Neural FCA

Yuriy Sosnin

December 15, 2022

1 Dataset & Binarination Strategy

The data I chose for this homework is a somewhat classic Credit Scoring binary classification dataset from Germany. It has 1000 objects and 9 attributes: one already binary ("Sex"), three numeric ("Duration", "Age", "Credit Amount") and 5 categorical ("Job", "Housing", "Saving accounts", "Checking account", "Purpose").

Target variable "Risk" represents risk of default and is encoded 1 for safe clients, and 0 for risky. The classification task is unbalanced: 70% of observations is 1. Considering this, I chose F1 Score as main classification metric. It is a common metric in such settings, where simple accuracy fails because of non-equal distribution of targets.

The other major desision was the choise of binarization strategy. Initally I thought of three approaches: manual, statistical and neural. Manual scaling would require me to appropriately divide numeric attributes into categories according to some "domain knowledge", e.g. age into "young", "middle", "elderly", etc. I decided to not proceed with this approach, to test Neural FCA algorithm in a more general setting, where such strategies would not be possible, for instance with a more numerous or non-interpretable attributes.

Secondly, I thought of utilising Neural Networks for this task. As main algorithm is a Neural Network itself, another module mapping continuous numeric attributes to binary features could in theory be made trainable with backpropagation as well. However, main Network requires a lattice, constructed from already prepared features, which makes training binarization module at the same time non-trivial. Perhaps useful concepts (and their connections) could be recalculated on-the-go as well, but it is too complex and defeats the purpose of designing interpretable NN algorithm.

In the end I decided to settle for statistical approach, simply dividing numerical attributes into bins with either uniform or quantile statistics, based on training sample.

2 Model Construction

The idea of this Neural FCA algorithm is to incorporate FCA lattice into the structure of a Neural Network, making it sparse, by providing an inductive bias for which connections are useful and which are not before training. In a sense, all major NN architectures are designed to reduce the number of trainable parameters in some meaningful way and make the network aware of some structure of the data. As I understand this would be an attempt to do it in general classification task.

The idea provided in base pipeline is as follows. First, Formal Context is created based on training part of the data. Second, a Concept Lattice is constructed from it. Then, a small number of concepts is selected based on a criterion (say, F1 score). They from a partially ordered set, which connections are transferred into a sparse linear layers of an MLP.

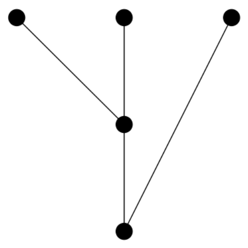

Figure 1: Example POSet

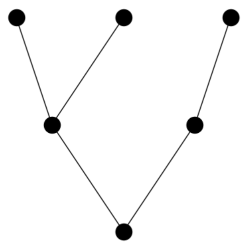

Figure 2: Example Network

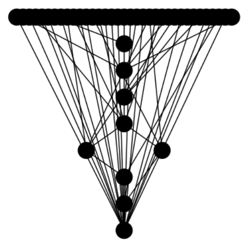

Figure 3: Actual POSet

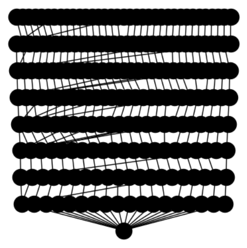

Figure 4: Actual Network

However, computing Concept Lattice from full Formal Concept with several hundred objects takes a considerably long time. A better solution is to not compute full lattice, but only predefined number of 'best' concepts according to some measure (e.g. stability). This way not only is computation much quicker, but also the next step is redundant, as the small number of concepts is already chosen (according to stability). Unfortunately, implementation of CbO in FCApy does not allow to this kind of parameterization, but algorithm called Sofia does, so I chose it.

After obtaining a partially ordered set of 'best' concepts, the next step is to construct a neural network from it. First, I add to this set all singleton concepts whose intent is every each single attribute, they will serve as an 'entrance' of the neural network, capturing attributes of given example; also, I add the final concept whose intent is all attributes, to represent the final layer of the network outputting predictions (because this is binary classification setting, one neuron is sufficient). Next, this poset needs to be converted into a sequence of sparse linear layers. To utilize a 'simple' straightforward PyTorch constructor, it is required to have connections layer by layer, i.e. each pair of layers being a bipartite graph, without skipping that is allowed in poset lattice. To achieve this, I fill the layers that are skipped by some of the concepts with their copies, connected only in one-to-one way.

This process is better described by example. Suppose we have a poset as in Figure 1. Notice that the rightmost connection skips second layer and is connected straight to the last one. To represent this poset as a sequence of sparse linear layers in neural network, we need to coerce it to Figure 2, adding one more node on second layer as an intermediate connection. Visualisations (rather ugly) of my actual model's poset and network can be seen in Figures 3 and 4. Finally, another graph of fitted neural network with colored edges can be seen in Figure 5.

3 Experiments

Neural FCA model was tested with different nonlinearities, number of concepts, and binarization strategies. Table 1 depicts F1 Scores with ReLU, Tanh and Sigmoid as activation functions;

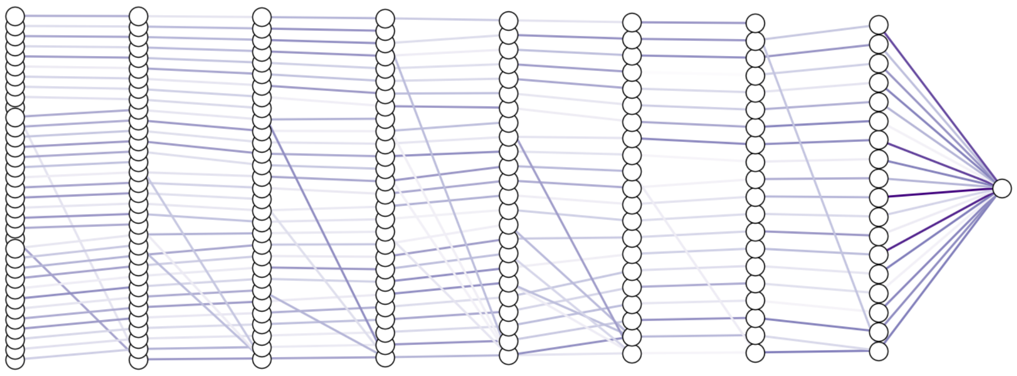

Figure 5: Weigths of fitted network, absolute values

ReLU and Sigmoid show the same result, while Tanh ends up with lower metric score. All the rest experiments are done with ReLU as nonlinearity.

Table 1: F1-Score of Neural FCA, on validation data

	F1 Score
ReLU	0.8201
Tanh	0.8097
Sigmoid	0.8201

As for binarization strategies, quantile and uniform modes were tested, but did not show any difference between them. The same, surprisingly, goes for number of best concepts provided by Sofia algorithm. 10, 20, 30, 40, and 50 concepts all yielded the same result, while taking progressively more time to train. For comparison with other models, the smallest network with 10 concepts was chosen.

Table 2: F1-Score on test data

	Binary	Numeric
KNN	**0.8520**	0.8455
Random Forest	0.8333	0.8339
Logistic Regression	0.8143	0.8212
MLP	0.8200	0.7579
Neural FCA	0.8439	-

In Table 2 there are scores for different ML models typically applied to binary classification on table data, compared to Neural FCA. As can be seen, Neural FCA does quite well, outperforming Random Forest, Logistic Regression and fully-connected MLP. However, simplest KNN with 50 neighbours outperforms them all on both binarized dataset and original dataset with numeric features.