Predicting Loan Approval Using Naive Bayes and Logistic Regression

By Tevin Parathattal
December 11, 2024
ITCS 3156: Introduction to Machine Learning

1 Introduction

1.1 Problem Statement

The problem I hope to solve is the need for more transparency in loan approval. Numerous variables are involved in loan approval, making it difficult for consumers to understand the process and the likelihood of their approval. Through machine learning, I want to create a model that can assess loan applications with high accuracy and explainability. Borrowers could gain greater confidence and understanding of their likelihood of approval, ultimately enhancing their ability to plan and achieve their financial goals.

1.2 Motivations

Loan approval is critical because it directly affects an individual's ability to access the means necessary for significant life events. For many, this could be purchasing a home, funding education, starting a business, or buying a vehicle. The likelihood of loan approval can seem very abstract, and it can feel almost random as to why some people get approved and some don't. The possibility of loan approval hinges on various factors, including your credit score, income, loan type, etc. Both lenders and borrowers need to understand these criteria. For lenders, evaluating loan approval is critical for minimizing risk, while for borrowers, the process often determines their ability to achieve goals that necessitate significant financial backing. Understanding the approval criteria can help individuals feel more confident when buying a home. According to Business Insider, the average American has $104,215 in debt, spanning student loans, mortgages, and credit cards (Streaks). While debt is more complicated than just loans, understanding loan approval can be crucial to understanding your finances. Machine learning can provide an avenue to tackle the complexities of loan approval.

1.3 Approach

To approach this problem, I will employ two different machine-learning algorithms: Logistic Regression and Naive Bayes. I will use a publicly available dataset on Kaggle that includes various factors and whether their loan was approved or rejected (Lo). I chose these algorithms because of their efficiency and speed. I will preprocess the data, ensuring the two algorithms can efficiently use it. Then, based on the initial results and the evaluation of the accuracy of each model, I will experiment. This could include but is not limited to, changing my approach in the data pre-processing stage by encoding categorical variables or considering methods to address the class imbalance. I will then evaluate my choice of algorithms and reflect on the efficacy. Finally, I will provide recommendations for future research into the topic and the other things that could've been done to improve the models. To look at my source code for this project, please refer to this link: https://github.com/tevinp23/MLFinalProject.

