Washington University in St. Louis

OLIN BUSINESS SCHOOL

Deep Learning for Prediction of Business Outcomes

Final Project
Lending Club Loan Default Detection

Project Group 28, Section 22

Group Members:

Zixin Zhou (MSFQ)

Jiaxi Zhou (MSFQ)

Jiaying Lu (MSFQ)

Siana Dong (MSBA)

January 4, 2020

Presentation recording:

https://wustl.zoom.us/rec/share/a_6pXUKBS8BPUyJNyyytp4Awsl1uxDjTAk855Ao6KW7MwH3NtFf8yAQBm6gxRxVL.RaumkXEpo99nmsGh

Contents

Abstract

Correctly identifying defaulters so that the lending club can decide whether a person is fit for sanctioning a loan or not in the future is important. We propose loan default detection methods with LendClub Issued Loans Data using supervised learning methods--Fully-connected Neural Network and Convolutional Neural Network to estimate the probability of the default, and self-supervised learning methods-- autoencoder and variational autoencoder based anomaly detection. In supervised learning, we solved the data unbalanced problem using undersampling and oversampling. In the anomaly detection modeling, we used the reconstruction error as an anomaly score for the autoencoder and variational autoencoder.

We used these principles to detect loan default. Experimental results show that undersampled Fully-connected NN, oversampled CNN and Autoencoder are good techniques to solve the problem. And overall, undersampled Fully-connected NN performs best.

1 Introduction

Because of insufficient or non-existent credit histories, many people have trouble getting loans from traditional financial institutions like banks. And these populations are often taken advantage of by unreliable lenders. To solve this problem and to broaden financial inclusion for the unbanked or underbanked people, peer to peer lending companies, such as LendingClub, are committed to providing a positive and safe borrowing experience for this population.

In order to make sure this underserved population has a positive loan experiences, it is of great significance to make full use of various data to predict the clients' repayment abilities. Doing so will ensure that clients who have the ability to repay will not be rejected. It will also be meaningful to help the lenders predict and manage its credit risks.

The goal of our project is to take advantage of deep learning models to help lenders predict which loan will default on its payment by analyzing their datasets, so that lending club can decide whether a person is fit for sanctioning a loan or not in the future. The data set we will use is Lending Club's loan data. Apart from that, we are committed to solve the imbalanced problem of the dataset.

2 Literature Review

2.1 Imbalanced problem and oversampling & undersampling

Brownlee explains how oversampling and undersampling could affect and solve the imbalance problem. He explains how theoretically each method will solve the problem and explains the concepts clear enough. From his theorems and practices one could get some inlightments about how to solve the imbalance problem. Brownlee explains how to apply SMOTE, which is an oversampling method, for the use of balancing data, classification, and With Selective Synthetic Sample Generation; but only focuses on undersampling for solving the imbalance problem. Challenging point he does not focus on is the application of undersampling in other areas besides balancing the data.

2.2 Convolutional Neural Network

In the research "Deep Dense Convolutional Networks for Payment Prediction in Peer-to Peer-Lending", Kim and Cho built a less complicated classification CNN model with the lending club data set. Though their research showed a higher comparison performance in the F1-score around 88%, their model is overfitting, which we introduce maxpooling and flatten layers to make the model construction more complicated.

2.3 Anomaly Detection Approach in Autoencoder/Variational Autoencoder

Anomaly detection, then, is the task of finding those patterns in data that do not adhere to expected norms, given previous observations. The capability to recognize or detect anomalous behavior can provide highly useful insights across industries. It is often used in tasks such as intrusion detection (in cyber-security), fault detection in various types of systems and military surveillance. Chandola and Banerjee (2009) concludes six types of techniques used to detect anomalies in their research: Classification based techniques, clustering based techniques, statistical techniques, information theoretic techniques and spectral techniques. Based on the method developed in the Z. Chen, C. K. Yeo (2018)'s research, autoencoder can be utilized in anomaly detection. In the Renstrom and Holmsten (2018)'s research, they compare the performance of the variational autoencoder, single autoencoder, and stacked autoencoder. It shows that stacked autoencoder performs better than the other. And the variational autoencoder did not prove particularly useful as a classification solution for fraud. Finding optimal hidden layers, a better choosing threshold method for the prototype is challenge, and they will be explored by the author in the future. Gokden (2020) shares his project work of Applying Anomaly Detection with Autoencoder to Fraud Detection, which generates good model performance. The Keras Blog gives tutorials of developing autoencoders in Keras with examples and codes.

3 Problem Description

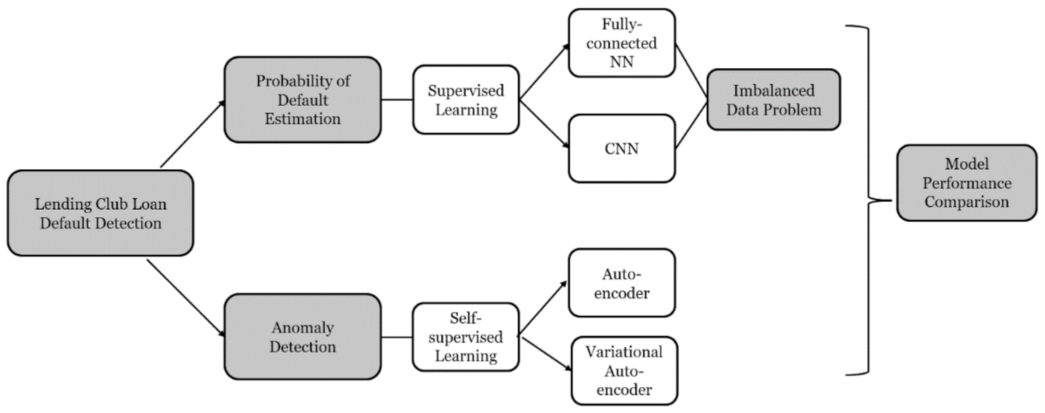


Figure 1 Problem Framework

The goal of our project is to detect the possible loan default of Lending Club. It can be divided into two parts, probability of default estimation and anomaly detection. We will use supervised learning models like fully-connected neural networks and convolutional neural networks to estimate the probability of default. We may come across imbalanced data problem in the process. We also plan to use self-supervised learning model including auto-encoder and variational auto-encoder model to detect the anomalies. Finally, we compare the outcomes of the models and choose the model with best performance.

4 Database Background and Data Preprocessing

The data set we will use is Lending Club's loan data. Lending Club enables borrowers to create unsecured personal loans from \$1000 to \$40000. There are 74 columns in the data set, including basic loan information like loan amount, interest rate and credit rating, and personal demographics like zipcode, home ownership and the purpose why they borrow. We will have loan data from 2007 to 2015 as our training data set and the rest loan data from 2016 to 2017 as test data set.

