NETWORK INTRUSION DETECTION-USING AUTOENCODERS

Sampath Vaddadi

INDEX

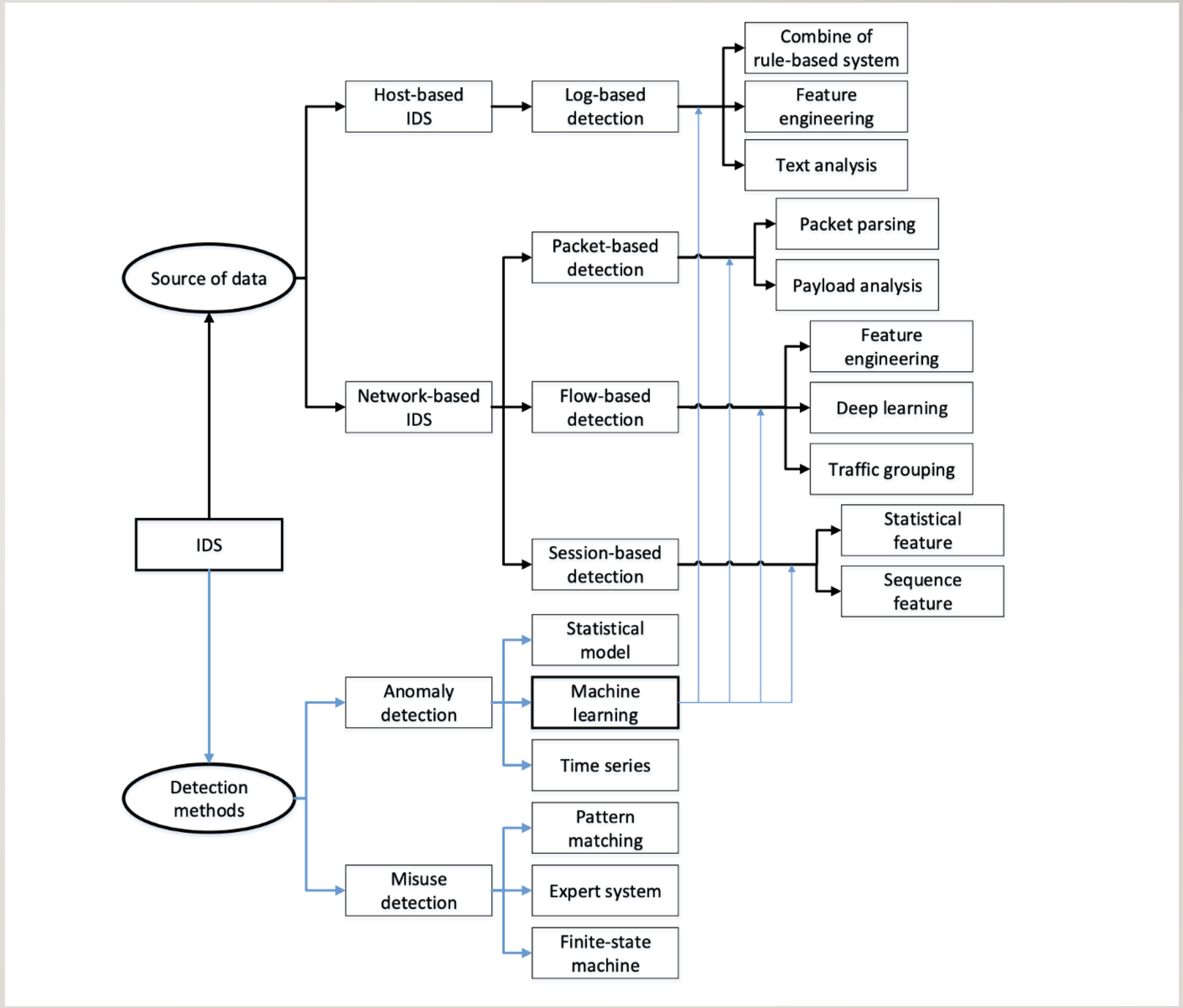
WHAT IS AN IDS?

- An intrusion detection system (IDS) which is an important cyber security technique, that monitors the state of software and hardware running in the network. Whenever there is an intrusion (attack/breach), it raises an alarm requiring the appropriate person to act.

OBJECTIVE

Networks play important role in modern life, and cyber security has become a vital research area. Despite decades of development, existing IDSs still face challenges in improving the detection accuracy, reducing the false alarm rate and detecting unknown attacks. To solve these problems, many researchers have focused on developing IDSs that use machine learning methods mainly because of their relatively lower dependence on domain expertise. The objective of this project is to develop an IDS (intrusion detection system) using self supervised learning and deep neural networks.

TAXONOMY OF IDS

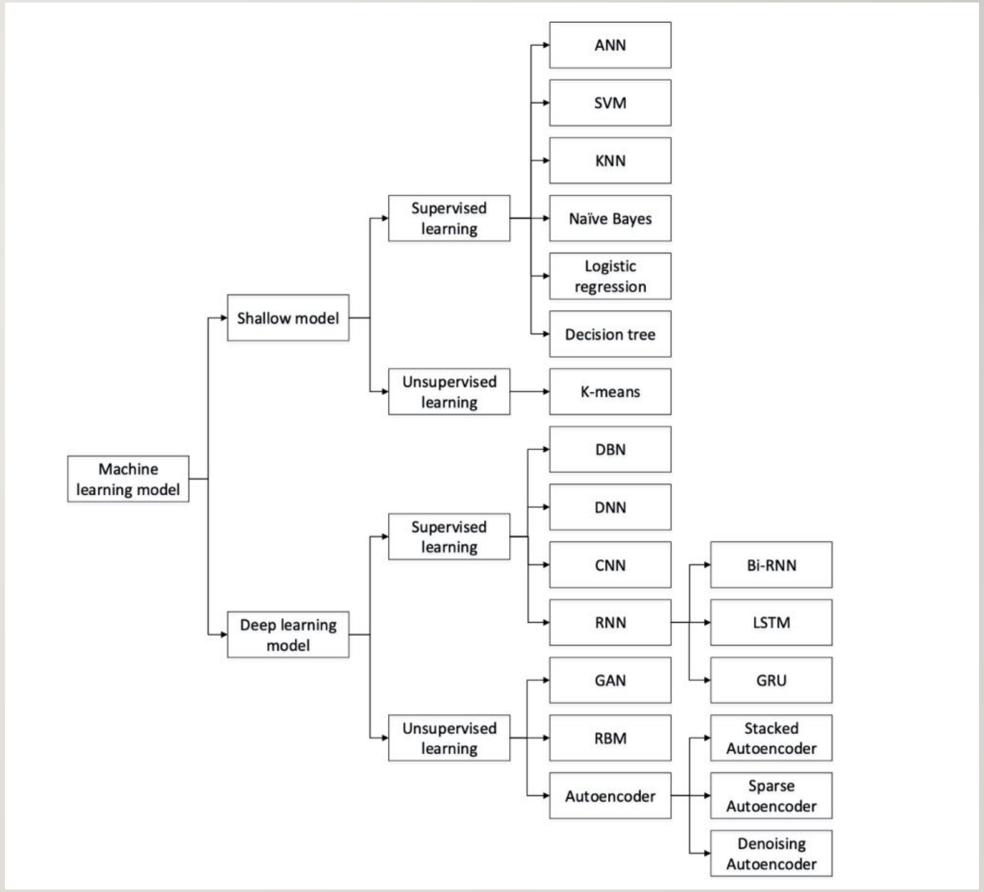


WHY DEEP LEARNING?

Deep learning-based IDSs can achieve satisfactory detection levels when sufficient training data is available, and deep learning models have sufficient generalizability to detect attack variants and novel attacks.

They seldom rely on domain knowledge; therefore, are easy to design and construct.

- In this work, we concentrate on the ML approach using deep learning networks and time-series principles. The basic requirement of solving time series problems using deep learning is the “data”. This data can be univariate/multivariate time-series data i.e. the data is recorded in a chronological order.