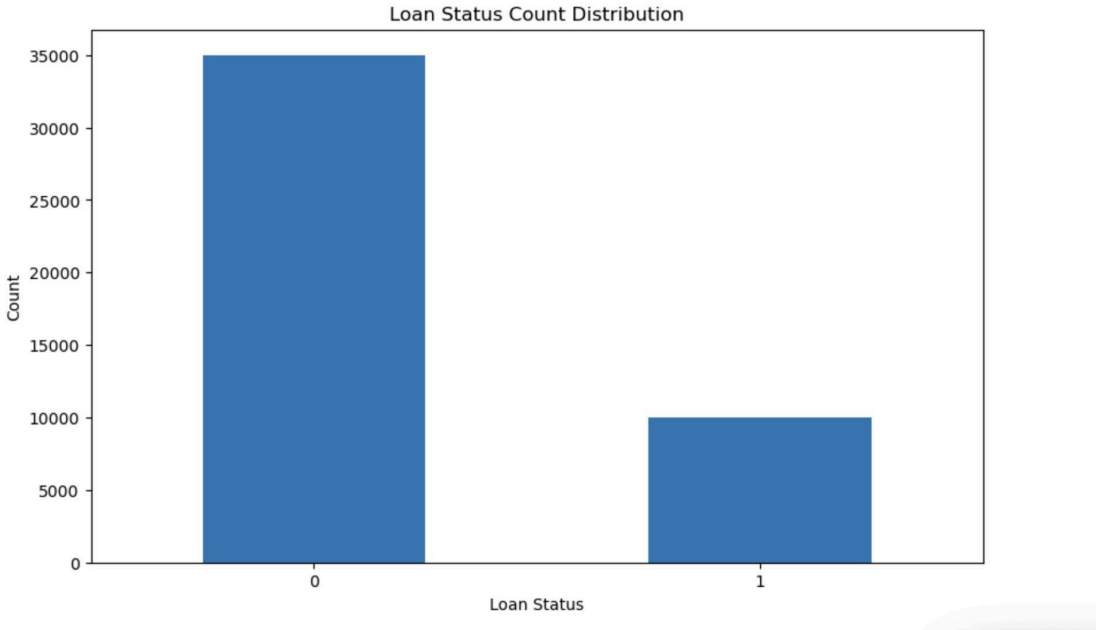
2 The Data

2.1 Introducing the Dataset

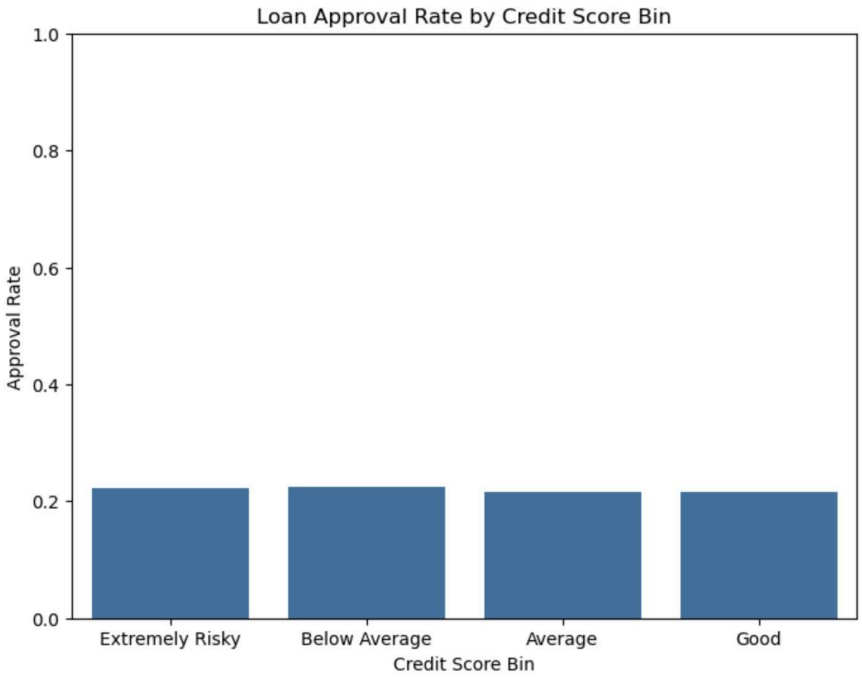
The Loan Approval Classification dataset from Kaggle is a well-structured dataset designed for binary classification tasks in financial decision-making (Lo). It includes numerical and categorical features, such as applicant income, loan amount, credit history, and education level, with the target variable indicating whether a loan application was approved. This dataset provides a practical basis for exploring the factors influencing loan decisions and evaluating predictive models in financial applications, making it perfect for this project. The dataset contains 45,000 samples and 14 features, satisfying the 10,000 sample and five feature requirements. Additionally, the dataset has no null values, so I did not have to employ any strategies to input missing values to ensure my models could handle the dataset.

2.2 Exploring the Dataset

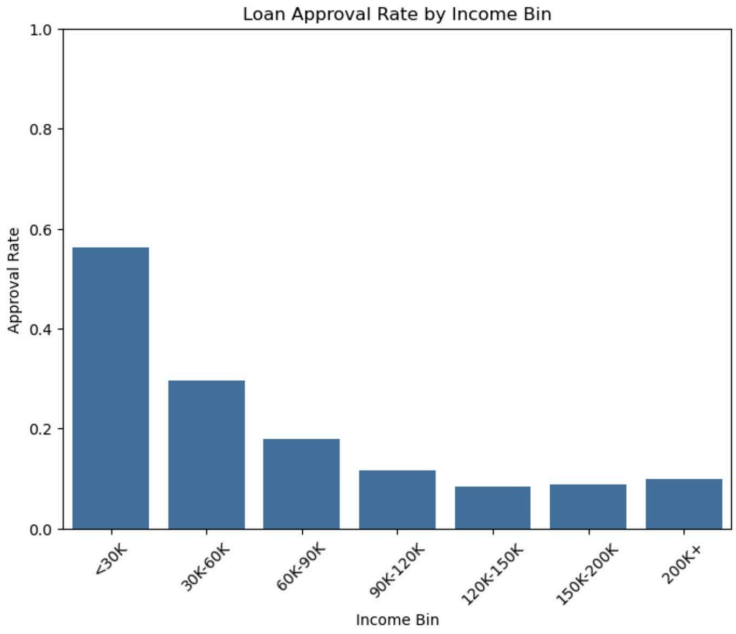
Before I got into the machine learning algorithms, it was essential to understand the dataset, and I solved this by visualizing it. The first thing I did was to see if the dataset was skewed regarding approvals. Since 2022, almost half of Americans who have applied for a loan or some financial product have been denied (Foster). This informed my decision to visualize the dataset in this way.



