Domain and scope of project

Financial loan servicing is a profitable business for several different types of companies, such as banks, credit unions, government agencies, and other financial institutions. Traditionally, these companies make money on loans through administrative fees and the yield spread between the rates from the larger banks they borrow from and the rates they charge borrowers. However, there's risk associated with this business. When people default on these loans, the lender is left on the hook. At best, the lender can go after the person's wages. However, this can be a lengthy process and will delay repayment. Ultimately, the lender's goal is to lend to as many people as possible, but only to people who aren't likely to default. This is where my machine learning project comes in.The project's goal is to help identify which potential borrowers the bank should lend money to. This project is relevant for financial institutions that lend unsecured personal loans to individuals and would like to utilize a machine learning algorithm to identify individuals who are unlikely to default on a loan. The key objective of this project was to create a machine learning algorithm that estimates the likelihood of an individual defaulting or not defaulting on a loan and then use this probability and other financial information to calculate the expected profit from each loan. If the expected profit is positive, then the loan will be approved. Otherwise, it will be denied.

From a technical perspective, this project allowed me to hone my python skills by working in Jupyter Notebook on the Machine Learning algorithm itself. Along with this, it allowed me to build on my front end development skills by learning Flask, HTML, and Pycharm and creating a user-interface for the loan default project on my personal website. I will go into more detail about the project below and within the jupyter notebook.

Data Description

The loan default dataset was found on Kaggle and can be located here: [Loan Default Dataset]

The dataset contains 255,347 rows and 18 columns in total. Here is a general description of each column:

Column Name	**Description**
LoanID	A unique identifier for each loan.
Age	The age of the borrower.
Income	The annual income of the borrower.
LoanAmount	The amount of money being borrowed.
CreditScore	The credit score of the borrower, indicating their credit worthiness.
MonthsEmployed	The number of months the borrower has been employed.
NumCreditLines	The number of credit lines the borrower has open.

InterestRate	The interest rate for the loan.
LoanTerm	The term length of the loan in months.
DTIRatio	The Debt-to-income ratio, indicating the borrower's debt compared to their income.
Education	The highest level of education attained by the borrower (PhD, Master's, Bachelor's, High School).
EmploymentType	The type of employment status of the borrower (Full-time, Part-time, Self-employed, Unemployed).
MaritalStatus	The marital status of the borrower (Single, Married, Divorced).
HasMortgage	Whether the borrower has a mortgage (Yes or No).
HasDependents	Whether the borrower has dependents (Yes or No).
LoanPurpose	The purpose of the loan (Home, Auto, Education, Business, Other).
HasCoSigner	Whether the loan has a Co-signer (Yes or No).
Default	The binary target variable indicates whether the loan defaulted (1) or not (0).

Overall, the information provides details about each borrower's life, financial wellbeing, their loan amount and whether the borrower defaulted on their loan or not.

Exploratory Data Analysis

The first step I took in my project was to explore the data. First, I want to understand the descriptive attributes of the borrowers in the dataset to get a better understanding of what I'm working with. Overall, the explanatory variables are very balanced. I see a fairly equal representation across the board for the categorical variables. Similarly, equal levels across a reasonable numeric range of the numerical variables are represented. To get my point across with an example, I see a roughly equal 25% representation of people for the four education categories. In other words, 25% of people's highest level of education was high school and another 25% was a Bachelor's with the same pattern repeating for a Master's and PHD. This equal split is similarly found in each variable.

Provided that I see equal representation across the board for most variables, it is important to note that this population is not necessarily representative of the United States.

Some notable differences being that:

- The median and mean income in the dataset is ~$82,500, while the median income in the US is ~$31,000 and the mean income is ~$54,000
- The average credit score in the dataset is 574, while the average credit score in the United States is ~700
- In the dataset, I saw that there was a roughly 25% representation of people whose highest level of education was either high school, Bachelor's, Master's, or a PHD. However, in the US around 62% have less than a bachelor's degree, 23% have a

bachelor's degree as their highest degree, and 14% of people have a master's or doctorate degree

It's important to consider differences between the dataset that trained an algorithm and the population it would be used on, so that we can understand any potential weaknesses in the algorithm.

