**DOKUZ EYLÜL UNIVERSITY**

**ENGINEERING FACULTY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**TURKISH SENTIMENT ANALYSIS WITH NEURAL NETWORKS**

**by**

**Berkay COŞKUNER**

**İlker SONER**

**Lecturer**

**Dr. Feriştah DALKILIÇ**

**January, 2021**

**İZMİR**

**TABLE OF CONTENTS**

**Chapter 1:Introduction……………………………………………………..……….…3**

**Chapter 2:****Dataset Preparation……………………………………………….…….....4**

**Chapter 3:Data Preprocessing……………………………….…………….……...…..6**

**Chapter 4:Models………………………………………………………………………9**

**Chapter 5:Conclusion and Results...…………………………………….……….......15**

**References…………………………...…………………………………….……….......16**

**CHAPTER ONE**

**INTRODUCTION**

Every day, more data is created than human processing. If all the people in the world sit down and try to read the data in 1 day, we still do not have time. For this reason, we need a more effective way to process the data. There is information that we cannot see with the naked eye. When you look at a table or reviews for a restaurant or comments for a product, you have a general idea, but you will never see the final result because you cannot read all of the comments. At this point, models are developed for processing the data. Our work here is to develop a model that can enable companies to test themselves. In this study, we tried to classify comments using Neural Networks. We tried to make an inference about whether a comment was positive or negative.

While making this classification, we used the comments on the Yemeksepeti.com website as our data. We prepared and pre-processed our data set ourselves. Afterwards, we developed 2 different Neural Network models and tried to predict which class (positive-negative) the comments belong to. We developed our first model in the Python programming language, on the PyCharm IDE, using the Tensorflow and Keras libraries. We developed our second model in R programming language, on RStudio IDE and using the neuralnet library.

The statistics we have obtained from the models we have developed are quite high and pleased us. We achieved around 85% accuracy in both models. We will examine the process and its results in more detail in the next sections.

**CHAPTER TWO**

**DATASET PREPARATION**

As we stated in the progress report, we prepared our own dataset from scratch. Briefly, we pulled all kinds of positive and negative comments from Yemeksepeti to apply sentiment analysis to them. We took comments from approximately 20 regions and 2000 restaurants. And we did not only use one city. We took different comments from İstanbul, Ankara, İzmir, Sakarya etc. For this first step, we used a framework called Selenium.

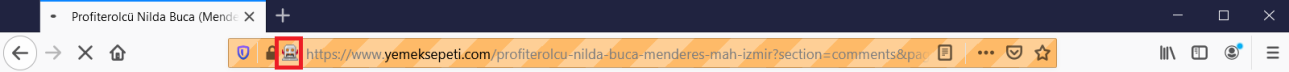


Figure 1:Selenium Working at Mozilla Firefox

Selenium is an open-source framework used for testing and automating web applications which is available for most of the browsers and programming languages. We used Selenium with Python and Mozilla Firefox. It basically travels around websites and gets what the user wants. Here is output of our code:

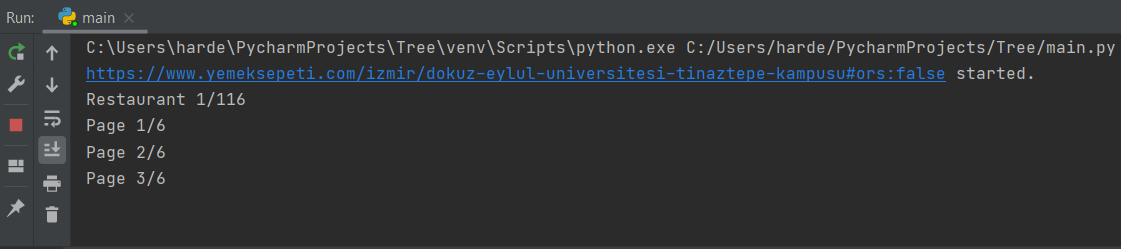


Figure 2:PyCharm Perspective

The first version of the comments was like this. As we explained above, we created 200 thousand comments like these as a data set using the selenium webdriver. The final comments are as follows:

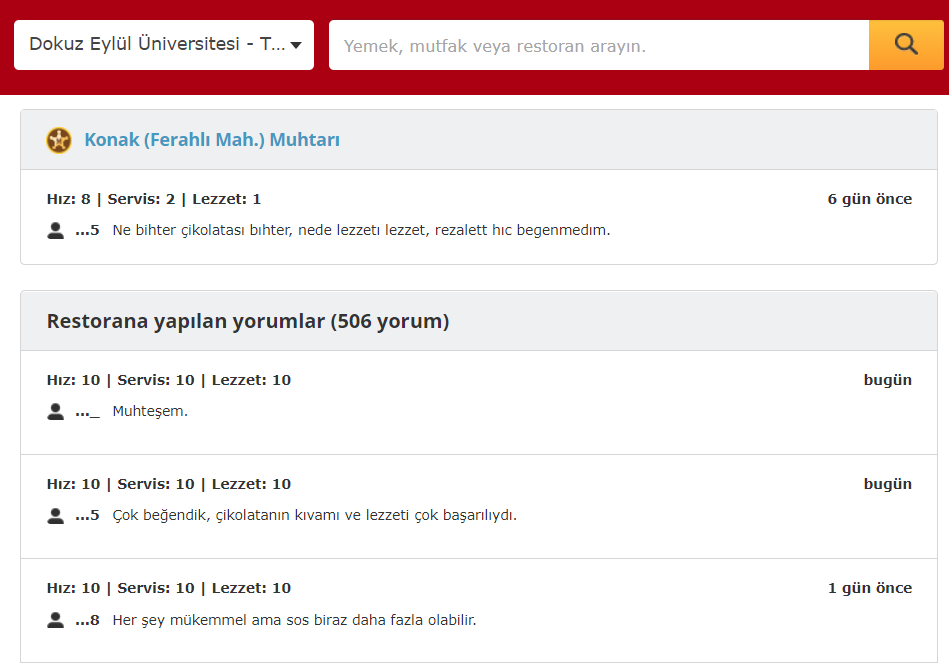


Figure 3:Before the Code Execution

There are 200 thousand comments in total on Dataset. The comments are evenly distributed, with 100k positive and 100k negative. The final version of the dataset:

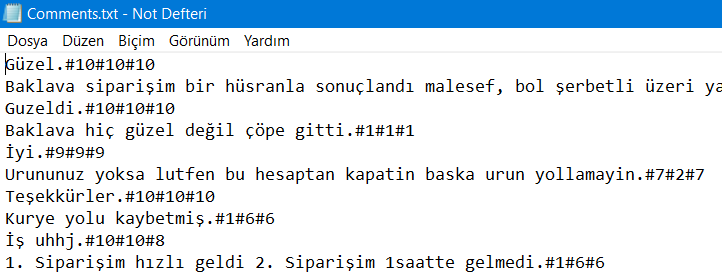


Figure 4:Dataset that We Used

**CHAPTER THREE**

**DATA PREPROCESSING**

After collecting our data from Yemeksepeti reviews, our first job was to determine whether the comments were negative or positive. As is known, there are 3 different scoring types in Yemeksepeti. These are speed, service, and taste. We took the average of these ratings for each review and gave positive attribute to those who scored 7 and above, and negative attribute for the remaining comments. After this process, the target attribute was ready.

The comments then had to be cleared of unnecessary spaces, spelling, punctuation, and stop words. First, we cleared out the spaces and unnecessary characters in our comments. Then, with a Turkish spelling correction library we found, we cleared our dataset from spelling mistakes as much as possible. While doing this, we aimed to make the library work better and we created a keyboard matrix to choose the most optimal spelling correction from this library which gives several outputs. When a person is typing incorrectly on the keyboard, they are more likely to press one of the keys next to the key they want to press. So, we put the letters around that key into a matrix for each letter on the keyboard. And we used this matrix to calculate the distance between the predicted word and the misspelled word while correcting the misspelling.