Based on this visualization, this dataset is skewed towards denials, which can be challenging when developing my models. Algorithms could become biased towards the majority class. I must figure out how to deal with that issue while training. Next, through my experience with financial wellness at UNC Charlotte, I've noticed that many people have preconceived notions about credit scores and how they affect a person's ability to take on debt. I separated credit scores into four categories: Extremely Risky, Below Average, Average, and Good.



Based on this visualization, the dataset is evenly distributed across credit score bins. However, we've already established that the dataset skews toward denied loans, showing that while credit scores may play a part in loan approval, they are not the only factor that goes into loan approval. That makes this problem increasingly tricky because of the factors that determine the likelihood of someone getting approved for a loan. Then, I decided to look at the situation from an income perspective. I created income score bins to see the distribution of income as well.



Based on this, most people in this dataset are relatively low-income. The visualizations have given us good information about the nature of the dataset and how the samples are distributed. Additionally, they help us understand some realities of loan approval, like how people with lower incomes often look for loans more than others because they are unable to pay for things out of pocket. However, it is essential to remember that this is just one dataset and may not indicate reality. I chose to take a more visual approach to the dataset because

3 Methods: Logistic Regression

3.1 Initial Implementation

Logistic regression is a machine learning algorithm used for classification problems. It predicts a probability that a given input belongs to a specific category. This target variable is categorical, so I believed this would be a good algorithm for this dataset. Logistic regression uses the sigmoid function to convert any real-valued number into a probability between 0 and 1. The sigmoid function is defined as:

$$\boldsymbol{\sigma}(z) = \frac{1}{1+e^{-z}}$$

Where z is the linear combination of the input features, weights, and bias, the function produces a probability. If that probability is greater than or equal to 0.5, it will predict 1. However, logistic regression doesn't deal with categorical variables. That means we must encode categorical variables like Gender, Intent of Loan, and others into numbers. I accomplished this by using one-hot encoding by using pd.get_dummies, I converted categorical variables like Gender into Male, using 0 and 1 as true and false. I also used drop_first = True to avoid multicollinearity by removing one category. An example of multicollinearity could be when I used pd.get_dummies. Instead of creating a Male and Female category, it only makes one.

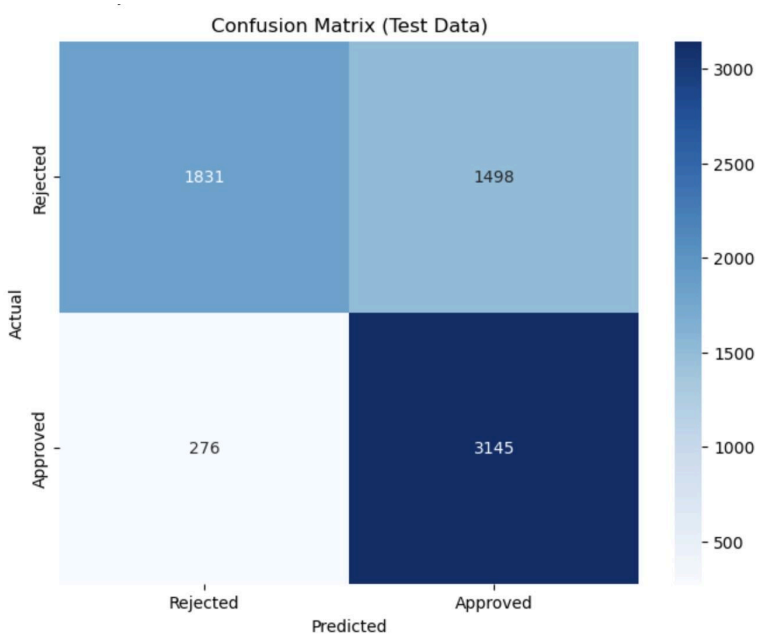
Then, I split the data into training, validation, and test sets. The training set fits the model, the validation set evaluates the performance during training, and the test set assesses the model's performance on unseen data. Then, I standardized the data using StandardScaler because logistic regression performs better with standardized data. The same scaling is applied to all three sets of data.

Then, I implemented logistic regression itself using sklearn by importing the model from the library. I used documentation from scikit-learn to help me establish logistic regression (scikit-learn). I instantiated the model with 1000 iterations for the algorithm to converge and random_state = 42. This is a pretty basic instantiation. I used model.fit to train it because it automatically finds the optimal weights and bias to minimize the log-loss function.