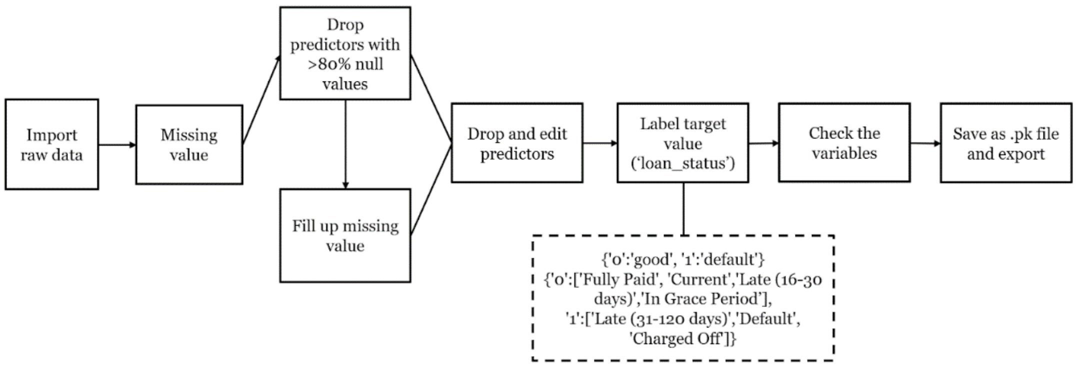

Import raw data
Missing value
Drop predictors with >80% null values
Fill up missing value
Drop and edit predictors
Label target value ('loan_status')
Check the variables
Save as .pk file and export
{'0':'good', '1':'default'}
{'0':['Fully Paid', 'Current','Late (16-30 days)','In Grace Period'],
'1':['Late (31-120 days)','Default', 'Charged Off']}

Figure 2 Data Preprocessing Flow Diagram

To prepare data for modelling, the first step is to drop the variables with >80% null values as they include little information. Then fill up missing values based on our understanding. Next, drop and edit each predictors to deal with non-numerical values. Then, label the target values ('loan_status'). '0' is assigned to the good loans, '1' is assigned to the default loans. At last, check the variables and save the data into a pickle format file. The table 1 & 2 in appendix showed the detail.

Besides, we used get dummies function in python that transform all non-numerical values to model-friendly format 0 and 1. And use sklearn.preprocessing package to assign labels to the data. Finally, there are 41 predictors left.

5 Supervised Learning: Probability of Default Estimation

5.1 Fully-connected Neural Network

(1) Model Desciption

Our logic of constructing models for this data set will have two lines: the first one is a simple neural network model followed by undersampling and oversampling methods; the second one is a convolutional neural network followed by undersampling and oversampling methods similarly.

The preprocessing of the data set has already excluded columns with unrelated information and handled the data format. Especially, we will focus on the "loan_status" column, which indicates whether the subject defaulted on the loan or not. Taking a close look on the "loan_status" column of the training data, we could see that the number of default transactions in training data is 52937 and number of transactions which do not default is 738087; number of default transactions in testing data is 52564 and number of transactions which do not default is 706270 – as we could see in the graph, the data has the problem of imbalanced classification and it might affect our result on correctly predicting the loan status on the test data set. That is the reason why we planned on using oversampling and undersampling to see whether they will solve the problem.

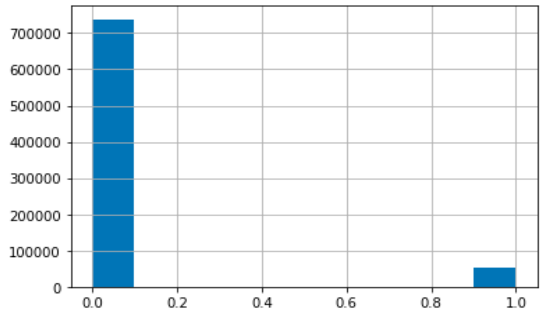


Figure 3 train data loan status histogram

(2) Data and Experimental Results

(a) Basic Model

We then classify X_train and X_test as columns excluding the “loan_status” column as they would be the columns in our model trying to simulate and explain the result. We classify the column “loan_status” as y_train and y_test, which are already preprocessed to binary in the previous steps. We would have to standardize X_train and X_test as there are 41 columns and each of the columns has a different range. Standardizing the columns using StandardScaler() would help stabilize the data and reduce the error. After making all four datasets from Dataframe to Array, they are ready to be implemented into a neural network model. We constructed the following model: a model with five dense layers and three dropouts. We used “relu” as activation for the beginning four layers and “sigmoid” for the last one to reduce vanishing problems. The result is satisfying after we adjust the parameters, epochs, and batch size, the result is around 97.48% accuracy (appendix figure 1). We will set this model as a basis for the convenience of comparing undersampling and oversampling models. From the confusion matrix for this model we show below, it could be seen that the number of default loans that are predicted by the model to be not default is still significant. An AUC of 0.92 indicates an excellent classifier as we could see below.

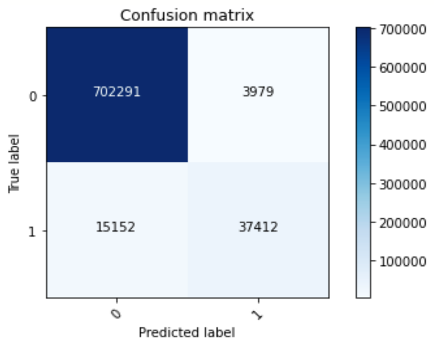


Figure 4 confusion matrix for neural network model

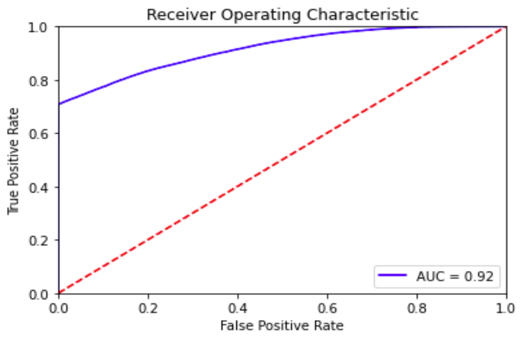


Figure 5 AUC-ROC for neural network

(b) Undersampling

As mentioned earlier, the data has the problem of imbalanced classification, the first method we would try to ease the problem is undersampling. Fernández stated in his book *Learning from Imbalanced Data Sets* that “undersampling, that consists of reducing the data by eliminating examples belonging to the majority class with the objective of equalizing the number of examples of each class (82).” There are many ways of undersampling, we choose to randomly select samples from the majority class, which is the non-default class, and delete them from both training and testing dataset. There are originally 791024 training data and 758834 testing data. After handling them with random choice, there are only 105874 training data and 105128 testing data left in our dataset. We then classify the data into under_X_train, under_X_test, under_y_train, under_y_test; standardizing under_X_train, under_X_test as we did earlier; and make the four classifications into Array.

We use the simple neural network we constructed earlier and make some adjustments on the original model trying to improve the accuracy. We discarded the dropout we used earlier in the model because undersampling has already decreased the sample size, we do not have to drop out further as too little data would affect our model in the opposite way. We also decrease the batch size accordingly as the dataset overall has less data.

The result of undersampling shows an accuracy of around 86.42% (appendix figure 2). We show the confusion matrix for undersampling model below. It could be seen that the possibility of incorrectly predicting good loans has decreased to zero, but the possibility of incorrectly predicting default loans has increased. Undersampling could ease the imbalance problem as AUC value further increased to 0.93, but it could not improve accuracy in the case of simple neural networks. We will further see how oversampling would affect our model in the following analysis.

