Cyber Security and AI - HW1

EMBER Semi Supervised

By:

- Gil Zeevi
- Joel Liurner
- Alon Hartanu

Abstract - Quick summary of work

In this paper and the attached notebook we inspect several methods and models in order to compete with the original EMBER dataset and its baseline results. We do that by presenting a lightweight dataset with only 184 features instead of the original 2381 features. We inspected several methods for feature selection and dimensionality reduction and then plugged the data into several models , including NN, just to find out that XGBoost outperformed them all.

Problem formulation

The following starting point was given:

1. A labeled benchmark dataset (which we call EMBER - Endgame Malware BEnchmark for Research) for training machine learning models to statically detect malicious Windows portable executable files. The dataset contains 1.1M samples splitted as follows: (a) For training 300k files malicious, 300k benign, 300k unlabled and (b) for testing 100k malicious and 100k files benign.
2. A baseline model based on the dataset and trained with gradient boosted decision tree model using LightGBM and no hyperparameter tuning which yielded an attractive ROC AUC 0.9964 and 86.8% detection at 0.1% False Positive Rate.

The dataset and initial code and baseline model is published on paper arXiv:1804.04637 and available on this repository.

The availability of the portable executable (PE) binary file samples is great progress to be able to study and create alternative models, however the complexity of each file structure allowed the possibility to extract huge amount of features and information. This makes the model and dataset a benchmark but not handy nor easily malleable.

Our purpose was to explore the dataset and find alternative detection models to compare results, finding its strengths and weaknesses. The problem to be tackled was to find another decent and acceptable model with other algorithm (with also high ROC AUC at 0.1% FPR) but based on a reduced amount of features. It would allow classifying the files or clustering them into malicius and benign. To do so we would require to engineer features and reduce dimensionality to make the dataset more accesible and computable while yielding a trustworthy but less heavy model.

Data description

The EMBER dataset is a collection of parsed files into json that contains the following data:

- A unique identifier of the sample (sha256 hash of the file)
- A label that indicates if it is benign, malicious or unknown
- Date of the first time the file was seen
- 8 groups of features that are derived from the PE files themselves. These comprise the features used for the baseline model and for our analysis and exploration. The first five groups are features extracted from parsing the file while the remaining three are format-agnostic features that do not require to parse the file.

Extracted features groups

1. *General file information*. This includes file size, virtual size, number of imported and exported functions, signature, relocations, resources, etc - mainly information extracted from PE file header.
2. *Header information*. It includes target machine, image characteristics, target subsystem, DLL characteristics, etc.
3. *Imported functions*. It includes a dictionary of the unique libraries of imported functions and individual functions.
4. *Exported functions*. It includes a list of the exported functions.
5. *Section information*. It includes the properties of each section of the file, based on the name, size, entropy, virtual size and other section characteristics.

Format-agnostic features groups

1. *Byte histogram*. The count of each of the 256 posible bytes in the file.
2. *Byte-entropy histogram*. It is the histrogram made out of the joint distribution of entropy H and byte value. The entropy is calculated for a sliding window over the data and pairing it with each byte instance in the window.
3. *String information*. Limited to 5 char strings, this feature group includes information about number of strings, their average length, a histogram of the printable characters within those strings, and the entropy of characters across all printable strings.

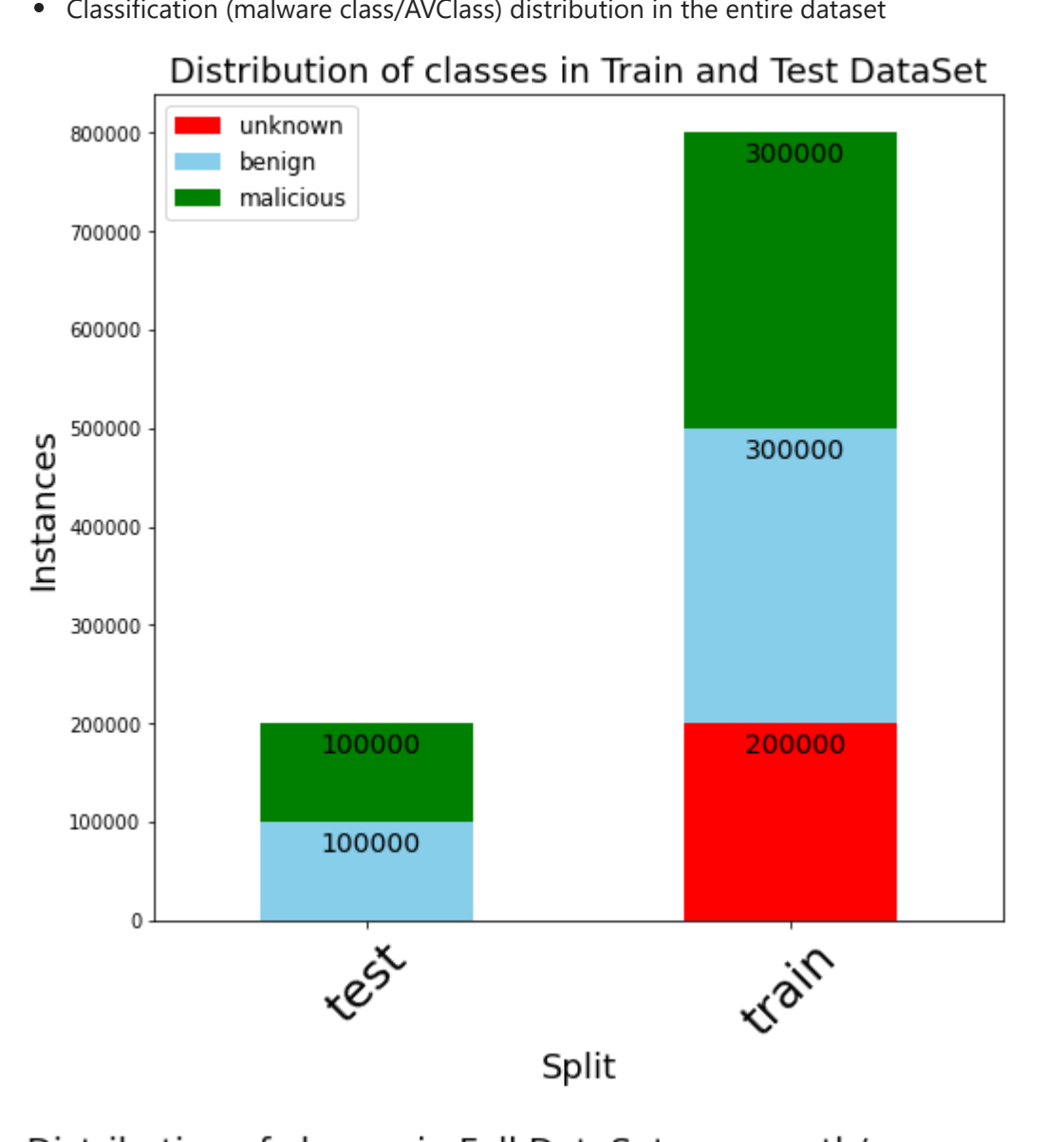
Model features are engineered from the raw features recently described and they are loaded into a feature matrix. Within these features, when strings were found, they were summarized and captured using the hashing trick (with different amount of bins per group and type) as referenced in the paper.