The accuracy score calculated the percentage of correct predictions after predicting the outputs for training and validation data. The accuracy scores for training and validation both ranged around 73-74%, indicating that this model is well-performing. It generalizes well to unseen data, and the model has achieved a good balance between bias and variance. Due to this, I chose not to fine-tune hyperparameters at this point.

3.1.1 Results of the Initial Implementation

The next thing I did was train the model on the testing set. Doing this lets me see how the model works on unseen data. The test accuracy came out at 74% as well. Then, I generated a confusion matrix and classification report to analyze the results.



Classification Report (Test Data):

	precision	recall	f1-score	support
False	0.87	0.55	0.67	3329
True	0.68	0.92	0.78	3421
accuracy			0.74	6750
macro avg	0.77	0.73	0.73	6750
weighted avg	0.77	0.74	0.73	6750

The testing model reached 74%, demonstrating an alright performance in classifying instances. The confusion matrix reveals that the model correctly identifies 1831 “Rejected” instances and 3145 “Approved” instances. However, it struggled with false positives, where “Rejected” cases were seen as “Approved.” With a recall of 92%, the model was great at identifying actual “Approved” cases. There is a lot of imbalance between the classes, indicated by the f-1 score, where “Approved” performed much better than “Rejected.”

3.2 Logistic Regression: Better Encoding

Based on my previous results, I realized I was doing a disservice to my model by taking a more lax approach to encoding. Therefore, instead of approaching encoding in a “one size fits all” approach, I approached it more intentionally with my categorical variables. I incorporated one-hot encoding(which I did in the last run) and realized that some categorical variables cannot be reduced to ones and zeroes. I used label encoding to provide a more straightforward classification. With one-hot encoding, there can be subtle inconsistencies with how logistic regression interprets the input. Label encoding provides something more manageable for the model to work with. By being more straightforward and more computationally efficient for categories like gender and previous defaults on file, I hoped this would change the results significantly.

Otherwise, my steps were the same. I split my data in the same way and then trained it. Based on my training and validation accuracy scores, my more intentional approach to encoding worked. Both training

and validation accuracy scores were around 86%, a big jump from the earlier implementation. The next thing I did was test my model on the testing data.

3.2.1 Results of Better Encoding

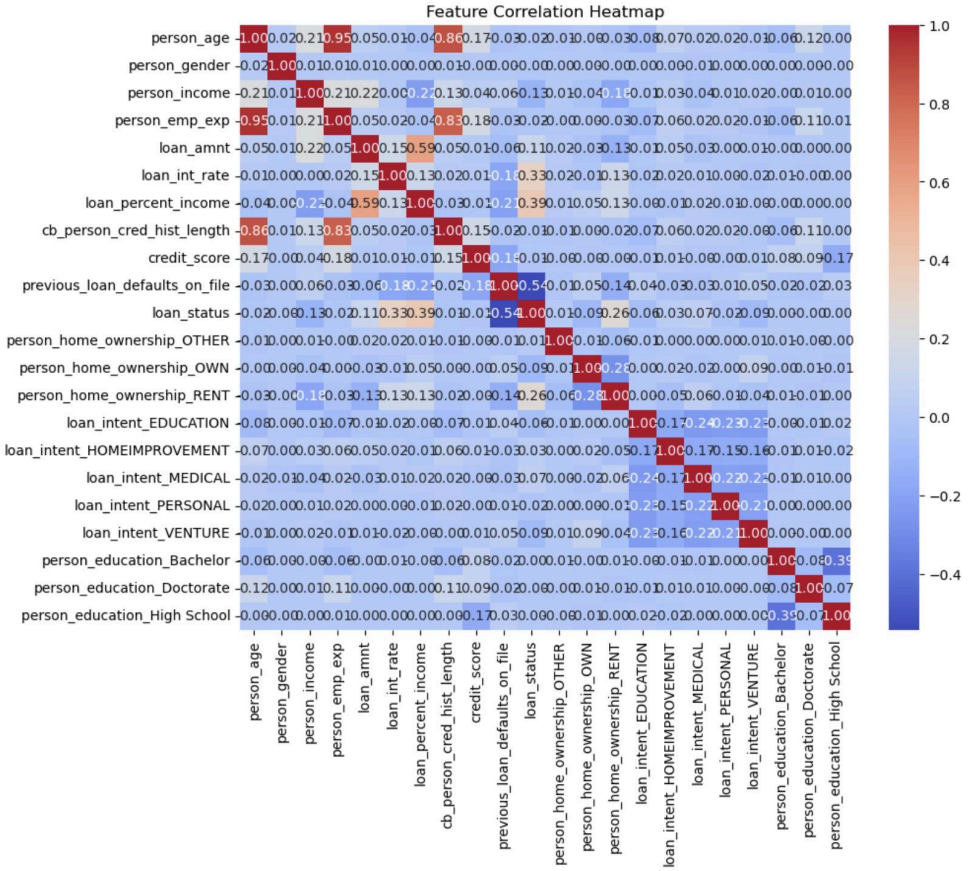
My results were much better than the earlier implementation. Testing accuracy only dropped 1% to 85%, which jumped 11% from the previous implementation. Next, I chose to generate a classification report and confusion matrix.


```
Classification Report (Test Data):
              precision    recall  f1-score   support

       False       0.85      0.99      0.92      5670
        True       0.68      0.12      0.20      1080

    accuracy                           0.85      6750
   macro avg       0.77      0.55      0.56      6750
weighted avg       0.83      0.85      0.80      6750
```