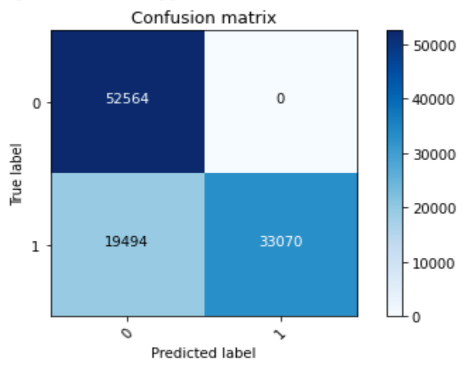


Figure 6 Confusion Matrix for Neural Network Model with Undersampling

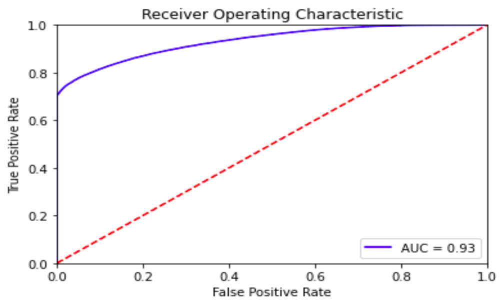


Figure 7 AUC-ROC for Neural Network Model with Undersampling

(c) Oversampling-SMOTE

The most common way to oversample the data is using Synthetic Minority Oversampling Technique (SMOTE). Similarly, we preprocess the four classifications as above: splitting them into X_train, X_test, y_train, and y_test, and then standardizing X_train and X_test. Before turning the four datasets from Dataframe to Array, we do one more step to oversample the data. We use SMOTE().fit_sample to resample the dataset. As we could see that there are originally 791024 training data and 758834 testing data. After processing with oversampling, there are 1476174 training data and 1412540 testing data.

We fit the data in array format into the neural network model. We have tried different batch sizes but the results do not seem to vary a lot. The ultimate accuracy is low – 76.53% (appendix figure 3). And from the confusion matrix below we could see that the possibility of incorrectly predicting default loans has further increased. After carefully analyzing the reason why oversampling does not seem to improve the result, we find that it would only be appropriate to increase the percentage of minority class, which is the default class, performing the test as the opposite of undersampling would bring a better result. However, it is hard to implement the ideal situation in real practice. In this case, resampling among the overall dataset would not help increase the accuracy and ease the imbalance problem. Next we would try the same series of models using convolutional neural networks.

```
Confusion matrix, without normalization
[[449402 256868]
 [ 74621 631649]]
```

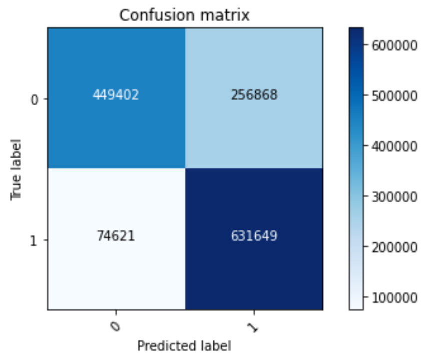


Figure 8 Confusion Matrix for Neural Network Model with Oversampling

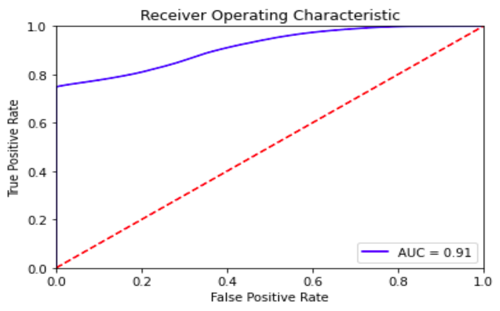


Figure 9 AUC-ROC for Neural Network Model with Oversampling

(3) Conclusion

- Utilizing dropout, regularization, change architecture, and early stopping, overfitting problem can be solved well
- Fully connected neural model has a good accuracy
- Undersampling would ease imbalance problem
- Oversampling might not be appropriate to solve the problem in this dataset

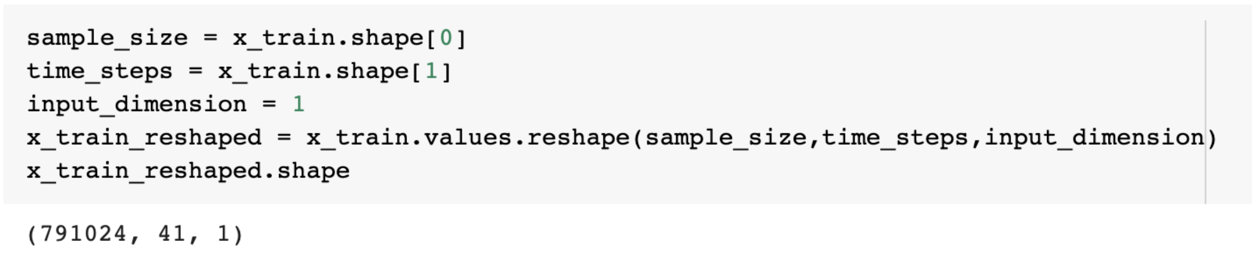
5.2 Convolutional Neural Network

(1) Model Description

The current data we are working on is a non-image data set. Therefore, we build a Conv1D model, instead of the traditional Conv2D model for the image data set. We use a sequential model with 3 Conv1D layers, one dropout layer, one MaxPooling1D Layer, and an output layer that returns a single loan default probability.

The original data already have split into training sets with 878916 and 758834 observations. There are plenty of observations in the training data. So, we have the train and validation split in a 9:1 ratio.

Prior to building the model, and training the model with the data set, reshaping the data set into the correct form is very important. The Conv1D layer expects input shape in 3D as [batch_size, time_steps, input_dimension]. However, the current data is in the shape of [batch_size, features]. For the current data set, it has 41 features and no timesteps. So, we convert features to timesteps and set the input_dimension to 1. To convert 2D of input data into 3D input, we reshape the data set as follows:



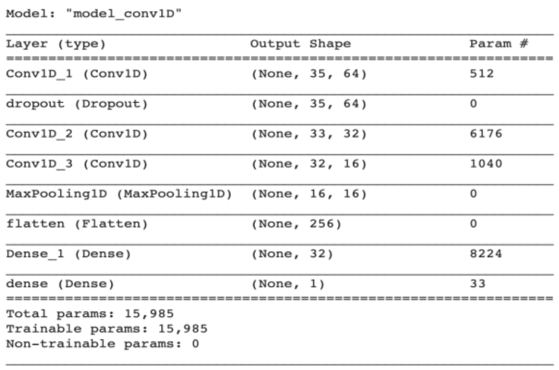
```
sample_size = x_train.shape[0]
time_steps = x_train.shape[1]
input_dimension = 1
x_train_reshaped = x_train.values.reshape(sample_size,time_steps,input_dimension)
x_train_reshaped.shape
```

```
(791024, 41, 1)
```

Figure 10 Data Reshaped Process

With the conversion, we have the train data set whose shape is [791024, 41,1].

