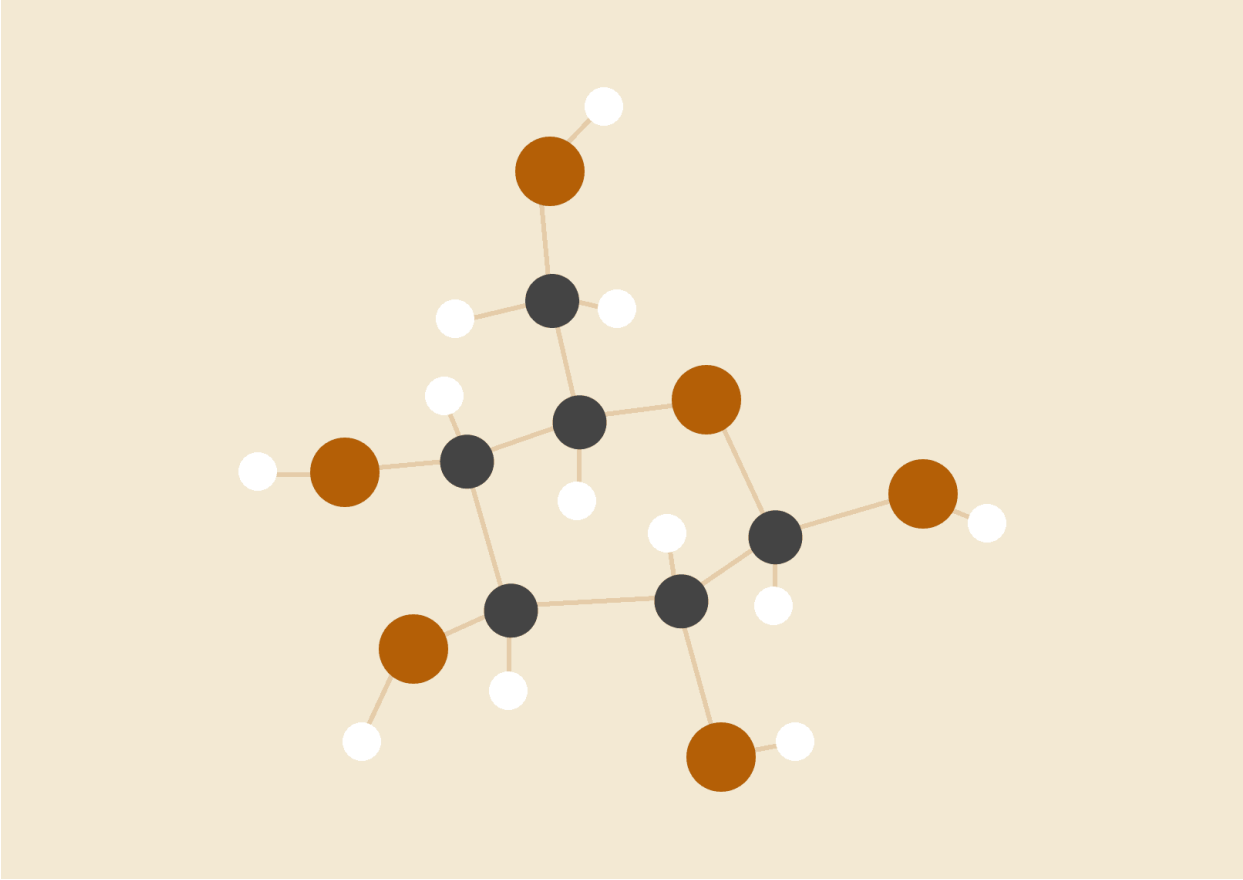
Neural Net and Tweets

Categorizing Congressional Tweets with Tensorflow



Derrick Jennings

05.02.2022

461

INTRODUCTION

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Data Preparation

First removed the full text column. This was done to reduce the scope of the problem. Then iterated over the “hashtags” column using pandas. Found the top 500 used hashtags and made a hashtag short list. Then added a variable for each tweet for each of the top hashtags. Needed to use the datafame.copy() method in pandas to prevent high fragmentation of the dataframe during insertion of extra columns. Then turned party_id into a 1 for Republican and 0 for Democrat. To do this used dataframe.loc[condition, ‘column’] = 0 or 1 depending on value. Also went through and replaced all nan values with 0. Then I normalized the data using the built in normalization function within tensor flow.

Network Configuration:

There are 5 layers in this configuration. The first 2 layers have 64 neurons, the third has 32 with a sigmoid activation function, the fourth has 2 neurons and the output layer has 1 neuron. The loss function was a MeanSquaredError loss function with a gradient descent optimizer with momentum.

1. Lorem ipsum dolor sit amet
2. Consectetuer adipiscing elit
3. Sed diam nonummy nibh euismod

Results:

First configuration was a sequential model with an input layer of 64 neurons and an output layer of one neuron. This produced about 69% accuracy on test data of 50,000. Changed the input layer to have as many neurons as there were columns in the input data and that lowered training accuracy by 1% from 68% to 67%.

Attempted a BinaryCrossentropy loss function as the literature suggested that this loss function was better used for cases in which there was only two possibly outcomes. It performed at around 58% accuracy in training. Then tried categoricalcrossentropy which was a low 50% accuracy in training and testing. MeanAbsoluteError loss function produced similar accuracy to MeanSquaredError loss function of about 67%.

Changed the optimizer from Adam to gradient decent with momentum optimizer showed faster learning that was consistent and didn't get stuck on a plateau. Then added one hidden layer of 64 neurons and noticed a 1% jump in training accuracy. Didnt notice much of a difference adding an additional hidden layer. When adding a hidden layer of 2 neurons raised training accuracy to 69% but the test accuracy stays the same. Removing the hidden layer of 32 neurons had no impact on accuracy.

Validation Strategy:

Took 50,000 rows for test data because that was about 10% of the total. Then took about 10,000 rows for the validation set. At first I was using 10,000 rows to test the model and was getting a higher accuracy by 1%.

Comments:

The results arent acceptable for any legitimate application but better than random so the model is picking up on something. I was surprised that 64 neurons as an input layer performed better than a 1:1 input layer. I think adding the text as data into the model might help improve accuracy. It seems that after I had gotten to a certain level of accuracy no matter what hyper perimeters I changed it only changed the way that the model approached that level of accuracy. This leads me to believe its a matter of the data its getting. Also I think capturing the hashtag information thats not from the top 500 might also help by adding another column that is true if a hashtag outside of the top 500 exist.

REFERENCES

https://www.kaggle.com/competitions/congressionaltweetcompetitionspring2022/data for data.

https://analyticsindiamag.com/ultimate-guide-to-loss-functions-in-tensorflow-keras-api-with-python-implementation/ for different loss functions.

https://www.tensorflow.org/overview Tensorflow documentation for building of model.