Loan Default Detection

By Vidya Velusamy, Praise Satyagrahi, and Troy Flood

The Objective

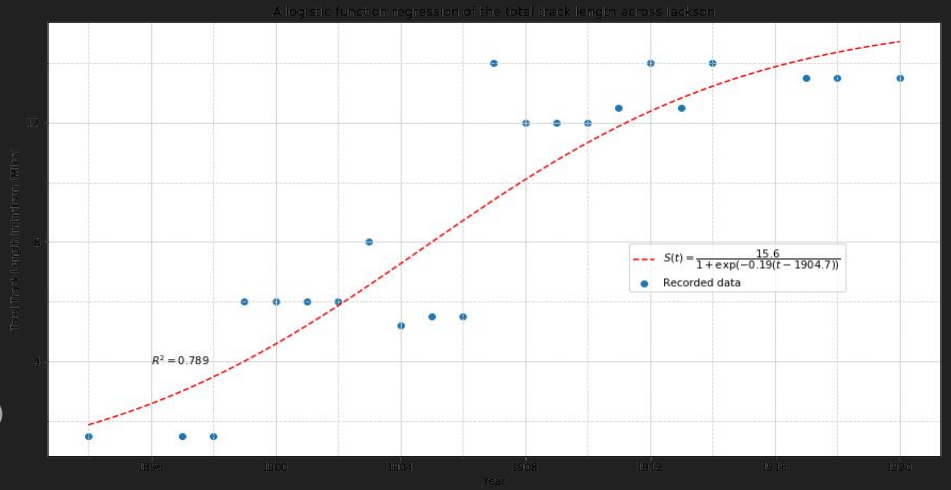
- By analyzing bank data we attempted to make a model that could predict the likelihood of a loan default.
- The Data was sourced from a Kaggle article “Loan Defaulter” from Gaurav Dutta - ("https://www.kaggle.com/datasets/gauravduttakiit/loan-defaulter?select=application_data.csv"
- This Data had a target column for whether a loan defaulted or not. It also included other features like how many children the loan holder has. Whether or not the holder has a house, or a car. Many features were considered in the model.
- In this presentation we will outline the methods for data cleaning, the different models we attempted, and the methods we tried to make it work properly.

The Data Cleaning/Normalization

- Created a Pandas dataframe from the CSV file.
- Performed cleaning on the data such as column dropping and hot encoding for categorical data.
- Split the data into training and testing sets.
- Created target and feature variables.
- Ran data through standard scaler.
- Fit and transformed data to prep for model.

Logistic Regression

- A statistical method used for binary classification problems.
- Predicts the probability of a categorical dependent variable based on one or more independent variables.
- The model correctly predicts the outcome 54%.
- This means that 46% of the predictions are incorrect.
- A 54% accuracy indicates potential issues with the model or the data.



(image not related to data)

