Dimensionality Reduction for Classification of Experimental Events

Shiven Tripathi
Department of Electrical Engineering
Indian Institute of Technology Kanpur

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Dimensionality Reduction for Classification of Experimental Events

I. Problem Definition

The problem is a black-box classification task on dataset containing 2 classes with each class having 28 features. Total samples in the dataset are 6 Million, out of which 500,000 were used for testing and validation each with the remainng serving as training set.

II. EDA and Preprocessing

The dataset was found to not contain any null or empty values. On plotting histograms some variables were found to be categorical. It is recommended that future work focus on building new features over categorical features to improve GBM methods. All features are processed through a standard scaling followed by min max scaling. This is an established step for processing data for DAE networks. For XGB models, another processing step of log transform had been tried out for certain features. This had been done after considering exponential like distribution of the original feature.

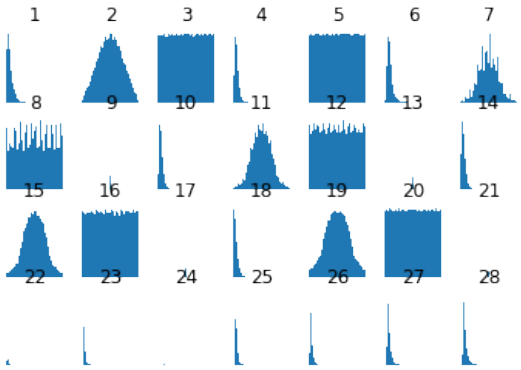

Fig. 1. Processed variables. Some may not be fully visible due to size of graph. Here 9,13,17,21 take only few unique values across samples.

III. Proposed Solutions

For the task of classification, Extreme Gradient Boosting and Deep Neural Networks were tried. After fine-tuning to get good numbers while using the entire feature space, dimensionality reduction was attempted. Deep Auto-encoder has been used exclusively for this purpose owing to its versatility in finding implicit functional dependencies between parameters which is useful for encoding the data-set to a lower dimensional latent feature space.

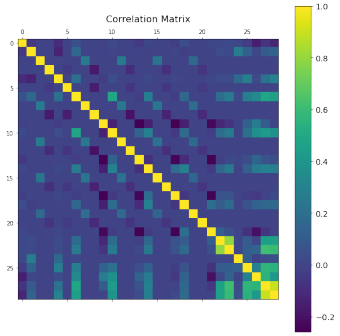

Fig. 2. Correlation Matrix between the features. Observe there is little correlation implying it would be very tough for our DAE to reduce dimensions.