This corresponds to the model's overall test accuracy of 85%. This is primarily attributed to the high accuracy noted while forecasting instances that belong to the "Rejected" class. From the confusion matrix, the model correctly classifies 5612 rejected instances as true negatives, while for approved instances, it classified them correctly in 125 instances, misclassified 955 as rejected, and only 58 of rejected instances as approved. This is reflected in the classification report, where the "Rejected" class has a high precision of 0.85 and an exceptional recall of 0.99, resulting in a strong F1-score of 0.92. However, the "Approved" class performs poorly, with a precision of 0.68, recall of 0.12, and an F1-score of 0.20, indicating that the model struggles to identify "Approved" instances correctly. While the overall model was much better, it struggled with true instances.

3.3 Logistic Regression: Correlation

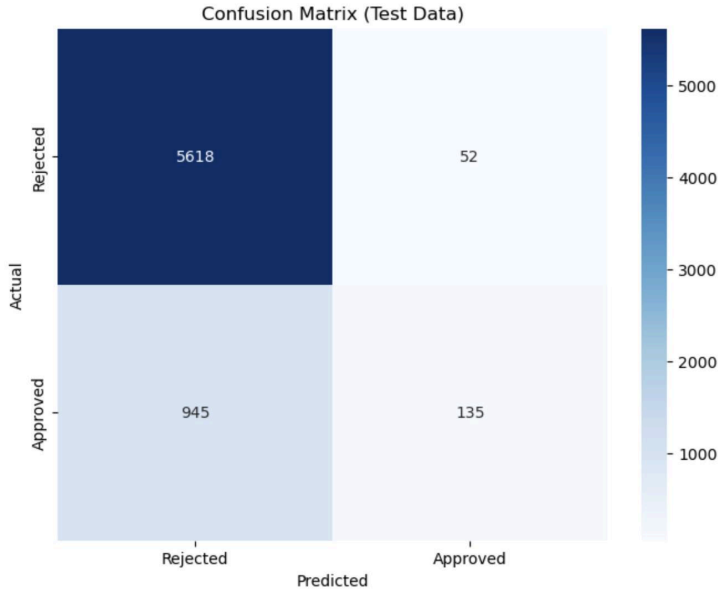
Variables can often be redundant. For example, the correlation between a person's age and years of employment is often very positive. To improve my model, I created a correlation matrix and dropped anything with a correlation of over 0.2.



I chose 0.2 because, as I will explain in my results, this provided the best improvement. The highly correlated features were person_income', 'person_emp_exp', 'loan_amnt', 'loan_percent_income', 'cb_person_cred_hist_length', 'loan_status', 'person_home_ownership_RENT.' I created a reduced dataset and then reran the same tests. I split the data in the same proportions and ran the tests. My training and validation accuracy both stood around 86%. This was about the same as last time.

3.3.1 Results from Dropping Highly Correlated Categories

After running my new model, it didn't perform significantly better. It had an accuracy of 85%, specifically 85.23%, which was a slight improvement but negligible. However, slight modifications were made to the precision of the "True" sections.