Kerras Model

Building and Training a Keras Model

Runs on top of TensorFlow,

More complex models

parameters

Layers: 2 layers

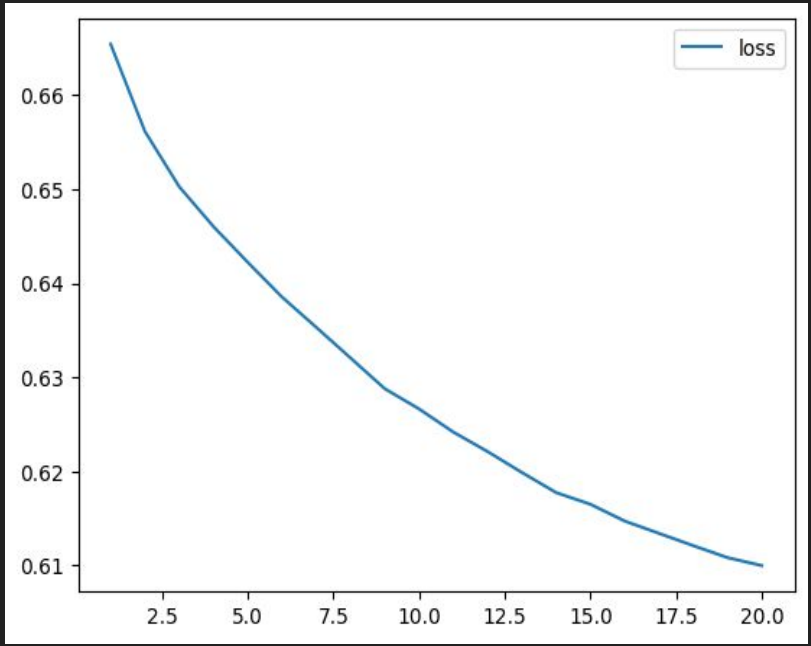
Units: 64,32

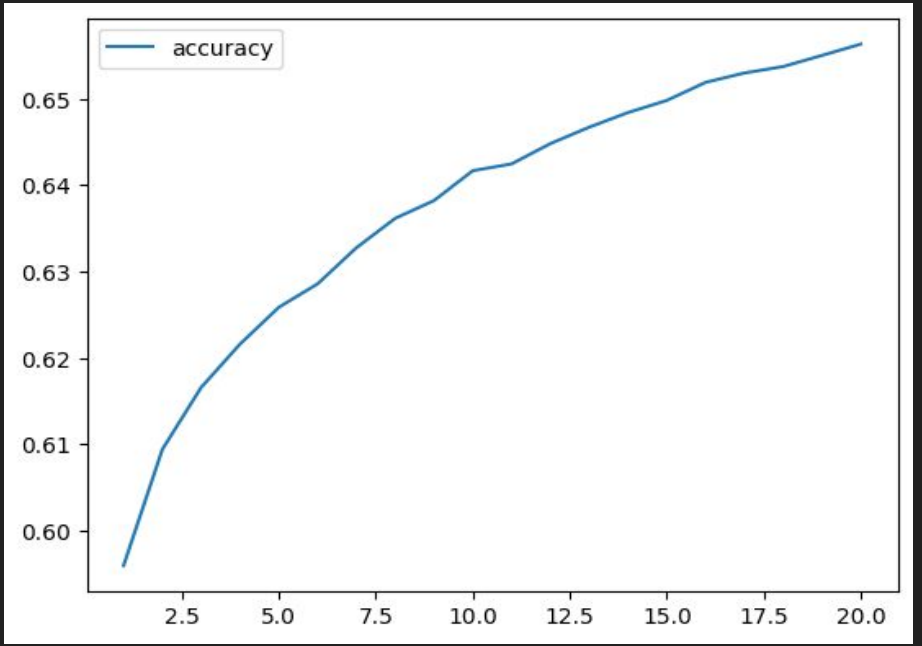
Epochs:20

Loss: 0.6172783970832825, Accuracy: 0.6477522253990173

The model achieved a loss of 0.6173 and an accuracy of 64.78% after training.