After that, we create the Conv1D model with the following construction:



```
Model: "model_conv1D"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv1D_1 (Conv1D)            (None, 35, 64)            512
_________________________________________________________________
dropout (Dropout)            (None, 35, 64)            0
_________________________________________________________________
Conv1D_2 (Conv1D)            (None, 33, 32)            6176
_________________________________________________________________
Conv1D_3 (Conv1D)            (None, 32, 16)            1040
_________________________________________________________________
MaxPooling1D (MaxPooling1D)  (None, 16, 16)            0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
Dense_1 (Dense)              (None, 32)                8224
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 15,985
Trainable params: 15,985
Non-trainable params: 0
_________________________________________________________________
```

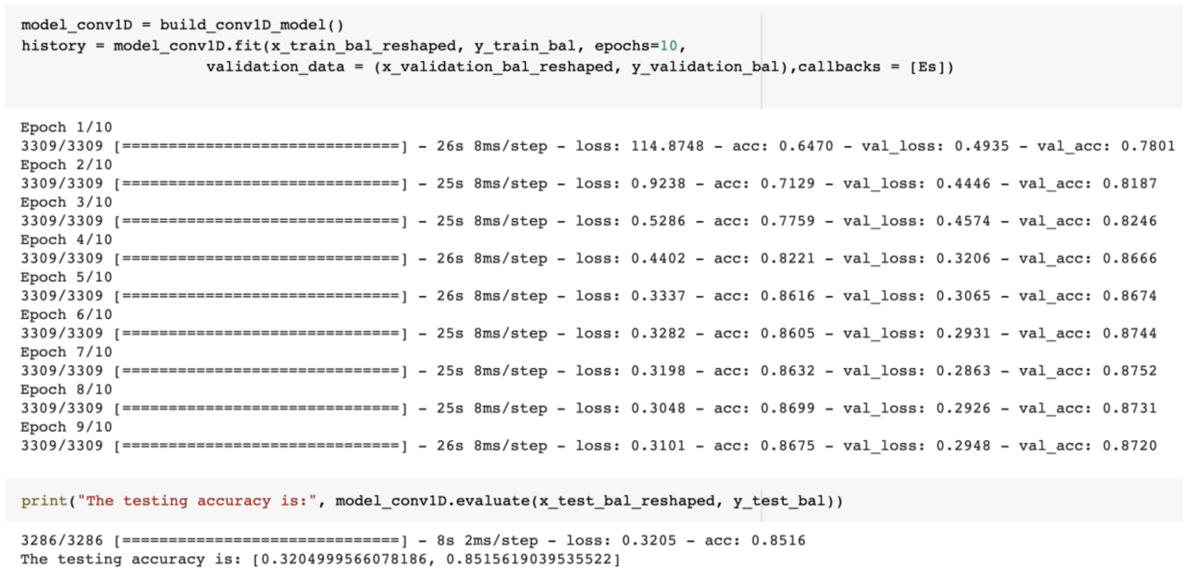
Figure 11 CNN Model Configuration

(2) Data and Experimental Results
(a) Basic Model

The training accuracy and validation accuracy when we fit the data into the model are both around 97%. The test accuracy is also about 98%, which is spectacularly good. However, when we count the number of 0 and 1 in the original data set, the data set is strongly imbalanced with more 1 than 0. Thus, the "perfect" result above cannot be taken into account.

(b) Oversampling and Undersampling

Having oversampling and undersampling to the original data set is our next step. Both two techniques balance the 0 and 1 occurrence by multiplying the minority class (oversampling) or decrease the majority class (undersampling). After the imbalance treatment, we have the same occurrence for 0 and 1. We fit the reshaped data into the model again and get the following result:



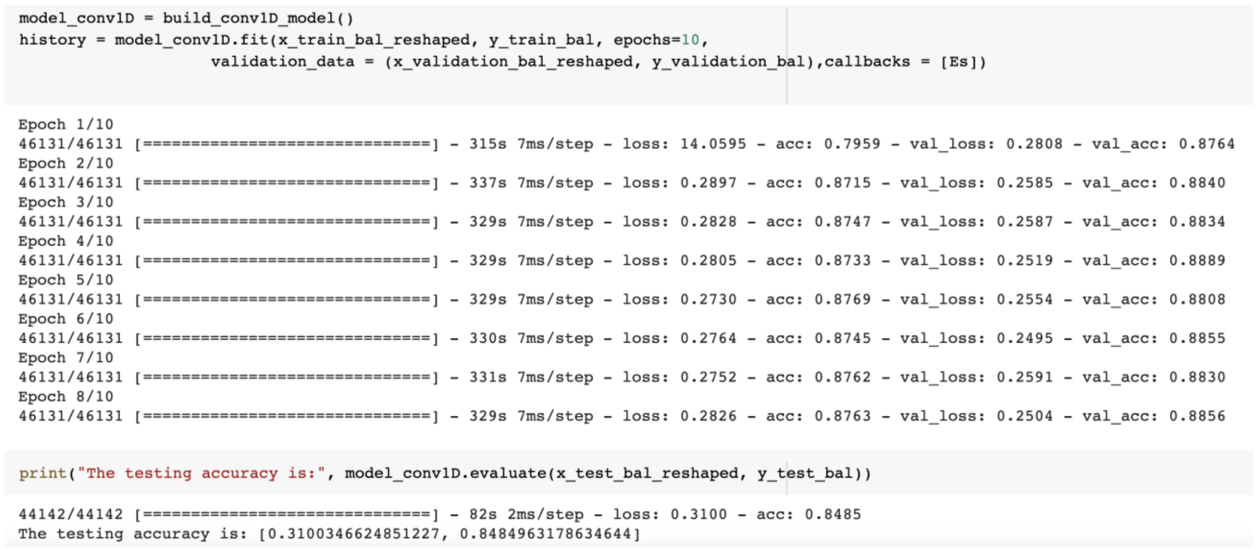
```
model_conv1D = build_conv1D_model()
history = model_conv1D.fit(x_train_bal_reshaped, y_train_bal, epochs=10,
                    validation_data = (x_validation_bal_reshaped, y_validation_bal),callbacks = [Es])
```

```
Epoch 1/10
3309/3309 [==============================] - 26s 8ms/step - loss: 114.8748 - acc: 0.6470 - val_loss: 0.4935 - val_acc: 0.7801
Epoch 2/10
3309/3309 [==============================] - 25s 8ms/step - loss: 0.9238 - acc: 0.7129 - val_loss: 0.4446 - val_acc: 0.8187
Epoch 3/10
3309/3309 [==============================] - 25s 8ms/step - loss: 0.5286 - acc: 0.7759 - val_loss: 0.4574 - val_acc: 0.8246
Epoch 4/10
3309/3309 [==============================] - 26s 8ms/step - loss: 0.4402 - acc: 0.8221 - val_loss: 0.3206 - val_acc: 0.8666
Epoch 5/10
3309/3309 [==============================] - 26s 8ms/step - loss: 0.3337 - acc: 0.8616 - val_loss: 0.3065 - val_acc: 0.8674
Epoch 6/10
3309/3309 [==============================] - 25s 8ms/step - loss: 0.3282 - acc: 0.8605 - val_loss: 0.2931 - val_acc: 0.8744
Epoch 7/10
3309/3309 [==============================] - 25s 8ms/step - loss: 0.3198 - acc: 0.8632 - val_loss: 0.2863 - val_acc: 0.8752
Epoch 8/10
3309/3309 [==============================] - 25s 8ms/step - loss: 0.3048 - acc: 0.8699 - val_loss: 0.2926 - val_acc: 0.8731
Epoch 9/10
3309/3309 [==============================] - 26s 8ms/step - loss: 0.3101 - acc: 0.8675 - val_loss: 0.2948 - val_acc: 0.8720
```

```
print("The testing accuracy is:", model_conv1D.evaluate(x_test_bal_reshaped, y_test_bal))
```

```
3286/3286 [==============================] - 8s 2ms/step - loss: 0.3205 - acc: 0.8516
The testing accuracy is: [0.3204999566078186, 0.8515619039535522]
```