Classification Report (Test Data):

	precision	recall	f1-score	support
False	0.86	0.99	0.92	5670
True	0.72	0.12	0.21	1080
accuracy			0.85	6750
macro avg	0.79	0.56	0.57	6750
weighted avg	0.83	0.85	0.81	6750

The logistic regression model achieved a test accuracy of 85%, reflecting its strong ability to classify most instances correctly. The confusion matrix shows that the model correctly identified 5618 Rejected instances as such (True Negatives) and 135 instances as Approved (True Positives). However, it misclassified 945 Approved instances as Rejected (False Negatives) and only 52 Rejected instances as Approved (False Positives). This is further reflected in the classification report, which shows perfect results for the "Rejected" class with a precision of 0.86, recall of 0.99, and F1-score of 0.92, whereas the class "Approved" has considerably poor metrics of precision, recall, and F1-score with values of 0.72, 0.12, and 0.21, respectively. The model simply fails to classify "Approved" examples correctly.

This indicates that the model isn’t good enough because while overall performance has increased, it simply isn’t good enough to be a genuine model. At the end of the paper, I will reflect on this model's performance and the experiments performed, but overall, the model is flawed.

4 Methods: Gaussian Naive Bayes

4.1 Initial Implementation

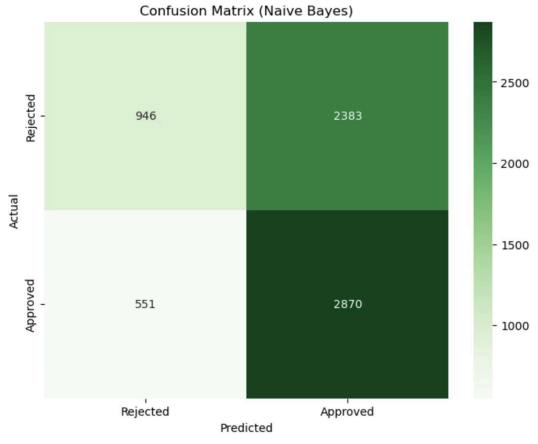
Naive Bayes is a machine learning algorithm that is built for classification problems like this one. It is a probabilistic algorithm based on Bayes' Theorem, which calculates a class's probability given the data's features. It automatically assumes all features are independent of each other. This is why it is called "naive." Despite the unrealistic assumptions, it often performs well in real-world scenarios. Additionally, with my background in math, Bayes' Theorem is crucial to the world of probability, and I was curious to see how it performed with this dataset. It's fast, easy to use, and works exceptionally well with large datasets like this one, but performance might suffer if features are highly correlated. I chose not to approach from the correlation perspective because when doing the same with logistic regression, I noticed very few categories were truly correlated. Additionally, correlation did not seem to be the most pressing issue I was currently dealing with.

In this implementation, the Naive Bayes algorithm was applied to classify data from the dataset. The data was first preprocessed using one-hot encoding (pd.get_dummies with drop_first=True) to convert categorical features into binary columns. The dataset was then split into training (70%), validation (15%), and test (15%) subsets. The Naive Bayes model was trained on the raw, unscaled training features (X_train, y_train). I did this because Naive Bayes doesn't need feature scaling.

Then, I implemented Naive Bayes itself using sklearn by importing the model from the library. I used documentation from scikit-learn to help me establish Gaussian Naive Bayes(scikit-learn). This model was simple, and I didn't do much editing besides encoding. The accuracy score calculated the percentage of correct predictions after predicting the outputs for training and validation data. The accuracy scores for training and validation ranged around 57%, meaning the model wasn't very good. I chose to see if there was a drop-off once the dataset saw the test set instead of reevaluating my approach just yet.

4.1.1 Results

The results were overall disappointing. The model didn't perform well at all, performing at around 57% accuracy, which is in line with the accuracy of the training and validation sets. As usual, I created a classification report and a confusion matrix to help me analyze the model's results more deeply than its overall accuracy.



```
Classification Report (Naive Bayes):
              precision    recall  f1-score   support

       False       0.63      0.28      0.39      3329
        True       0.55      0.84      0.66      3421

    accuracy                           0.57      6750
   macro avg       0.59      0.56      0.53      6750
weighted avg       0.59      0.57      0.53      6750
```

