MSDS 411: Unsupervised Learning

Assignment 3

August 4, 2025

Unsupervised pretraining using autoencoders for logistic regression

Yingyu Mao

Abstract

In this paper, I explored using unsupervised pretraining with autoencoders to improve credit risk assessment on a German credit card dataset. The goal is to predict which credit applicants are likely to be "good" (repaying debts) versus "bad" (defaulting). I built basic, sparse, and denoising autoencoders to embed features of the German credit card dataset and compared the performance of logistic regression models trained using the original data to using the embeddings. Furthermore, interaction features are generated to augment the feature space and stacked autoencoder architectures were trained with the new features. The augmented features reduced the performance of logistic regression models, but using the latent embeddings from a stacked autoencoder slightly improved logistic regression model performance.

Keywords: Unsupervised pretraining, autoencoder, feature embedding

1.0 Introduction

In the highly competitive credit card industry, a key challenge for companies is to expand their customer base while simultaneously mitigating financial risk. The ability to accurately predict which applicants will be "good" customers (i.e., those who repay their debts) versus "bad" customers (i.e., those who default) is crucial to profitability.

At a hypothetical company, the financial implications of misclassification are significant. It has been determined that the cost of granting credit to a bad customer is five times greater than the opportunity cost of denying credit to a good customer. Historically, the company has relied on traditional supervised learning techniques, such as logistic regression, to make predictions about credit applicants. However, these methods may be suboptimal. A more sophisticated approach, specifically the application of unsupervised learning methods, could lead to improvements in prediction accuracy, thereby significantly reducing the overall risk and costs. In this paper, I will explore the potential of unsupervised pretraining with autoencoders applied to a German credit card case dataset (Hofmann 1994) to redefine the credit risk assessment strategy and reduce the overall cost of misclassification.

2.0 Methods and Literature Review

An autoencoder is a type of artificial neural network designed for unsupervised learning, where the primary goal is to learn an accurate, compressed representation of input data. Its architecture consists of two main parts: an encoder and a decoder. The encode takes the input data and compresses it into a lower-dimensional representation, known as the "latent space" or "bottleneck". This process forces the network to capture the most essential features of the data while discarding noise and redundant information. The decoder takes the compressed representation from the latent space and attempts to reconstruct the original input data. By forcing the data through the low-dimensional latent space, the autoencoder learns a compact and meaningful encoding rather than simply copying the input.

Mathematically, an autoencoder can be described as learning two functions: an encoding function f and a decoding function g. Given an input vector x, the encoder maps it to a latent representation $z = f(x)$. The decoder then takes this latent representation and maps it back to a reconstruction of the input, $\hat{x} = g(z) = g(f(x))$. The goal of training the autoencoder is to minimize the difference between the original input x and the reconstructed output $\hat{x}$. This difference is measured by a loss function that reflects reconstruction error, often the mean squared error (MSE) or binary cross-entropy, depending on the nature of the data.

A neural network without activation functions constitutes the basic autoencoder. It learns linear projection of the original feature to a latent space (f and g are linear functions). When MSE is used as the loss function, it is essentially the same as Principal Component Analysis (PCA). With nonlinear activation functions, the projection also becomes nonlinear. The most common activation function that performs well is ReLu, due to its strong gradient for positive values and stable backpropagation than the sigmoidal function. ReLu suffers from dying ReLu problem where some neurons stop outputting anything other than 0, especially when the learning rate is large. Leaky ReLu solves this problem by adding a small slope for values < 0 (Xu, et al. 2015). Effective parameter initialization for leaky ReLu has been shown to be uniform or normal distribution

according to He initialization rules (He, et al. 2015) that control equal variance of the input and output of a layer.

Parameter sparsity generated by L1 regularization in neural networks has been used as a feature selection mechanism to filter out irrelevant dimensions (Goodfellow, Bengio and Courville 2016). The latent space of a sparse autoencoder can have a higher dimension than the input but with a sparsity constraint added to the loss function. This constraint encourages only a small number of neurons in the hidden layers to be active at any given time, forcing the model to learn a sparse representation of the original features (Géron 2022).