A. Vanilla Deep Neural Network

A 5 Layer Deep Neural Network consisting of Linear Dense layers had been constructed for the task. For all but the last layer, `'ReLu'` activation was used, while for the last `'sigmoid'` activation was used. Using `Dropout` helped reduce chances of overfitting to the training set. It should be emphasised that methods like DNNs are particularly adept at identifying implicit relations between features and therefore are expected to give the best results.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 256)               7424
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_12 (Dense)             (None, 32)                4128
_________________________________________________________________
dense_13 (Dense)             (None, 8)                 264
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 9
=================================================================
Total params: 44,721
Trainable params: 44,721
Non-trainable params: 0
```

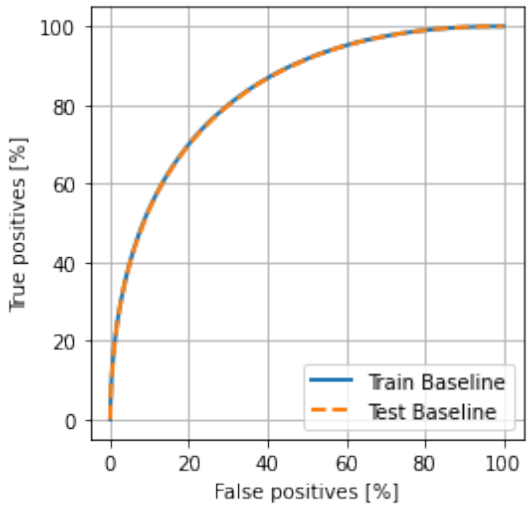

Fig. 3. ROC curve for vanilla DNN model

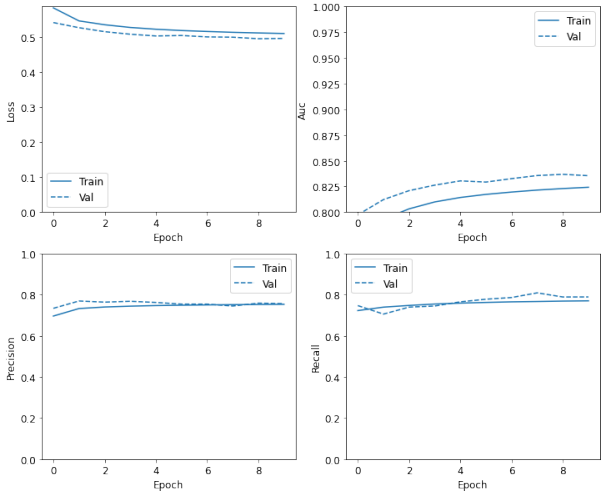

Fig. 4. Various metrics plotted for vanilla DNN model

B. Vanilla Extreme Gradient Boosting

We used `XGBClassifier()` from the XGBoost Library which is a ready implementation of the mentioned architecture. One of the advantages of XGB is the minimal finetuning which is needed before establishing a quick baseline.

```
XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,colsample_bynode=1,
colsample_bytree=1, gamma=0,learning_rate=0.1,
max_delta_step=0, max_depth=3,
missing=None, n_estimators=100, n_jobs=1,
nthread=None, objective='binary:logistic',
random_state=0,reg_alpha=0, reg_lambda=1,
scale_pos_weight=1, seed=None,
silent=None, subsample=1, verbosity=1)
```

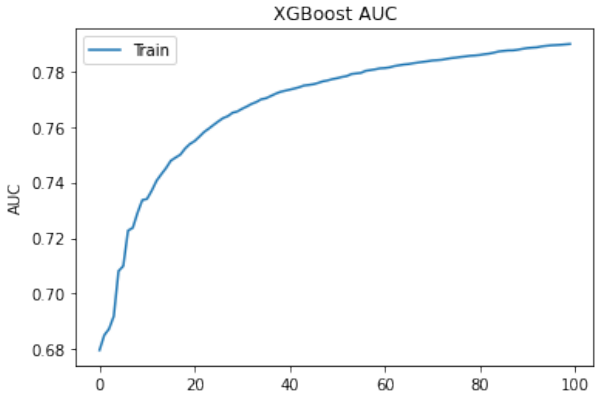

Fig. 5. AUC Metric plotted for XGB

C. Deep Autoencoder Model

A 5 layer Deep Encoder Model composed of Linear Dense units had been used. Bottleneck size of 8 has been tested to be optimal compared to 5/6/7/8. Sizes greater than this had not been attempted to justify adequate reduction of dimensions of the feature space. This model had been picked for optimal restoring accuracy and following results are with this parameter choice only.

