

Table of Contents

1. Definition

- 1.1. Project Overview
- 1.2. Problem Statement
- 1.3. Metrics

2. Analysis

- 2.1. Data Exploration
- 2.2. Exploratory Visualization
- 2.3. Algorithms and Techniques
- 2.4. Benchmark

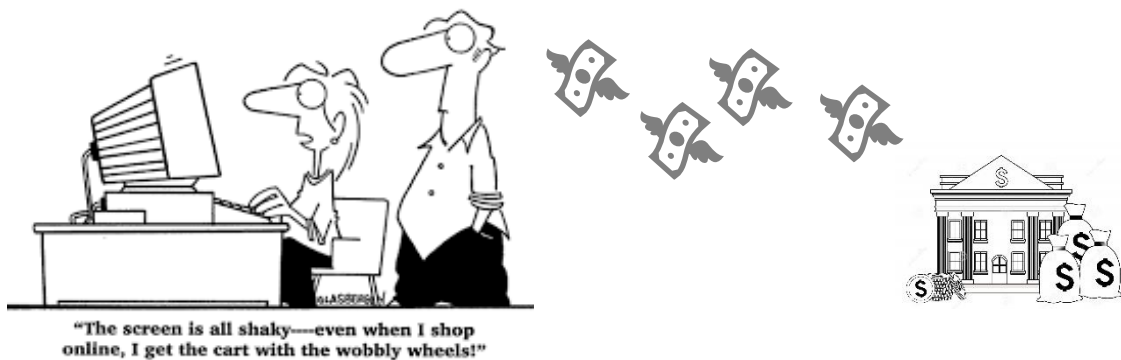
3. Methodology

- 3.1. Data Preprocessing
- 3.2. Implementation
- 3.3. Refinement

4. Results

- 4.1. Model Evaluation and Validation
- 4.2. Justification

The User storyboard is the as follows:



It's Black Friday, and there deals you CANNOT miss on electronics. Your PC is so slow and you've always wanted to upgrade to a laptop, you keep shopping online until you find one that fits your required specs and it's 50% OFF. Only problem is it costs \$1500 upfront and you only have five days to use the discount.

You decide to apply for a loan from RougeBank branch, and since you have a good credit score you're likely to be approved fast.

User Story 1: As a **bank client**, I would like to get the loan approved fast (within the five days) so I don't miss the opportunity to buy.

User Story 2: As a **bank**, I should be able to accurately assess credit worthiness of the client

Then time passes



You wait two days, three days, five days, ten days, and finally after fourteen days, you get a decision back that you're loan has been approved.

You're obviously frustrated that you missed your opportunity to buy the laptop. The next time someone asks you to bank with RougeBank, you'll not want say no, and as a **bank**, I have just lost a client to my competitor



Proposed solution:

Go to the RougeBank myloan app page and fill out the form. The application is processed on your laptop/mobile device You get the decision instantly ✓



And so you're happy and can buy your favorite deals, well next Black Friday...

1. Definition

1.1. Project Overview

The goal of this project is to develop a machine learning web application that is used in as a banking service to facilitate loan application process. The motivation behind this project is to shorten the amount of time it should take for a loan application to be processed by banks and the decision to be sent to the client. The primary reason why “banks” take considerable amount of time (ranging from 2-4 weeks) to process small-medium loan applications is the presence of legacy procedures that exist for such business process. With the advent of machine learning and sophisticated web frameworks, the process of developing such web apps is becoming increasingly easier. This project is a stepping stone into a much larger and promising customer experience improvement journey. The rise of fintech applications as this one is a golden opportunity for banks to transform their UX and increase their client base.

This project is divided into three main milestones. The first milestone is the development of machine learning algorithm for the dataset in question. The second milestone is the web application development. I'll be using Python web framework called Django to build the web app's backend and build the frontend using pure CSS and HTML. The last milestone is the improvement and reflection whereby I attempt to add more features into the web app and talk about some future direction for it.

1.2. Problem Statement

To reiterate, the main problem I intend to tackle in this project is the longevity and complexity of loan applications to be approved by banks. I specifically mention banks as the primary financial institution in question because there are other FIs that leverage more technology-based models such as peer to peer lending or crowdsourcing to lend smaller amounts of money (100-30,000) range.

1.3. Metrics

The main metric that will be focused on is the false positive rate of the chosen algorithm. Imagine the following scenario, you are a client at RougeBank and you have a very promising business project that requires a \$10,000 investment on your behalf. You go to the bank's website to apply for the \$10,000 loan and getting approved instantly. This feels great and you'll be able to bank in on that profitable business project of yours without any delay. Next day you go to the bank to collect your money only to realize that the bank has made an error and sent you the following message: “We apologize for your inconvenience, there was a technical error in our systems, the bank cannot grant you the loan amount specified.”, pretty disappointing, right? You'll probably never going to bank with RougeBank again and you'll probably tell your disappointing story to your business associates and family members and the word will spread about the bad reputation RougeBank now has. This is why the false positive rate should be minimized as much as possible without compromising the model generality. In more technical words, the bias-variance tradeoff should be minimized as well.

