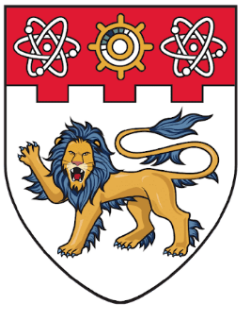

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

MH4510 Statistical Learning and Data Mining

Know Your Loan Potential

Crimson Twilight
Agarwal Lakshya U2123904L
Aruna Vadivelu U2130195C
Tho Hong De U2140435K
Tiew Ding Xuan U2140462J

15 November 2024

Abstract

In this study, we develop and evaluate machine learning models to predict loan approvals and estimate interest rates using financial data. The objective is to provide a robust decision-support tool for financial institutions by accurately classifying loan approval decisions and predicting viable interest rates. We applied the following models: Random Forest, XGBoost, and Neural Networks for loan classification, Principal Component Regression (PCR), Linear Regression with Autoencoders and Neural Networks for interest rate prediction. Results showed that ensemble methods such as Random Forest and XGBoost outperformed Neural Networks on loan classification tasks, while PCR provided strong predictive power for interest rate estimation. These models offer practical solutions to streamline financial decision-making, reduce risk, and improve loan processing efficiency. The findings provide a foundation for implementing data-driven, scalable predictive tools within lending frameworks.

Contents

1 Introduction

Financial institutions face increasing demands for efficient and accurate loan processing to meet customer expectations and manage risk effectively. Predicting loan approvals and estimating appropriate interest rates are critical in this decision-making process. Traditional assessment methods can be resource-intensive and may not fully leverage the predictive insights available in vast financial datasets. In this project, we harness machine learning to automate and enhance loan evaluation processes, aiming to improve approval accuracy and interest rate estimation. We employ several machine-learning techniques tailored to different aspects of the task. For loan approval classification, we use Random Forest and XGBoost due to their ability to handle complex, non-linear relationships and deliver robust predictions, as well as Neural Networks. For interest rate estimation, we apply Principal Component Regression (PCR) to reduce dimensionality and mitigate multicollinearity, along with Linear Regression with Autoencoders and Linear Neural Networks.

The following sections outline the data preprocessing steps, model configurations, and evaluation metrics used. Our findings demonstrate the effectiveness of these models in producing reliable classifications and predictions, offering valuable insights and potential applications for automated financial decision-making.

1.1 Objectives

Our project comprises two parts:

- Classify loan approvals through Random Forest, XGBoost, and Neural Networks.
- Predict interest rates using Principal Component Regression, Linear Regression with Autoencoders and Linear Neural Networks.

2 Exploratory Data Analysis

2.1 Dataset Description

The Synthetic Dataset for Risk Assessment and Loan Approval Modeling was obtained from the Kaggle database (Zoppelletto, n.d.). Due to the sensitivity of real-world data, this dataset is generated synthetically. It consists of a comprehensive set of financial and personal attributes for each loan applicant, ranging from demographic details, and financial metrics to credit behaviour indicators. It has 20000 data points and 36 variables.

The variables are as shown below:

- *ApplicationDate, Age, AnnualIncome, CreditScore, EmploymentStatus, EducationLevel, Experience, LoanAmount, LoanDuration, MaritalStatus, NumberOfDependents, HomeOwnershipStatus, MonthlyDebtPayments, CreditCardUtilizationRate, NumberOfCreditLines, NumberOfCreditInquiries, DebtToIncomeRatio, BankruptcyHistory, LoanPurpose, PreviousLoanDefaults, PaymentHistory, LengthofCreditHistory, SavingsAccountBalance, CheckingAccountBalance, TotalAssets, TotalLiabilities, MonthlyIncome, UtilityBillsPaymentHistory, JobTenure, NetWorth, BaseInterestRate, InterestRate, MonthlyLoanPayment, TotalDebtToIncomeRatio, LoanApproved, RiskScore*

We excluded the variables ApplicationDate and BaseInterestRate as the date of application is not relevant to our predictions and base interest rates are highly correlated with our regression task's response variable, *InterestRate*.