Any dataset can contain a certain level of noise in the form of irrelevant information. Denoising autoencoders are designed to be more robust to noise. The network is trained by first corrupting the input data with noise (e.g., adding some Gaussian noise or using a dropout layer), and then asking the autoencoder to reconstruct the original, uncorrupted input. This forces the model to learn robust features that can handle noisy or incomplete data.

There are other types of autoencoders, such as Variational Autoencoder that learns the underlying generative process of a dataset. Since generating new data is not the main goal of the classification task at hand, I will only focus on previous types of autoencoders for feature embedding.

Regardless of the architecture of an autoencoder, its parameters are optimized using backpropagation and gradient descent setup. Multiple optimizers have been developed over the years to improve the search speed for the global minimum. In this paper, I chose to use Nadam optimizer, with adaptive momentum plus the Nesterov trick (Dozat 2015), to perform all training.

3.0 Model Training and Results

3.1 Exploratory data analysis and data transformations

The German credit card dataset contains 14 categorical features and 7 numerical features. The response to be predicted is the credit classification of 70% ‘good’ cases and 30% ‘bad’ cases. I first mapped these labels to 0 and 1, respectively. `Credit_amount` has skewed distribution, so I used log transformation on this feature. The categorical features were one-hot encoded to generate 54 Boolean features. The numerical features were scaled but not centered using `MinMaxScaler`. Instead of regrouping some of the categories in `purpose` column as suggested in the example code provided in the assignment instruction, I kept the original values. The final number of features used in this study was higher than that generated by the example code.

3.2 Baseline model trained with original features

The baseline logistic regression model was trained with L2 regularization while accounting for the class imbalance of the response variable. The model was trained with 5 folds of stratified train and test set pairs for cross validation. For each stratified fold, the model was evaluated by calculating the average precision, recall, F1 score, and the cost of incorrect classification at different cutoff values as shown in Table 1. The lowest cost is not at the predicted cutoff 0.086 (accounting the class imbalance in the response variable and the cost of misclassification is 5 times higher for the ‘bad’ label) but around 0.2 to 0.4, which yields balanced precision and recall scores as indicated by the higher value of F1 score.

Table 1 Model performance with original features

	Baseline logistic regression ROC-AUC = 0.780738				PCA-encoded logistic regression ROC-AUC =0.761714				Basic, no activation reconstruction_error = 0.011846 ROC-AUC = 0.766929			
cutoff	**precision**	**recall**	**f1_score**	**cost**	**precision**	**recall**	**f1_score**	**cost**	**precision**	**recall**	**f1_score**	**cost**
0.086	0.32722	0.99	0.491827	125.2	0.319927	0.996667	0.484339	128.2	0.319586	1	0.484346	127.8
0.2	0.383411	0.94	0.544455	108.8	0.36608	0.94	0.526758	115.8	0.371604	0.956667	0.535175	110.2
0.3	0.420175	0.87	0.566465	111	0.408903	0.876667	0.557398	113	0.408312	0.893333	0.560187	109.6
0.4	0.474396	0.80333	0.59623	112.4	0.454879	0.81	0.582163	115.2	0.458079	0.81	0.584895	114.4
0.5	0.524001	0.72	0.606291	123.4	0.485811	0.693333	0.571033	136.2	0.48223	0.7	0.57096	135.2
	Basic, leaky ReLu activation reconstruction_error = 0.014261 ROC-AUC = 0.765643				**Sparse reconstruction_error = 0.013547 ROC-AUC = 0.78**				**Denoising reconstruction_error = 0.018465 ROC-AUC = 0.770976**			
cutoff	**precision**	**recall**	**f1_score**	**cost**	**precision**	**recall**	**f1_score**	**cost**	**precision**	**recall**	**f1_score**	**cost**
0.086	0.316112	0.99333	0.479574	131	0.302157	1	0.464075	138.6	0.319135	0.99	0.482638	129.8
0.2	0.366291	0.96667	0.531172	110.4	0.335537	0.99	0.501074	120.8	0.374473	0.95	0.537085	110.4
0.3	0.404956	0.89	0.556474	111.4	0.388234	0.93	0.547533	109	0.406656	0.856667	0.55123	118
0.4	0.457348	0.82	0.586988	112.4	0.44121	0.843333	0.578902	111.2	0.457288	0.783333	0.577155	120.6
0.5	0.486194	0.73	0.58351	127.4	0.503245	0.72	0.591892	127	0.507724	0.696667	0.58718	131.6
	Stacked reconstruction_error = 0.028702 ROC-AUC = 0.76419				**Mixed layers reconstruction_error = 0.015667 ROC-AUC = 0.757667**				**Mixed loss function ROC-AUC = 0.765595**			
cutoff	**precision**	**recall**	**f1_score**	**cost**	**precision**	**recall**	**f1_score**	**cost**	**precision**	**recall**	**f1_score**	**cost**
0.086	0.313374	1	0.477142	131.6	0.316914	0.996667	0.480858	130	0.31746	1	0.481928	129
0.2	0.364183	0.96667	0.528924	111.4	0.361407	0.943333	0.522392	117.2	0.361842	0.916667	0.518868	122
0.3	0.403299	0.88333	0.553386	113.4	0.402885	0.873333	0.551141	115.6	0.407692	0.883333	0.557895	112
0.4	0.447256	0.81667	0.577519	115.6	0.44222	0.79	0.566804	122.8	0.485149	0.816667	0.608696	107
0.5	0.483556	0.71	0.575043	132.6	0.484554	0.673333	0.563424	141	0.530864	0.716667	0.609929	123