In addition to the false positive rate, the accuracy should also be matched or better improved above the current industry benchmark. While it's extremely difficult to get an accurate number for the accuracy of current models utilized in banks, it's safe to say that a minimum of 70% accuracy is achieved by the models currently in use.

2. Analysis

2.1. Data Exploration

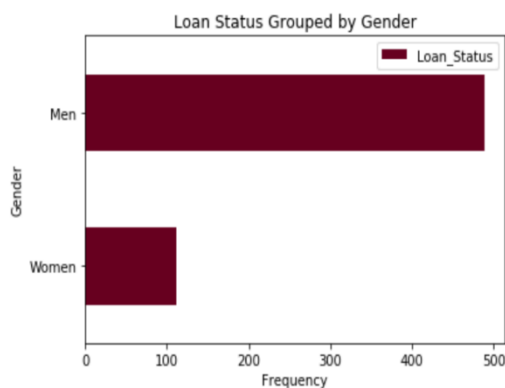
2.1.1. Meta Data

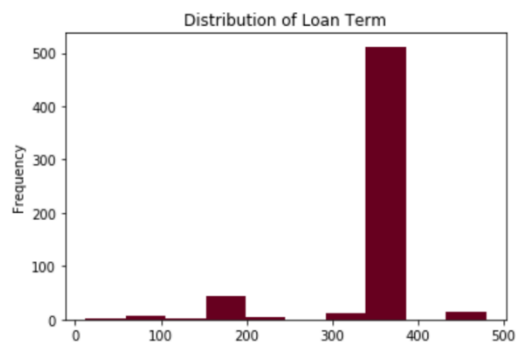
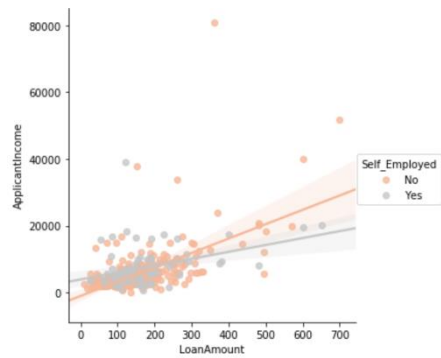
Before starting with any data exploration, it's always a good practice to obtain some metadata about the dataset. This includes the datatypes of each variable as well as the number of missing values. The dataset is made up of 614 rows and 13 columns, the variables are distributed with a 50-80% mix of numerical vs categorical variables. There are some values missing across the rows. There are multiple ways to deal with this problem, however, I choose to remove these missing rows altogether for two reasons. First, it would be illogical from a business perspective to have incomplete information about the client when they fill out the loan application. While the potential for error entry or missing values is likely as they fill out the application form, I will add a feature to deal with this problem later on. The second reason to remove these observations is to optimize the data fed into the algorithm so it can be trained on representative data. In the business context at hand, it's more important to feed high-quality data into the model than to preserve the highest number of datapoints possible.

2.1.2. Descriptive Data Analysis

There's a trend of high variation in the numerical variables used in this dataset. The minmax range of coapplicant income and applicant income are significantly large. Additionally, there seems to be another trend of skewness to towards the right in all variables, with some varying degrees. For example, loan term amount particularly stands out as it's z-value (assuming a normal distribution with mean 0 and standard deviation 1) is $342.00/65.12 = 7.42$. These trends will be later confirmed in the data visualization section.

2.2. Exploratory Visualization





The purpose of this section is to unveil hidden insights about the clientele this bank serves. By looking at multiple dimensions of data using seaborn and matplotlib, some interesting observations about the dataset were revealed. To begin with, there is an imbalanced representation of loan classes. There are more applications that have been approved than those that have been rejected. This is a common problem in machine learning tasks concerning rare events. This will be accounted for by oversampling the minority class (“No”) later in data preprocessing. Some other interesting observations arise. For example, majority of the clientele is men and more men have been approved than rejected compared to women who share an almost equal split of acceptance versus rejection. The majority of accepted loans are for suburban areas and educated applicants. Moreover, the applicant and co applicant incomes are negatively correlated and most of the observations are cluttered for low ranges of applicants and co applicant incomes, with larger difference showing at higher range. Although the graduates and non-graduates are not competing for the same loan amounts, they are competing for the same loan terms. Also, self-employed applicants are asking for higher loan amounts compared to non-self-employed, indicating a potential gap in the bank in terms of business loans offered by the bank. In the bad credit history category, there’s more non self-employed applicants, indicating that these applicants may be trying to improve their finances. The bank is currently targeting applicants who are at their early marriage lives with no or a single dependent, so likely age is between 25-40 years.