- Response variables (Y): *LoanApproval* (Classification Task), *InterestRate* (Regression Task)
- Predictor variables (X): *Age, AnnualIncome, CreditScore, EmploymentStatus, EducationLevel, Experience, LoanAmount* (Regression Task),*LoanDuration, MaritalStatus, NumberOfDependents, HomeOwnershipStatus, MonthlyDebtPayments, CreditCardUtilizationRate, NumberOfOpenCreditLines, NumberOfCreditInquiries, DebtToIncomeRatio, BankruptcyHistory, LoanPurpose, PreviousLoanDefaults, PaymentHistory, LengthofCreditHistory, SavingsAccountBalance, CheckingAccountBalance, TotalAssets,*

TotalLiabilities, MonthlyIncome, UtilityBillsPaymentHistory, JobTenure, NetWorth, InterestRate (Classification Task),*MonthlyLoanPayment, TotalDebtToIncomeRatio, LoanApproved, RiskScore*

For LoanApproval, '0' represents Not Approved while '1' represents Approved.

2.2 Data Preprocessing

- **Ordinal Encoding Categorical Variables:** Variables such as EducationLevel are encoded ordinally as it signifies an increase in the depth of knowledge and duration of study.
- **One-Hot Encoding Categorical Variables:** Variables such as EmploymentStatus, MaritalStatus, HomeOwnershipStatus, and LoanPurpose have no inherent ranking or order, hence, these are encoded using One-Hot Encoding.
- **Scaling and Normalisation:** For numerical variables such as Age and MonthlyIncome, scaling prevents variables with larger magnitudes from disproportionately influencing the model's predictions.
- **Splitting data into training and test sets:** We randomly split the processed dataset into 80% training set and 20% test set.

2.3 Data Visualisations

In our regression models, *InterestRate* is the response variable. To understand the relationship between *InterestRate* and all other features, we plotted a scatter plot as seen below.

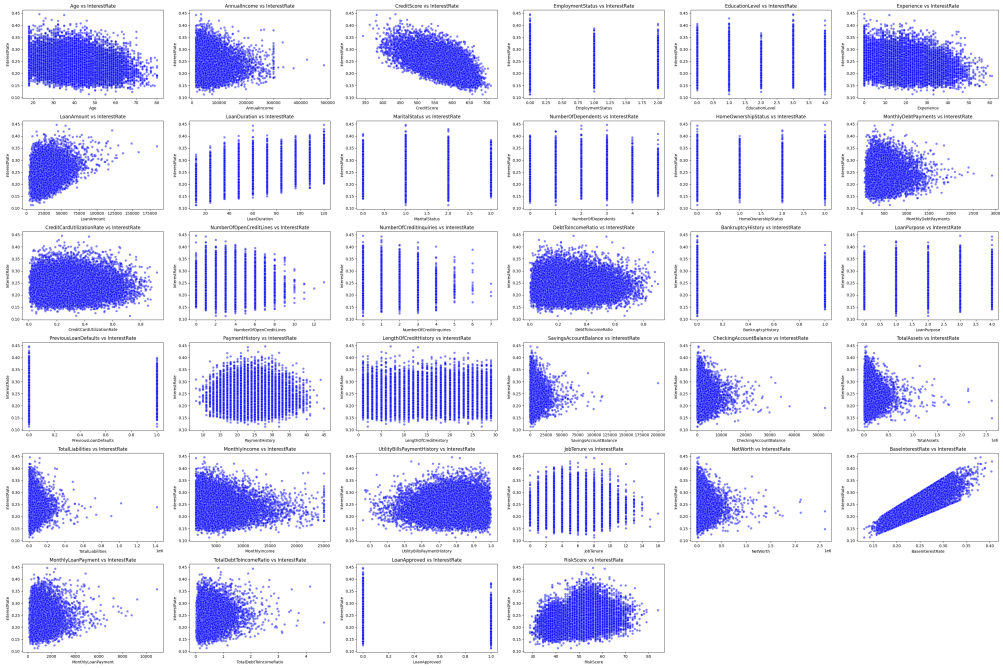

Figure 1: Scatterplot of *InterestRate* against all other features

From the scatterplot, we observed that our features exhibit a correlation with the response variable. Hence, we derived the correlation matrix below to get a better understanding of collinearity in our data.