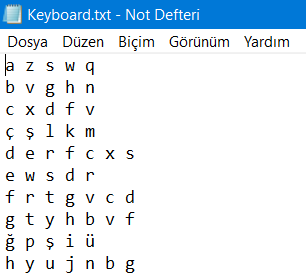


Figure 5:Keyboard Representation

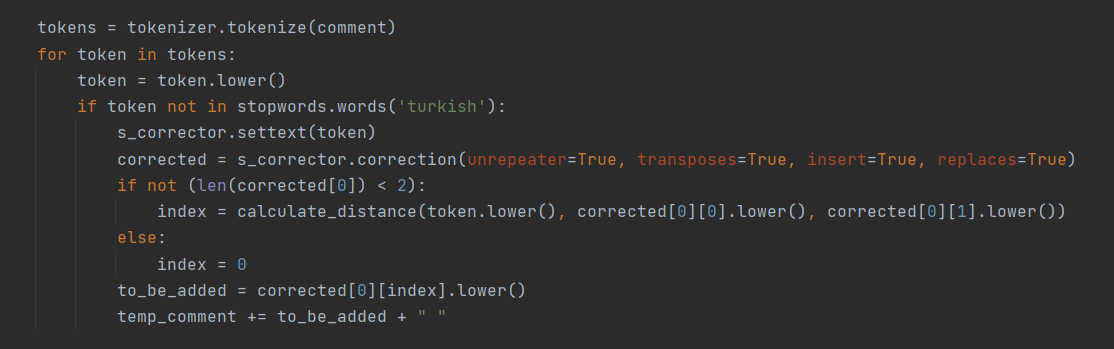


Figure 6:Comment Correction Algorithm

After our data set was cleaned, we divided our data set into two for train and test sets using a 90 percent rate. We have translated these words into vectors using Word2Vec. The working principle of Word2Vec basically comes from calculating the probability of a word occurring by looking at the words around it. To include a word in our data set, it must occur at least 10 times. The algorithm we use to create word vectors is Skipgram. We chose 64 as the vector length, a length we experimented with. Here is the word vectors.

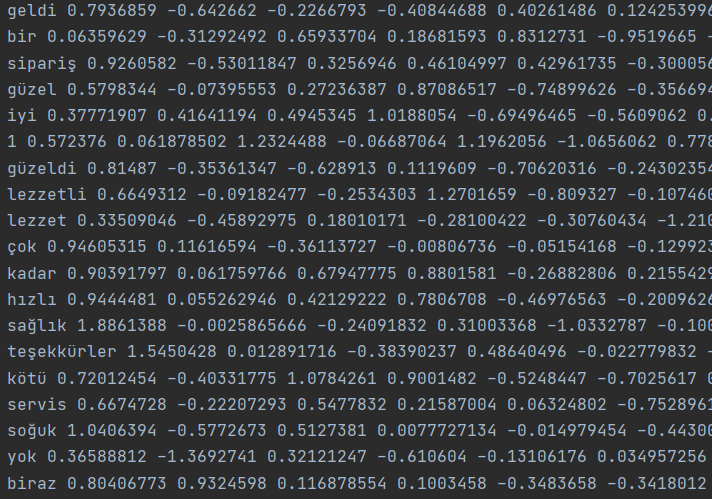


Figure 7:Final Dataset

Using these vectors, we have likewise used 64-length vectors of each comment. We want to explain our algorithm here through an example. As an example comment:

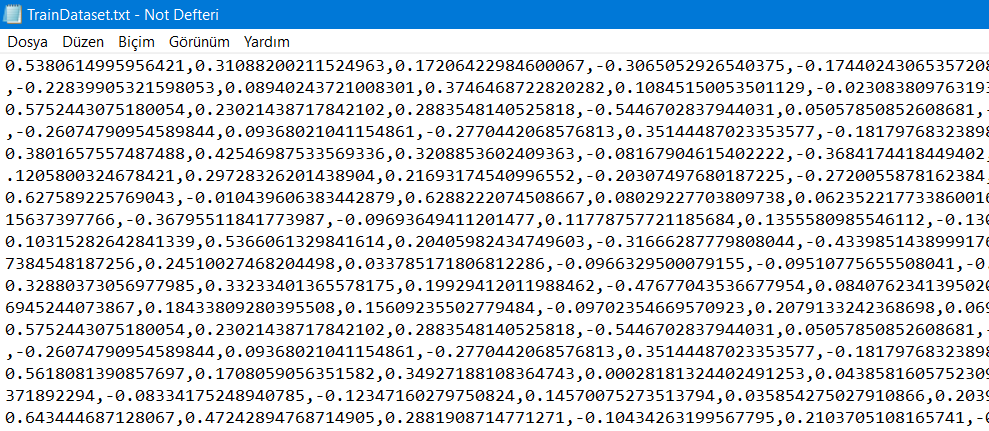
“Yemeklerinizi çok beğendim,teşekkürler.”