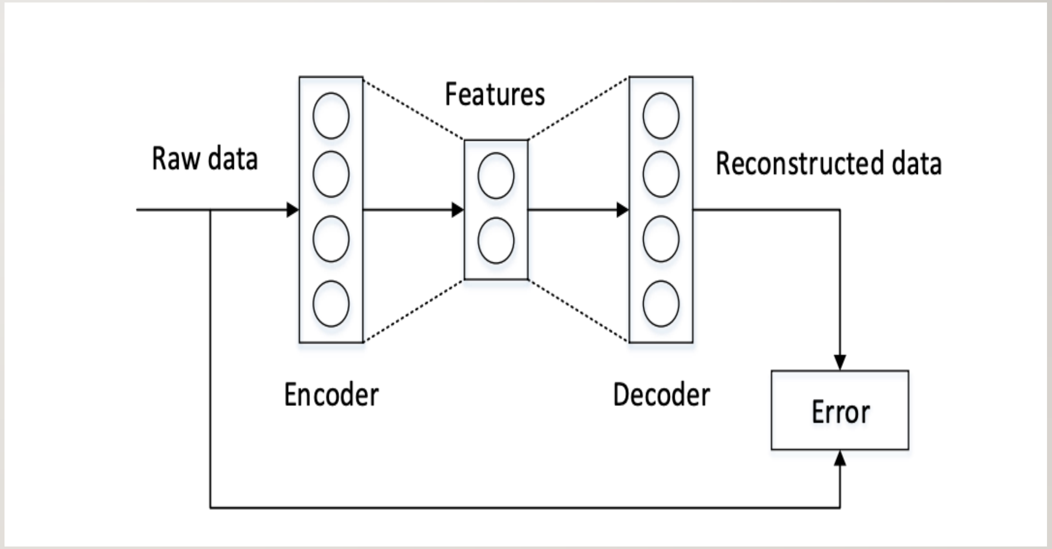


APPROACH

- In this approach, the problem of detecting attacks is framed as a task of anomaly detection.
- The reason behind this is that attacks generally occur very rarely, but when they do occur, their signature (to be precise, distribution of the data) is quite different from that of a normal operation condition.
- This change in signature is reflected in the time-series data. We use an Auto-encoder model to learn the distribution of a normal condition data.
- Using this model, we can identify if the incoming data has a significantly different signature.

AUTO-ENCODERS

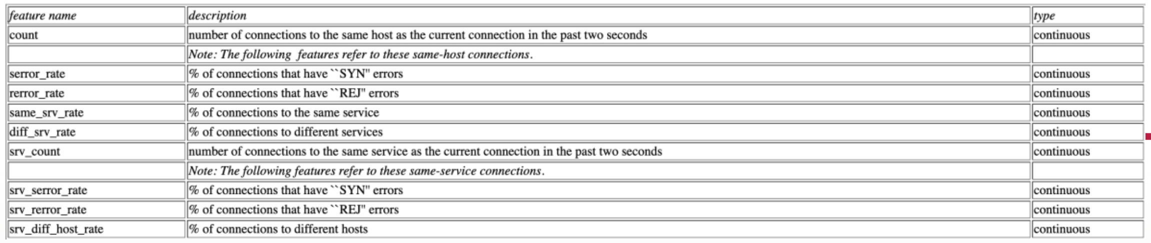
- Auto-encoders are neural networks comprising two symmetrical components, an encoder and a decoder.
- The encoder extracts features a.k.a latent representations from raw data, and the decoder reconstructs the input data from these latent representations.
- During training, the divergence between the input of the encoder and the output of the decoder is gradually reduced and hence, the latent representations coming out of encoder gradually tend to represent the essence in the original data while throwing away all the higher dimensional details.



DATASET DESCRIPTION

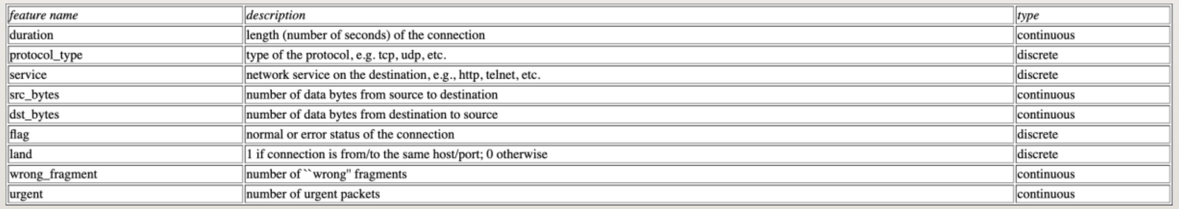
- To model our intrusion detection learning task we use the KDD99 dataset. It is the most widespread IDS(Intrusion Detection System) benchmark dataset.
- Its compilers extracted 41-dimensional features from a raw dataset called DARPA1998 dataset containing both raw TCP(Transmission Control Protocol) packets and labels since raw packets are not of much use for ML models. Because of this reason, a new dataset was curated called KDD99 dataset.

Traffic features computed using a two-second time window



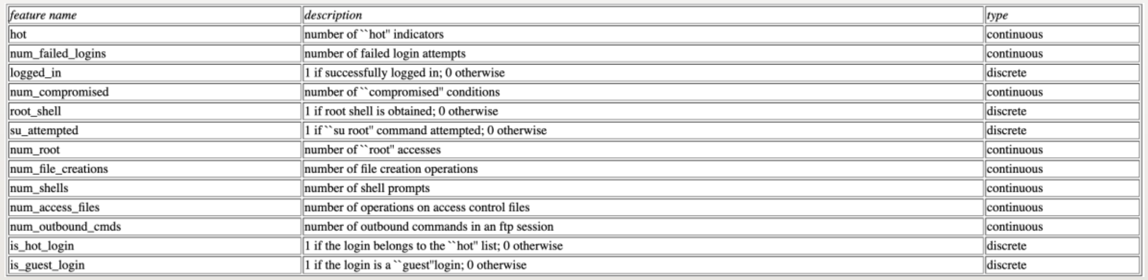
feature name	*description*	*type*
count	number of connections to the same host as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-service connections.	
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Basic features of individual TCP connections



feature name	*description*	*type*
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous
urgent	number of urgent packets	continuous

Content features within a connection suggested by domain knowledge

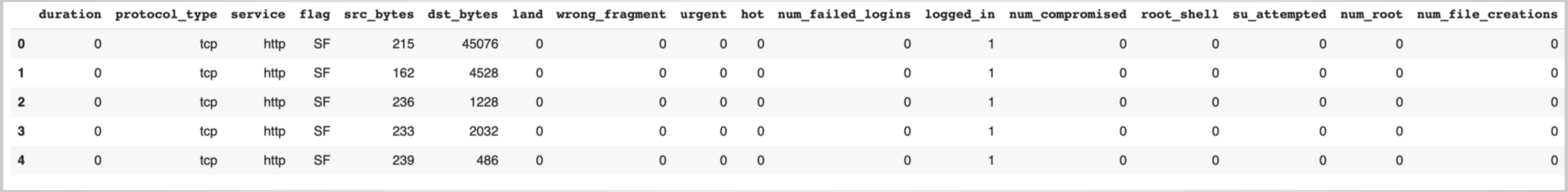


feature name	*description*	*type*
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete

EXPLORATION AND DATA PREPROCESSING

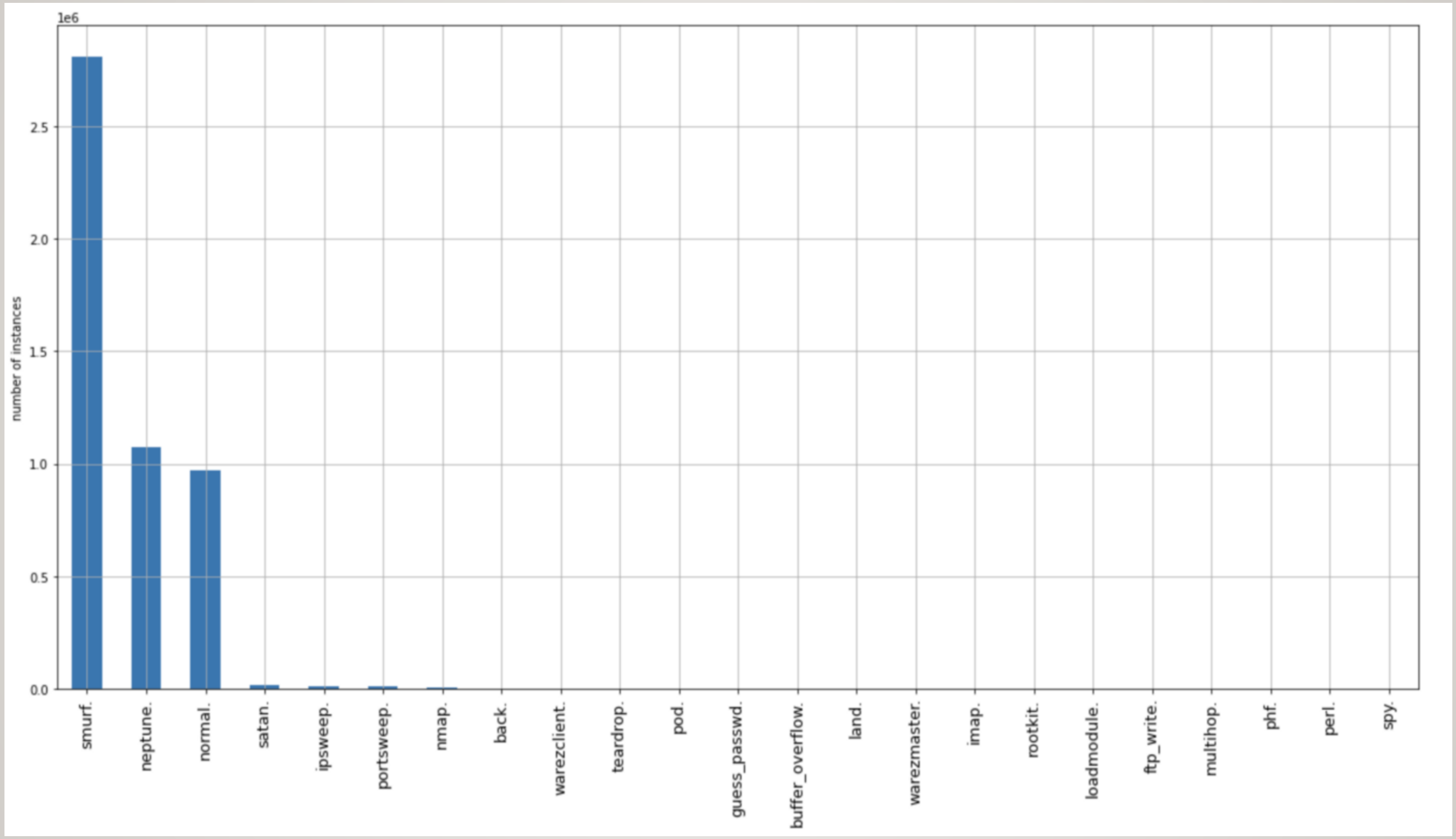
- This training data set has more than 4 million rows, out of which only ~20% are normal.
- There are different types of attacks such as: back dos, buffer_overflow u2r, ftp_write r2l, guess_passwd r2l, etc.
- In the test set, in addition to the attacks present in the training set, there are some new attacks present. These new attacks are variants of existing attacks in the training set. This is done on purpose to measure the model's effectiveness on unseen attacks. However, we do not intend to classify different types of attacks and proceed by treating all of them as anomalies.

- First 5 rows of the data frame

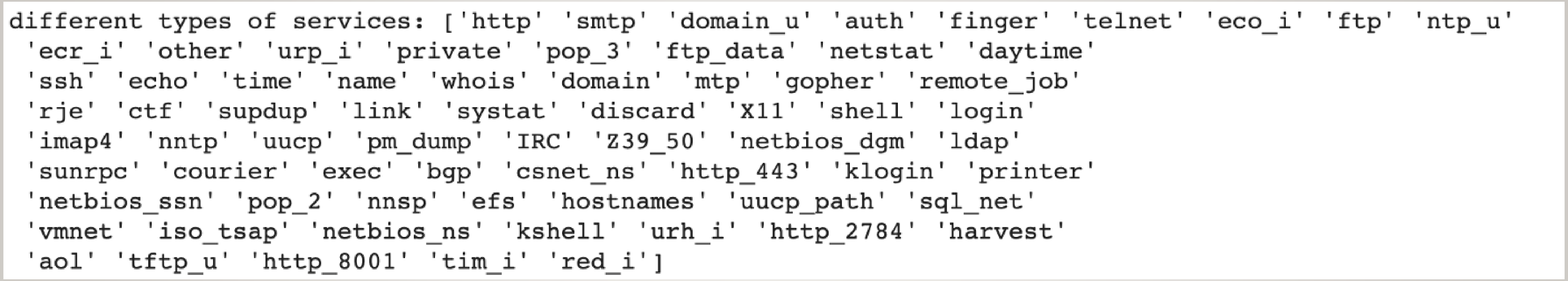


	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	root_shell	su_attempted	num_root	num_file_creations
0	0	tcp	http	SF	215	45076	0	0	0	0	0	1	0	0	0	0	0
1	0	tcp	http	SF	162	4528	0	0	0	0	0	1	0	0	0	0	0
2	0	tcp	http	SF	236	1228	0	0	0	0	0	1	0	0	0	0	0
3	0	tcp	http	SF	233	2032	0	0	0	0	0	1	0	0	0	0	0
4	0	tcp	http	SF	239	486	0	0	0	0	0	1	0	0	0	0	0