Figure 12 Result of CNN with Oversampled Data



```
model_conv1D = build_conv1D_model()
history = model_conv1D.fit(x_train_bal_reshaped, y_train_bal, epochs=10,
                    validation_data = (x_validation_bal_reshaped, y_validation_bal),callbacks = [Es])
```

```
Epoch 1/10
46131/46131 [==============================] - 315s 7ms/step - loss: 14.0595 - acc: 0.7959 - val_loss: 0.2808 - val_acc: 0.8764
Epoch 2/10
46131/46131 [==============================] - 337s 7ms/step - loss: 0.2897 - acc: 0.8715 - val_loss: 0.2585 - val_acc: 0.8840
Epoch 3/10
46131/46131 [==============================] - 329s 7ms/step - loss: 0.2828 - acc: 0.8747 - val_loss: 0.2587 - val_acc: 0.8834
Epoch 4/10
46131/46131 [==============================] - 329s 7ms/step - loss: 0.2805 - acc: 0.8733 - val_loss: 0.2519 - val_acc: 0.8889
Epoch 5/10
46131/46131 [==============================] - 329s 7ms/step - loss: 0.2730 - acc: 0.8769 - val_loss: 0.2554 - val_acc: 0.8808
Epoch 6/10
46131/46131 [==============================] - 330s 7ms/step - loss: 0.2764 - acc: 0.8745 - val_loss: 0.2495 - val_acc: 0.8855
Epoch 7/10
46131/46131 [==============================] - 331s 7ms/step - loss: 0.2752 - acc: 0.8762 - val_loss: 0.2591 - val_acc: 0.8830
Epoch 8/10
46131/46131 [==============================] - 329s 7ms/step - loss: 0.2826 - acc: 0.8763 - val_loss: 0.2504 - val_acc: 0.8856
```

```
print("The testing accuracy is:", model_conv1D.evaluate(x_test_bal_reshaped, y_test_bal))
```

```
44142/44142 [==============================] - 82s 2ms/step - loss: 0.3100 - acc: 0.8485
The testing accuracy is: [0.3100346624851227, 0.8484963178634644]
```

Figure 13 Result of CNN with Undersampled Data

For both oversampled and undersampled data, there is a slight accuracy decrease in an acceptable range. Oversampled performs slightly better with test accuracy of 85%.

For better evaluation of the result, we also draw the ROC/AUC curve for two processed data sets. The curve results are as follows:

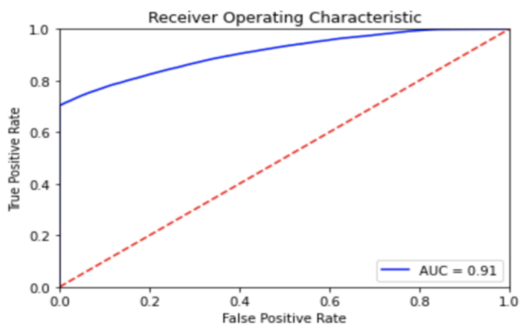


Figure 14 ROC/AUC Curve for Undersampled Data

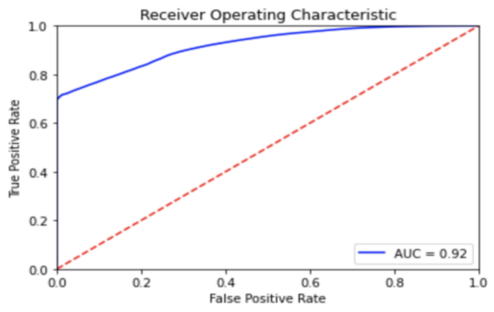


Figure 15 ROC/AUC Curve for Oversampled Data

(3) Conclusion

Both ROC/AUC curves perform well. Oversampled data performs slightly better with AUC score equals to 0.92. Therefore, the CNN model works the best with oversampled data.

6 Self-supervised Learning: Anomaly detection

In the second part of the modeling, we consider using self-supervised learning to solve a problem so that we don't need to solve imbalanced data problem, and we don't need labeled data, which is expensive in real-world business so that the model can be applied to more scenarios.

6.1 Anomaly Detection Approach in Autoencoder/Variational Autoencoder

(1)Introduction

Anomaly or outlier detection is the detection of points in a data set that is different from all other "normal" points. It is often used in tasks such as intrusion detection (in cyber-security), fault detection in various types of systems, and military surveillance.

With unsupervised learning, machines do not possess example input-output pairs that allow them to learn a function that maps the input features to outputs. Instead, they learn by finding structure within the input features. Because, as mentioned previously, labeled anomalous data is relatively rare, unsupervised approaches are more popular than supervised ones in the anomaly detection field. That said, the nature of the anomalies one hopes to detect is often highly specific. Thus, many of the anomalies found in a completely unsupervised manner could correspond to noise, and may not be of interest for the task at hand.

(2)Method

The principle of anomaly detection by using autoencoders is as follows: First, train the autoencoder on a training set that only contains the non-anomaly points. Because the autoencoder is now fitted on the normal data set, and the autoencoder has not yet encountered any anomalies, the reconstruction should perform better on normal data than on anomalies. Hence we can determine whether data points are anomalies or not by calculating the reconstruction error used to train the autoencoder. Generally the reconstruction errors of anomalies are higher than the reconstruction error of the normal data. Hence we can determine an upper bound α for the reconstruction error. If the reconstruction error of point x_i higher than α then x_i is an anomaly, otherwise it is not.

6.2 Autoencoder

Autoencoders are neural networks designed to learn a low-dimensional representation, given some input data. They consist of two components: an encoder that learns to map input data to a low-dimensional representation (termed the bottleneck), and a decoder that learns to map this low-dimensional representation back to the original input data. By structuring the learning problem in this manner, the encoder network learns an efficient "compression" function that maps input data to a salient lower-dimensional representation, such that the decoder network is able to successfully reconstruct the original input data. The model is trained by minimizing the reconstruction error, which is the difference (mean squared error) between the original input and the reconstructed output produced by the decoder. In practice, autoencoders have been applied as a dimensionality reduction technique, as well as in other use cases such as noise removal from images, image colorization, unsupervised feature extraction, and data compression.