Accuracy and Loss Plots





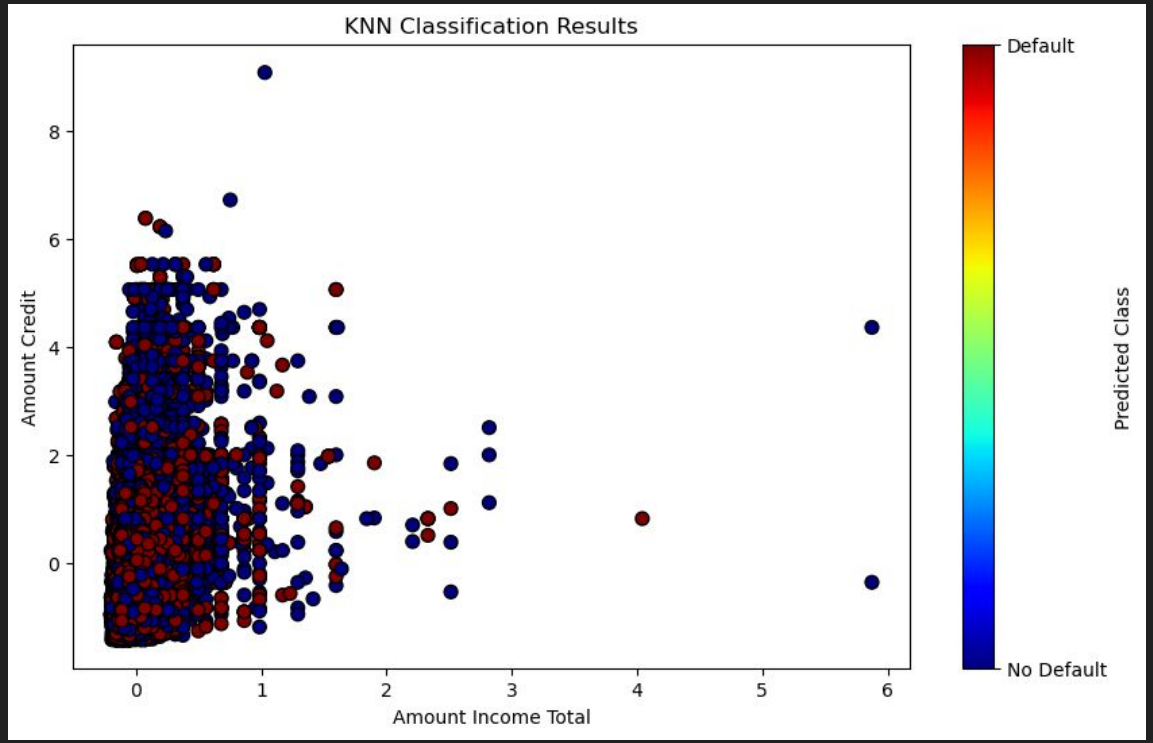
KNN Model

- Trained and tested this model for predictions using the training and testing data.
- The precision for the 0's were 0.99 and for the 1's, it was 0.83.
- Overall accuracy score was 0.90.
- Used the RandomOverSampler by installing the imbalanced-learn library to get a better accuracy score.

KNN MODEL CLASSIFICATION REPORT

	precision	recall	f1-score	support
0	0.99	0.80	0.89	38623
1	0.83	0.99	0.90	38320
accuracy			0.90	76943
macro avg	0.91	0.90	0.90	76943
weighted avg	0.91	0.90	0.90	76943

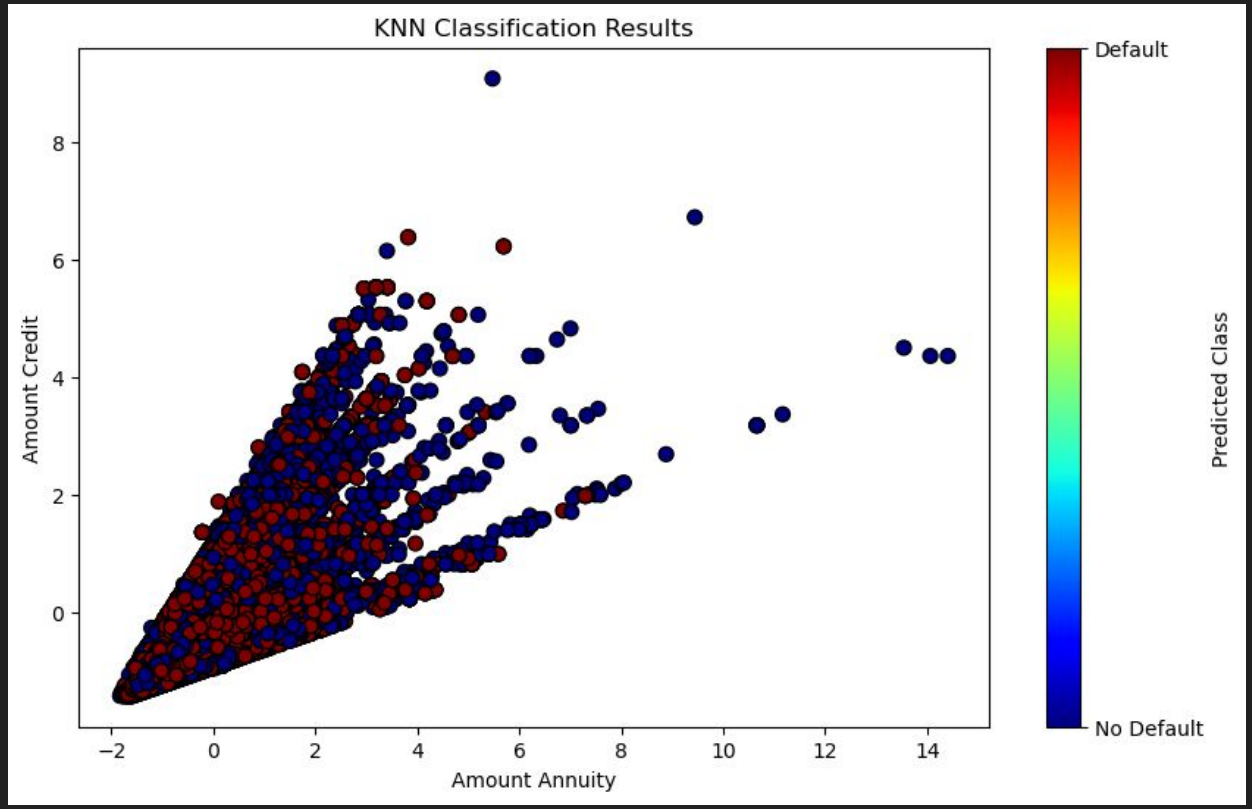
KNN Scatter Plot



- Blue: No Default
- Red: Default
- This scatter plot represents if someone will default or not based on their income total and the amount of credit they want.

