CoxCom Payment Fraud Detection Proof of Concept Documentation

# Section I: Introduction

This project was designed to see if CoxCom can leverage machine learning to predict and identify credit card fraud within the billing system. The technologies used and how they fit into the entirety of the project are as follows:

* **Anaconda:** Anaconda is a Python distribution that pretty much had everything we needed in one place. The primary utilities that we used from Anaconda are Jupyter Notebook, Python 3, TensorFlow, Pandas, and NumPy.
* **Jupyter Notebook:** This was the environment that was used to work with what we had. We used Jupyter to execute code cell-by-cell.
* **Python 3:** Used as the primary language since it had so much support for data scientists already.
* **TensorFlow:** This was the machine learning library that was utilized to create the model that would be trained and used to give CoxCom predictions into whether a given amount of accounts are fraudulent or not.
* **Splunk:** Splunk was leveraged to give us real-world data to test against. This was given in the form of text dumps.

The project is split into two portions. The first portion is the actual machine learning model. The second portion is a parser designed to parse logs from Splunk. There are two machine learning models that are trained and ready for further improvement. One is created with the Estimator API and the other with the Keras API. Details and differences will be described in-depth moving forward.

# Section II: TensorFlow Model

The TensorFlow model (which will simply be referred to as “model” now) is a combination of different parameters used to get the best predictions possible. The first thing that is included in the model is getting the data to fit in the model. The first few cells that are present after the log parser, which should always be the second cell in any of the notebooks, is where the training data is generated. NumPy is then used to convert the lists into arrays to convert the values within the array to values between 0 and 1 (greater than 0 and less than or equal to 1) to push the data through the models. From here, there is going to be a variable (or variables) that request a “shape”, “input dimension”, “units”, or anything of that nature. That variable(s) is dependent on how many features you have. In this case, we have 5 features (please reference the parser for a description of the project’s features). For this project, 5 or whatever number of features that were used, were put into a “shape”, “input dimension”, and/ or “unit”. From here, this is where we diverge into either the Estimator API or the Keras API.

## Section II.I: Estimator API – “Fraud Detection Model 2”

The notebook name of “Fraud Detection Model 2” contains the model that uses the Estimator API. With this model, there seems to be a lot of code, but control and visibility are added benefits. The main control point that seems to stick out is a developer can set where checkpoints can be saved. Other than that, the code may look a lot denser that other implementations, but every aspect of the model is visible.

## Section II.II: Keras API – “Keras Fraud Detection Model 2”

The notebook name of “Keras Fraud Detection Model 2” contains the model that uses the Keras API. With this model, the code is a lot simpler and implementation of a model is a lot simpler, but a lot of visibility of what the code is doing is taken out. The main point of using Keras is that it is simple, customizable, and easy to adjust, but some issues might occur if control is a large factor.

Now that both models have been covered, we now move onto the actual data and predictions of the model. Below each model, a classification report and a confusion matrix are provided. The classification report describes how accurate the model was against the actual results (known as “targets” in the models) and how well was the model able to recall data based on its training. As for the confusion matrix, please refer to the following link: <https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/>. Following the image provided on the website, the top portion for our case is all “prediction” while the left-hand side of the matrix is “actual”. This should help in understanding the confusion matrix, especially in the case of the models specific for this project. The last few cells are basically giving predictions to what accounts are suspected of fraud and what accounts may be fraudulent.

# Section III: Splunk Log Parser

The log parser that is included in both models (as well as a standalone file) was created to parse a Splunk log dump that was in a .txt file. The parser, however, has a few issues to it that need to be addressed before it is brought into use. The issues are as follows:

* The main issue is that the Splunk log format has changed. To utilize the parser again, all that needs to change are the various regex variables that are in the parser. Once they are fitted to the new format, the parser should be good to be utilized again.
* The final issue is the counter functions that are present throughout the parser. Within the class of “AccountInfo” there are five functions that are counters: “idCounter”, “ipCounter”, “payCounter”, “avgPayCounter”, and “ccCounter”. If an AccountInfo object has more than one infraction, the counters may drastically increase those infraction counts. For example, if Account Number 11110000 has 14 User IDs, the counter may miscount it to 30 User IDs.

Other than those issues, the parser should work as requested/ needed.

# Section IV: Issues

The only issues that are within the project are mentioned in Section III which are listed below again.

* The main issue is that the Splunk log format has changed. To utilize the parser again, all that needs to change are the various regex variables that are in the parser. Once they are fitted to the new format, the parser should be good to be utilized again.
* The final issue is the counter functions that are present throughout the parser. Within the class of “AccountInfo” there are five functions that are counters: “idCounter”, “ipCounter”, “payCounter”, “avgPayCounter”, and “ccCounter”. If an AccountInfo object has more than one infraction, the counters may drastically increase those infraction counts. For example, if Account Number 11110000 has 14 User IDs, the counter may miscount it to 30 User IDs.