```
    AE_4D_300_LeakyReLU(
(en1): Linear(in_features=28, out_features=300)
(en2): Linear(in_features=300, out_features=200)
(en3): Linear(in_features=200, out_features=100)
(en4): Linear(in_features=100, out_features=50)
(en5): Linear(in_features=50, out_features=8)
(de1): Linear(in_features=8, out_features=50)
(de2): Linear(in_features=50, out_features=100)
(de3): Linear(in_features=100, out_features=200)
(de4): Linear(in_features=200, out_features=300)
(de5): Linear(in_features=300, out_features=28)
(tanh): Tanh())
```

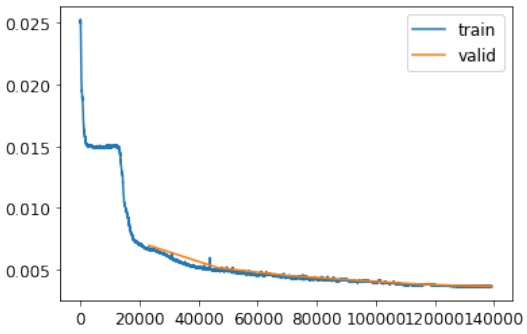

Fig. 6. Training and validation set loss for the DAE.

We ended up reducing the dataset from 4GB to only 200MB on disk storage. This is a huge reduction which justifies giving up some accuracy.

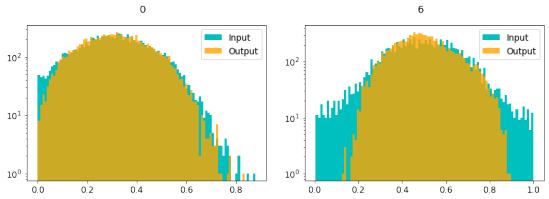

Fig. 7. Reconstruction of some variables from the test set. One variable very accurately reconstructed while the other has high bias for high values.

D. Deep Autoencoder followed by Deep Neural Network

We directly use the encoded output (8 features) of our DAE network as input at this stage. A 4 Layer Deep Neural Network consisting of Linear Dense layers had been constructed for the task. For all but the last layer, `'ReLu'` activation was used, while for the last `'sigmoid'` activation was used. Using `Dropout` helped reduce chances of overfitting to the training set.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               1152
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 520
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 9,937
Trainable params: 9,937
Non-trainable params: 0
```

E. Deep Autoencoder followed by Extreme Gradient Boosting

We directly use the encoded output (8 features) of our DAE network as input at this stage. To that input, we used `XGBClassifier()` from the XGBoost Library which is a ready implementation of the mentioned architecture. One of the advantages of XGB is the minimal finetuning which is needed before establishing a quick baseline.

```
XGBClassifier(
base_score=0.5, booster='gbtree',
colsample_bylevel=1,colsample_bynode=1,
colsample_bytree=0.8, gamma=0.1,
learning_rate=0.1, max_delta_step=0,
max_depth=5,min_child_weight=1,
missing=None,n_estimators=300, n_jobs=1,
nthread=None,objective='binary:logistic',
random_state=0,reg_alpha=0, reg_lambda=1,
```

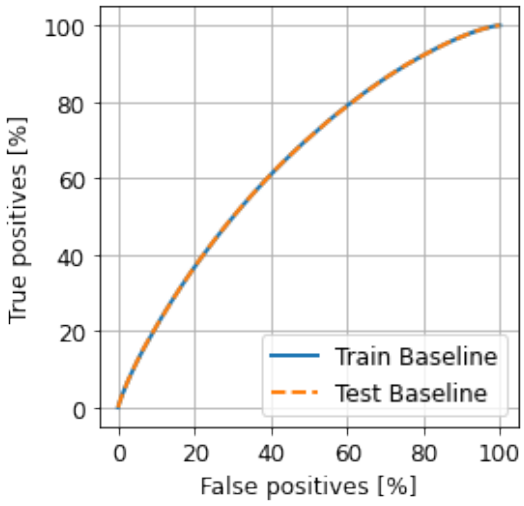

Fig. 8. ROC curve for DAE DNN model

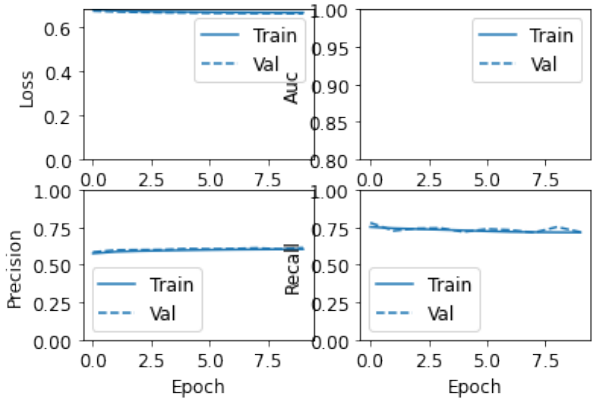

Fig. 9. Various metrics plotted for DAE DNN model

```
scale_pos_weight=1,seed=None,silent=None,
subsample=0.8, verbosity=1)
```