I trained 3 single-layer autoencoders, basic, sparse, and denoising, as well as a stacked and mixed type autoencoder using the same train-test split strategy described above. The architectures of these autoencoders are listed below. I did not find better performance with either ReLu or leaky ReLu in comparison with no activation. Therefore, no activation function is used for the single-layer architectures.

	Basic	**Sparse**	**Denoising**	**Stacked**	**Mixed**
Input	Input (61)	Input (61)	Input (61)	Input (61)	Input (61)
Encoder	Dense (30)	Dense (60) + L1 regularization	Dropout (0.3) Dense (40)	Dense (40) Dense (20)	Dense (120, leak ReLu, He initialization) BatchNormalization Dropout (0.2) Dense (60, leak ReLu, He initialization) BatchNormalization Dropout (0.2) Dense (30, leak ReLu, He initialization)
Decoder	Dense (61)	Dense (61)	Dense (61)	Dense (40) Dense (61)	Dense (120, leak ReLu, He initialization) Dense (61)

The model was then complied with an MSE loss and a Nadam optimizer with a scheduled learning rate decay:
`tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=1e-2, decay_steps=500, decay_rate=0.95)`
The training was also carried out with an early stopping callback function to regulate overfitting.
The evaluation results for the model after 5-fold cross validation are shown in Table 1. The embeddings by autoencoders did not improve the logistic model performance at different cutoffs, nor did it achieve lower misclassification cost, despite their low reconstruction errors. Sparse autoencoder embeddings performed the best, which indicates the effectiveness of L1 regularization, although it is still slightly worse than the baseline model built upon the original data. This shows that with relatively low-dimension feature space, the benefit of dimension reduction through embeddings is not immediately obvious. This makes sense logically, as the goal of an autoencoder is to reconstruct the original dataset. During the process of encoding, there will always be some information loss that could potentially negatively impact prediction. This notion is supported by the fact that the PCA-transformed data with the same 30 latent features resulted in similar prediction performance of the logistic regression model.
Complex architecture, stacked and mixed autoencoders, performed worse with prediction, although maintained similar reconstruction errors. It is likely that the data does not have complex topology that needs nonlinear embedding. A simple PCA is as sufficient as a basic autoencoder.
Since MSE is not a suitable loss function for the one-hot-encode binary data in the dataset, I took a further step to test out a mixed loss function approach, where the decoder contains two outputs, one for the numerical features and the other for the binary-encoded features. `MSE` and `BinaryCrossEntropy` loss functions were used for the neurons representing the numerical features and those representing the binary-encoded features, respectively. This approach did not improve the logistic regression prediction (Table 1).