- Histogram showing the number of instances of different attack types



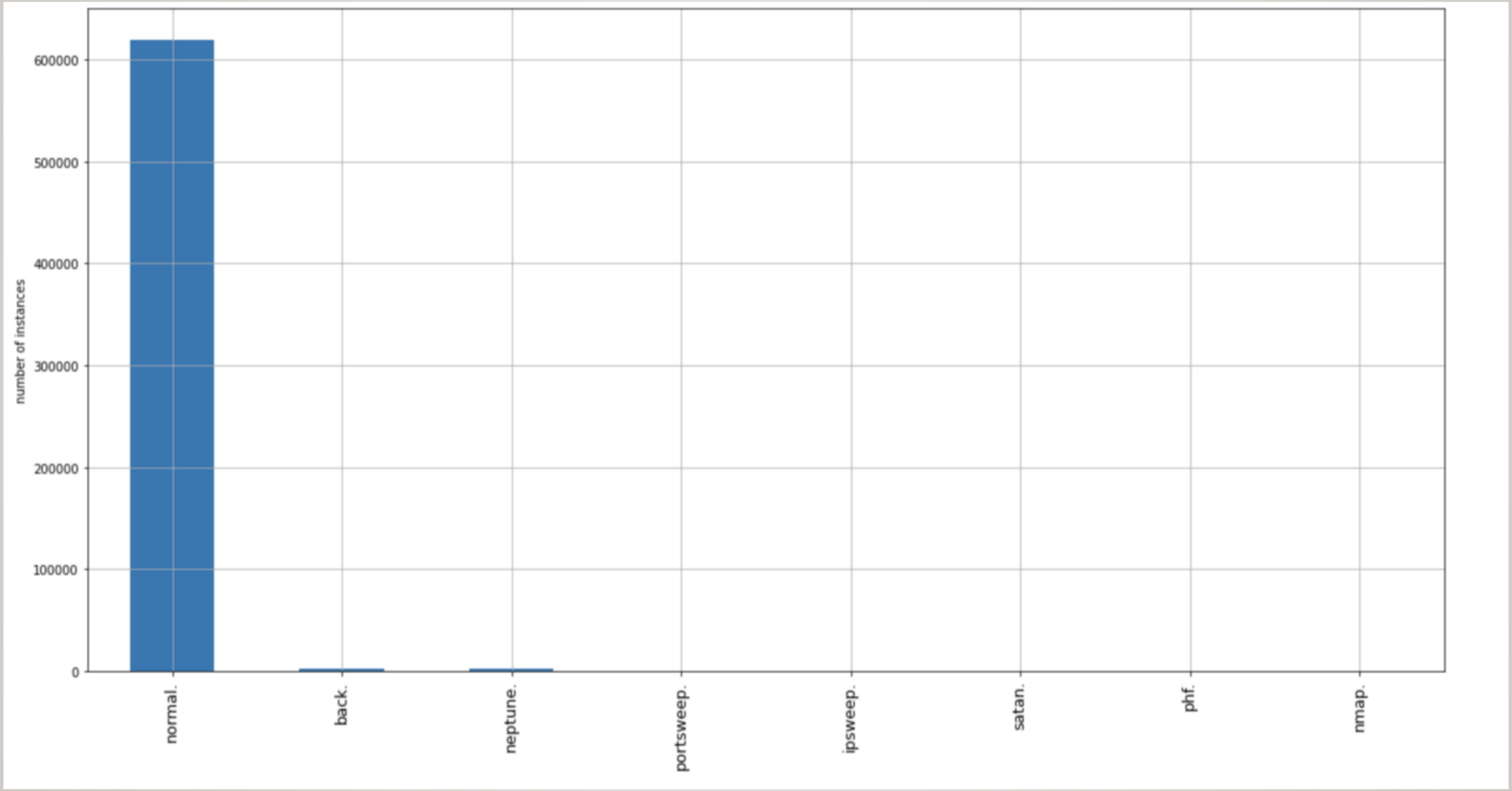
- Different types of services present in the data



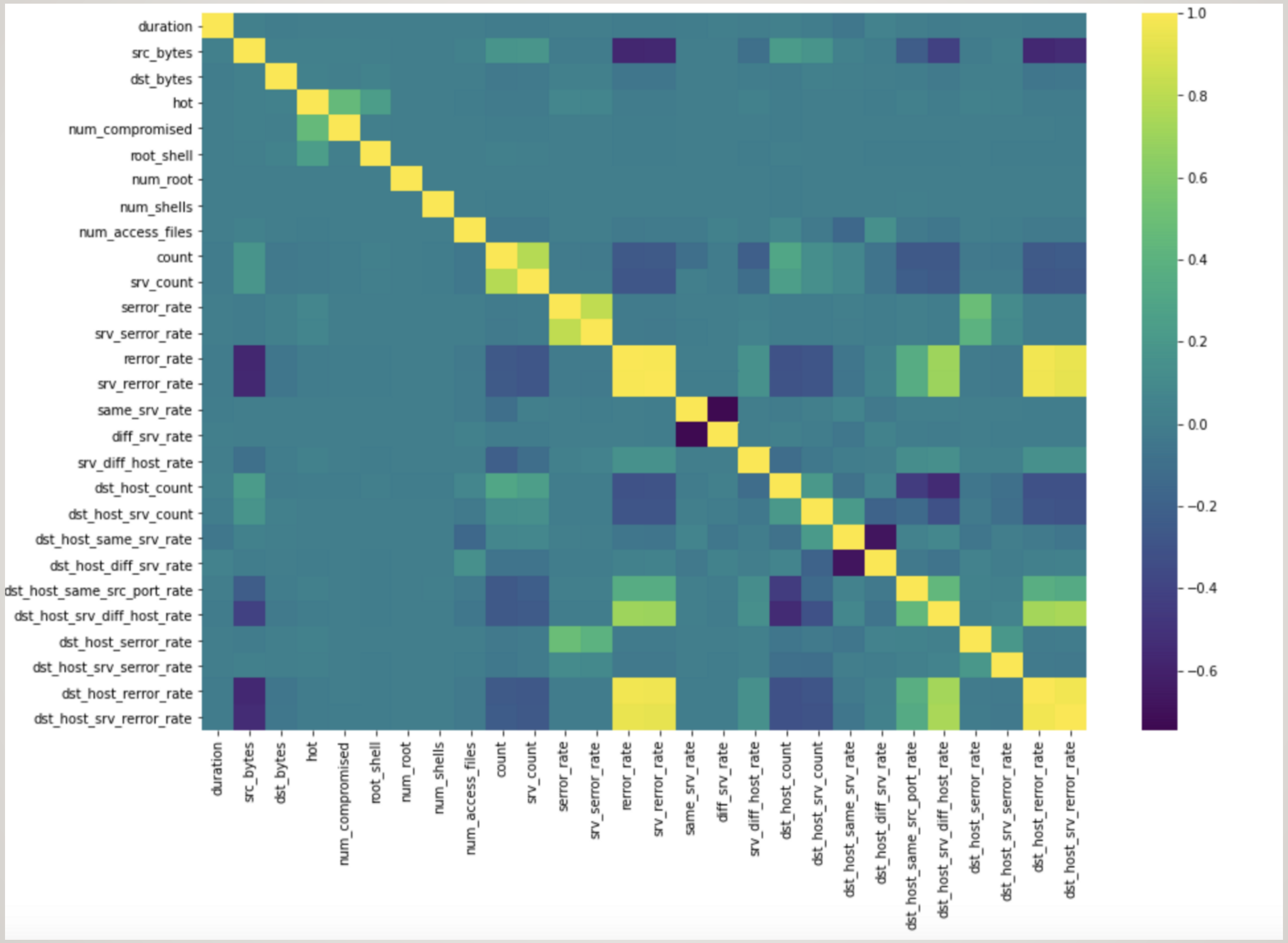
```
different types of services: ['http' 'smtp' 'domain_u' 'auth' 'finger' 'telnet' 'eco_i' 'ftp' 'ntp_u'
 'ecr_i' 'other' 'urp_i' 'private' 'pop_3' 'ftp_data' 'netstat' 'daytime'
 'ssh' 'echo' 'time' 'name' 'whois' 'domain' 'mtp' 'gopher' 'remote_job'
 'rje' 'ctf' 'supdup' 'link' 'systat' 'discard' 'X11' 'shell' 'login'
 'imap4' 'nntp' 'uucp' 'pm_dump' 'IRC' 'Z39_50' 'netbios_dgm' 'ldap'
 'sunrpc' 'courier' 'exec' 'bgp' 'csnet_ns' 'http_443' 'klogin' 'printer'
 'netbios_ssn' 'pop_2' 'nnsp' 'efs' 'hostnames' 'uucp_path' 'sql_net'
 'vmnet' 'iso_tsap' 'netbios_ns' 'kshell' 'urh_i' 'http_2784' 'harvest'
 'aol' 'tftp_u' 'http_8001' 'tim_i' 'red_i']
```

- In the above picture, we can see that the data contains different types of services.
- Examining the data belonging to different services shows that they have a different distribution for the normal condition.
- Hence, we consider only the 'http' service type as it is the most encountered service type in the internet traffic. Doing so also re-establishes our assumption that attacks/anomalies are rare in nature.