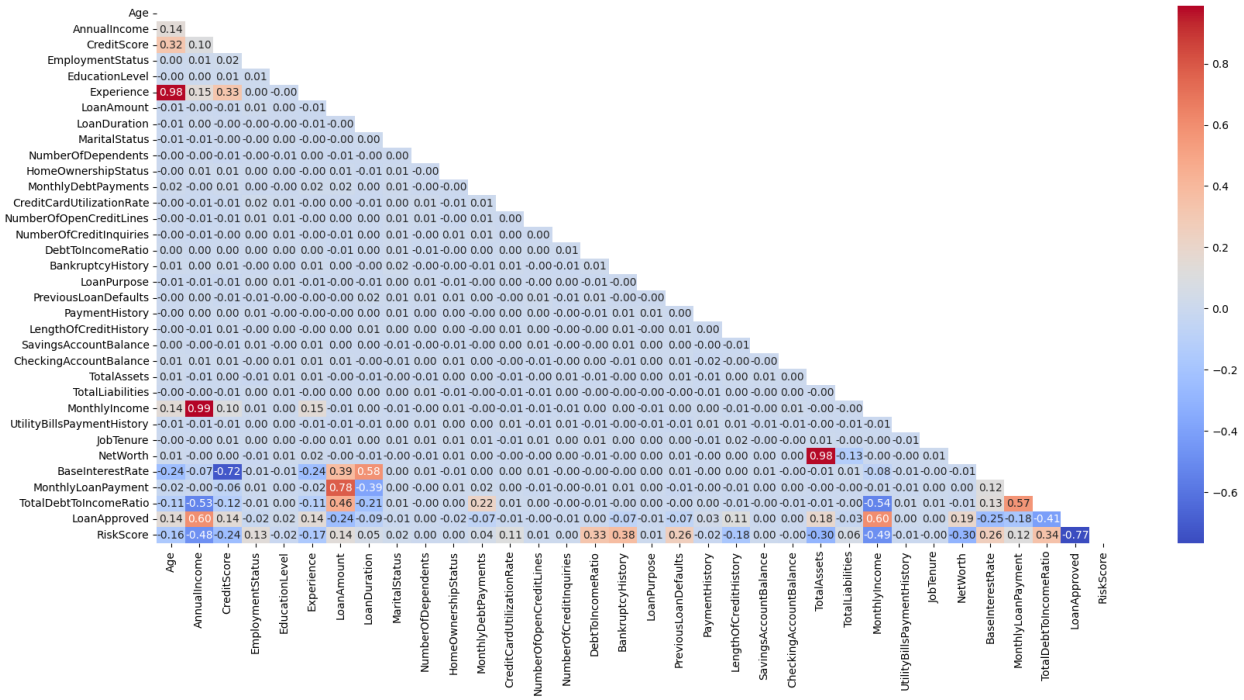

Figure 2: Correlations between all pairs of features

From the correlation matrix we see that multicollinearity exists in the data with particularly strong collinearity ($\geq$0.6) existing between these variables:

- Experience and Age
- MonthlyIncome and AnnualIncome
- BaseInterestRate and CreditScore
- MonthlyLoanPayment and LoanAmount
- NetWorth and TotalAssets
- LoanApproved and MonthlyIncome
- RiskScore and LoanApproved

Since our dataset has a large number of variables, we proceeded to conduct dimension reduction in all our models to handle the multicollinearity.

3 Classification Models

3.1 Random Forest

The target variable for classification is *LoanApproved*, while all other features in the dataset serve as inputs for the decision nodes. To optimize the model's performance and minimize prediction errors, we conducted hyperparameter tuning. Specifically, for the following hyperparameters:

- `n_estimators`: Number of trees in the forest.
- `max_depth`: Maximum depth of each tree.
- `min_samples_split`: Minimum samples required to split a node.

- `min_samples_leaf`: Minimum samples required to be at a leaf node.

Using `GridSearchCV` for hyperparameter tuning, we determined the optimal values to be `n_estimators = 100`, `max_depth = None`, `min_samples_split = 5`, and `min_samples_leaf = 1`. After adjusting these parameters and performing cross-validation, the test accuracy increased to 98.98%, indicating improved model performance on unseen data, with an F1 score of 99% and 98% for "Not Approved" and "Approved," respectively.

3.2 XGBoost

XGBoost (Extreme Gradient Boosting) is a machine learning algorithm that iteratively builds models, using decision trees as base learners. It enhances the traditional gradient boosting framework with regularization, scalability, and parallelization, making it highly efficient for regression and classification tasks (Chen, Guestrin, 2016). XGBoost optimizes a combined objective function, which includes a training loss and a regularization term. The objective function is given as:

$$O = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{T} \Omega f_k$$

where:

- $L(y_i, \hat{y}_i)$ is the loss function that measures the difference between true and predicted values.
- $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \|w\|^2$ is the regularization penalty term.