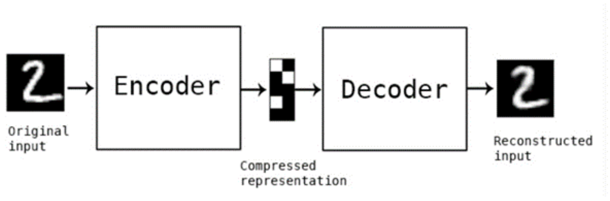


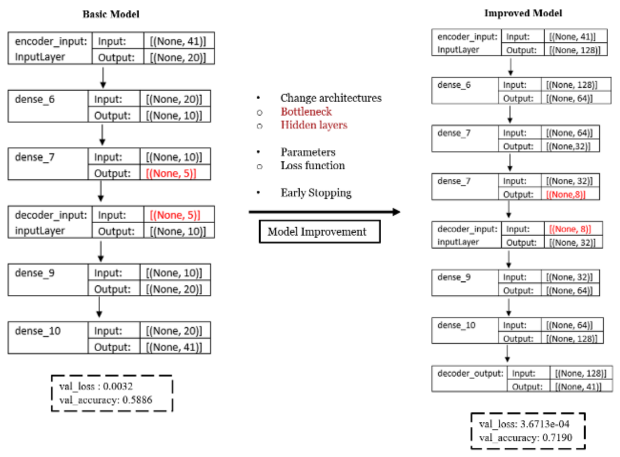
Figure 16 Autoencoder

(1) Model Description

The model is developed from a basic one, with three hidden layers in both encoder and decoder, and the bottleneck is 5, the loss function is MSE, the optimizer is 'adam'. Using this architecture, the model can have a 0.0032 validation loss and 0.5886 accuracy.

To improve the model, first, change the architecture by adding bottleneck and hidden layers. It works in model improvement. Also, loss function-'binary crossentropy' is tried, but 'mse' functions better in this case. And to prevent overfitting and find the optimal epochs, early stopping is applied.

The currently best model has four hidden layers which is more complex than the basic model, and bottleneck increases from 5 to 8.



Basic Model
encoder_input: InputLayer | Input: [(None, 41)] | Output: [(None, 20)]
dense_6 | Input: [(None, 20)] | Output: [(None, 10)]
dense_7 | Input: [(None, 10)] | Output: [(None, 5)]
decoder_input: inputLayer | Input: [(None, 5)] | Output: [(None, 10)]
dense_9 | Input: [(None, 10)] | Output: [(None, 20)]
dense_10 | Input: [(None, 20)] | Output: [(None, 41)]
val_loss : 0.0032
val_accuracy: 0.5886
Change architectures
Bottleneck
Hidden layers
Parameters
Loss function
Early Stopping
Model Improvement
Improved Model
encoder_input: InputLayer | Input: [(None, 41)] | Output: [(None, 128)]
dense_6 | Input: [(None, 128)] | Output: [(None, 64)]
dense_7 | Input: [(None, 64)] | Output: [(None,32)]
dense_7 | Input: [(None, 32)] | Output: [(None,8)]
decoder_input: inputLayer | Input: [(None, 8)] | Output: [(None, 32)]
dense_9 | Input: [(None, 32)] | Output: [(None, 64)]
dense_10 | Input: [(None, 64)] | Output: [(None, 128)]
decoder_output: | Input: [(None, 128)] | Output: [(None, 41)]
val_loss: 3.6713e-04
val_accuracy: 0.7190

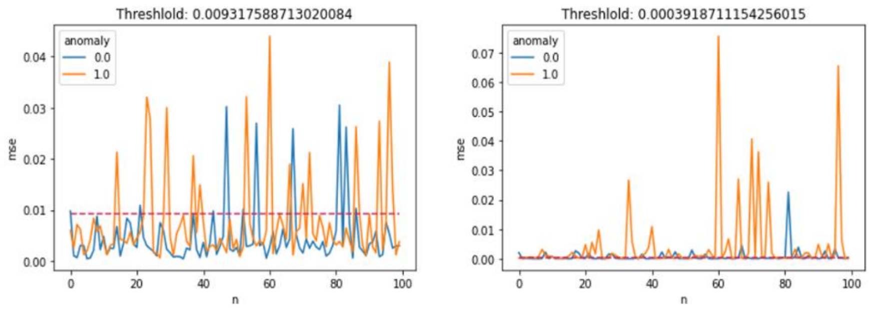
Figure 17 Autoencoder Framework

(2) Results and Evaluation

From the basic model to the improved model, mean-squared-error decreases from 0.0032 to 3.6713e-04, and accuracy increases from 0.5886 to 0.7190. And the models are trained well without overfitting and underfitting problem.

To evaluate the performance better, ROC-AUC curve is applied. For the basic model, AUC value is 0.7184. For the improved model, AUC value increases to 0.8461, which means current model can have good classification ability.

As we utilize threshold-based anomaly detection, based on the following graphs, we can see a significant improvement in anomaly detection using the improved model.



Basic Model Improved Model

Figure 18 Threshold-based Anomaly Detection Graph of Autoencoder

Overall, Autoencoder can perform well in this anomaly detection problem after improving model.

6.3 Variational Autoencoder

A variational autoencoder (VAE) is an extension of the autoencoder. Similar to an autoencoder, it consists of an encoder and a decoder network component, but it also includes important changes in the structure of the learning problem to accommodate variational inference. As opposed to learning a mapping from the input data to a fixed bottleneck vector (a point estimate), a VAE learns a mapping from an input to a distribution, and learns to reconstruct the original data by sampling from this distribution using a latent code. The encoder network learns the parameters (mean and variance) of a distribution that outputs a latent code vector Z, given the input data (posterior). On the other hand, the decoder learns a distribution that outputs the original input data point (or something really close to it), given a latent bottleneck sample (likelihood). Typically, an isotropic Gaussian distribution is used to model this reconstruction space. The parameters of the model are trained via two loss functions: a reconstruction loss forcing the decoded samples to match the initial inputs, and the KL divergence between the learned latent distribution and the prior distribution, acting as a regularization term.

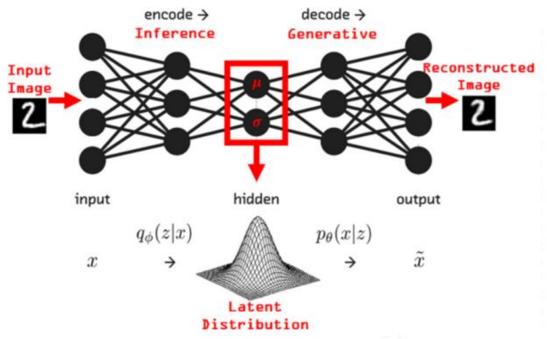


Figure 19 Variational Autoencoder

(1) Model Description

Next, the variational autoencoder is constructed to see if it can have better performance than the autoencoder.

The model is developed from the basic one with one hidden layer (units = 64) in encoder and decoder. We use the combination of reconstruction error and KL loss to train the model, and set the optimizer as 'adam'. It can have validation loss of 10.6706, and validation accuracy of 0.3098. Using approaches of changing architecture, I find that the model performances improve little. And using different optimizer, I find that RMSprop doesn't work for this data, and Adamax is sort of better than Adam, but the difference is small actually.