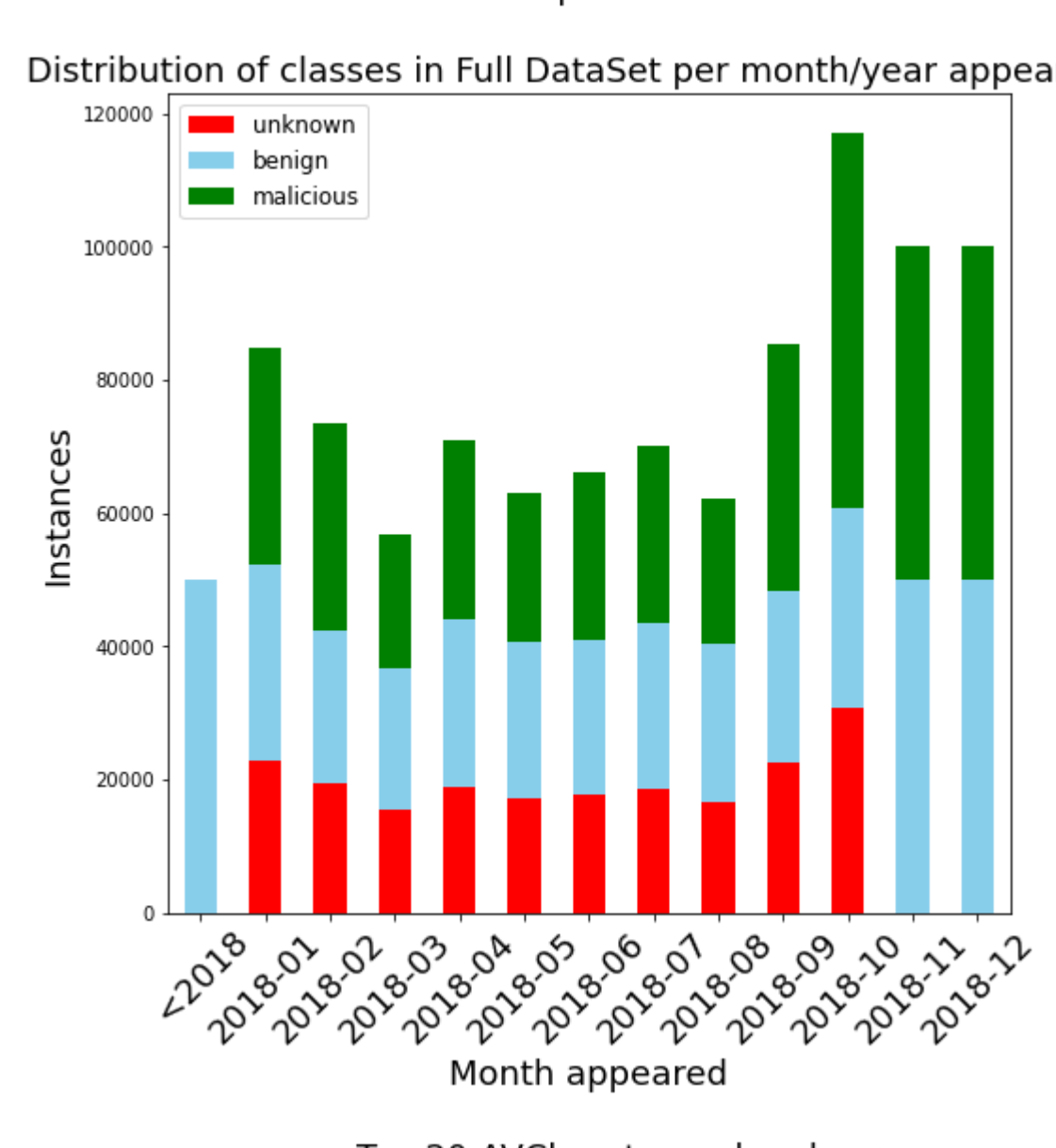
The total number of features is 2381. We used the code provided to extract the raw data and create the same model features to start our base analysis.

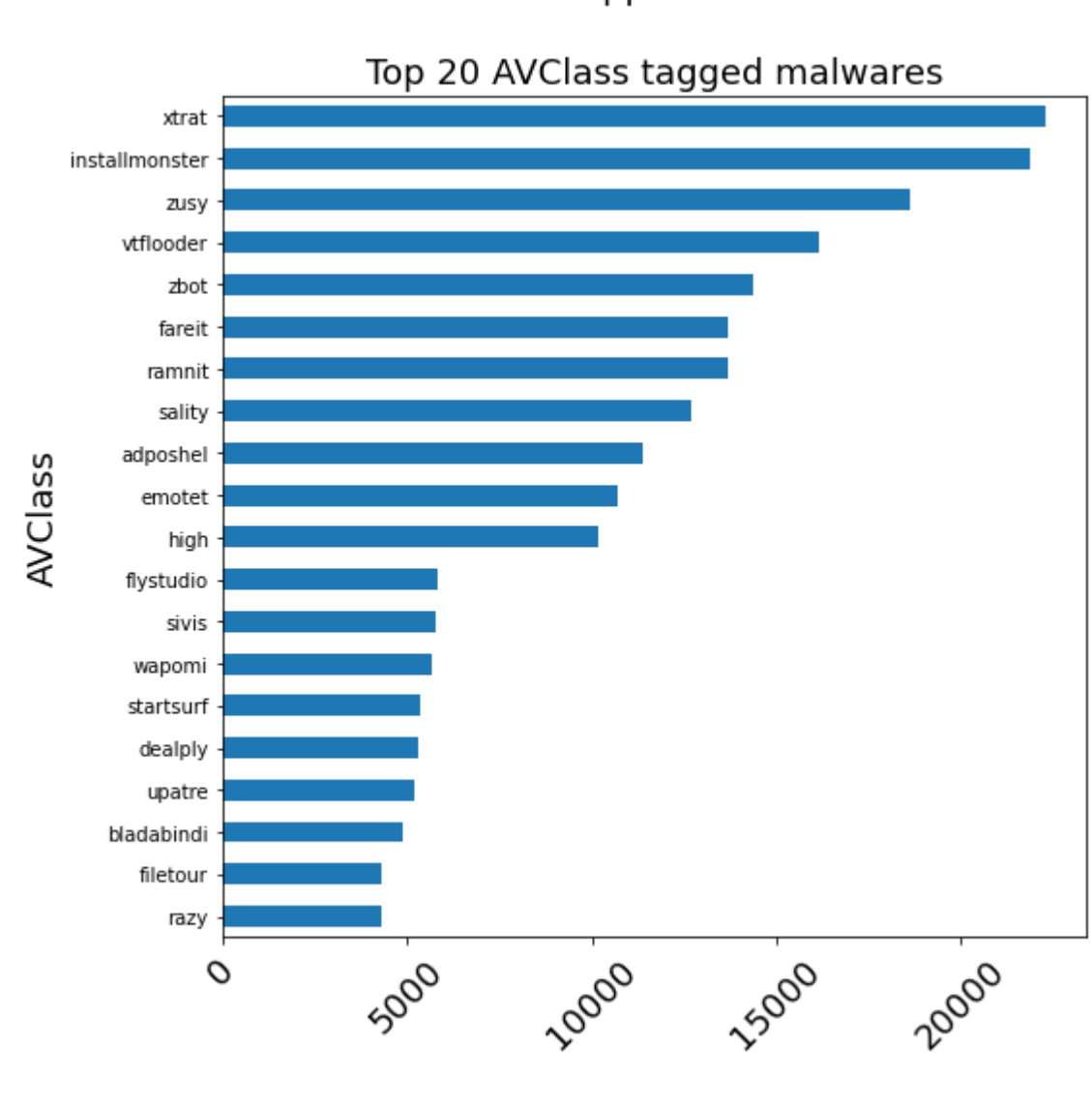
Exploratory data analysis (EDA)

We explore the data in the following ways:

- Classification (malware/benign) distribution in the train and test datasets
- Classification (malware/benign) distribution in the entire dataset along the years
- Classification (malware class/AVClass) distribution in the entire dataset







Baseline Description

For creating a baseline model, the authors used supervised learning methods, hence removing unknown labeled instances. The baseline ember model which is provided is a gradient boosted decision tree (GBDT) trained with LightGBM with default model parameters (100 trees, 31 leaves per tree) on the 2017 dataset and optimized via grid search on the 2018 dataset. all of the 2381 features were used in the baseline model with no feature selection/engineering/dimensionality reduction meaning the vectorized data was used *as is*.