The regularization term handles model accuracy and prevents overfitting with γ controlling the penalty for adding additional tree leaves and λ penalizing large leaf weights. To optimize the objective function, XGBoost applies a second-order Taylor expansion on the loss function:

$$L(y_i, \hat{y_i}^{t-1} + f_t(x_i)) \approx L(y_i, \hat{y_i}^{t-1}) + g_t f_t(x_i) + \frac{1}{2} h_t f_t^2(x_i)$$

- $g_i = \frac{\partial L}{\partial \hat{y}_i}$ is the gradient of the loss function.
- $h_i = \frac{\partial^2 L}{\partial \hat{y_i}^2}$ is the second-order derivative.

Each tree is built greedily by selecting splits that maximize the gain in the objective. The gain for a split at node j is:

$$Gain = \frac{1}{2}[\frac{G^2 L}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}] - \gamma$$

where:

- G_L, G_R are the sum of gradients for the left and right child nodes respectively.
- H_L, H_R are the sum of Hessians for the left and right child nodes.

The tree then stops growing when the gain is below a threshold or when a maximum depth is reached. XGBoost then uses L1 and L2 regularization to improve generalization. After adding a tree, XGBoost scales its contribution by a learning rate η, which prevents overfitting by slowing down the learning process:

$$\hat{y_i}^t = \hat{y_i}^{t-1} + \eta f_t(x_i)$$

In our XGBoost model, we optimised the process to minimise costs associated with misclassification. We tuned the hyperparameters using grid search with 5-fold-cross validation on the training data:

- `max_depth`: Specifies the maximum depth of a tree. It controls the complexity of the model by limiting the number of nodes in the tree.
- `learning_rate`: Controls the step size at each iteration while moving toward a minimum of the loss function.
- `gamma`: Specifies the minimum reduction in loss required to create a new split.
- `reg_lambda`: Controls the L2 regularization term on weights. L2 regularization penalizes large weights, helping to avoid overfitting.
- `scale_pos_weight`: Balances the positive and negative weights, especially when the classes are imbalanced.

The optimal hyperparameter values upon completing Grid Search were `max_depth = 2`, `learning_rate = 0.5`, `gamma = 0`, `reg_lambda = 2`, and `scale_pos_weight = 1`.

For performance metrics, we used accuracy.

Model	**Accuracy**
Random Forest	0.71
XGBoost	0.99

Table 1: Performance metrics for Random Forest and XGBoost with hyperparameter tuning

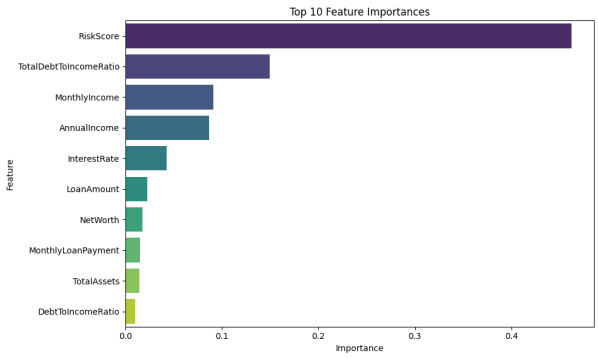

Figure 3: Top 10 feature importances using Random Forest

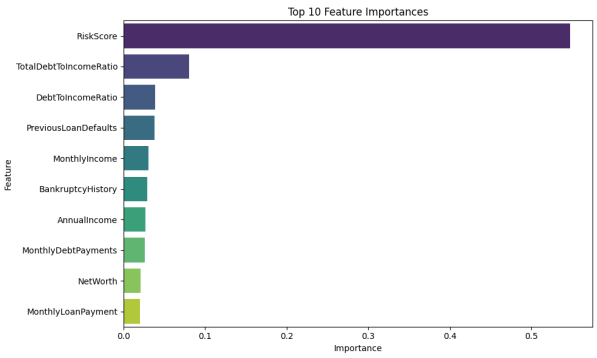

Figure 4: Top 10 feature importances using XGBoost

The two images present the top 10 feature importances derived from two distinct models: Random Forest and XGBoost. While both models identify overlapping key features, there are variations in their rankings and the relative importance assigned to certain features, highlighting the differences in how each model evaluates feature relevance.