The words in this comment will become the following after our word correction algorithm: “yemek beğenmek teşekkürler”

Once this is done, we add the vectors of each word and take the average. This vector becomes our comment vector.

Sample Calculation: (Vector(yemek)+Vector(beğenmek)+Vector(teşekkürler))/3

And finally, our interpretation vectors are prepared. Because it consists of a large number of numbers, we cannot show all of them, but we show a part of a txt file consisting of approximately 170 thousand lines with 64 numerical values ​​and 65th value added as the value 0 or 1. Here is the final version of the dataset before it goes to the Neural Network Models.



**CHAPTER FOUR**

**MODELS**

1. **Phyton**

As we mentioned before, our first model is our model that we developed using Tensorflow and Keras libraries on Phyton.

In the data preparation section, we mentioned that we convert each interpretation into 64 long vectors. Our model takes 90% of the 200 thousand train datasets. 180 thousand comments are equally distributed, there are 90 thousand positive and 90 thousand negative comments. We trained our model with these data. The image of our model is as follows;

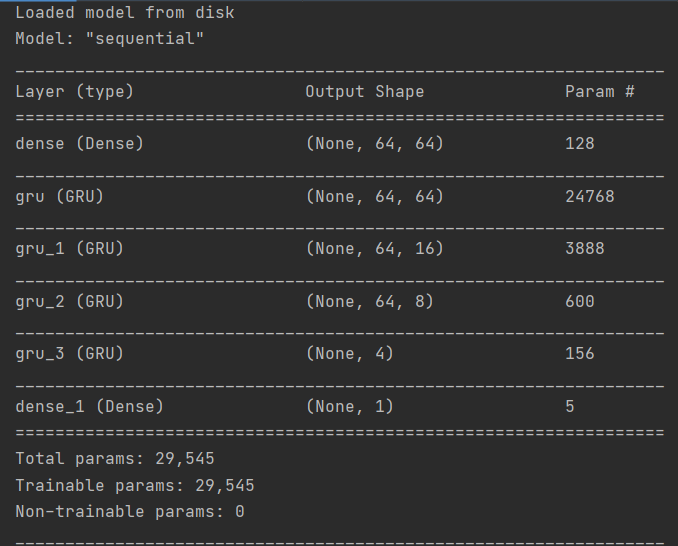


Figure 8:Phyton Model Summary

The statistics we get when we run the model are as follows:

Accuracy: 0.8552015423774719

Precision: 0.84687684

Recall: 0.86821239

F-Score: 0.8574119

ROC: 0.855164137593773

MSE: 0.14479842971463083

RMSE: 0.3805238884940482

MAE: 0.14479842971463083

R-Squared: 0.42080151436146573

Avg. Accuracy score of CV: 0.848575939

We think we achieved good accuracy despite working for approximately 4 hours. Below is an example of how the comments entered are classified as an example:

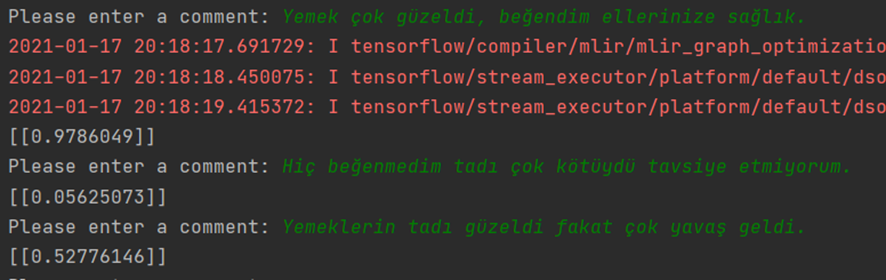


Figure 9:Classification Testing

When a clearly beautiful comment is made, the model can say that this comment is positive at a rate of 97%. Likewise, when a clearly bad comment is made, 99.5% can say that it is a bad comment. In the last case, we asked our model to classify an interpretation that even anyone could get in between, and our model was in between with 53%. The correct classification of this interpretation is positive. Since the output given by our model is rounded to the nearest integer, it guesses correctly. Only the percentage of confidence is low.

1. **R**

After creating a successful RNN model using tensorflow and keras in Python, we created a neural network model using the neuralnet library in R.