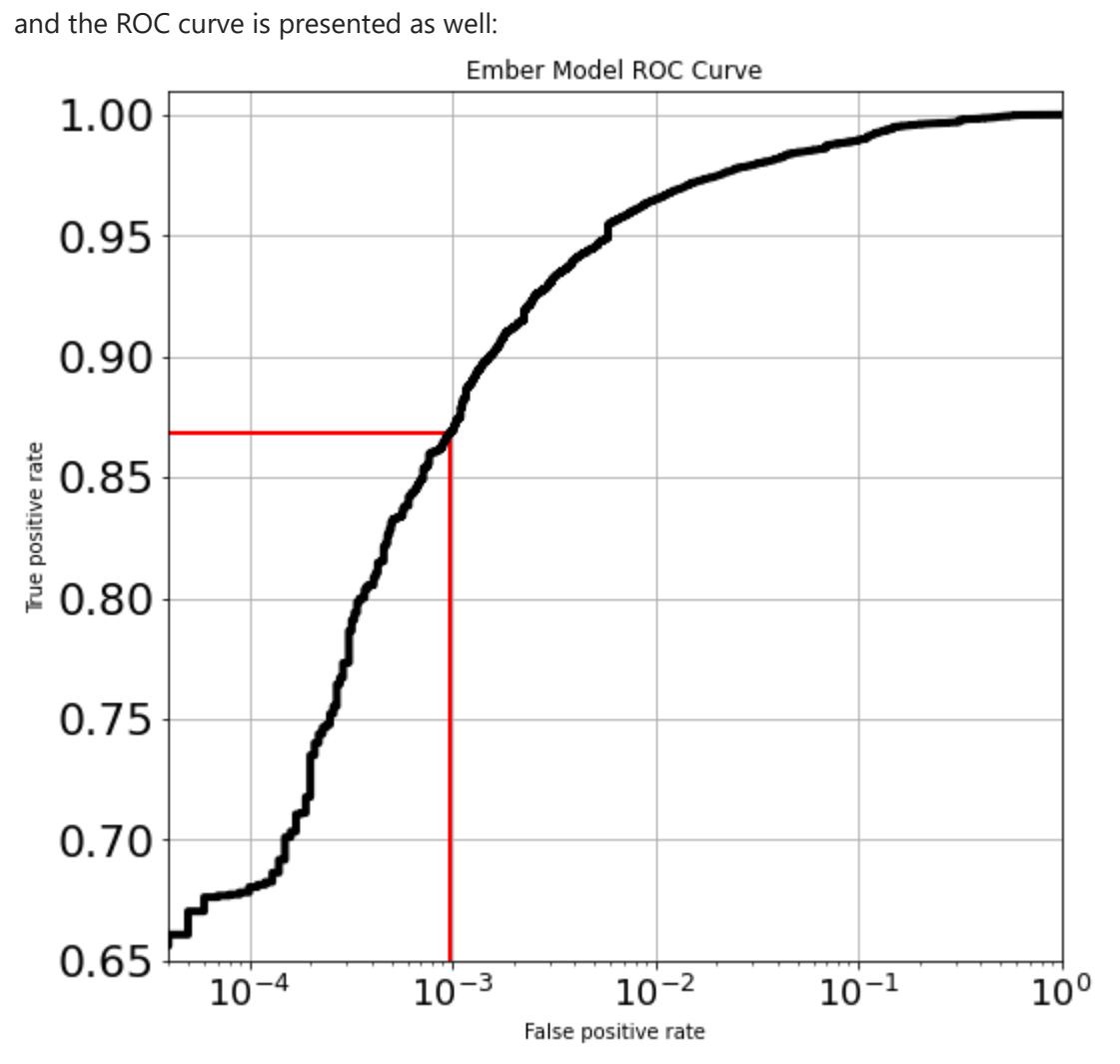
The chosen parameters were the following:

```
best_params = {
  "boosting": "gbdt",
  "objective": "binary",
  "num_iterations": 1000,
  "learning_rate": 0.05,
  "num_leaves": 2048,
  "feature_fraction": 0.5,
  "bagging_fraction": 1.0,
  "max_depth": 15,
  "min_data_in_leaf": 50,
}
```

The performance of this model on the test set were really high and are presented here as following:

```
ROC AUC: 0.9964289467999999

Ember Model Performance at 1% FPR:
Threshold: 0.8336
False Positive Rate: 1.000%
False Negative Rate: 3.502%
Detection Rate: 96.498%

Ember Model Performance at 0.1% FPR:
Threshold: 0.9996
False Positive Rate: 0.098%
False Negative Rate: 13.192%
Detection Rate: 86.80799999999999%
```

and the ROC curve is presented as well:



The area under the ROC curve is a good method for comparing binary classifiers and the ember benchmark model achieves a score of 0.99642 on the test set.

Feature extraction and engineering , Dimensionality reduction

Dimensionality Reduction

Initial Aproximation: Chi-Squared, Mutual information and PCA

In general we would like to state that Feature engineering is one of those hard parts of Data Science that has no universal solution. We will try to tackle EMBER with several methods but there are so many more methods and we cannot anticipate beforehand what is the best one to go.

We care about robustness on one hand but on scalable algorithms on the other hand as we are using local,limited resources, machines, as opposed to those who were used building the EMBER dataset.

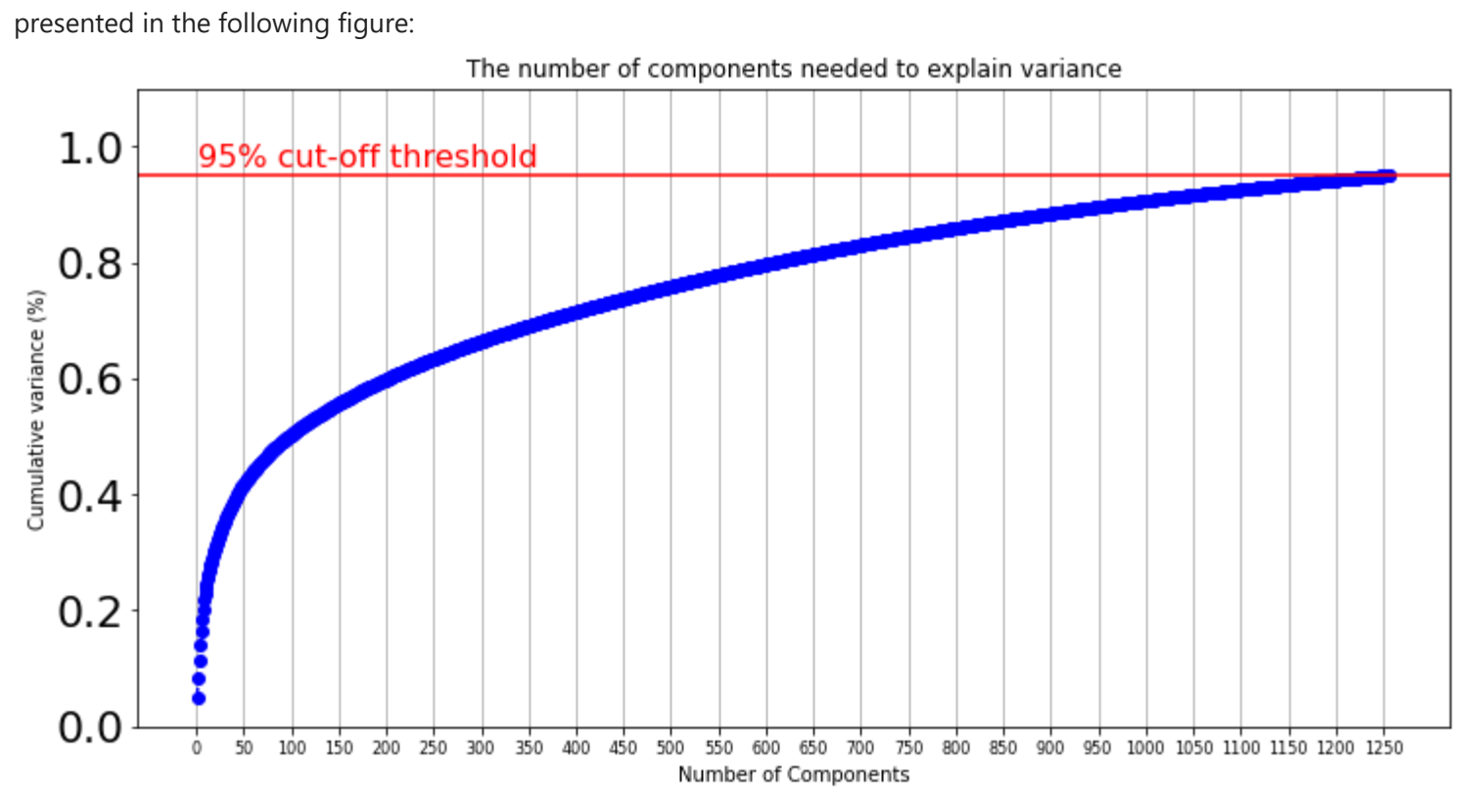
We first wanted to inspect filter methods: We thought on two main possible ways (there are many more of course):

1.
   - Applying MinMaxScaler to features
   - Chi-Square filtering method
2.
   - Estimate mutual information **After feature reduction applied, most important features can be found by using some sort of a decision tree** but first, we need to inspect how variant is our data so we can properly scale it to avoid noisy occurances:
     ```
     Minimum Value in Training data: -4278190080.0
     Maximum Value in Training data: 4294967296.0
     Mean Value in Training data: 574316.9375
     Standard deviation in Training data: 29699554.0
     ```
     This data varies alot, and need to be scaled, the question is how.

Applying first filter method - chi square:
For chi square, MINMAX scaling has to be made because it bounds the transformed features by [0,1]. This is due to Pearson's chi square test (goodness of fit) does not apply to negative values. By playing with this filtering and applying to different models, we found that this methods didnt fit for the EMBER data due to its need of positive values matrix. applying MinMaxScaler causes to lose essential information in the encoded features or the chi square selection isnt right for this raw data.