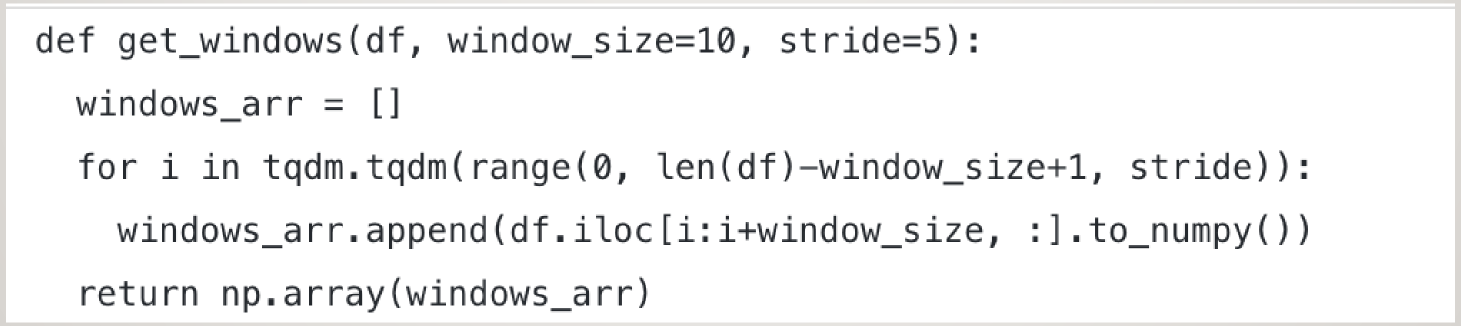
- Histogram showing the number of different attack types in HTTP service



- Correlations present in ‘normal’ data.

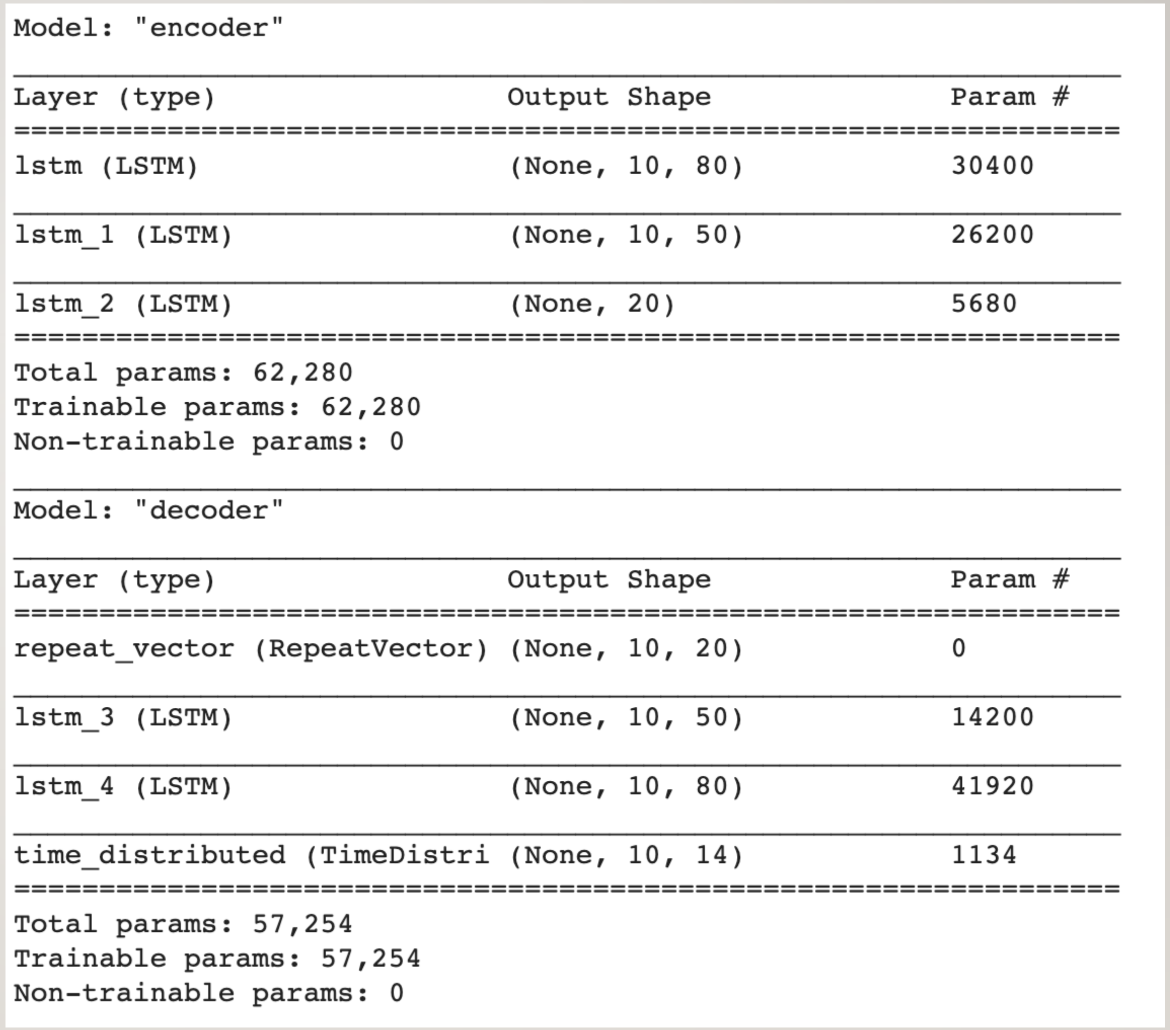


- Although, most features are non-correlated with one another, there are few strong correlations present.
- To remove these correlations and to bring all the features to same scale, we scale the data using Standard Scaler and reduce the dimensionality of the data using PCA (Principal Component Analysis).
- This also ensures that all the correlations are removed as the PCA components are orthogonal to one another.
- Finally, windows are extracted from this data using a specific window length and stride; using the function in the picture. Data flow between consecutive windows is optional and can be ensured via stride between windows (i.e. stride should be less than window length).



```python
def get_windows(df, window_size=10, stride=5):
  windows_arr = []
  for i in tqdm.tqdm(range(0, len(df)-window_size+1, stride)):
    windows_arr.append(df.iloc[i:i+window_size, :].to_numpy())
  return np.array(windows_arr)
```