The conclusion from this section is that RougeBank is currently targeting the following clientele:











	Income	<20,000		Employment	Graduated
	Marital status	Married		Mean Loan Term	>360
	Gender	Men		Mean Loan Amount	10-15,000
	Dependents	Zero to one		Education	Graduated
	Property Area	Suburban		Credit History	Good

Figure 1: Demographic characteristics of target segment

2.3. Algorithms and Techniques

Before choosing the algorithm model, it's worthwhile recapping the kind of problem in this dataset. Firstly, this is an imbalanced class problem with unequal class representation, and the majority class occurs twice as much as the minority. Secondly, the numerical variables are not following a perfect gaussian distribution. Thirdly, it's likely not a problem that can be classified linearly as evident from the graphs in the visualization section above. In terms of the business problem, our focus is on the false positive occurrence (precision of minority class should be maximized). Additionally, to ensure partial explainability to the model linked directly to client satisfaction, it's less favorable to use black box models. This defines my criteria of model choice to be used.

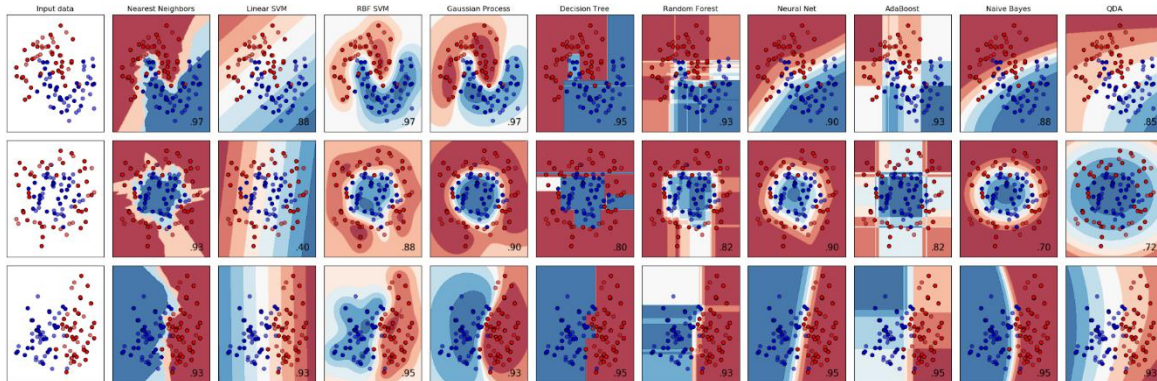


Figure 2: Classification models comparison. Source: Sklearn Documentation

The snapshot above is taken directly from the sklearn documentation for classifier comparison. Based on problem characteristics described above, I would assume that algorithms such as RFB SVM, Random Forest, Neural Network and Naïve Bayes would be a good set to start with. The reason being that the shape of input data is closest to the first row of the table.

2.4. Benchmark

Linear classifiers are most easy to understand and interpret to users. It's also common standards for banks to use logistic regression in their credit analysis. This algorithm has been adopted for more than a decade by financial institutions (ResearchGate, 2010). Therefore, as a benchmark I use logistic regression class from sklearn library. Without any modification to the default parameters, the accuracy obtained is 85% and false positive instances are 9 out of 67, an equivalent 13%.

3. Methodology

3.1. Data Preprocessing

Let's list out the problems we need fixed during this step. The target variable has imbalanced class representation, there are missing values that should be dropped, some variables have skewed distributions and lastly, the categorical variables should be one hot encoded in preparation for modelling. First, to deal with the imbalanced class, I used the imbalanced learner library to fit a SMOTE (Synthetic Minority Oversampling Technique) to take the minority class, the rejected applications, and increase its instance count from 148 to 322. The reason why I chose to oversample than to under-sample is to keep the integrity of data and

provide as many points for model training as possible. In the refinement section, I talk about using a more improved version of SMOTE, ADASYN later in this report. To deal with the skewed distribution and variation among numerical variables, I decided to standardize the values using MinMaxScaler. A second reason why this is done is because I intend to use a neural network for training.

3.2. Implementation

After performing the data preprocessing steps above, I pickle the scaled and transformed dataset to have it ready for use instantly. I use the sklearn module to load the algorithm classes and the evaluation metrics, such as accuracy score and confusion matrix. Prior to implementing the algorithm, I split the data into training and testing subsets with test size of 10%. The goal is to feed the most data into the model for accurate outcomes.