3.3 Interactive feature engineering and autoencoder dimension reduction

To further investigate the embedding outcome of autoencoders on higher dimension data, I engineered second degree interaction features using `PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)` from `sklearn.proprocessing`. This resulted in more than 2000 features. Since the data instances are only 1000, less than 2000 features, I performed a feature selection step prior to interaction feature engineering. This is based on the notion of hierarchical models in which higher-order terms are included only if the related lower-order terms are included (Fienberg 2007).
The relevant features were selected using a univariate feature selection module `SelectKBest` implemented in `sklearn.feature_selection`. The one-hot encoded categorical features that are independent of the `class` labels were identified using `chi2` test with a p-value cutoff of 0.1. The numerical features that are independent of the `class` labels were identified using ANOVA F-value computed with `f_classif` function.
After applying `PolynomialFeatures` and deleting the columns with all 0, there were 461 features in total to be embedded with autoencoders. Due to the more complex feature space, I used stacked autoencoders that are either basic, sparse, or denoising as shown in the table

below. Activation function was added back to sparse and denoising autoencoders to stabilize gradient descent learning.

	Basic	**Sparse**	**Denoising**
Input	Input (461)	Input (461)	Input (461)
Encoder	Dense (100) Dense (30)	Dense (100, leaky ReLu, He normal initialization) Dense (200, leaky ReLu, He normal initialization) + L1 regularization	Dropout (0.3) Dense (100, leaky ReLu, He normal initialization) Dense (30, leaky ReLu, He normal initialization)
Decoder	Dense (100) Dense (461)	Dense (100, leaky ReLu, He normal initialization) Dense (461)	Dense (100, leaky ReLu, He normal initialization) Dense (461)

The same training regime was used as mentioned in Section 3.2. The evaluation results are shown in Table 2. Interestingly, simply removing irrelevant features improves the logistic regression model performance. The performance was slightly reduced after augmenting the feature space with interaction features, as indicted by lower recalls and area under the Receiver Operating Characteristic (ROC) curve, as well as higher cost of misclassification. The basic stacked and the denoising autoencoders both slightly improved the logistic regression model performance using only 30 dimensions of latent features in comparison to 461 features (>15-fold dimension reduction), as indicated by higher AUC and overall lower cost of misclassification. PCA-encoded data with 30 dimensions resulted in similar results as the stacked autoencoder.

Table 2 Model performance with selected and engineered features

	Logistic regression with selected features ROC-AUC = 0.791357				**Logistic regression with interaction features ROC-AUC = 0.775786**				**PCA-encoding with interactino features logistic regression ROC-AUC = 0.788667**			
cutoff	**precision**	**recall**	**f1_score**	**cost**	**precision**	**recall**	**f1_score**	**cost**	**precision**	**recall**	**f1_score**	**cost**
0.086	0.325152	0.993333	0.489867	125.8	0.3699	0.94	0.530764	114.2	0.329893	0.996667	0.495644	122.6
0.2	0.376743	0.936667	0.53725	112	0.422096	0.86	0.566141	112.6	0.377505	0.933333	0.53749	112.4
0.3	0.417765	0.883333	0.567113	108.8	0.453379	0.79	0.57581	120.2	0.41538	0.876667	0.563536	111
0.4	0.476292	0.83	0.604907	105.8	0.484111	0.723333	0.579727	129.4	0.470677	0.806667	0.594275	112.4
0.5	0.527625	0.723333	0.609868	122	0.527961	0.656667	0.585039	138.4	0.523783	0.733333	0.610874	120
	Basic, stacked test_loss = 0.014962 ROC-AUC = 0.788643				**Sparse, stacked, leaky ReLu test_loss = 0.017623 ROC-AUC = 0.769548**				**Denoising, stacked, leaky ReLu test_loss = 0.011705 ROC-AUC = 0.782143**			
cutoff	**precision**	**recall**	**f1_score**	**cost**	**precision**	**recall**	**f1_score**	**cost**	**precision**	**recall**	**f1_score**	**cost**
0.086	0.326185	0.996667	0.491479	124.6	0.3	1	0.461538	140	0.312598	1	0.476277	132
0.2	0.37431	0.933333	0.534146	113.8	0.3	1	0.461538	140	0.35541	0.953333	0.517656	118
0.3	0.417895	0.883333	0.567295	108.8	0.309197	0.996667	0.471865	134.8	0.409238	0.906667	0.563835	106.6
0.4	0.465284	0.81	0.59061	113	0.36714	0.956667	0.529999	112.6	0.450796	0.83	0.584085	111.8
0.5	0.534689	0.716667	0.612278	122.4	0.490782	0.703333	0.577858	133	0.519383	0.716667	0.601836	124.8