LSTM AUTO-ENCODER MODEL

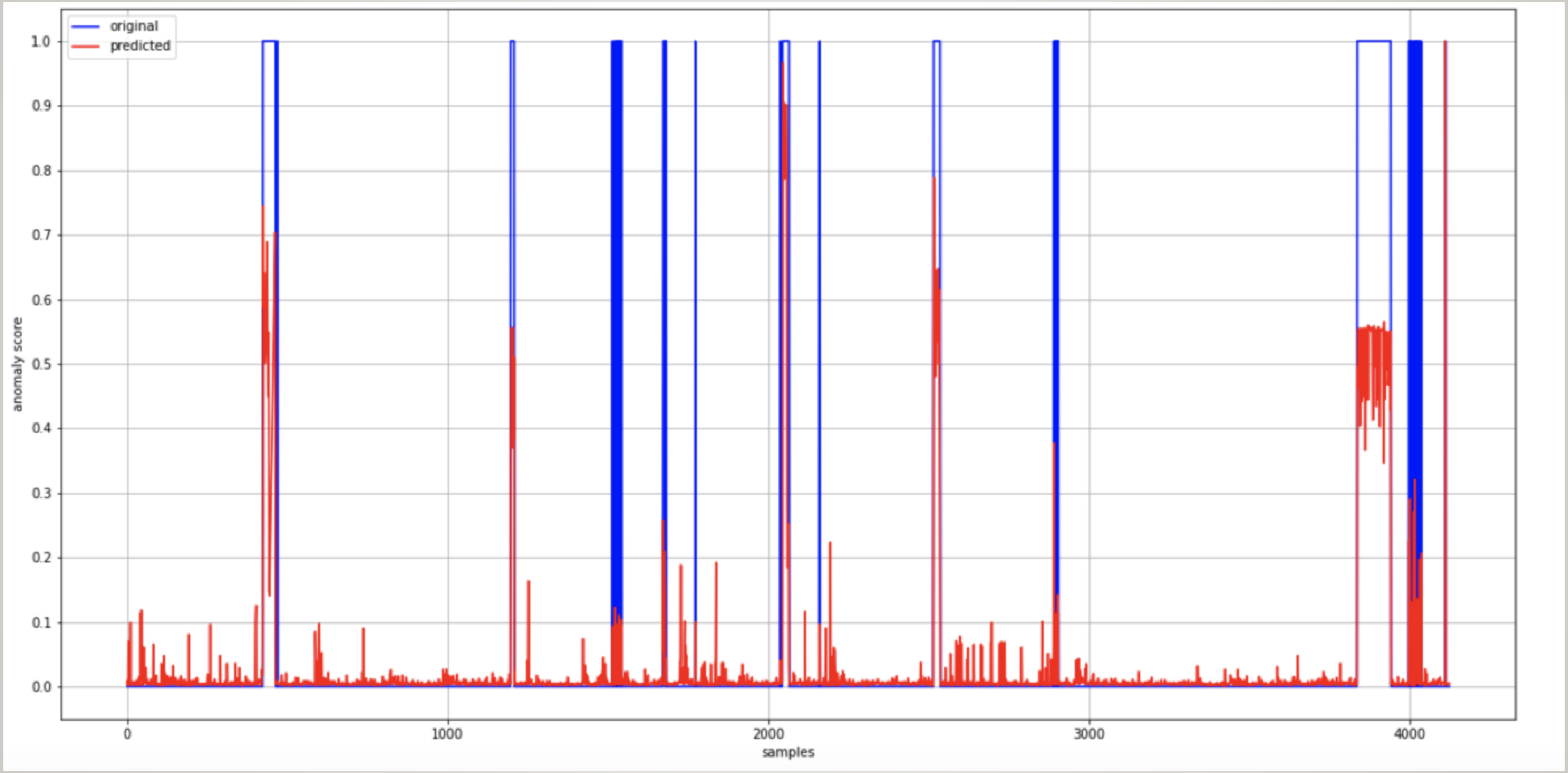


```
Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 10, 80)            30400
_________________________________________________________________
lstm_1 (LSTM)                (None, 10, 50)            26200
_________________________________________________________________
lstm_2 (LSTM)                (None, 20)                5680
=================================================================
Total params: 62,280
Trainable params: 62,280
Non-trainable params: 0
_________________________________________________________________
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
repeat_vector (RepeatVector) (None, 10, 20)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 10, 50)            14200
_________________________________________________________________
lstm_4 (LSTM)                (None, 10, 80)            41920
_________________________________________________________________
time_distributed (TimeDistri (None, 10, 14)            1134
=================================================================
Total params: 57,254
Trainable params: 57,254
Non-trainable params: 0
```

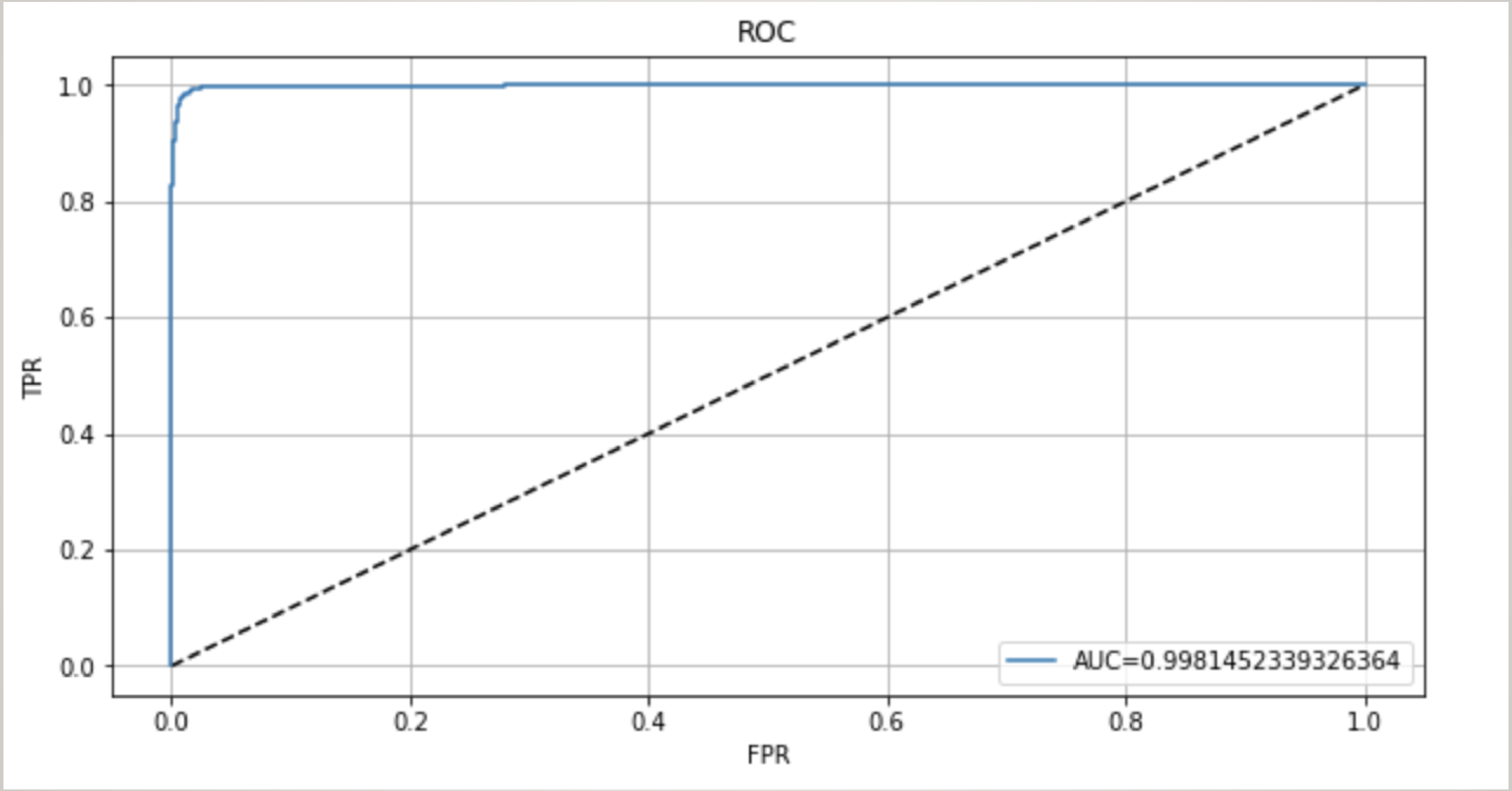
INFERENCE AND THRESHOLD SETTING

- Reconstruction error is used a metric to predict the likely hood of a sample/instance being anomalous.
- The reason behind this is that we used only the normal data to train the auto-encoder. During inference, if the model encounters an anomalous sample, the encoder compresses it to a latent representation that is similar to that of normal data (it has been trained to only compress normal data, hence its weights are tuned to only do that).
- This latent representation misses the information related to anomalous characteristics and the decoder reconstructs this as a normal sample resulting in large reconstruction error.
- After calculating reconstruction errors for all the samples in the test data, we can scale them to [0, 1] range called the anomaly score