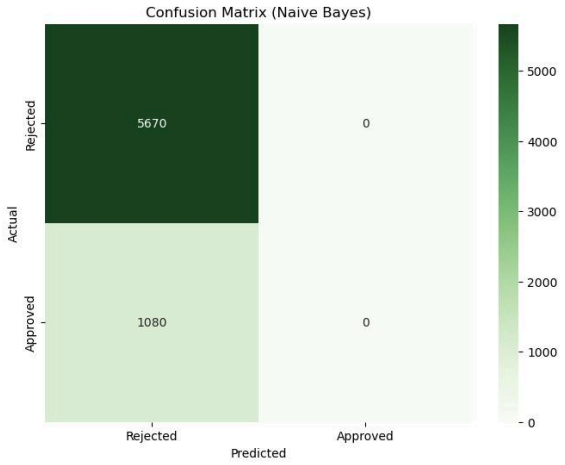
Based on these results, we can take away some positives and negatives from this. While our overall accuracy wasn't good, the recall for our "True" section was 84%, which is pretty good. Of all the actual positive instances, our model was able to predict 84% of them. Compared to the earlier logistic regression model, this is a marked improvement. However, every other category struggled significantly. Of the actual false instances(ones that were rejected), the model could only predict 28% of them. This model needs significant improvement, so I experimented with it similarly as I did logistic regression.

4.2 Encoding

With logistic regression, I saw a massive increase in performance due to using more intentional forms of encoding. As previously mentioned, this dataset has multiple categorical variables, and by taking a more deliberate approach to encoding, we can create a better model. Similarly to earlier, I made sure to categorize specific categories with Label Encoder, using the label encoding library, and with others I used one-hot encoding. Otherwise, my steps were the same; I split the dataset, and my training and validation accuracy scores were positive. Using better encoding, the model's overall accuracy increased from 74% to 85%. This was encouraging, and similarly, I utilized classification reports and the confusion matrix to evaluate the model's performance on the test data.

4.2.1 Results of Naive Bayes Better Encoding

The results were interesting, to say the least. Some things about it were amazing, and others not so much. To evaluate the model effectively, the classification report is crucial as it provided massive insights into why the model performed so much better, and why its not all good.



```
Classification Report (Naive Bayes):
              precision    recall  f1-score   support

       False       0.84      1.00      0.91      5670
        True       1.00      0.00      0.00      1080

    accuracy                           0.84      6750
   macro avg       0.92      0.50      0.46      6750
weighted avg       0.87      0.84      0.77      6750
```

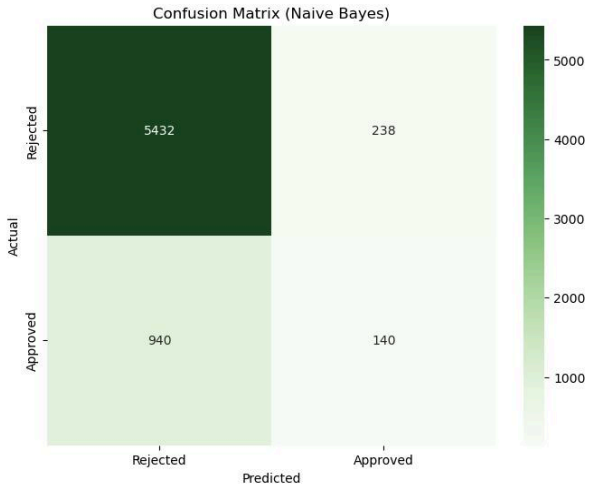
The model excels at identifying "False" cases, achieving high precision and recall. However, it completely fails to identify "True" cases, resulting in a low F1 score for this class. This model seems to have an apparent bias towards the majority class, something that might be fueled by the more “False” instances in the dataset. The model performs well overall, but its practical applications are likely limited due to its inability to figure out if anything is false. The 1.0 precision for “True” is extremely misleading because precision measures false positives. When an instance is predicted as “True,” is it true? As we can see, in this case, the model failed to identify true instances at all. I believe that the next step to trying to balance this out is looking at class imbalance.

4.3 Addressing Class Imbalance

Class imbalance is an issue in this scenario and I approached this situation by using priors. Based on our earlier data visualizations, we know that there are significantly fewer instances than the other, so by setting appropriate priors, which are probabilities, that represent the initial belief or estimate of each class occurring, I can try to improve the model and address class imbalance. It can help give more weight to the minority class in this scenario. After editing the initialization of the models by inserting priors=[0.75,0.25], which essentially says that I believe that the “False” class is roughly 75% of the dataset, I subsequently trained my model. I chose 75% through various experiments. I started at around 60% and chose to increase it until I hit the spot where increasing it further worsened the model's accuracy. I followed the same steps as before to split my dataset into training, validation, and testing sets. My training and validation scores got slightly worse, dropping to around 83%. However, this doesn’t necessarily tell the whole story. That’s why it's important to analyze the results using a classification report and confusion matrix.