Applying a different filter method - mutual information selection:
In general mutual informations is better practice than just applying the most common Pearson's correlation because pearsons only spot linear correlations whereas the real world data tends no to be linear at all.
We found out that mutual information selection scale awfully for this size of database. It should act as first stage filter before applying more complex methods and should remove the less explained features, but considering the time it took to run this method, we decided to give up on it and go straightly to PCA.

Principal Components Analysis (PCA):
Firstly, we scale the data with standard scaler which is probably more adequate.It arranges the data in a standard normal distribution which in general is allowed due to central limit theorem where there are roughly more than 30 observations.
Then, we Applied PCA which captured 95% of the features variance. On one hand we seek to reduce the number of features (which are denoted as the dimension) as much as possible, but on the other hand we want to maintain high variance of our feature in order not to make our model biased. By picking to maintain a variance of 95%, the number of features need to explain the variance are 1257 and also presented in the following figure:

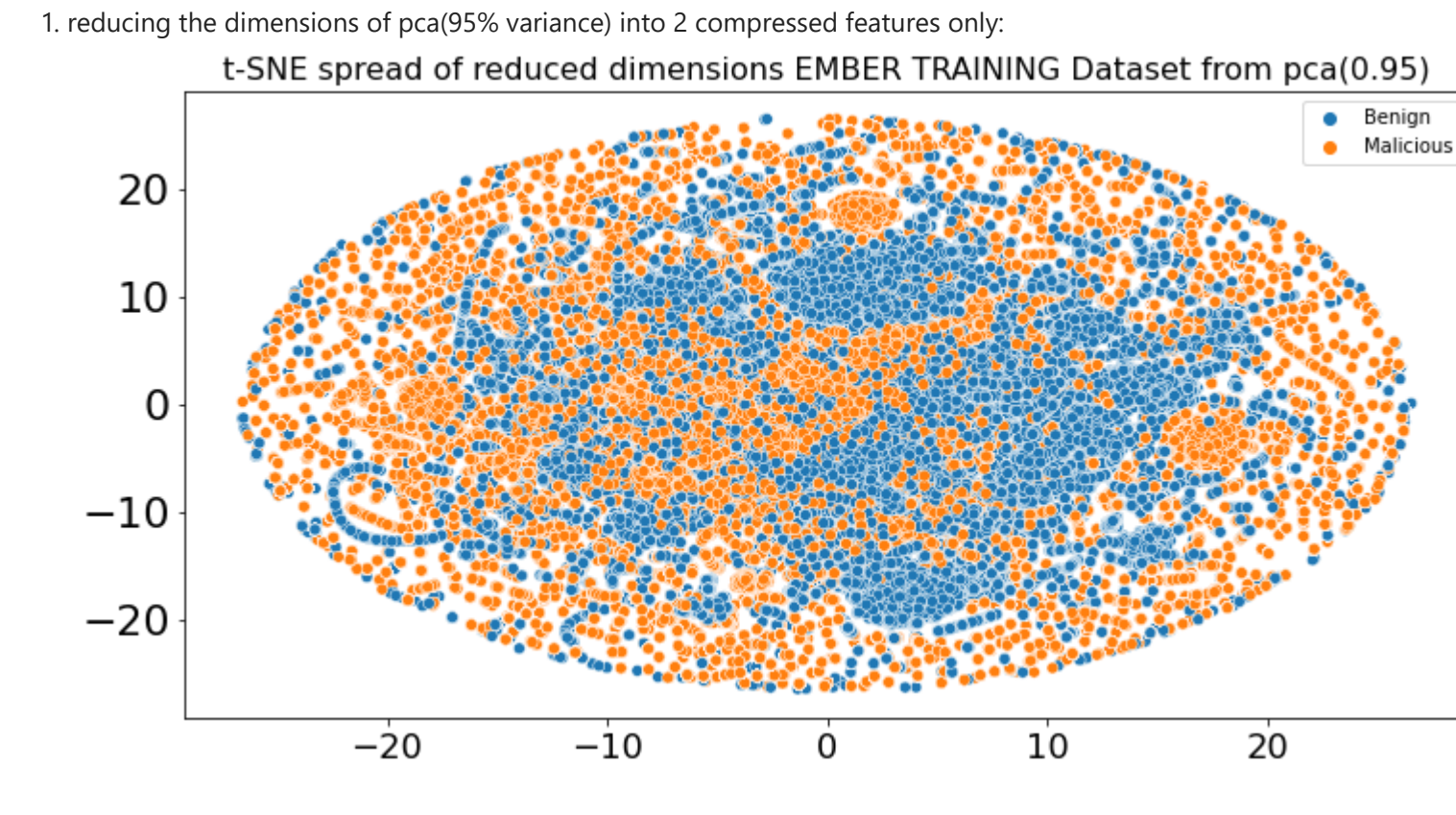


So we've managed to maintain 95% of the features variance and decresead the dimensions almost in half to 1257

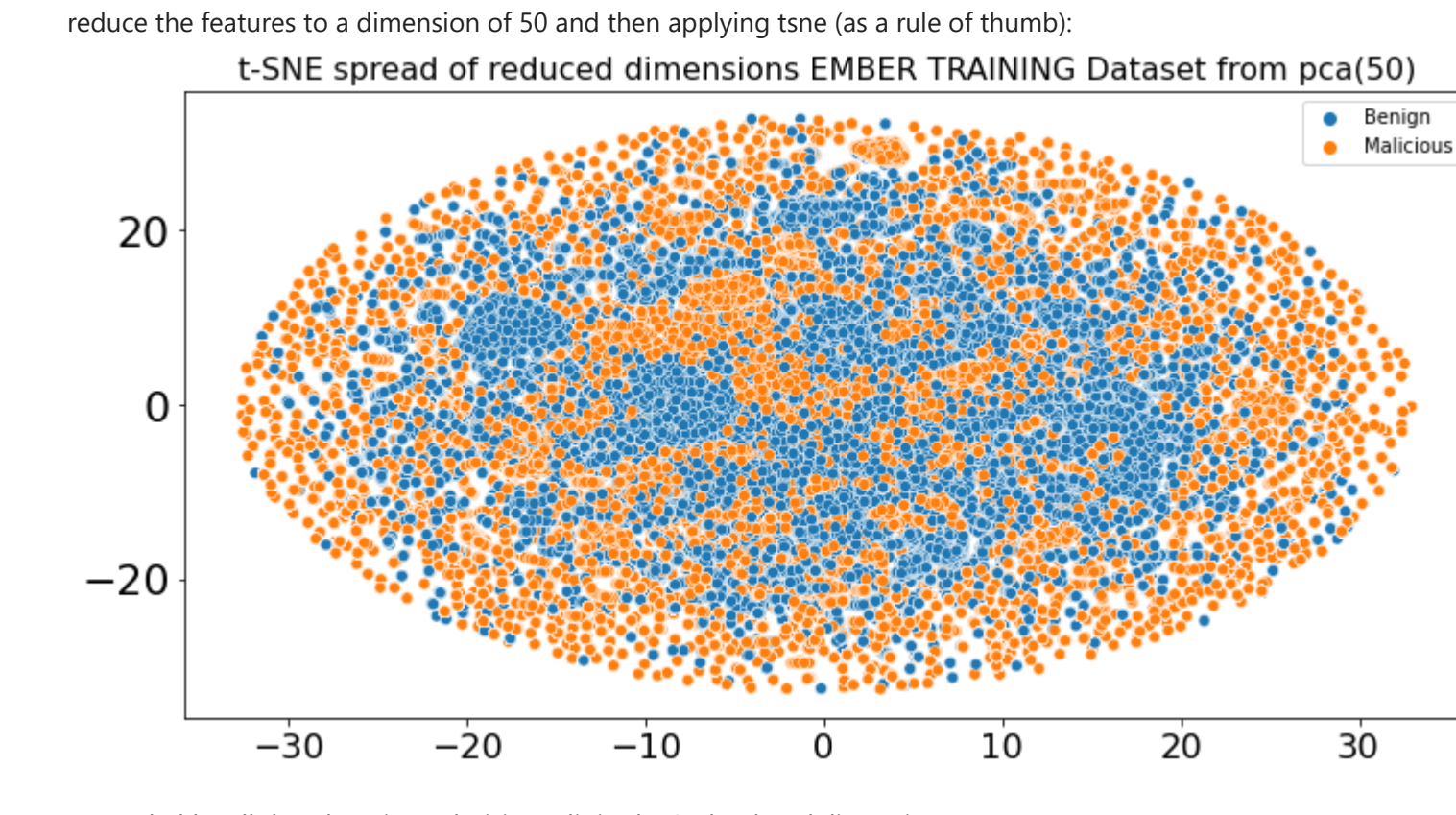
Complement study with TSNE

Applying TSNE - t-Distributed Stochastic Neighbor Embedding:
Now , lets try and reduce the dimensions even more and reduce them into two dimensions and try and inspect if there is a clear seperation between the malware and the benign samples. The biggest advantage of TSNE is that we can visualize the scatter of the samples due to a 2D dimension. We will try two approaches:

1. reducing the dimensions of pca(95% variance) into 2 compressed features only:



1. Reducing the full dataset into a pca of dimension of 50. If T-sne is desired for a high dimensions database, it is first recommended to reduce the features to a dimension of 50 and then applying tsne (as a rule of thumb):



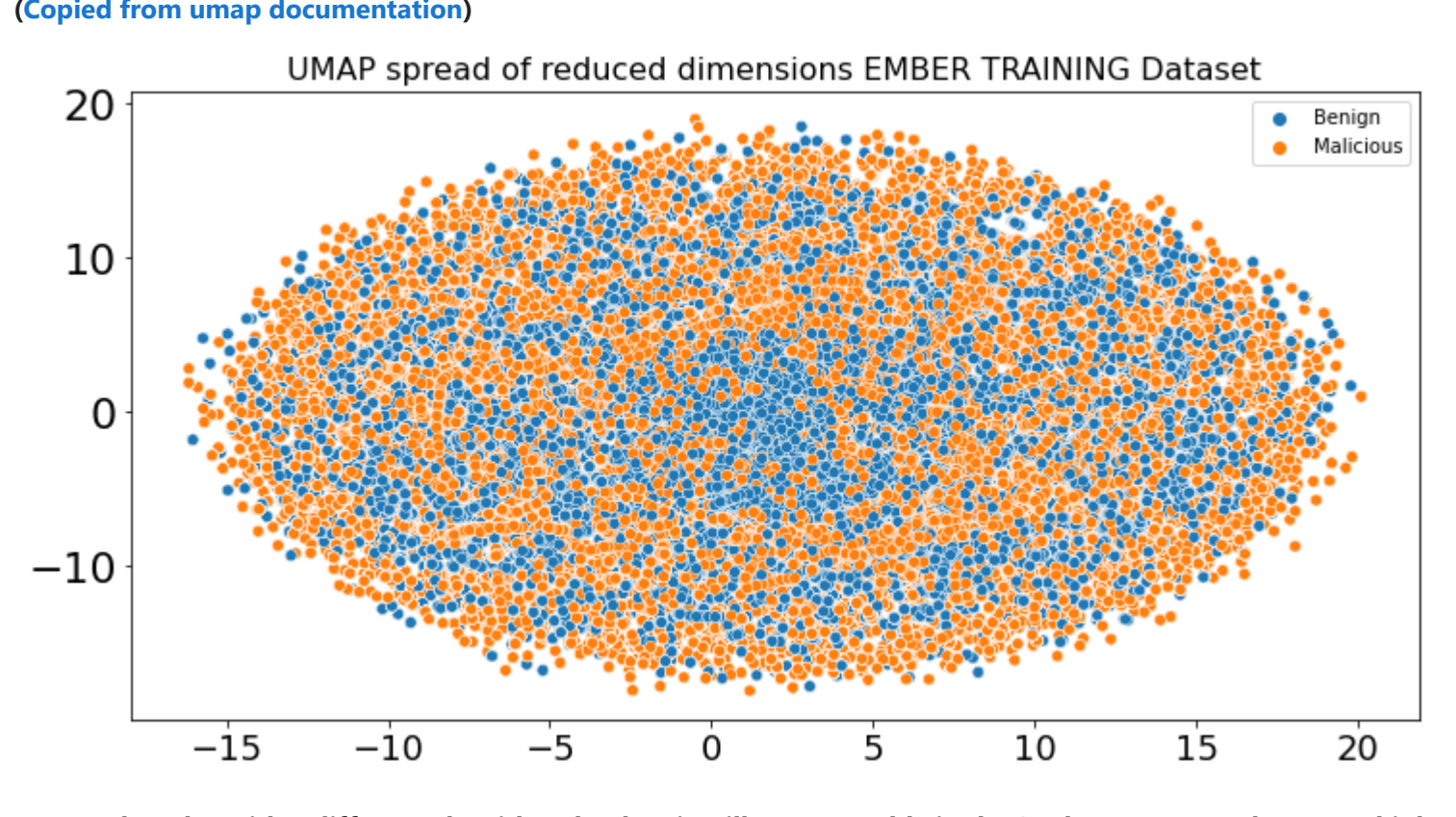
We can probably tell that there is no decisive split in the 2nd reduced dimension.

Attempt with UMAP

Applying UMAP - Uniform Manifold Approximation and Projection for Dimension Reduction:
Lets try a different approach, using UMAP.
Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique that can be used for visualisation similarly to t-SNE, but also for general non-linear dimension reduction. The algorithm is founded on three assumptions about the data:

- The data is uniformly distributed on Riemannian manifold;
- The Riemannian metric is locally constant (or can be approximated as such);
- The manifold is locally connected.

From these assumptions it is possible to model the manifold with a fuzzy topological structure. The embedding is found by searching for a low dimensional projection of the data that has the closest possible equivalent fuzzy topological structure.
(**Copied from umap documentation**)



We see that also with a different algorithm the data is still not separable in the 2D. hence, we need to go to higher dimension!

Decided method: Feature selection using XGBOOST

A benefit of using ensembles of decision tree methods like gradient boosting is that they can automatically provide estimates of feature importance from a trained predictive model. Feature importance scores can be used for feature selection and it is done by using a specific python package that takes a model and can transform a dataset into a subset with selected features.
This package(resides in scikit-learn of course...) can take a pre-trained model, such as one trained on the entire training dataset. It can then use a threshold to decide which features to select. This threshold is used to consistently select the same features on the training dataset and the test dataset.
By doing that, we extract the most important features of EMBER and create a lightweight dataset with a minimal number of features on one hand, but still robust on the other hand as well.

We will use the reduced data from pca into a variance of 95% which holds 1257 features and apply XGBOOST model on those and use his feature importance properties to select the best features.
We chose XGBoost for his robustness and the fact that we can utilize GPU with XGBoost thus we're able to find the most important features pretty quickly.

Finally after applying this method for feature selection, we've managed to reduce our dimension into 184 dimensions out of 2381! with the following description:

```
Number of New features: 184
Minimum Value in New Training data: -2414.445556640625
Maximum Value in New Training data: 8227.6240234375
Mean Value in New Training data: 2.466495516273426e-06
Standard deviation in New Training data: 2.548919439315796
```

Clustering the dimentionality reduced data

We can look on the training dataset clustering as a supervised task where we can acquire the clustering using y vector. We will use supervised algorithms hence we will solve classification problem.

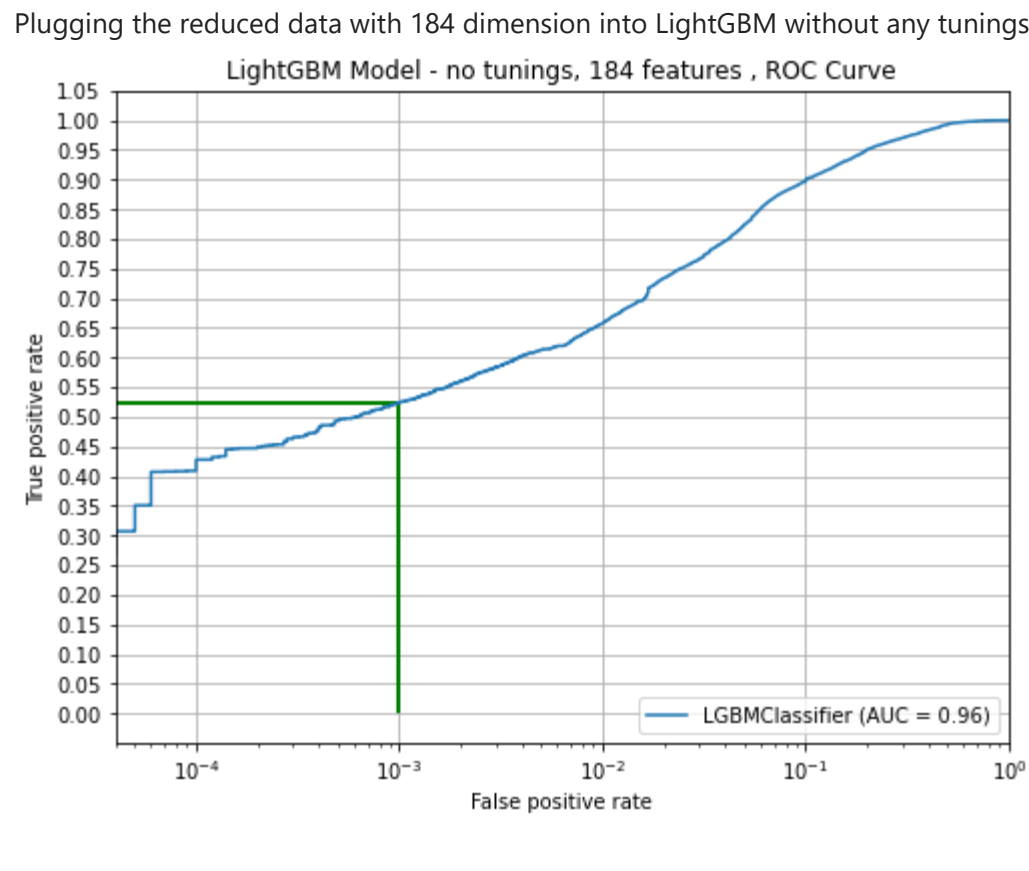
We will inspect two different(Not that different actually) Gradient boosting models. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models. It's called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. The following models were inspected:

> **LightGBM** - used by EMBER paper. a gradient boosting framework that uses tree based learning algorithm that is designed to be distributed and efficient with many advantages such as Faster training speed, higher efficiency and Lower memory usage.
>
> **XGBoost** - stands for eXtreme Gradient Boosting and aims to provide a regularizing gradient boosting framework. it aims to provide a "Scalable, Portable and Distributed Gradient Boosting.

Both have gained much popularity as the algorithms of choice for many winning teams of machine learning competitions. Another reason to use those models ,except their robustness of course, is the ability to train them on GPU which will fasten the training and will make hyperparameters tuning alot faster and more capable.

LightGBM

Plugging the reduced data with 184 dimension into LightGBM without any tunings:

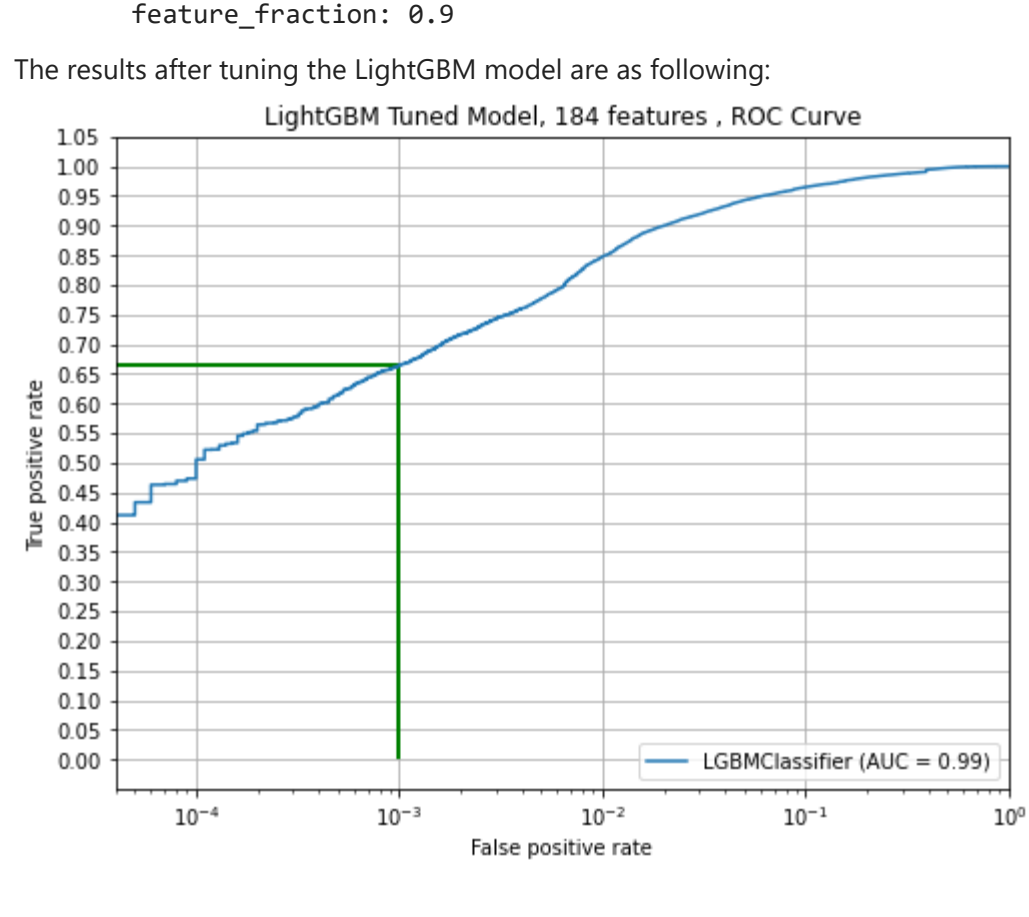



```
ROC_AUC Score: 96.460%

Ember Model Performance at 1% FPR:
Detection Rate: 65.769%

Ember Model Performance at 0.1% FPR:
Detection Rate: 52.410%
```

Performing hyperparameters tuning on the GPU:

```
Best parameters chosen are:
    device_type: gpu
    n_estimators: 10000
    learning_rate: 0.2936141158201311
    num_leaves: 180
    max_depth: 8
    min_data_in_leaf: 2100
    lambda_l1: 15
    lambda_l2: 85
    min_gain_to_split: 0.4261257659917306
    bagging_fraction: 0.8
    bagging_freq: 1
    feature_fraction: 0.9
```

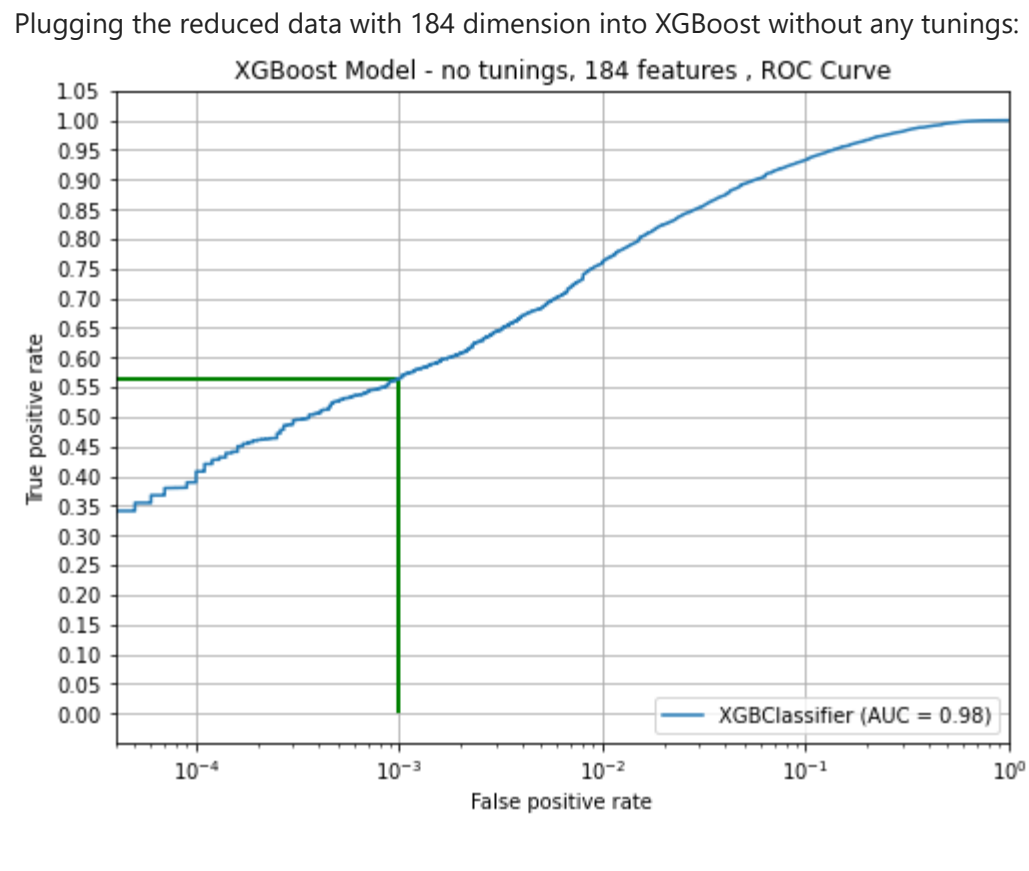
The results after tuning the LightGBM model are as following:




```
ROC_AUC Score: 98.615%

Ember Model Performance at 1% FPR:
Detection Rate: 84.776%

Ember Model Performance at 0.1% FPR:
Detection Rate: 66.490%
```