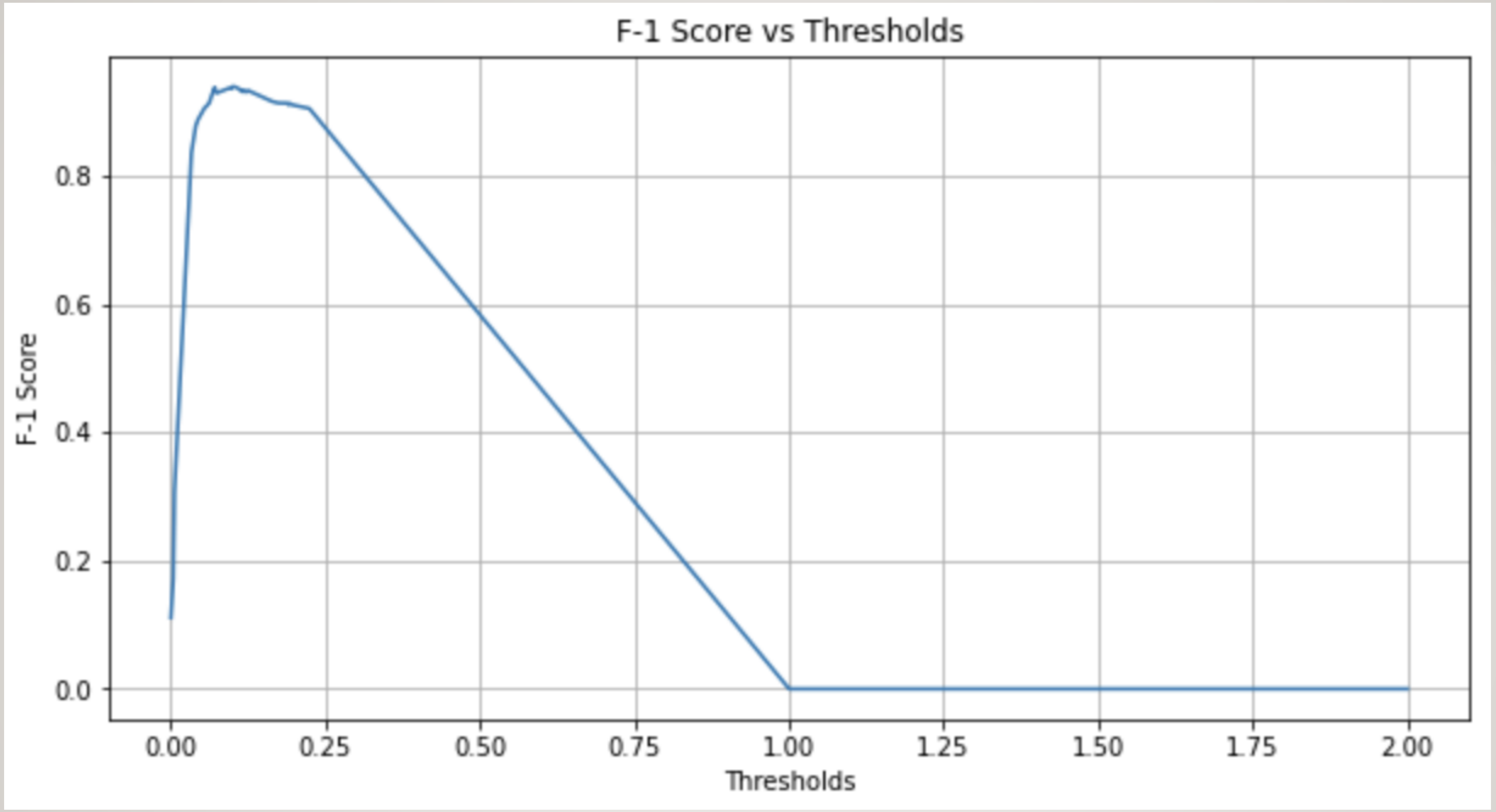
- Predicted and actual anomaly scores for validation samples



