Description:

Feature engineering process:

After looking at the data, we took manually the following features for each segment:

* Frequency commands within the segments
* Frequency of commands with only letters
* Frequency of commands with alphanumeric
* Frequency of commands with only numbers
* Frequency of commands with special chars – (+, -, \_, ., %, [, ])
* Frequency of commands with capital letters
* Frequency of commands contains only capital letters

Our ML approaches:

We understood this mission can be handled with different types of learning, we decided to examine 2 different approach:

1. Supervised learning – labels were given to the segments, by that XGBoost classifier may solves this problem. In order to solve the problem like this, we split the labeled data to train/validation sets (80/20) randomly, we made sure the frauded segments split up the same between the sets. Then we applied the model on the unlabeled data.

This approach yields to very few FP, but a lot of FN.

We tried first this approach first because it was very easy and straight-forward to train the classifier but after looking at the cross validation results we understood we need a model who may model the distribution of the data in a better way, what lead us to the next approach -> LSTM.

1. Unsupervised learning – LSTM encoder-decoder. As also mentioned in the course lectures, seq2seq attention model (LSTM encoder-decoder with attention layer) tends to perform well on anomaly detection problems.  
   Our model architecture:  
   Encoder – embedding layer followed by one lstm layer: each command in the segment presented as one-hot vector, by that the embedding layer is responsible on learning a better representation to the commands (We also thought on training a different embedding model as word2vec, if we had more time we would do that, we believe the embedding representation is very important here to the model performance.

Decoder - embedding layer followed by attention layer, followed by one lstm layer, and then 2 dense layers. We did a lot of evaluations with different combinations of layers for the decoder model, the 2 main things which help us the most (in terms of performance) were the attention layer and adding more than once dense layers after the LSTM layer. We also tried to use dropout because we thought if we make the model to use only part of its neurons it might improve the results but actually that wasn't helpful in practice.

Split data –

The first 50 segments of each user took for train the model, the next 100 segments of users 0-9 were taken as validation set to set the decision threshold.

We tried 3 different approaches to trained the LSTM:

1. One model for all the users.
2. One model for each user (**To capture user behavior analytics**).
3. Combination on 1+2, we took the model we trained for all the users, then we trained it again (fine tuning) each time on specific user. This approach yields our best performance.