3.3 Neural Networks

We used a neural network with a sigmoid activation function in the last layer to perform binary classification. However, we modified the loss function using the loan amount associated with the data point. This is to minimize the net dollar loss due to misclassification.

To make the model viable, we looked at data distribution and found that approximately 74% of the data had loans not approved. Moreover, the data was highly skewed to have most higher amounts of loans not approved. Thus, fitting a model on this data would result in a very good naive classification of all loans being not approved. Thus, we first removed outliers of loan amount and did oversampling to fix the issue of class imbalance.

The accuracy of a neural network with such data is approximately 62%. We do not report the loss value as after scaling and transformation, the custom loss is uninterpretable. All hidden layers used the *tanh* activation function and had 128 neurons with the dropout probability of a connection as 0.3 to prevent overfitting.

Figure 5: Training and Validation Loss in Neural Network

4 Regression Models

4.1 Principal Component Regression

Principal Component Regression (PCR) is a regression model that uses principal components as regressors. It involves performing Principal Component Analysis (PCA) in which the optimal number of components is derived through k-fold cross-validation and subsequently performing regression using the principal components as predictors. PCA allows us to handle the multicollinearity in our data as the uncorrelated linear combination of original variables is used as regressors instead of the original variables which are correlated. Furthermore, dimension reduction reduces the complexity of our data and prevents overfitting.

The optimal number of components derived from the lowest 5-fold cross-validation error was 30. The explained variance and cumulative explained variance graphs were plotted as seen below.

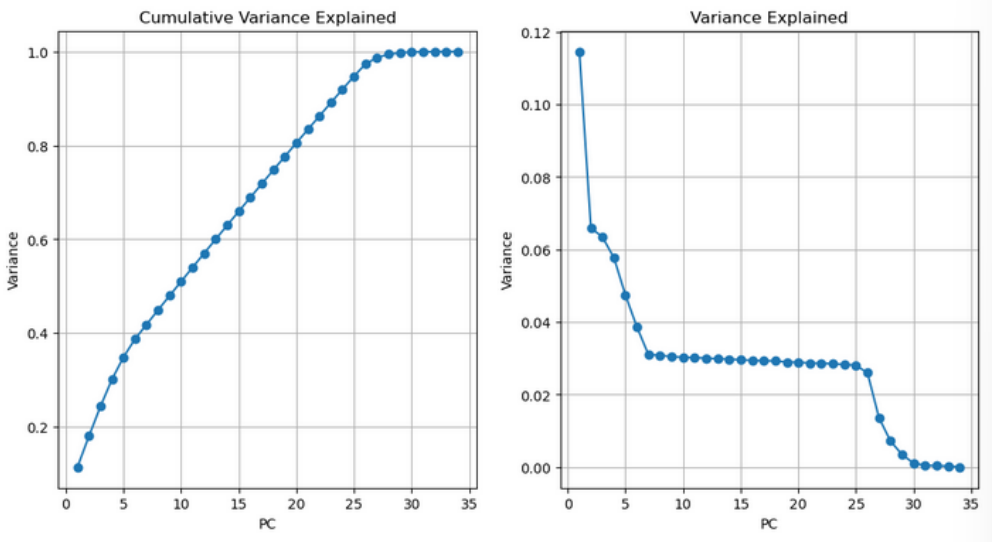

Figure 6: Explained Variance and Cumulative Explained Variance by each principal component

From the cumulative explained variance plot, we observed two other kinks at PC=6 and PC=25. Hence, we performed regression using PC=6 and PC=25 to determine which model would be the best fit for our data. The results derived were:

Number of Principal Components	Mean Squared Error (MSE)	R^2
6	0.001256	0.300392
25	0.000866	0.517592
30	0.000498	0.722613

Table 2: Summary of results for models using varied number of principal components

The model using 30 principal components had the lowest MSE and highest R^2 value.

4.2 Linear Regression with Autoencoders

Autoencoders are used as an alternative method of dimension reduction. The purpose of this emphasis on dimension reduction is to attempt to capture some variables that contribute to the variance of loan approval the most.