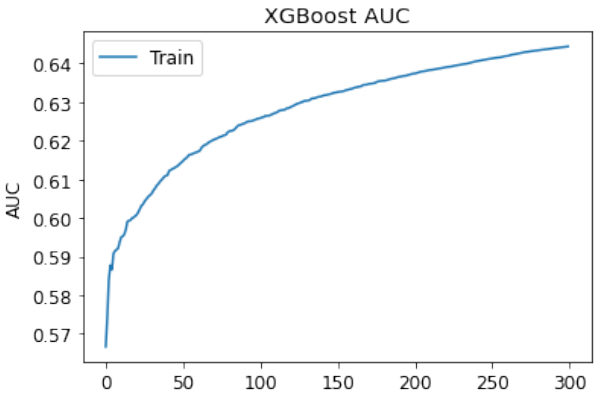

Fig. 10. AUC Metric plotted for DAE XGB model

Model	Test Metrics Accuracy	AUC
DNN	0.753	0.835
XGB	0.714	0.790
DAE + DNN	0.609	0.646
DAE + XGB	0.607	0.644

TABLE I. COMPARING ON TEST SET FOR HIGGS_6M DATASET

IV. CRITERIA FOR ASSESSING SOLUTIONS

All classification solutions are judged on their performance on the ROC curve. Additionally for methods on which Autoencoder has been applied to reduce dimnesionality, we judge the accuracy of information retrieval by the decoder to determine robustness of the encoder model.

V. RESEARCH METHODOLOGY

I first started out with exploring the different classes. Plotting histograms, I realised that some of them could be used as categorical classes.

On observing features, it was found that some exhibited exponential distributions and therefore it was felt necessary to log transform them. After that all features were standard scaled and then min max scaled. An additional step to categorical features of rounding them to integers was done to make it easier for the model to distinguish between their categories.

A train-test-valid split of 0.84-0.08-0.08 was carried out to measure performance.

At this point the datasets were ready for classification by the Vanilla models. The DNN models were written in Tensorflow and the XGB models were used as they are from the XGBoost library. The DAE model was trained on Fastai, an abstraction over PyTorch.

I then proceeded to training and documenting performance on test set for the Vanilla models. Then I shifted to building a good DAE and experimented with Bottleneck sizes. After finding a reasonably good model, I proceeded to train the DNN and XGB on encoded models.

All training was performed on Google Colab's GPUs. Due to varying allocation of GPU resources, a proper benchmarking of time could not be done, something which would be undertaken in future work. A reference to Table I.

VI. CONCLUSIONS AND RECOMMENDATIONS

On the basis of the above criteria of measuring AUC for classification we can observe that DNN based models have an edge over XGB models. A significant dip in accuracy is noted for the DAE+ models. A justification of this could be that the DAE itself has not been adequately finetuned to be representative of the information of the entire feature space. Another explanation could be to explore larger bottlenecks though that begs the question of reducing the compression ratio in the first place.

Owing to the fact that the bulk of this work had been completed in 4 days of part time work, significantly more efforts could be dedicated towards identifying useful architectures appropriate to the task.

The author is highly optimistic that such a proposal for work spanning over 3 months would definitely produce meaningful results for the scientific community.

APPENDIX A
DEALING WITH LARGE DATASETS

The dataset is large enough that if loaded without care it may exceed RAM capacity. This problem was observed when converting the original dataset to the encoded version after passing through DAE model. A solution was found of generating the encoded version chunk wise. Another solution was tested of training models batch-wise. This is trivial for a DNN but for XGB, incremental training had to be implemented.