Again, we used 40 thousand of our own data as our train set and 19 thousand as our test set. The reason we did not use our original dataset and shrunk our dataset here was because the neuralnet library was running slowly. Even using the 40 thousand train sets took hours. We tried to run it on a 100 thousand data set, and when it exceeded 17 hours and it was still not finished, we gave up. Cross validating is really slow in neural networks, but we did cross validation in our 40 thousand in size dataset, even though it takes hours.

Since we have 64 vectors per column in our neural network dataset, we have 64 neurons on the input layer. There are 2 hidden layers and one of these layers has 4 and the other has 2 neurons. And of course, because we expect a positive or negative result as a result, we have only one neuron in our output layer.

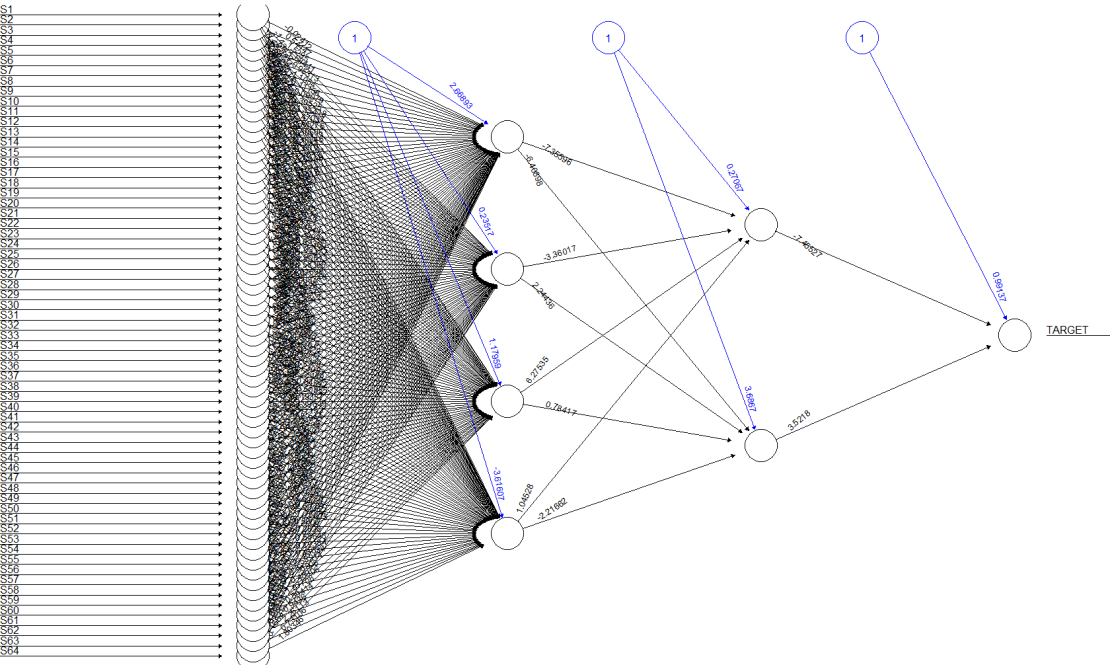


Figure 10:R Neural Network Model

After running, the evaluation metrics for neuralnet is as following:

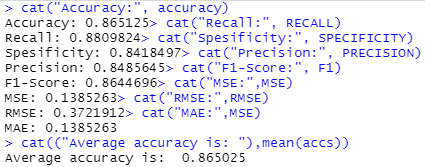


Figure 11:Results of R Model

Accuracy for a single one and the average for the cross validated one is quite satisfying even though we used a smaller dataset. But if we look at the testing below, we can see that this model works worse than the Python one.