- ROC of the process

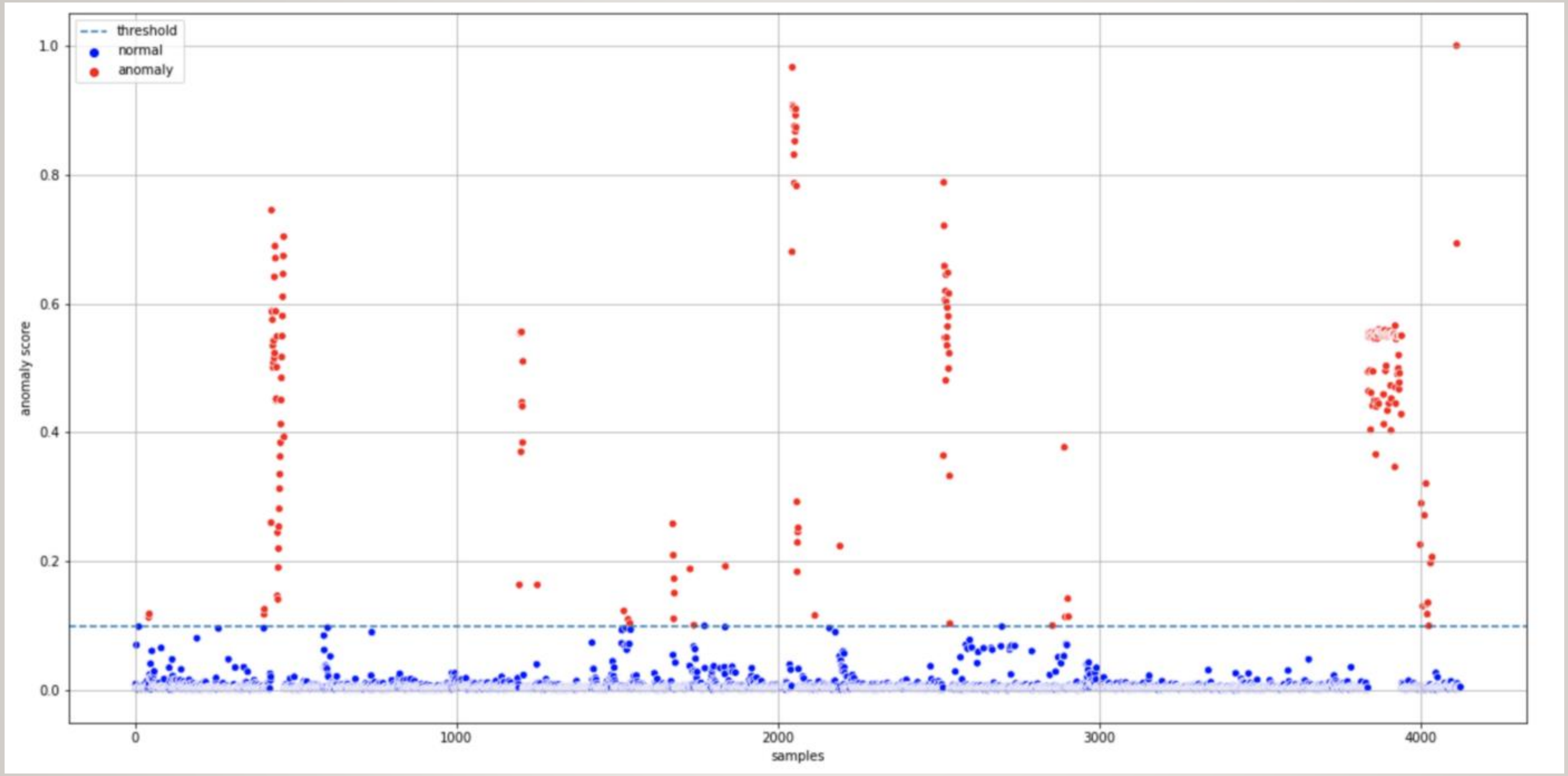


- F-1 scores for different thresholds



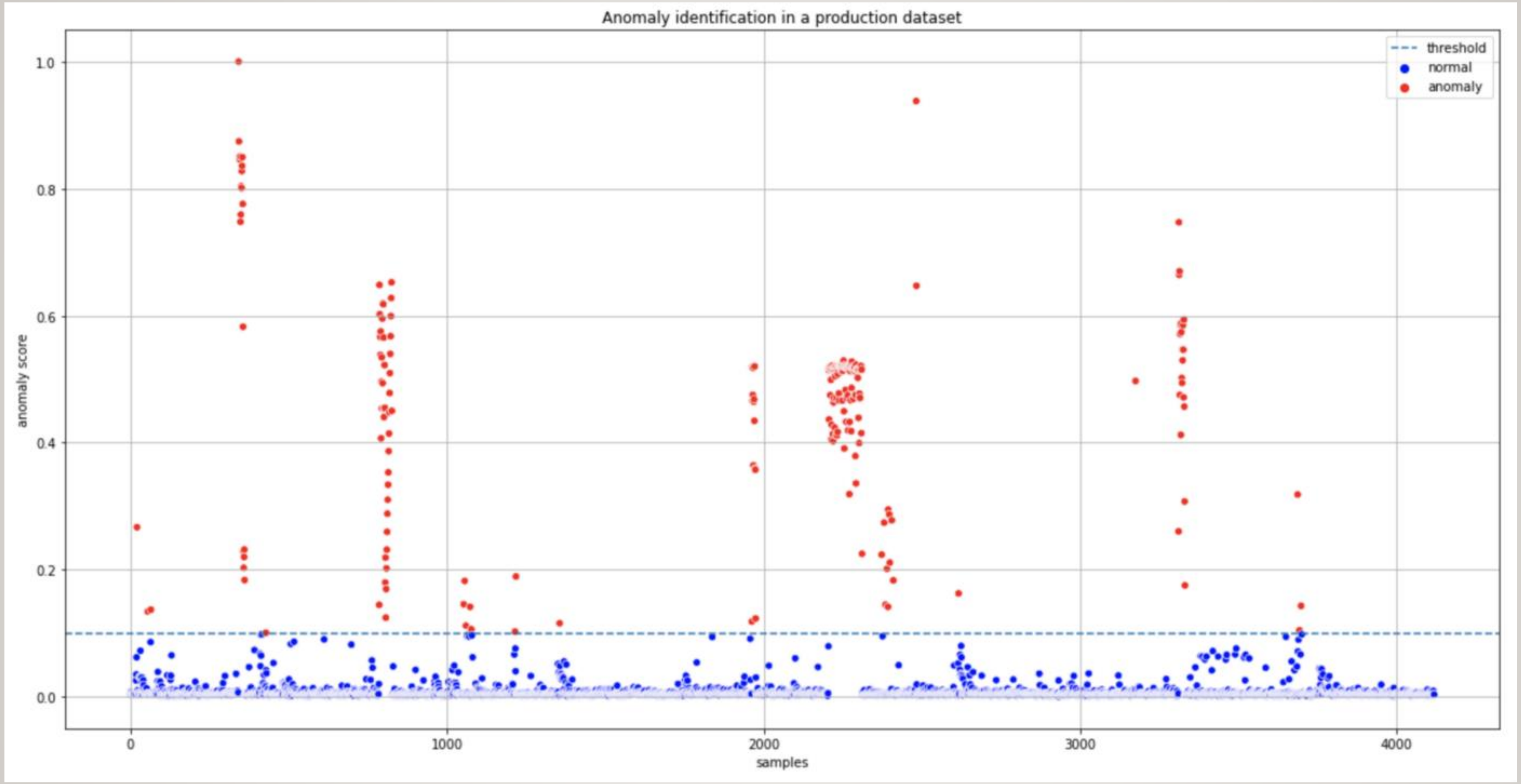
The plot shows that best threshold=**0.0999448** which is the value of threshold that results in best f-1 score.

- Results for validation set



Threshold=**0.0999448**

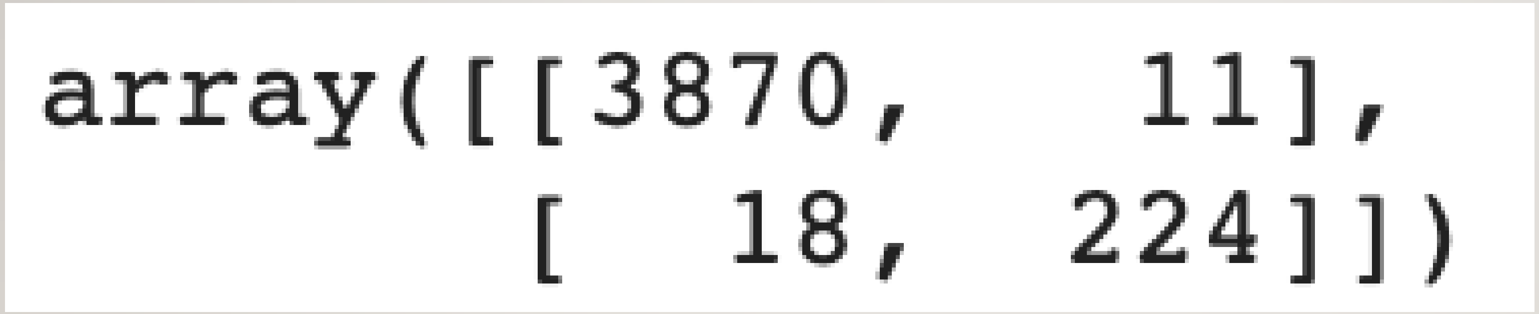
- Results for test set



Threshold=**0.0999448**

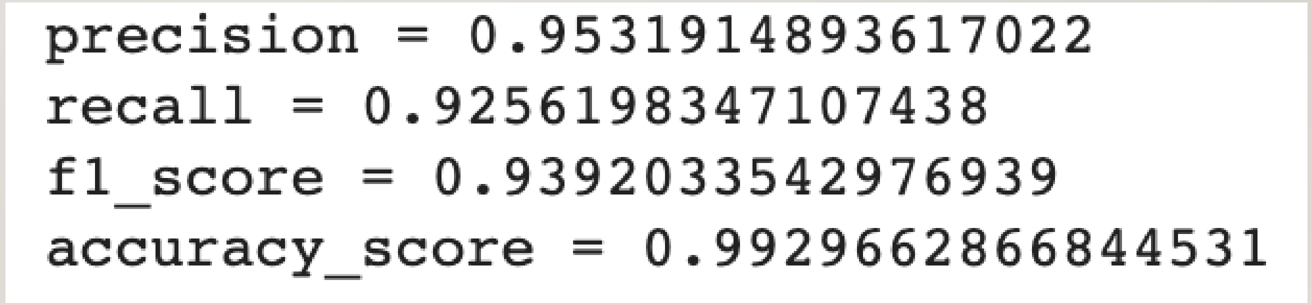
CLASSIFICATION METRICS

- F-1 scores for different thresholds



```
array([[3870,   11],
       [  18,  224]])
```

- Classification metrics for best threshold



```
precision = 0.9531914893617022
recall = 0.9256198347107438
f1_score = 0.9392033542976939
accuracy_score = 0.9929662866844531
```

NOTES

- Looks like the model can identify anomalies which have large reconstruction error in the production set as well.
- However, a domain expert must intervene and diagnose the anomaly and plan remedial actions.
- Hence, this type of applications can only act as an aid to a cybersecurity expert and cannot fully replace their role. We still have a long way to reach that point.

FUTURE WORK

Although this is an effective way of detecting intrusions in a network, there are several superior deep learning techniques:

- GANs can be used in place of auto-encoders to model the data. If trained right, GANs can more accurately capture the data distribution. They can also be used to augment the dataset by generating new examples of normal or anomalous classes.
- Different variants of auto-encoders such as sparse auto-encoder, variational auto-encoders can be used. They have special constraints on the latent representations and lead to extraction of more robust and efficient latent representations.
- The above-mentioned methods are pure ML method with little/no domain knowledge. In any field, combining domain knowledge with ML has always resulted in the best outcomes.

CONCLUSION

- We have seen that with right se of features, availability of large amounts of data and knowledge of deep learning algorithms, we can develop a model that is efficient in detecting anomalies.
- These days, many of the IDS software are relying on these techniques mainly because they require little or no domain knowledge to function satisfactorily.
- Congratulations on making it to the end of an elaborate article. Now go and demonstrate the power of neural networks to Cyber security specialists😀.

REFERENCES

- https://www.mdpi.com/2076-3417/9/20/4396/pdf
- https://blog.keras.io/building-autoencoders-in-keras.html
- http://odds.cs.stonybrook.edu/http-kddcup99-dataset/
- http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html