CSC 180-01 Intelligent Systems (Fall 2023)

Project 2: Modern Low Footprint Cyber Attack Detection

William Moosakhanian, ID# 302537495

Xiong Moua, ID# 220278332

Due: October 13, 2023

Problem Statement

The problem statement of this project is to create a model that can determine between bad connections, intrusions or attacks, and good normal connections.

Methodology

The methodology of our project can be divided into two sections. For starters, when we initially created our models, we neglected to read the directions and created the multi-class classification models and confusion matrices. Later, when we discovered our mistake, we went back to redo our code to create the binary classification models as required.

The way we created our multi-class model was as follows: First we read all of our data and created the necessary data frames to house the training dataset and the testing dataset respectively. In this, we used *pandas.Dataframe.unique* to find and sort the unique values for each dataframe. The idea was to remove anything that wasn't present in the test data but was in the training data, which would prevent any inconsistencies during the model training and prediction process. We achieved this goal with the use of two nested for loops to find these values, concatenating them to a list which we could later manipulate. Afterwards, we used another nested for loop to traverse through each dataframe and find the indices of the respective rows that contained all the values in the aforementioned list. An issue we encountered was the list contained a large number of duplicates and all the indices were offset by one because of the way excel labels the cell numbers. Nonetheless, we removed them, and performed normalization on the numeric features and one-hot encoding on the categorical features.

The design of our FNN model was not as difficult compared to the other models. We used our first project as a guide. The number of neurons is equal to the amount of features in our model. In our case it was 199. We created a loop that would go through all the different optimizers and activators to reduce redundancy. This works great but there was an issue with the local optimum. The local optimum resulted in the model being unable to correctly load the weights. We got stuck, so we just decided to move on and kept it at only 1 for the local optimum. All the relevant information about the model such as the scores, the prediction values, etc. is stored in a list so we can reference it for use later. We decided to print out all of the FNN models because our model was overfitted for the basic model.

Designing the CNN model for this project was quite tricky. First we designed the model in four dimensions like the example given in the lab. Although the model was technically logically correct, it was not working as intended. Thus, we looked through lab 10 and found more information on reshaping the data size for the input. Professor Chen also helped us when we asked for some assistance on the topic by providing some clarification. For the CNN we essentially turned the x input into a 1-D array and the y-input into a 2-D array. We then used 6 combinations of activation functions and learning algorithms as well as 8 layers per model. We then printed everything out on confusion matrices and with precision/f1 score.

For the binary classification problem, we had a slightly different approach to preprocessing the data. We created a new column called ‘label_check’ which contains a numeric value that would determine whether an attack had taken place or not. If it was an attack, the value was set to 1, otherwise, it was set to 0 for normal circumstances. When we performed label encoding the values switched and this did cause some technical issues because the values were transposed. We corrected this with *pandas.Dataframe.replace* which swapped the numbers around and fixed the problem. Afterwards we performed one hot encoding and calculated the z-scores, and used the dataframe for the models.

Experimental Results and Analysis

Models	Activation	Optimizers	Layers	Neurons	Accuracy
Logistic Reg.	None	None	None	None	1.0
SVM	None	None	None	None	None
FNN	relu	adam	4	199, 66, 22, 2	1.0
FNN	sigmoid	adam	4	199, 66, 22, 2	1.0
FNN	tanh	adam	4	199, 66, 22, 2	1.0
FNN	relu	sgd	4	199, 66, 22, 2	1.0
FNN	sigmoid	sgd	4	199, 66, 22, 2	1.0
FNN	tanh	sgd	4	199, 66, 22, 2	1.0
CNN	relu	adam	8	16, 32, 64	1.0
CNN	sigmoid	adam	8	16, 32, 64	1.0
CNN	tanh	adam	8	16, 32, 64	1.0
CNN	relu	sgd	8	16, 32, 64	0.999
CNN	sigmoid	sgd	8	16, 32, 64	1.0
CNN	tanh	sgd	8	16, 32, 64	1.0
FNN_Add_1	relu	adam	4	189, 63, 21, 10	0.878
FNN_Add_2	relu	adam	4	189, 63, 21, 10	0.56
CNN_Add_1	relu	adam	8	16, 32, 64	0.88

