Class Struggle

Attempts to classify domain names using classifiers trained on raw datasets have been discussed as much as using regular expressions to parse HTML, so let's add a few more words.

Where is my data?

I want to find domains similar to those on the CERT Polska warning list [1]. Obtaining a representative set of legitimate domains, e.g. from The Majestic Million [2], is unrealistic. As a result, if I wanted to use any classifier, my training dataset:

- is radically unbalanced - phishing domains (the ones I'm interested in) are extremely rare compared to legitimate ones,
- since the legitimate class would be artificially constructed, my classifier might label as phishing domains it has never seen before in training.

One might think that a classifier with an overrepresentation of the expected domain type will correctly label them and recognize the rest as another class. Unfortunately, classifiers that learn to separate feature spaces without counterexamples may push the decision boundary arbitrarily far to minimize loss. This is like a diagnostic test labeling everyone as sick - it achieves 100% sensitivity, finding all the truly sick.

You can observe this even in a simple example with a FF MLP network reproducing the XOR problem [3].

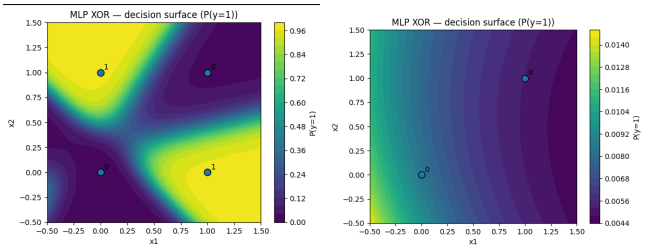

Figure 1: Complete XOR (left) and the decision boundary with an incomplete training set.

Autoencoders to the rescue

An autoencoder [4] is a neural network that learns to reconstruct its training input. The encoder compresses data into a hidden representation z, and the decoder tries to reconstruct the original input from it. The model is trained without labels by minimizing the reconstruction error [5].

Thus, it doesn't need counterexamples; it doesn't learn decisions but representations.

During inference [6], we pass new data through the network and measure the reconstruction error - a large value means that the sample is different from the training data. The inner layer z serves as an embedding - a compact representation of the data structure.

Yet EDA - on limitations

Phishing domains are often created to resemble legitimate ones. Training on the entire corpus would result in an autoencoder that reconstructs legitimate domains.

I must restrain my ambitions. To do this, I perform EDA [7] through clustering [8].

From the dataset, I select the cluster(s) of interest - in my case, domains with certain characteristics [9].

Preprocessing FTW

For the autoencoder to understand the data, it must be vectorized. Counting lengths, dashes, etc., loses semantic information, so I used TF-IDF statistics [10] on n-grams [11].

The TF-IDF vectorizer doesn't preserve positional information or relationships between n-grams. We can enforce minimal positional awareness by positional n-grams through adding start and end markers [12].

Metrics and GOTO EDA

How to choose hyperparameters? I'm not a scientist, so I do it empirically. For network layer dimensions, remember that an autoencoder should compress the representation - a bottleneck is necessary [13]. Typically, if your validation loss increases while training loss decreases, you've trained too long or your model is too large [14].

A nice feature of the autoencoder is that we can apply PCA [15] to its embeddings and try to infer what the autoencoder has learned.

You can find the repository with runnable code here: https://github.com/artur-augustyniak/class_struggle/.

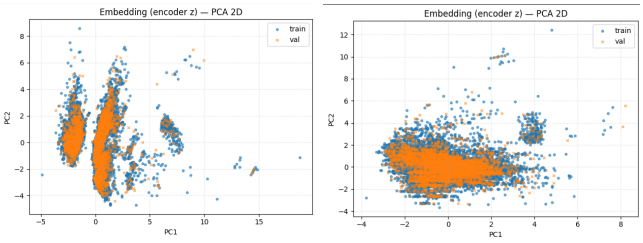

Figure 2: A small AE forced to learn classes (left) and a larger one capable of generalization. In each case, we are interested in the overlap between validation and training embeddings.

1 https://cert.pl/en/warning-list/
2 https://majestic.com/reports/majestic-million
3 https://github.com/artur-augustyniak/class_struggle/blob/main/notebooks/xor_nn_decission_boundary.ipynb
4 https://github.com/artur-augustyniak/class_struggle/blob/2287420fa8b5882b3fc603b6660bb50810b764f1/notebooks/3_autoencoder_train_eval_inference.ipynb?short_path=827cbee#L608
5 https://github.com/artur-augustyniak/class_struggle/blob/2287420fa8b5882b3fc603b6660bb50810b764f1/notebooks/3_autoencoder_train_eval_inference.ipynb?short_path=827cbee#L741
6 https://github.com/artur-augustyniak/class_struggle/blob/2287420fa8b5882b3fc603b6660bb50810b764f1/notebooks/3_autoencoder_train_eval_inference.ipynb?short_path=827cbee#L916
7 https://github.com/artur-augustyniak/class_struggle/blob/2287420fa8b5882b3fc603b6660bb50810b764f1/notebooks/1_eda.ipynb
8 https://scikit-learn.org/stable/modules/generated/sklearn.cluster.HDBSCAN.html
9 https://github.com/artur-augustyniak/class_struggle/blob/2287420fa8b5882b3fc603b6660bb50810b764f1/notebooks/2_eda_based_data_selection.ipynb
10 https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
11 https://en.wikipedia.org/wiki/N-gram
12 https://github.com/artur-augustyniak/class_struggle/blob/2287420fa8b5882b3fc603b6660bb50810b764f1/notebooks/3_autoencoder_train_eval_inference.ipynb?short_path=827cbee#L498
13 https://github.com/artur-augustyniak/class_struggle/blob/2287420fa8b5882b3fc603b6660bb50810b764f1/notebooks/3_autoencoder_train_eval_inference.ipynb?short_path=827cbee#L56
14 https://github.com/artur-augustyniak/class_struggle/blob/2287420fa8b5882b3fc603b6660bb50810b764f1/notebooks/3_autoencoder_train_eval_inference.ipynb?short_path=827cbee#L707
15 https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html