4.0 Discussion

In this paper, I tested the encoding outcomes of different autoencoder architectures by using them for training a logistic regression model that predicts the credit class label of the German credit card dataset. The result shows that a simple-layered basic autoencoder is sufficient for

reconstruction of this original data. All tested architectures with or without nonlinear activation function, single or deeper layers, with or without dropout, resulted in reduction of performance of the downstream logistic regression model. This indicates the information lost during encoding might be useful to get better prediction results. Interestingly, the embeddings of linear autoencoders performed slightly better than the nonlinear autoencoders, but similar to PCA-encodings. This suggests the dataset in this study might not have nonlinear topology. The lack of significant improvement of the downstream supervised learning agrees with the findings made by Fanai and Abbasimehr's study, where autoencoders were used on a modified numerical form of the same dataset (Fanai and Abbasimehr 2023). Their PCA encodings had worse performance, but it could be due to the modified datatype used for training.

After augmenting the feature space with engineered interaction features, logistic regression performed slightly worse on the augmented data. Embeddings from the stacked and denoising autoencoders slightly improved logistic regression learning. The stacked autoencoder was also a linear model and performed slightly better than the denoising nonlinear model. As comparison, PCA encoding generated results closer to the stacked model.

In conclusion, autoencoders are effective tools for dimension reduction. A basic linear autoencoder behaves like PCA. Adding nonlinear activation function and making deeper architectures can enable nonlinear embeddings. A suitable autoencoder architecture for a dataset depends on its unique data structure. One would find better use of autoencoders when learning latent features of datasets with higher dimensions and structural complexity, such as image and audio data. However, since autoencoders (like PCA) are an unsupervised learning method, their primary goal is to reconstruct the input data. This differs from supervised learning, where the goal is to improve the final prediction of labels. Therefore, an autoencoder may faithfully reconstruct the original data but not necessarily aid a downstream supervised prediction goal. If the original data does not contain enough relevant features for the prediction task, then unsupervised pretraining cannot magically construct the missing key features. Further improvement of the prediction can only be achieved by identifying additional relevant features.

The model I would choose to present to a hypothetical management client for this classification problem should be the best performing model, which was not trained on any of the autoencoder-embeddings, but on the selected relevant features. I would also recommend them to explore other features relevant to the two classes to improve the model.

References

Dozat, Timothy. 2015. "Incorporating Nesterov Momentum into Adam." https://cs229.stanford.edu/proj2015/054_report.pdf.

Fanai, Hosein, and Hossein Abbasimehr. 2023. "A novel combined approach based on deep Autoencoder and deep classifiers for credit card fraud detection." *Expert Systems with Applications* 217:119562.

Fienberg, Stephen E. 2007. "Three-dimensional Tables." In *The Analysis of Cross-Classified Categorical Data*, by Stephen E. Fienberg, 43. New York: Springer Science+Business Media, LLC.

Géron, Aurélien. 2022. "Chapter 17. Autoencoders, GANs, and Diffusion Models." In *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. "Regularization for Deep Learning." In *Deep Learning*, 224-270. MIT Press.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." *arXiv* 1502.01852.

Hofmann, Hans. 1994. https://www.openml.org/search?type=data&sort=runs&id=31&status=active.

Xu, Bing, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. "Empirical Evaluation of Rectified Activations in Convolutional Network." *arXiv* 505.00853.