I also performed univariate and multivariate analysis on the dataset to understand the relationships between defaulting and the several variables I know about each borrower.

Some general relationships that I walked away with was:

- Those who were unemployed or only part-time workers were more likely to default than those who were self-employed or full-time. Full-time employees particularly had a lower default rate.
- Those with higher education levels were less likely to default on average.
- Those who have a higher income and credit score are less likely to default on average.
- Those with a higher DTI, interest rate, and loan amount are more likely to default on average.
- Lastly, a potential surprise was that those who do not have dependents are more likely to default on average.

Below, I will walk through my general thinking as I worked on the machine learning algorithm in my Jupyter Notebook. More detail will be found in the notebook, if you'd like to dive into any piece further.

Data Preparation and Cleaning

There were no missing values in the dataset, so the only preparation needed was to one hot encode the categorical variables. After one-hot-encoding the data, I brought all the data back together, but removed the LoanID variable, as it is just an identifier and not a variable to be used for prediction. Lastly, I split the dataset into feature and response datasets and further split them into testing and training data. The data is now ready for model training.

Model Training

I trained both a random forest and gradient boosting classifier. I performed grid search for parameter selection and evaluated the model through 3 fold cross validation. At first, I scored the models based on accuracy. This resulted in the final random forest classifier predicting that everyone wouldn't default. Similarly, the gradient boosting classifier heavily skewed toward predicting that no one would default. This resulted in high accuracy (around 89%), but very low precision, recall, F1 Score, and AUC curve. For the Random forest classifier, all of these scores were zero. However, the precision was actually fairly high for the gradient boosting model. While this model was predicting very few defaulters, it did have 60% precision when doing so.

Overall, scoring based on accuracy did not seem appropriate. Accuracy is not always a reliable metric, when the response variable is imbalanced. This is the case for this dataset, where those who default represent only around 12% of the data. Therefore, I rescored the models based on F1 score during parameter selection. I chose F1 score, because there was no reason to lean more on recall or precision specifically. After finishing the grid search for a random classifier and gradient boosting classifier, I trained both models on the chosen parameters.

Model Evaluation

I evaluated the model on the below metrics:

Models	Item	Score
Random Forest	Accuracy	0.807167
	Precision	0.196494
	Recall	0.21661
	F1 score	0.206063
	AUC curve	0.550457
	Log Loss	6.94786
Gradient Boosting	Accuracy	0.819013
	Precision	0.221091
	Recall	0.224576
	F1 score	0.22282
	AUC curve	0.560617
	Log Loss	2.195316

Random classifier confusion matrix		
	Predicted Negative	Predicted Positive
Real Negative	39,944	5,226
Real Positive	4,622	1,278

Gradient Boosting confusion matrix		
	Predicted Negative	Predicted Positive
Real Negative	40,502	4,668
Real Positive	4,575	1,325

In the above exhibit, I compare the Random Forest and Gradient Boosting model across a number of different metrics. While I optimized the model based on F1 score, it's important to compare among a number of different metrics in order to get a comprehensive look of the model's performance. Based on the metrics above, I will move forward with the gradient boosting model. The models have fairly similar metrics, but the gradient boosting model edges out the random forest model in all of them, so we'll move forward with that one.

In the grand scheme of things, it's important to contextualize how well the model is doing. The models are by no means perfect and do have difficulty in acknowledging who is likely to default. Let's contextualize accuracy, precision, and recall to get this point across. Accuracy is around 82%. This means that 82% of the time it is correctly labeling a defaulter or non-defaulter. However, precision is 22%. This means that when the model labels someone as a defaulter, it is only correct 22% of the time. Similarly, recall is 22%. This means that out of all of the defaulters in the test set, the model was only able to identify 22% as such. Overall, the model has difficulty identifying defaulters correctly. It is important to keep this consideration in mind, if this model was to ever be employed in the real world.