XGBoost

Plugging the reduced data with 184 dimension into XGBoost without any tunings:

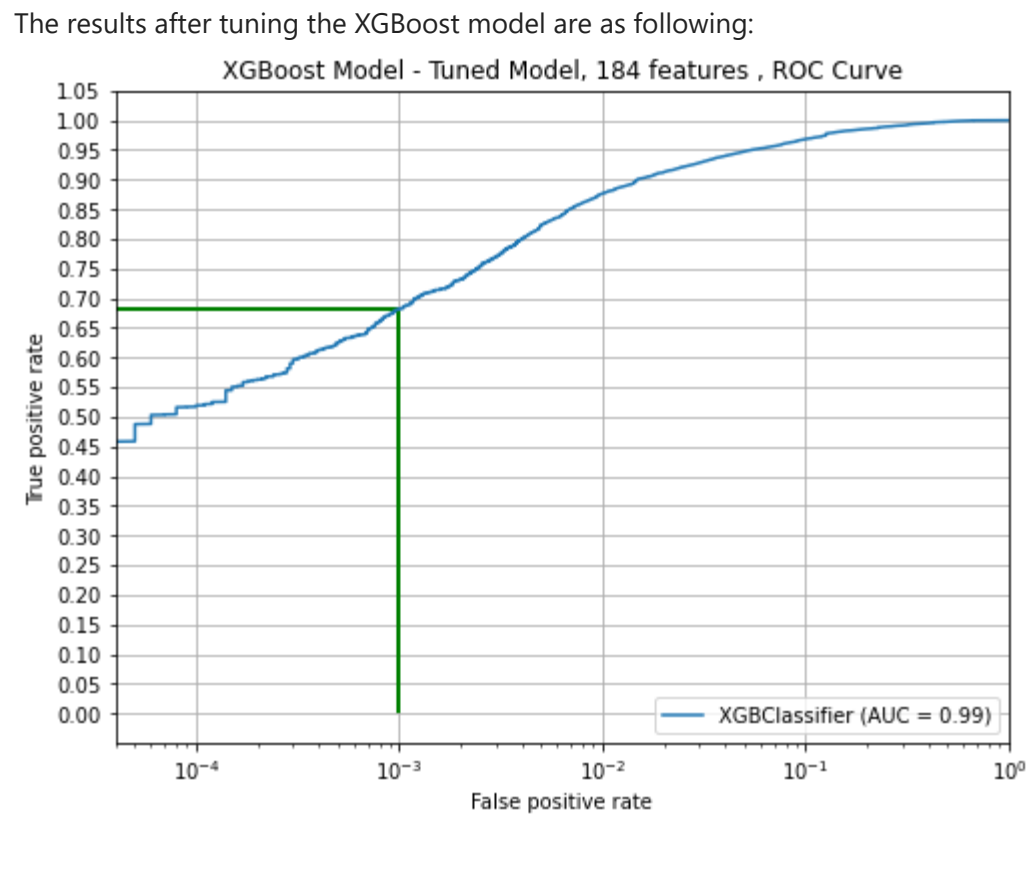



```
ROC_AUC Score: 97.652%

Ember Model Performance at 1% FPR:
Detection Rate: 76.129%

Ember Model Performance at 0.1% FPR:
Detection Rate: 56.494%
```

Performing hyperparameters tuning on the GPU:

```
Best parameters chosen are:
    'colsample_bylevel': 0.7999999999999999,
    'colsample_bytree': 0.5,
    'gamma': 0.2,
    'learning_rate': 0.1,
    'max_depth': 8,
    'min_child_weight': 5,
    'n_estimators': 10000,
    'reg_alpha': 0.0,
    'reg_lambda': 0.0,
    'subsample': 0.8999999999999999
```

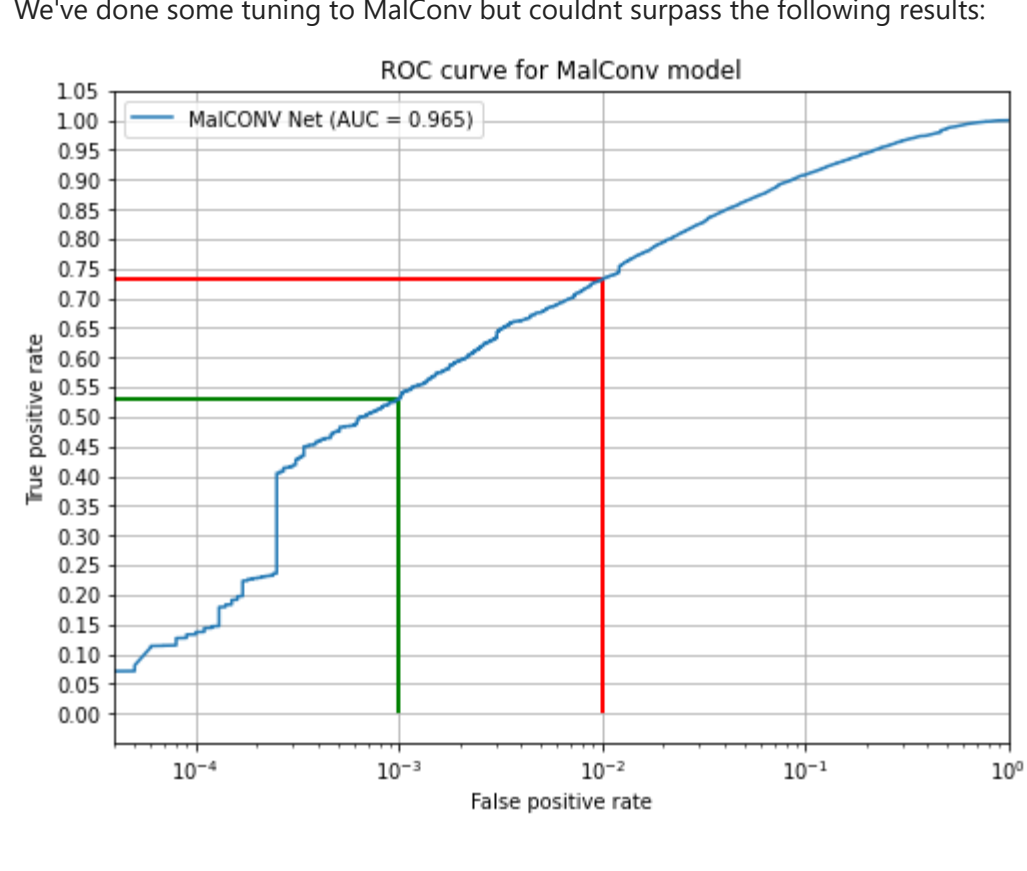
The results after tuning the XGBoost model are as following:




```
ROC_AUC Score: 98.852%

Ember Model Performance at 1% FPR:
Detection Rate: 87.655%

Ember Model Performance at 0.1% FPR:
Detection Rate: 68.194%
```