Output Dimension of Encoder	Mean Squared Error (MSE)	R^2
6	0.0013	0.2782
25	0.0006	0.6552
30	0.0005	0.6921

Table 3: Summary of results for autoencoders

Autoencoders essentially are neural networks, forced to learn the most information about the given data points through a reconstruction loss function (commonly the mean squared error) while making one of the middle layers with a small number of hidden neurons. The decoder part, after the small dimension layer, is symmetric in terms of several neurons compared to the first half where the line of symmetry is in the "squeezed layer".

The reconstruction loss for Mean Squared Error (MSE) can be written as:

$$\text{Reconstruction Loss} = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$$

where:

- n is the number of samples,
- $\mathbf{x}_i$ is the i-th input sample,
- $\hat{\mathbf{x}}_i$ is the reconstructed output for the i-th input, and
- $\|\cdot\|^2$ represents the squared Euclidean norm.

The entire Neural Network, including the encoder and the decoder, is trained based on the reconstruction loss. The method is similar to gradient descent. After a stable value of loss is reached, we take the weights and structure of the encoder part and use it on the original data after data scaling to get an output in reduced dimensions.

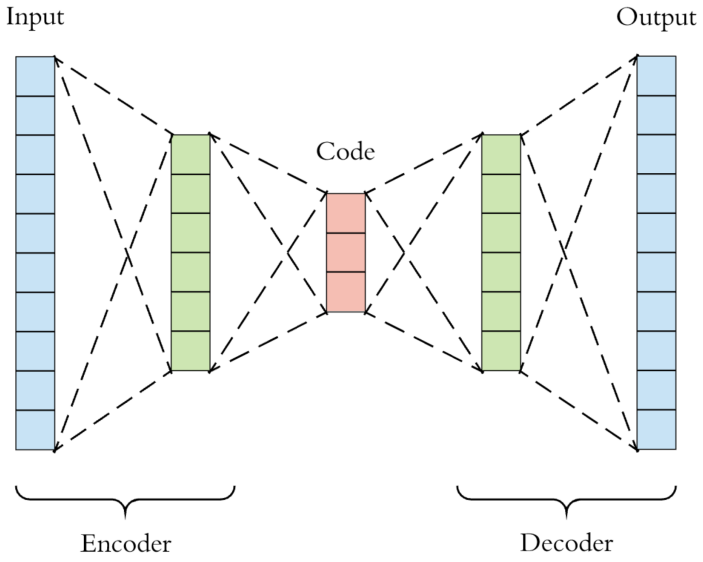

Figure 7: Sample Autoencoder

4.3 Neural Networks

In this section, we explore several linear neural network architectures to predict interest rates. Each model was designed to handle the complexities of the dataset, incorporating best practices such as dropout for regularization, ReLU activation for non-linear transformations, and tuning of hyperparameters to optimize performance.

Here is an overview of the results obtained for the various architectures:

Model	Mean Squared Error (MSE)	R^2	Collab Training Time (Mins)
Basic NN	0.000594	0.68668	0.26
Basic NN + PCA	0.0005683	0.69879	28
Deeper NN + PCA	0.0007707	0.57062	38
Wider NN + PCA	0.0007643	0.57419	50
NN with Auto-Encoders	0.0008100	0.54873	40

Table 4: Summary of results for Neural Networks

The first architecture used was a **Basic Neural Network in Keras**.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	2,368
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 1)	33

Figure 8: Model architecture for Basic Neural Network

The optimal number of principal components selected from 5-fold cross-validation with the lowest RMSE was 25.

For the Final Keras Neural Network with Optimal PCA Components, we obtained MSE = 0.0005783475107011066 and R^2 = 0.6777972111678993.

Hence, the basic neural network seems to yield good results when conducted with optimal principal components and appears to be a good fit.

The next architecture implemented was a **Deeper Neural Network**.

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 128)	2,688
dropout_3 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 64)	8,256
dropout_4 (Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 32)	2,080
dropout_5 (Dropout)	(None, 32)	0
dense_7 (Dense)	(None, 1)	33

Figure 9: Model architecture for Deep Neural Network

The optimal number of principal components selected from 5-fold cross-validation with the lowest RMSE was 28.