Ultimately, the gradient boosting model is going to be used to predict the probability of someone not defaulting. Note when using the predict_proba function, I will take the probability in index 0, which represents the probability of someone *not* defaulting. I will then calculate expected profit based on the equation below:

Profit = Probability x LoanAmount x LoanTerm / 12 x InterestRate / 100 - (1-Probability) x LoanAmount

The formula calculates the expected profit from interest if they do not default minus the expected loss experienced if they do default (1-probability). If profit is in the positive, then the final model will approve the loan. If profit is in the negative, then the model will deny the loan.

This part of the model will be implemented in the User-Interface and will be talked about in the next section.

User-Interface Integration

<u>General overview:</u>

I placed a navigation bar at the top of the website that allows the user to click through a biographical homepage, a resume page, a user-interface for the loan default project, and an external links page that brings you to my github.

<u>Construction of website:</u>

I utilized the integrated development environment (IDE) Pycharm in order to create the above website with Flask and Heroku. Additionally, I read more on bootstrap (a popular CSS Framework) in order to create a user-friendly website with nice aesthetics.

First, let's discuss the python file.

The goals of python file were as follows:

- Bring in gradient boosting model
- Create any functions needed for website/user-interface
- Connect to any html pages of the website

I accomplished bringing in the gradient boosting model by using pickle. Essentially, I saved the model down to my computer from my Jupyter Notebook and then I was able to load the same model into the python file in Pycharm via pickle.

I defined the profit function (previously discussed) in this python file, as it will be used in the loan user-interface. Lastly, I created routes for each page in the website in this file and defined what should happen when each website route is called. The ones created for this website are the homepage, resume page, the page for the loan default project, and the page for the external links. I won't discuss the first three pages in too much detail, as each route simply calls the html file for that page, but feel free to peruse the information.

However, the route for the loan default page is a little bit more complex. First, I need to utilize POST requests because the user will be submitting information on the loan default page. As you can see in the python file, the first part of the loandefault_page function defines what should be done when information has been submitted on the loan default page. This part of the function defines what will be done with the submitted data and how it will determine whether the loan application will be denied or approved. Let's walk through it.

The first part of the if statement takes in the information submitted on the loan default web page. This information is saved to info_submit. Next, I need to one-hot-encode this information in order to use this information as an input for the model. As you can see, this code works similarly to how it did in the Jupyter Notebook.

Next, let's discuss the variable called empty_df. I run into a tricky issue when encoding the data submitted on the loan default page. I am able to create encoded data for the variables selected, but I do not have encoding for the categorical data that was not selected. For example, if the user chooses that they have their bachelor's then I do not have information on the user *not* choosing High School/Master's/PHD. To solve this problem, I created a dataframe for all the variables the gradient boosting model takes with all information filled with zeros. Next, I combine this information with the data that was submitted and remove the empty information. This results in a dataframe with the information that was submitted and includes all the appropriate variables.

At this point, I am ready to use the model and the function I created earlier. I will need the probability of not defaulting, the loan amount, the loan term, and the interest rate for the function. I can get the loan amount, loan term, and interest rate from the data that was submitted on the loan default page by the user. I'll calculate the probability of not defaulting by utilizing the 'predict_proba' function of the gradient boosting model and submitting the user-submitted-data to it. Once I have all of this information, I submit the information into the loan function, which will return whether the loan has been approved or denied.

Walkthrough of user-interface:

If you click on "Loan Default Project" on the navigation bar, you will be brought to the user-interface of the loan default project. Here you will find instructions on how to use the tool. Simply put, the user will fill out the same type of information that the gradient boosting was trained on, but that information will be used to determine whether their loan is approved or not. Note that the variables have been programmed such that you can only submit information that makes sense provided the variable. For example, you can't submit a negative age or a credit score outside of the normal 300-850 range.

After filling out all of the information for the loan application, the user will hit submit and the application status will change to whether they've been approved or denied their loan.

Conclusion

Overall, my project focused on a real-life business issue that banks, credit unions, government agencies, and other financial institutions interact with everyday when people apply for loans. It helped me hone the machine learning skills that I was taught in the Master's program at Eastern. Additionally, it let me expand my knowledge by learning Flask, HTML, and Pycharm.