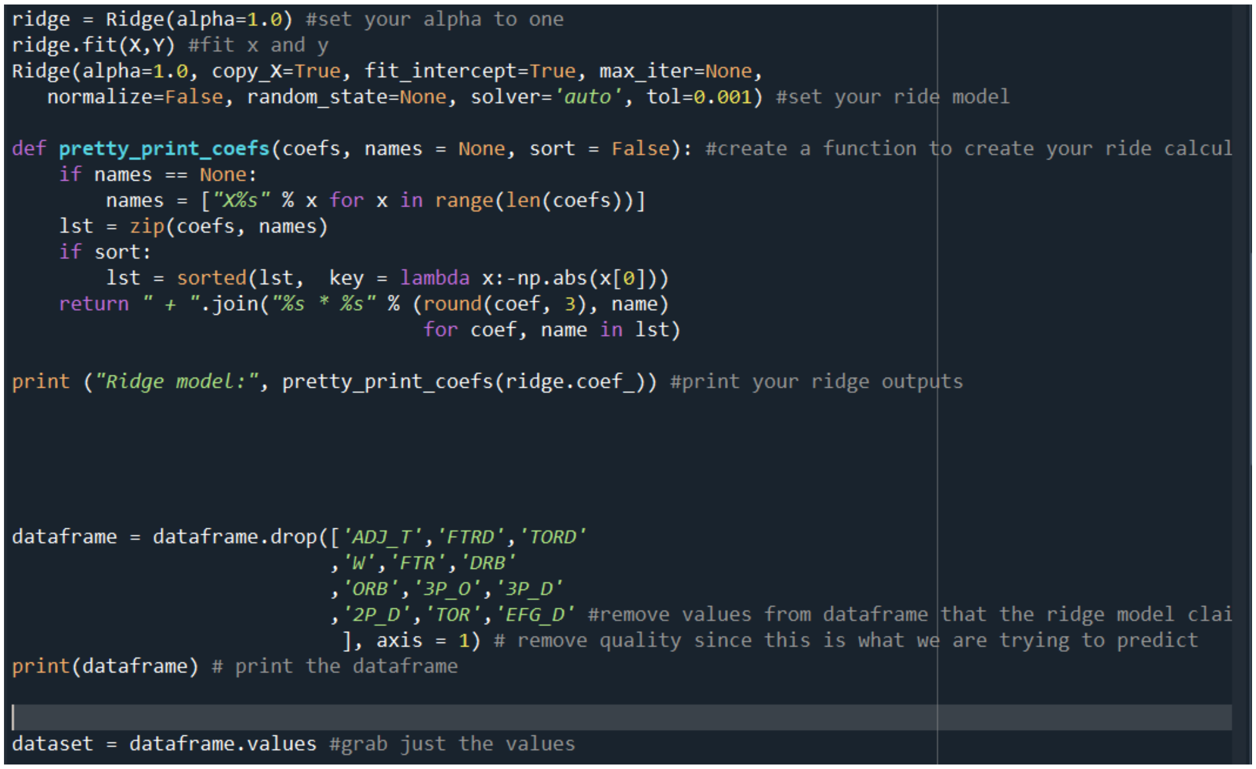
```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from matplotlib import pyplot
import datetime
import tensorflow as tf
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn import preprocessing
from sklearn import model_selection
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge


"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#load data set
dataframe = pd.read_csv("C:\\Users\\thoma\\OneDrive\\Documents\\AI\\M2\\cbb.csv")
```

In the above code we load in the necessary libraries and load in our excel document containing the data.