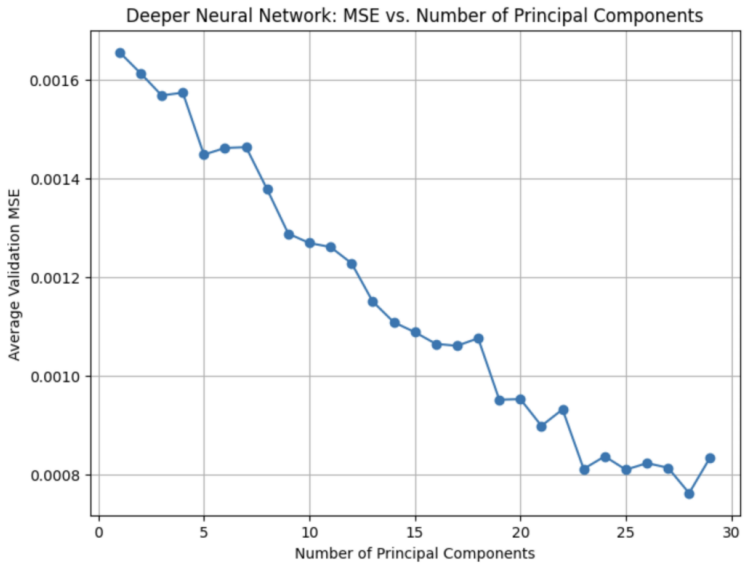

Figure 10: Optimal principal components for Deeper Neural Network

For the Keras Deeper Neural Network with Optimal PCA Components, we obtained, MSE = 0.000770731604587545 and R^2 Score = 0.5706182393037283.

The deeper neural network seems to yield decent results when conducted with optimal principal components but still has a lower R^2 than the basic neural network, which is less computationally intensive.

The third architecture used was **Wider Neural Network**.

This model was used as an attempt to learn richer feature representations. We theorised that the increased number of neurons in the hidden layers might provide more capacity for capturing complex patterns while preserving dropout would ensure the model remains generalizable.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	7,680
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 1)	65

Figure 11: Model architecture for Wide Neural Network

For the Wider Neural Network with Optimal PCA Components, we obtained MSE = 0.0007643171354712071 and R^2 Score: 0.5741917998359697.

The wide neural network seems to yield similar results when conducted with optimal principal components, but in addition to taking much more time, it still has a lower adjusted R^2 value than the basic neural network.

Lastly, we used **Neural Networks with Auto-Encoders**.

Aside from PCA, we used autoencoders as another dimensionality reduction technique. Autoencoders would be able to capture more complex patterns than linear methods such as PCA. We conducted a 5-fold cross-validation and selected the optimal encoding dimensions with the lowest MSE.

For hyperparameter tuning, the autoencoder was tested with different encoding dimensions (5, 10, 15, 20, 25, 30) to find the optimal balance between compression and information retention.

We use cross-validation with the lowest MSE to select the optimal encoding dimension of 20.

For the Auto-Encoder and Neural Network with Optimal Encoding Dimension architecture, we obtained MSE = 0.0008100132591576224 and R^2 = 0.5487340634090747.

5 Conclusion

5.1 Classification of Loan Approvals

In conclusion, while both XGBoost and Random Forest demonstrate high accuracy, Neural Networks offer a prospective advantage due to their customizability in defining loss functions, which is critical in our context. This flexibility allows us to prioritize minimising incorrect approvals of large loan amounts over ensuring the correct approval of smaller loans, aligning better with our project objectives.

5.2 Prediction of Interest Rate

Among all the models implemented, PCR seemed to yield the best adjusted R^2 and MSE. However, the basic neural network with principal components also showed similar results, with it only being slightly less accurate than PCR.

Hence, we find that different iterations and permutations of our neural network implementation could be able to achieve better results. With more time and computational resources, a grid-search algorithm can be executed to properly identify the optimal hyperparameters, such as the number of hidden layers or neurons per layer.

Although PCR yielded slightly better results, we find that the neural network approach shows great potential, especially for deciphering our data which tends to have a lot of noise as evidenced by the scatterplot in Figure 1.

6 References

- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.
- Zoppelletto, L. (2024, September 7). Financial Risk for Loan Approval. Kaggle. `https://www.kaggle.com/datasets/lorenzozoppelletto/financial-risk-for-loan-approval/data?select=Loan.csv`
- Saha, A. (2018, October 9). *Applied Deep Learning - Part 3: Autoencoders.* Towards Data Science. Retrieved from https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798