KNN Scatter Plot 2

- Blue: No Default
- Red: Default
- This scatter plot represents if someone will default or not based on the annuity amount and the credit amount.



Random Forest Model

- Fit the data to the random forest model.
- Make predictions using the SKlearn Random Forest library.
- This model produced acceptably accurate results and was perfect on True Positives and False Negatives.
- This required Random Over Sampling in order to prevent the model from prediction “0” every time.

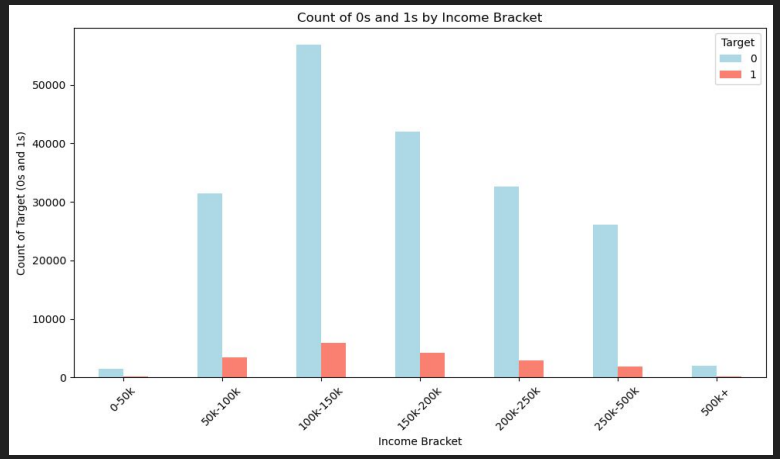
Accuracy Score : 0.9739807389886019

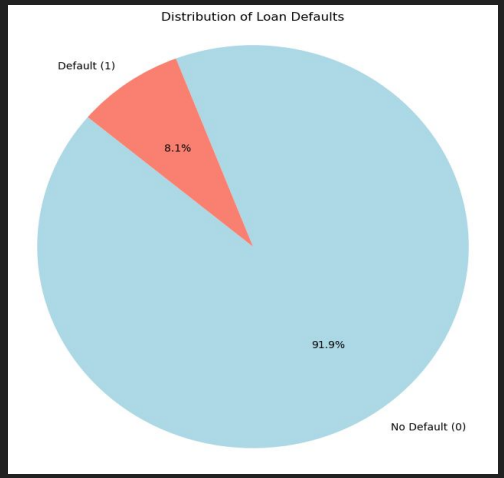
Classification Report

	precision	recall	f1-score	support
0	1.00	0.95	0.97	38623
1	0.95	1.00	0.97	38320
accuracy			0.97	76943

Issue We Faced

- Initially, our models were unanimously giving us 91% accuracy, and poor score on False Negatives.
- The model had particular trouble correctly predicting False Negatives.
- By charting the original data we were able to notice that 91% of the data was result 0 or non default, the exact same as our model score.





The Solution

- We tried many things that did not work. The first being stratification. This slightly improved the score on False Negatives but not enough to make any model acceptable.
- We determined the top 5 and 10 columns weighing on the model and isolated the model to just those columns. Again slight improvement, but not enough.
- Finally, we tried Random Under and Random Over Sampling. This brought overall accuracy up 6 points and greatly lowered False Negative inaccuracy.

Data Loading

Inserting One-Hot Encoded Data into a SQL Database

One-hot encoding is crucial for preparing categorical data for machine learning.

SQL databases can efficiently store one-hot encoded data.

Direct integration with libraries like SQLAlchemy simplifies data handling.

Conclusion

- For our purposes, only the KNN and Random Forest Models would be acceptable.
- Kerras and Logistic Regression did not correct even after over sampling.
- Given the performance of KNN and Random Forest, we recommend focusing on these models for future iterations. Further tuning of hyperparameters could enhance their accuracy even more. Models like Logistic Regression and Keras will require significantly more adjustments or may not be suitable for our current task.