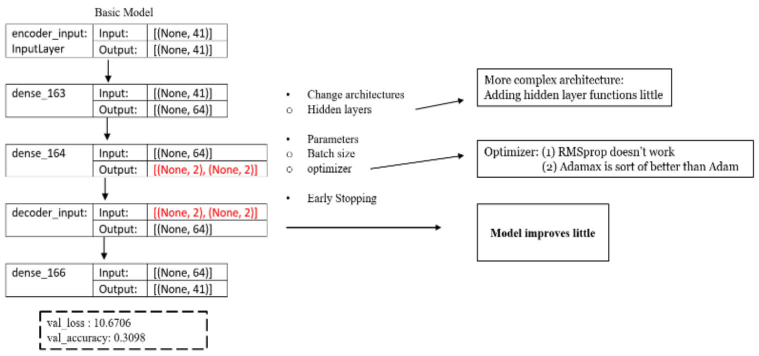

Basic Model
encoder_input: InputLayer | Input: [(None, 41)] | Output: [(None, 41)]
dense_163 | Input: [(None, 41)] | Output: [(None, 64)]
dense_164 | Input: [(None, 64)] | Output: [(None, 2), (None, 2)]
decoder_input: | Input: [(None, 2), (None, 2)] | Output: [(None, 64)]
dense_166 | Input: [(None, 64)] | Output: [(None, 41)]
val_loss : 10.6706
val_accuracy: 0.3098
• Change architectures
○ Hidden layers
More complex architecture: Adding hidden layer functions little
• Parameters
○ Batch size
○ optimizer
Optimizer: (1) RMSprop doesn't work (2) Adamax is sort of better than Adam
• Early Stopping
Model improves little

Figure 20 VAE Framework

(2) Results and Evaluation

For the VAE, the model loss is around 10.6 and validation accuracy is around 0.3. And it is hard to be improved by changing architecture and common optimizers. To measure the model better, we use ROC-AUC curve that the AUC value is around 0.68. And based on the threshold classification graph, there is some classification ability for the current model, but not much.

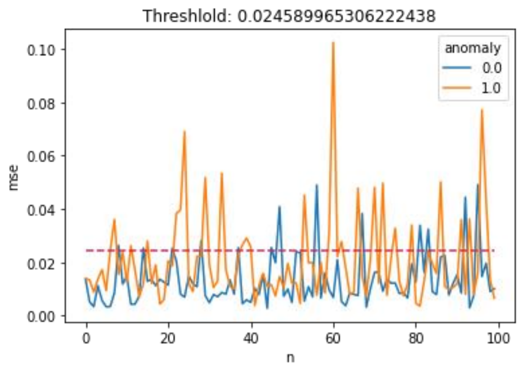


Figure 21 Threshold-based Anomaly Detection Graph of VAE

6.4 Conclusion

Autoencoder can perform well in this anomaly detection problem after improving model using the methods of changing the architecture, parameters and early stopping.

To improve the model further, some methods can be tried to the autoencoder anomaly detection:

- Use pretraining model to train the autoencoder
- Use other optimizer to train model (simulated annealing, genetic algorithm, particle swarm optimization, etc.)

VAE doesn't work well so far, no matter how to change the architecture and parameters.

As it is demanding on the decoder, we still need to explore other methods to improve the model. The potential solution to improve the model performance is using other complex optimizers to train model.

7 Model Performances Comparison

To find the best model to solve our default detection problem, accuracy of the test data set and AUC metrics are utilized as the performance measurement. Best performance of the fully-connected Neural Network is the one using undersampling tool. It has test accuracy of the 0.8642, and AUC of 0.93. Best performance of the Convolutional Neural Network is the one using oversampling. It has 0.8485 test accuracy and 0.92 AUC value. For the self-supervised learning modeling, autoencoder performances better than the variational autoencoder. And the best model can have the accuracy of 0.6988 and 0.8461 AUC.

In conclusion, for current data set and algorithms, undersampled fully-connected neural network can have best performance.

Table 1 Model Performance Comparison

Model	Accuracy of the test data	AUC value
Fully-connected NN-Undersampling	0.8642	0.93
CNN-Oversampling	0.8485	0.92
Autoencoder	0.6988	0.8461

8 Conclusion, Discussion and Future Work

(1) Fully-connected Neural Network: undersampling would ease imbalance problem, and functions better than the oversampling tools for fully-connected NN and this dataset.

(2) CNN: oversampled data performs better with the CNN model in both accuracy and AUC/ROC curve.

(3) Autoencoder/VAE: Based on current algorithm, autoencoder performs better than VAE.

Overall, for current deep learning algorithms and dataset, undersampled fully-connected NN works best for this loan default detection problem with the high accuracy of 0.86. We can achieve our project objectives that predict loan defaulters well so that lending club can decide whether a person is fit for sanctioning a loan or not in the future.

And at the next step, we will explore better approaches to improve the performance of the autoencoder and variational autoencoder.

Appendix

Tables 1 Data preprocessing: How to treat missing values

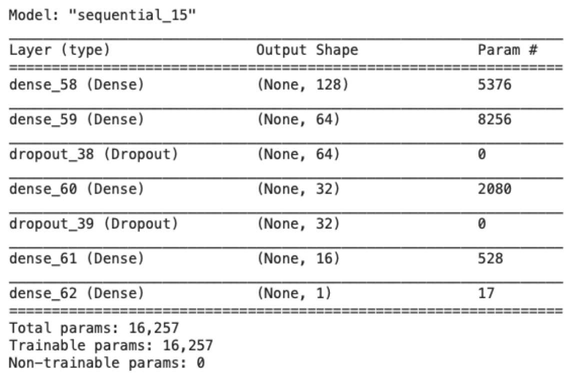
Variable	**Descriptions**
loan_status (Y variable)	{'0':'good', '1':'default'} {'0':['Fully Paid', 'Current','Late (16-30 days)','In Grace Period'], '1':['Late (31-120 days)','Default', 'Charged Off']}
id	Dropped
member_id	Dropped
desc	Dropped
title	Dropped
purpose	describe a reason for the loan
url	Dropped
application_type	Application type variable shows whether the loan is individual or join: 0 to individual and 1 to jointly applied loan
term	Was changed to numerical, with value 36 and 60 represents term of 36 months and 60 months
grade	Assigned number 0-6 to grade A-G
sub_grade	Assigned number 0-34 to grade A1-G5
emp_title	Dropped
emp_length	Length of employment: {'n/a':0, '< 1 year':1, '1 year':2, '2 years':3, '3 years':4, '4 years':5, '5 years':6, '6 years':7, '7 years':8, '8 years':9, '9 years':10, '10+ years':11}
home_ownership	Description on whether the home is owned by the individual; drop the rows of ANY
verification_status	Indicated whether the source of income of a client was verified.
pymnt_plan	Dropped
zip_code	Dropped
addr_state	Dropped
list_status	Listing status of the loan: 1 to free, 0 otherwise
policy_code	Dropped
recoveries	1 to recoveries not equal to 0, otherwise 0
collection_recovery_fee	Dropped

Table 2 Data preprocessing: how to treat non-numeric values

Variable	**Descriptions**
dti_joint	Dropped, missing value is over 99%
annual_inc_joint	Replace NaN with 0
verification_status_joint	Dropped, missing value is over 99%
open_acc_6m	Dropped, missing value is over 97%
open_il_6m, open_il_12m, open_il_24m, mths_since_rcnt_il, total_bal_il, il_util, open_rv_12m, open_rv_24m, max_bal_bc, all_util, inq_fi, total_cu_tl, inq_last_12m, collections_12_mths_ex_med	Dropped, missing value is over 99%