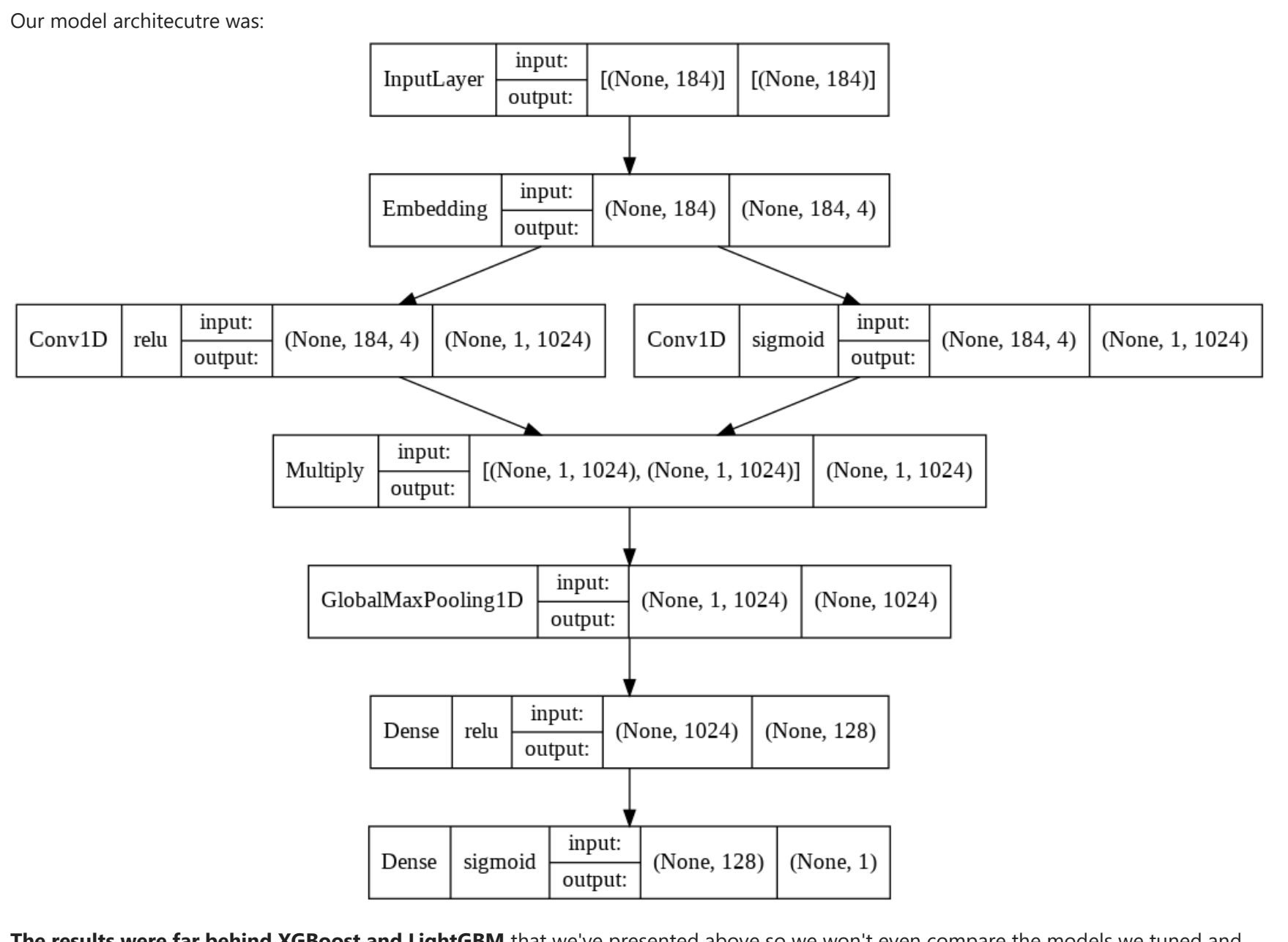
Applying Neural network on EMBER 2018 reduced dimensions dataset

We also applied on the reduced data a NN called MalConv which was also used in comparison to the baseline LightGBM model in the original paper.
MalConv looks for both benign and malicious byte patterns in order to make a decision.
MalConv mechanism works in a manner that Files are passed to MalConv as a sequence of integers representing the bytes of the file (0–255). An embedding layer inside MalConv maps each byte to a vector of numbers. The sequence of vectors can then be processed by additional neural network layers. The model outputs two numbers representing the probabilities of an input being benign and malicious.
We've done some tuning to MalConv but couldnt surpass the following results:




```
ROC_AUC Score: 96.5%

Ember Model Performance at 1% FPR:
Detection Rate: 73.17%

Ember Model Performance at 0.1% FPR:
Detection Rate: 53%
```

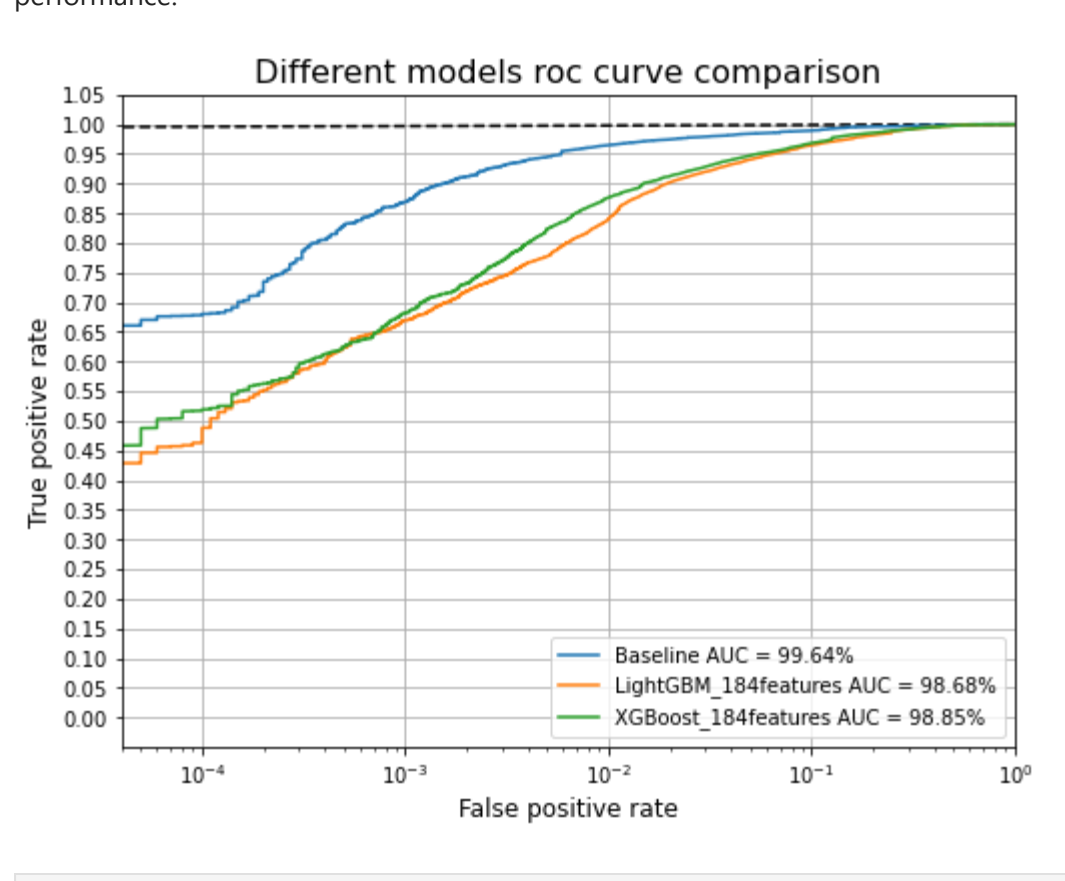
Our model architecutre was:



The results were far behind XGBoost and LightGBM that we've presented above so we won't even compare the models we tuned and the baseline with malconv.
Nevertheless there probably is more room to improvement in this area of NN classifications.

Summarizing the results

The following figure provides a summary of the different models and how they compare. We did not include the NN due to its poor performance.




```
In [ ]:
import pandas as pd
dat=pd.DataFrame(data= [['AUC','99.64%','98.68%','98.85%'],['0.1% FPR Detection','86.8%','66.945%','68.194%'],[
dat.set_index('metric')
```

Out[ ]:

metric	Baseline	LightGBM	XGBoost
AUC	99.64%	98.68%	98.85%
0.1% FPR Detection	86.8%	66.945%	68.194%
1% FPR Detection	96.49%	84.03%	87.65%

Limitations and Discussion

- The EMBER dataset is very complete and the amount of possible features to be extracted from the files is huge. It is clear that many of those features are not that relevant for this classification task. We also agree that trying to classify between benign and malicious is not a simple task and has no clear boundary.
- Even though we werent able to surpass the original model which was trained on 2381 features we were able to present a lightweight dataset (compared to original) with only 184 features which is able to represent the real non reduced data in a manner that one should argue the tradeoff of things.
- Our lightweight data is scalable in any local machine compared to the original one which has to be run on high resources machine due to memory leakage of both RAM and GPU. The accuracy tradeoff is pretty decent where all things considered.
- With these analysis and results, we can stand on the new reduced dataset and acceptable baseline. We believe that there are three different paths to be considered and worth exploring given a next iteration on EMBER and/or more time on the project:
  - 1) Explore deeper on the Malconv Neural Network that, given the state of art, should be able to produce better results than the ones we achieved.
  - 2) Try to perform clustering on malware class. This can be done by filtering the benign samples or by consider those harmless as a malware class itself. We are currently bundling all malware into same bag as malicious but it might be worth exploring further into its differences. It mght be the case that the differences between *xtrat* and *zbot* malware is as big as differences between *xtrat* and benign samples.
  - 3) Given the results, run different models and work on the unlabeled samples, to be able to cluster it and predict its classification.