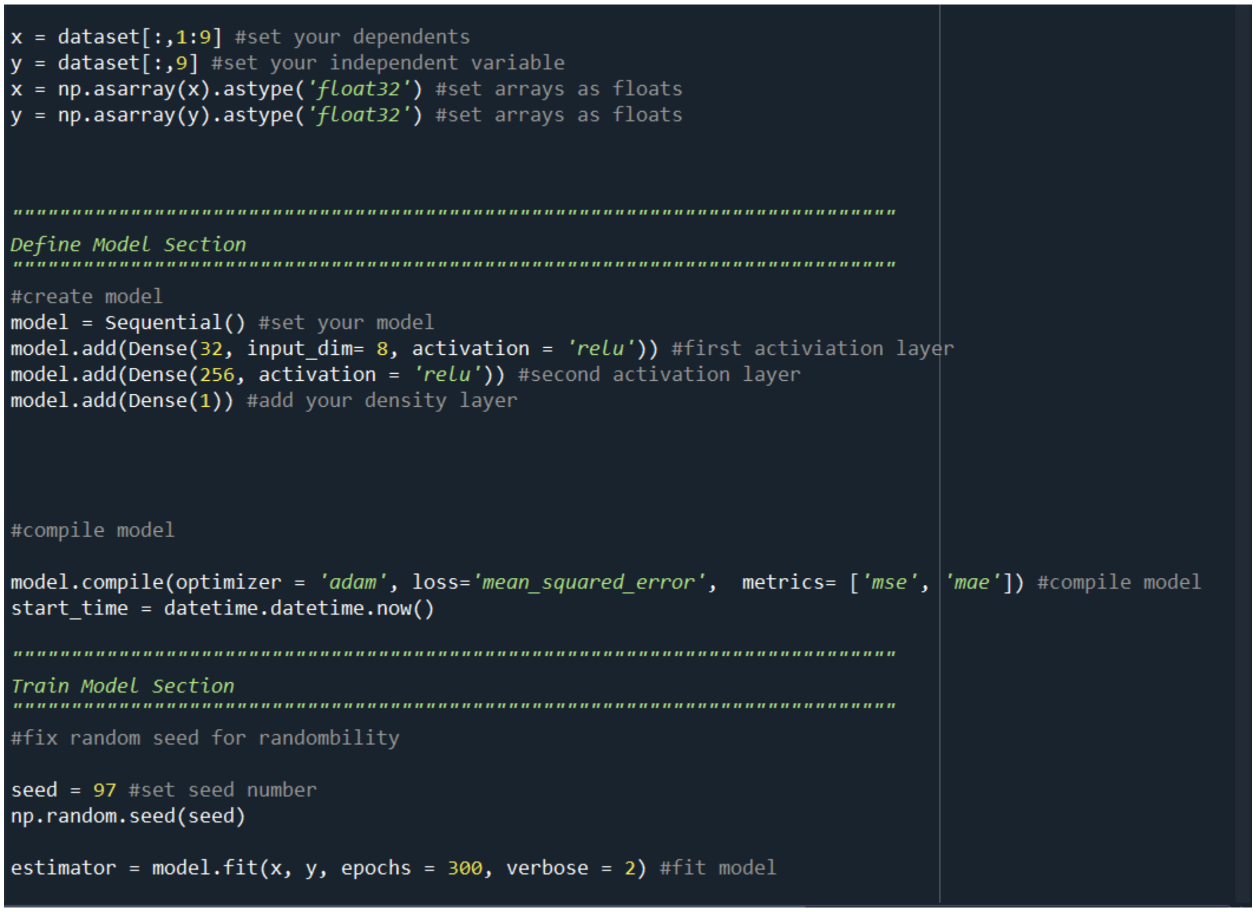
```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Pretreat Data Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

x_array = np.array(dataframe['WAB']) #adjust the WAB datatype
normalized_arr = preprocessing.normalize([x_array]) #set the data so it is preprocessed
print(normalized_arr) #print the value

x = dataframe['WAB'] #name column WAB as x

#Normalized Data
normalized = (x-min(x))/(max(x)-min(x)) #normalize data column
print(normalized) #print the string of normalized values
dataframe['WAB'] = normalized #replace with the normalized values



dataframe = dataframe.replace(np.nan,0) #replace any values that are nulls with zeros
print(dataframe) #print the dataframe
array = dataframe.values #set the array
X = array[:,1:21] #set your dependent variables
Y = array[:,21] #set your independent variables
X = np.asarray(X).astype('float32') #make the variables floats
Y = np.asarray(Y).astype('float32') #make the variables floats
```

In the code above we first normalize the values for the WAB column. We do this to run our ride model. We then ensure that there are no values with zero and change the numbers to floats.



```
ridge = Ridge(alpha=1.0) #set your alpha to one
ridge.fit(X,Y) #fit x and y
Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001) #set your ride model

def pretty_print_coefs(coefs, names = None, sort = False): #create a function to create your ride calcul
    if names == None:
        names = ["X%s" % x for x in range(len(coefs))]
    lst = zip(coefs, names)
    if sort:
        lst = sorted(lst,  key = lambda x:-np.abs(x[0]))
    return " + ".join("%s * %s" % (round(coef, 3), name)
                                   for coef, name in lst)

print ("Ridge model:", pretty_print_coefs(ridge.coef_)) #print your ridge outputs




dataframe = dataframe.drop(['ADJ_T','FTRD','TORD'
                           ,'W','FTR','DRB'
                           ,'ORB','3P_O','3P_D'
                           ,'2P_D','TOR','EFG_D' #remove values from dataframe that the ridge model clai
                            ], axis = 1) # remove quality since this is what we are trying to predict
print(dataframe) # print the dataframe


dataset = dataframe.values #grab just the values
```

Next, we run our ridge model. Based on the printed results, we remove certain columns that dop not have a great effect on the final model.