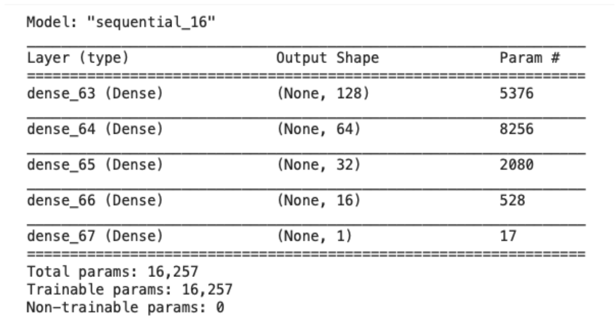
mths_since_last_delinq	Whether the specific action took place (delinquency, public record, worse rating. Binary categories of Yes (1), No (0).
mths_since_last_major_derog	
mths_since_last_record	
tot_coll_amt	Have missing values for the same observations. Replace NaN with 0.
tot_cur_bal	
total_rev_hi_lim	
revol_util	Revolving line utilization rate (the amount of credit the borrower is using relative to all available revolving credit.) Replace NaN with 0
last_credit_pull_d	Dropped. Most recent month LC pulled credit for this loan
issue_d	Dropped. The date loan was issued
next_payment_d	Dropped. The date of next payment
last_pymnt_d	Dropped
pymnt_received	Binary categories of Yes (1), No (0).
acc_now_delin	Replace NaN with most common values.
open_acc	
total_acc	
pub_rec	
delinq_2yrs	
inq_last_6mths	
earliest_cr_line	

Figure 1: Model Loss of the Improved Autoencoder Model



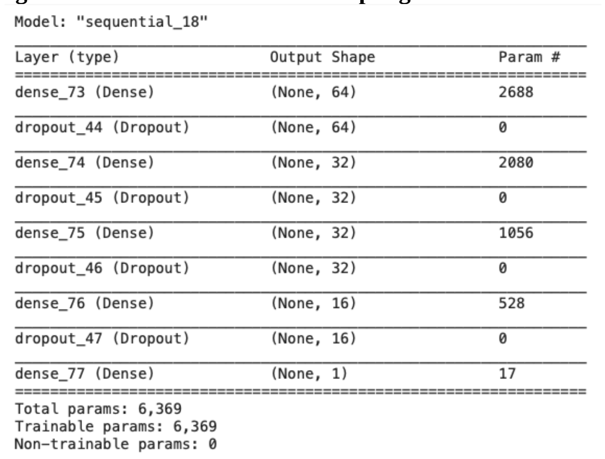
```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_58 (Dense)             (None, 128)               5376
_________________________________________________________________
dense_59 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_38 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_60 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_39 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_61 (Dense)             (None, 16)                528
_________________________________________________________________
dense_62 (Dense)             (None, 1)                 17
=================================================================
Total params: 16,257
Trainable params: 16,257
Non-trainable params: 0
```

Figure 2: CNN model with undersampling



```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_63 (Dense)             (None, 128)               5376
_________________________________________________________________
dense_64 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_65 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_66 (Dense)             (None, 16)                528
_________________________________________________________________
dense_67 (Dense)             (None, 1)                 17
=================================================================
Total params: 16,257
Trainable params: 16,257
Non-trainable params: 0
```

Figure 3: CNN model with oversampling



```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_73 (Dense)             (None, 64)                2688
_________________________________________________________________
dropout_44 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_74 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_45 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_75 (Dense)             (None, 32)                1056
_________________________________________________________________
dropout_46 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_76 (Dense)             (None, 16)                528
_________________________________________________________________
dropout_47 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_77 (Dense)             (None, 1)                 17
=================================================================
Total params: 6,369
Trainable params: 6,369
Non-trainable params: 0
```

Figure 4 Model Loss of the Basic Autoencoder Model

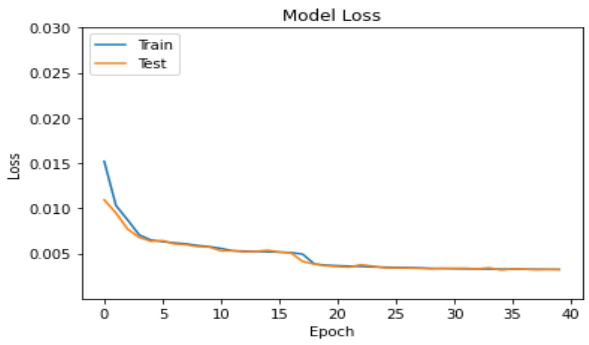


Figure 5 Model Loss of the Improved Autoencoder Model

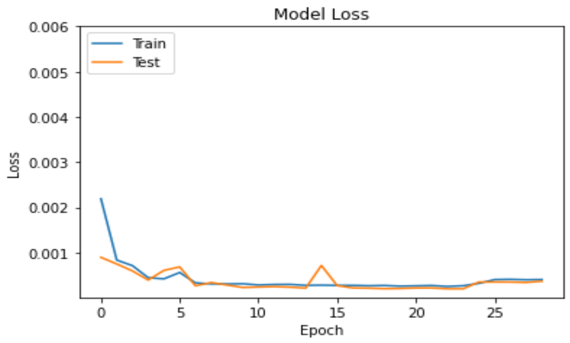
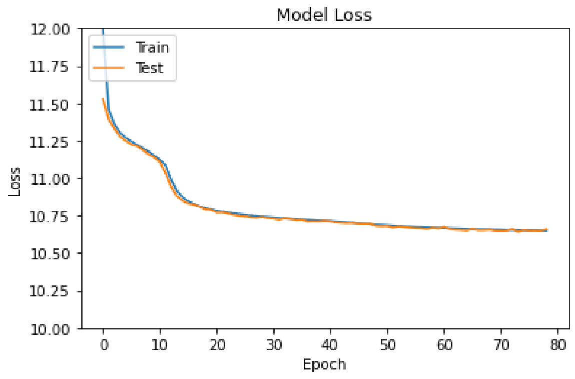


Figure 6   Model Loss of the Basic Variational Autoencoder Model


Model Loss
Train
Test
Loss
Epoch
12.00
11.75
11.50
11.25
11.00
10.75
10.50
10.25
10.00
0
10
20
30
40
50
60
70
80

References

[1] Brownlee, Jason. “SMOTE for Imbalanced Classification with Python.” Machine Learning Mastery, 20 Aug. 2020, machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/.
[2] Berk Gokden, Applying Anomaly Detection with Autoencoders to Fraud Detection, towards data science 2020: https://towardsdatascience.com/applying-anomaly-detection-with-autoencoders-to-fraud-detection-feaaee6b5b09
[3] Deep Dense Convolutional Networks for Repayment Prediction in Peer-to-Peer Lending, http://sclab.yonsei.ac.kr/publications/Papers/IC/2018_SOCO_JYK.pdf
[4]https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
[5] LendingClub Issued Loans Data: https://www.kaggle.com/husainsb/lendingclub-issued-loans
[6] Renström, Holmsten: Fraud Detection on Unlabeled Data with Unsupervised Machine Learning, KTH 2018
[7] The Keras Blog-Building Autoencoders in Keras: https://blog.keras.io/building-autoencoders-in-keras.html
[8] Tom Sweers, Autoencoding Credit Card Fraud, Radbound University 2018
[9] Understanding of Convolutional Neural Network (CNN) — Deep Learning
Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. ACM Comput. Surv., 41(3):15:1–15:58, July 2009
[10] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau. Autoencoder-based network anomaly detection. In 2018 Wireless Telecommunications Symposium (WTS), pages 1–5, April 2018.
[11] Packages: Python 3.0 numpy; pandas; matplotlib; seaborn; sklearn; mlxtend.plotting; pathlib2; pickle; keras; tensorflow;