CNN_Add_2	relu	adam	8	16, 32, 64	0.82

Firstly we would like to say that our basic model is overfitted. The results gathered are potentially biased. The logistic regression model was quite bad because it overfitted to an extreme. The SVM model gave us technical issues and we have no results for it. The FNN models all performed identically at 1.0 with an accuracy of around 99%. This was too high and it is our opinion that the FNN binary classification models were overfitted as well. Interestingly, when we originally made our multi-class FNN models, they all had an accuracy of around 84-87%. With the CNN binary models, we experienced the same issue of overfitting and only one model was slightly different which was relu and sgd.

The story was not the same for our additional features. The FNN model for multi classification performed with around 87.8 accuracy. The best model was chosen out of six different combinations and fitted to the confusion matrix. The CNN model was also very close at 88% and utilized relu and adam as its activation function and respective learning algorithm.

For oversampling features, our FNN model had the worst accuracy at 56%, but the CNN model was much better at 82%. Although it was at 82%, it still performed the best out of all of the models because it was able to predict more values. We think this is because the original data set was too small and caused a severe bias towards the ‘Normal’ category. We added in 60,000 samples total and that ultimately resulted in a better distribution of predictions.

Task Division and Reflection

The task was divided evenly among the two members of the team. Xiong worked on data preprocessing for the basic models, logistic regression model, CNN model, and additional features number one, multi classification . Will worked on the SVC model, FNN model, and data preprocessing for additional features number two, oversampling and undersampling. He added oversampling for Worms, Shellcodes, and Analysis attacks, by adding up to 20,000 entries for each type of attack. This bloated the data set to around 235,000 entries.

While working on this project, we learned to read the directions more carefully. We started off the project with multi classification rather than a binary classification. Unfortunately, we often found ourselves repeating tasks that could have easily been transformed into a reusable function. However, with designing these methods, we found that our program produced more errors and would set us back with hours of debugging, so we were forced to cut our losses and do things the repetitive way. One example of this was with the CNN weights not loading properly in a for loop whereas with the FNN model, they functioned perfectly fine within the loop.

Understanding how the CNN model worked was also quite difficult. Lab 10 was incredibly helpful in understanding how to make the model 1-D rather than 3-D compared to an image model, but it took us a while to properly fix and tune the model so it would perform decently well.

The largest technical difficulty we had was when we discovered halfway through this project that tensorflow was not using our GPUs but rather it was using our CPUs. The primary reason for this was because tensorflow 2.10 was the last version to natively support GPUs on Windows, and the version we had installed was 2.14. Will attempted to fix the issue on his end, but it completely broke his installation of Anaconda, and it set the group back a few hours because we were scrambling to repair it. Luckily, Will has an i7 8700k which was able to train the models in a decent amount of time, but this will be something we will try to fix for project 3 if we have spare time. There are also multiple issues that we did not account for. One of them is the local optimum. Our for loop does not load in the weights correctly for the next models to be trained. We did not have enough time to go back and refactor our code so that it works.

References

API References

Merging: https://pandas.pydata.org/docs/user_guide/merging.html

Mask: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.mask.html

Copy: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.copy.html

Head: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html

Value_Count:
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.value_counts.html

Shape: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shape.html

Indexing: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html

Boolean_Indexing:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#indexing-boolean

Read CSV: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

Duplicates: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.duplicated.html

Replace: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.replace.html

SVM: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.svm

Metrics: https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics

Logistic_Regression:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Unique Values: https://favtutor.com/blogs/pandas-unique-values-in-column

List Comprehensions: https://www.w3schools.com/python/python_lists_comprehension.asp