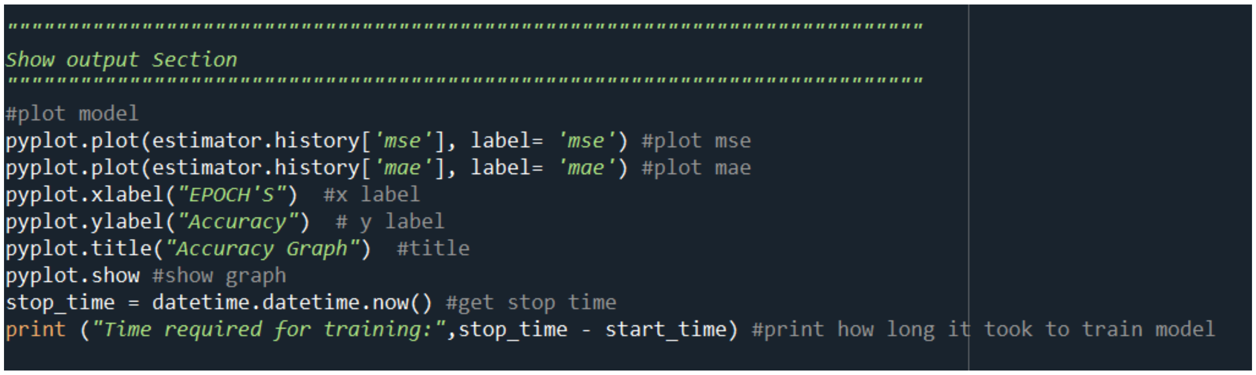


```
x = dataset[:,1:9] #set your dependents
y = dataset[:,9] #set your independent variable
x = np.asarray(x).astype('float32') #set arrays as floats
y = np.asarray(y).astype('float32') #set arrays as floats



""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Define Model Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#create model
model = Sequential() #set your model
model.add(Dense(32, input_dim= 8, activation = 'relu')) #first activiation layer
model.add(Dense(256, activation = 'relu')) #second activation layer
model.add(Dense(1)) #add your density layer




#compile model

model.compile(optimizer = 'adam', loss='mean_squared_error',  metrics= ['mse', 'mae']) #compile model
start_time = datetime.datetime.now()

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Train Model Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#fix random seed for randombility

seed = 97 #set seed number
np.random.seed(seed)

estimator = model.fit(x, y, epochs = 300, verbose = 2) #fit model
```

Next, we set our dependent and independent variables, and make them floats.

We then define the model, we set the model, then add two activation layers and a density layer. We then compile the model and want to derive the mse and mae metrics.

We then set the seed, setting it to 300 epochs.



```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Show output Section
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#plot model
pyplot.plot(estimator.history['mse'], label= 'mse') #plot mse
pyplot.plot(estimator.history['mae'], label= 'mae') #plot mae
pyplot.xlabel("EPOCH'S")  #x label
pyplot.ylabel("Accuracy")  # y label
pyplot.title("Accuracy Graph")  #title
pyplot.show #show graph
stop_time = datetime.datetime.now() #get stop time
print ("Time required for training:",stop_time - start_time) #print how long it took to train model
```

Lastly, we print a graph plotting the number of epochs on the x axis and the accuracy on the y axis.

Testing Procedure

To test our dataset, we used the final basketball statistics from the 2019 season. Testing on new data from a separate year not contained in the dataset will validate the model. This data contained the same columns and 353 rows of data. The same statistics are contained in both the training and testing data. We prepped the data the same way (normalizing WAB) and ran the ridge model. To keep testing the same, we removed the same variables in both sets.

Our results for our training dataset were similar to the results of the training data. With 300 Epochs, we returned an mse of .2012 and an mae of .2934. This model only took nine seconds to run, which is due to having fewer datapoints contained in the training set. Because of this, we can confirm that the model accurately predicts the finishing position of the 2019 NCAA basketball tournament.

I was impressed that the training developed an overall effective model that I would feel confident in using to predict future NCAA tournament brackets.

Other Considerations and conclusion

Overall, artificial neural networks help individuals determine the quality of their processes and evaluations. In the example, we were looking to predict the result of the men's college basketball tournament. The disadvantage to this is that adjusting models and tuning them takes time to be able to correctly predict and classify. In the example here, predicting basketball results is not a dangerous evaluation. Stock markets and weather forecasting can ultimately lead to financial burdens. There is no way to ensure that a model will be completely accurate, and this should be a consideration when making decisions using artificial neural networks.

While building this model, we had a few other considerations while building. The testing strategy involved backpropagation. This is defined as that the learning process is triggered by the results generated each time, we run the new model. Throughout the paper you saw me edit the inputs of the model and test the result. This is the process of backpropagation. In the future, it may be helpful to use feedforward or recurrent networks to avoid bias in our result.

Additionally, this model uses supervised learning techniques. We already know the results of the NCAA basketball tournament each year. This would be unsupervised learning if we were to predict the results before the tournament kicks off.

Input and output number were predetermined by the dataset. The input being the x values and output being the y value. We use the x for our input and y as our output because we removed different columns to decrease our MSE and MAE values.

Our training dataset contained 2455, containing tournament years from 2010 through 2018. We tested on the 2019 season which has 353 rows of data

Recommendations

Be cautious when using neural networks in predictions. No artificial intelligence platform will ever be completely accurate, so please consider before making decisions.

Appendix

The flow chart was created using excel and then copied and pasted into this document.