4.3.1 Results of Addressing Class Imbalance

My results might’ve been worse overall; however, many factors go into evaluating a model's accuracy, which is why the classification report is so important. Our earlier iteration of the model had an 84% accuracy rate, but performed horribly on “true” instances. Now, it’s time to see if it did the same thing this time.



Classification Report (Naive Bayes):

	precision	recall	f1-score	support
False	0.85	0.96	0.90	5670
True	0.37	0.13	0.19	1080
accuracy			0.83	6750
macro avg	0.61	0.54	0.55	6750
weighted avg	0.78	0.83	0.79	6750

Overall, the results aren't amazing. While in this iteration, the model had some success with its ability to analyze "true" instances, the F1 score was only 0.19. While it wasn't as skewed as before, it still isn't an objectively good model. While addressing the class imbalance in other ways may help improve the model, it's more likely that Naive Bayes is simply too limited for this dataset.

5 Comprehensive Results Analysis

Overall, the models created had decent accuracy. They both performed around 84% by the end of the experimentation, but they both had deep flaws. In the logistic regression model, after dropping correlated features, we ended up with a model that struggled to identify “true” cases. The recall was at 12%, which is simply below par. The same thing happened with the Naive Bayes model: the model struggled with identifying “true” cases but did great with “false” ones. The pattern between both models seems to indicate a heavy bias toward those who did not get approved, which makes sense when considering the dataset skewed toward those not approved for loans. My methods were chosen out of curiosity and logical reasoning, but they were limited. The models did improve significantly from my experimentation, which was an encouraging sign. From the initial iteration of the model to the end, accuracy improved by roughly 20-30% in both machine learning algorithms. What piqued my interest was how encoding made Naive Bayes switch entirely. While the first iteration didn’t perform well, it performed much better for “true” cases than it ever did again.

6 Conclusion

This project was a comprehensive learning experience. I often struggled with making my models perform better and finding solutions to problems with my particular dataset. One thing that was hard to work with was how skewed the dataset was to rejected cases. While this might indicate the nature of loan approval, which was mentioned earlier in the paper, it still made it harder for my models to perform. I’ve learned a lot in this project, specifically to look at more complex algorithms. Loan approval can be a very abstract concept to grapple with, and there could’ve been alternatives when it came to the algorithms I chose.

Logistic regression is inherently limited because it assumes a linear relationship, so if loan approval is more complicated than that, it might not capture it well. On the other hand, Naive Bayes assumed that features are independent of each other, but this might not necessarily be true. Overall, both models had their limitations, and I believe I’ve learned a lot of valuable things about more complex and intensive algorithms. Logistic regression and Naive Bayes are considered easy-to-use and fast algorithms, but I believe that to approach this problem more effectively, a more complex algorithm with a lot more tweaks is needed than I accomplished in this project. I believe a strong neural network could be a more comprehensive solution to this problem. However, it would take more knowledge than I currently have about the subject to implement effectively.

Despite the challenges, I believe the opportunity to go through the entire machine learning pipeline and then analyze it was an extremely valuable experience because I couldn’t get better without challenges. I feel like I have a much better understanding of the limitations and advantages of the machine learning algorithms I chose, and if faced with this in the future, a better way of approaching machine learning problems more effectively. It truly brought out a lot of critical thinking that will serve me well in the future.

References

Foster, Sarah. “Survey: Half of Loan Applicants Have Been Denied as High Rates Pinch Consumers.” *Bankrate*, 4 Mar. 2024, www.bankrate.com/credit-cards/news/credit-denials-survey/.

Lo, Ta-wei. “Loan Approval Classification Dataset.” *Kaggle.com*, 2024, www.kaggle.com/datasets/taweilo/loan-approval-classification-data.

scikit-learn. “Sklearn.linear_model.LogisticRegression — Scikit-Learn 0.21.2 Documentation.” *Scikit-Learn.org*, 2014, scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

Streaks, Jennifer. “Average American Debt in 2024: Household Debt Statistics.” *Business Insider*, 31 July 2024, www.businessinsider.com/personal-finance/credit-score/average-american-debt.

Acknowledgments

This paper was developed using various online resources cited above, previous classwork from Professor Aileen Benedict's ITCS 3156: Introduction to Machine Learning class at UNC Charlotte, a dataset from Taiwei Lo on Kaggle, and Grammarly. I did not use artificial intelligence tools to develop this paper or the source code. I used these resources as references to develop my code, statistics to back up assumptions, and data on which I built my source code and this paper.