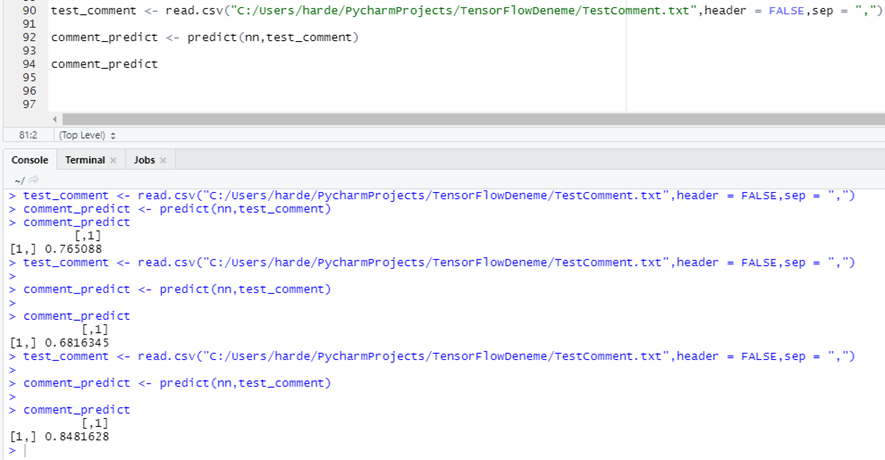


Figure 12:R Model Comment Testing

The 3 prediction we see in the image were completely correct in the Python part. But here the model classified the negative 2nd comment as positive and made it wrong. In the last comment, we expected it to stay in the middle, but the model sure said it was a good one.

As a result, the accuracy graph in the cross validation we made is as follows.

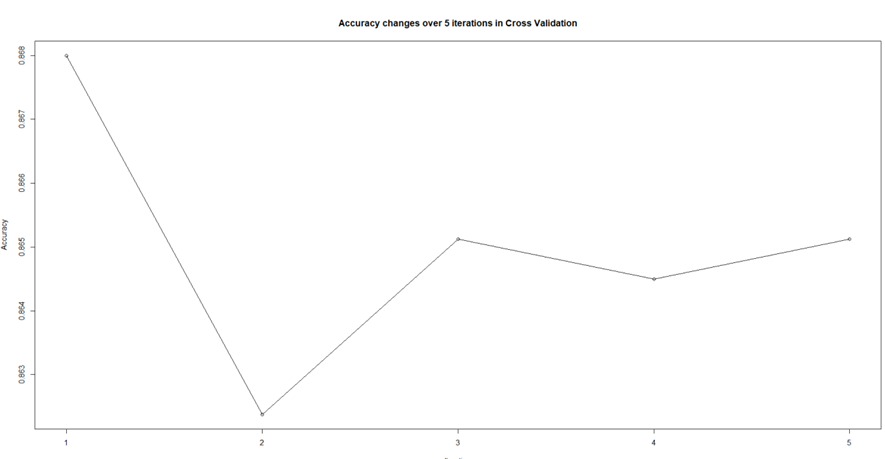


Figure 13:Accuracy of Cross Validation

**CHAPTER FIVE**

**CONCLUSION AND RESULTS**

As a result, there is not much difference in accuracy between the recurrent neural network model we use in Python using Keras and Tensorflow and the neural network model we use in R. Both have satisfying error rates. But when we last tested the models by giving a word to both models, our Python model gave more successful and sharp results, while our R model gave more unstable results even if the prediction was correct. Of course, this is because the infrastructure of the Python model is more powerful. In the Python model, the number of neurons in hidden layers, the technology used, and the recurrent neural network provide a great advantage for the model. The R model is not recurrent, and the number of neurons is less than the Python model. And though it takes longer. While the number of layers in Python and the number of neurons in the layers is very high (64,64,16,8,4,1), there were very few neurons and layers in R (64,4,2,1). We used GRU in hidden layers in Python. But since we did not have such a chance in R, all layers would be tried. The average accuracy of both models is the same, but the Python model has trouble classifying intermediate values, classifying outliers better. In R, the opposite situation is in question. It categorizes intermediate values very successfully while having difficulty in extreme values.

In fact, there are keras and tensorflow for R. We tried this before using the Neuralnet library, but abandoned it because it was a little different from Python and we got errors that had never been seen in the literature.

As a result, 2 models have their own strengths and weaknesses. You need to choose the model according to what kind of situation you will face. It was a project that we enjoyed at every stage. It has improved us a lot.

**REFERENCES**

Church, K. W. (2017). Word2Vec. *Natural Language Engineering*, *23*(1), 155-162.Zhou, L.,

*NeuralNet*. (n.d.)Rdocumentation. Retrieved January 17, 2021, from https:/www.rdocumentation.org/packages/neuralnet/versions/1.44.2/topics/neuralnet

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

*NeuralNet*. (n.d.). Rdocumentation. Retrieved January 17, 2021, from https:/www.rdocumentation.org/packages/neuralnet/versions/1.44.2/topics/neuralnet