For the naïve bayes algorithm, there were multiple types included sklearn library, but the most suitable one to the case of imbalance dataset is the ComplementNB, according to the documentation. This is because the classical Naïve Bayes model assumes independence between features, which is most likely an oversimplification to the data. Thus, ComplementNB corrects for this assumption by adding a smoothing parameter. The results obtained from this algorithm are underperforming relative to the benchmark. The accuracy deteriorated significantly to 65% whereas false positive rate jumped to 16%.

Following that, I implement the RFB SVM algorithm which creates a hyperplane to separate the classes. There are multiple kernel functions that can be used to create the optimal hyperplane, however, I choose to stick to default options and leave hyperparameter optimization for later. The accuracy achieved is 85% and the FP rate is 15%, again underperforming benchmark.

Ensemble learning models are gaining a lot of increase popularity in the data science world. Combining several machine learning algorithms into one predictive model usually produces more accurate results than a single model would, as many Kaggle competitions proved (Medium, 2020). Random forest is a bagging technique trains several decision tree models, each on a different subset of data, in parallel, then uses a majority vote for final prediction, in this case classification outcome. This can significantly reduce variance, bias or improve accuracy. This algorithm succeeded in beating the benchmark accuracy by additional 3% and FP rate improved significantly to 5%. The last algorithm I use is the neural network. To gain more exposure into different Python libraries, I decided to use Keras to build the Neural Network architecture. It also gives me a lot more control over hyperparameter tuning. I had run into a problem with kernal integration on sagemaker. At first, I was using mxnet backend, but the API couldn't integrate with it at all. After consulting stackoverflow and github issues, and a lot of playing around, I was able to call the API using pytorch backend and the library was successfully imported. I build a sequential 3-layer NN with relu activation functions and the outer layer with a sigmoid activation function. The number of neurons inside each layer is randomly chosen, while keeping in mind the overfitting issue. As mentioned, Keras gives a lot of autonomy over hyperparameter optimization and network architecture. Initially, I fit the training data in a single epoch, the default value and this produces 65% accuracy and 38% FP rate. This is obviously underperforming the benchmark, but these results should be improved with hyperparameter optimization. Specifically, by increasing the number of epochs, the NN would be allowed to train better on the data fed into it.

3.3. Refinement

I start this process by refining the neural network. I increased the number of epochs gradually and through trial and error, I was able to achieve a very improved result. By running the NN 100 epochs, accuracy is improved 65% and the FP rate becomes 7%. I also try increasing the complexity of the architecture slightly by adding more layers, but this hasn't improved the result significantly given the increased risk of overfitting. Next, I add a dropout layer with 5% rate to reduce risk of overfitting but the results again deteriorated. Lastly, I change the optimizer to Stochastic Gradient Descent instead of Adams, but this also hasn't shown a large improvement. Thus, the optimal NN algorithm is achieved by optimizing number of epochs while keeping everything else constant. The SVM can be hypertuned by changing the regularization parameter C and the kernel function. For this task, I pass the parameters as a dictionary and use gridsearch to exhaustively search the parameter space and obtain the best estimator model. This is the model corresponding to C=10 and a polynomial kernel function. The results of this model outperforms the benchmark in both metrics, 90% accuracy and 9% FNR. Lastly, I choose to tune the random forest model by changing the criterion for splitting as well as the number of trees. I passed a list ranging from 25 to 150 trees and a tuple for both criteria, Gini and Entropy. The results are outperforming the benchmark on both metrics. The best estimator model has 100 trees built and entropy criterion for splitting.

4. Results

4.1. Model Evaluation

The models implemented will be validated on both the accuracy and false positive rate metrics. So far, I have eight models implemented. A final dataframe of all the algorithms with respective metrics arranged in descending order of FPR, my primary metric for business objective. The final model I choose is the tuned random forest since it optimizes both metrics among all the others. The parameters of this model are 25 trees and entropy criterion.

	False Positive Rate	Accuracy Score
Random Forest	5.0	0.880597
Random Forest Tuned	5.0	0.865672
Neural Network	5.0	0.820896
Support Vector Classifier Tuned	6.0	0.895522
Support Vector Classifier	10.0	0.835821
Naive Bayes	11.0	0.656716
Naive Bayes Tuned	11.0	0.656716

Figure 3: Model evaluation table

4.2. Justification

The final random forest model is not perfect, and it's crucial for developers to know where the failure happened so it can be taken into consideration in production. With some dataframe manipulation, I create two csv files of the false positive observations for the benchmark and the final model. Note that the numerical values are scaled, so these are not the actual values. There are two important things to grasp from here, one is where both models failed and the second is where the final model failed.

Index No.	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Female	Gender_Male	Married_No	Married_Yes	Education_Graduate	Education_Not Graduate	Self_Employed_No	Self_Employed_Yes	Property_Area_Rural	Property_Area_Semiurban	Property_Area_Urban
286	0	0.242176871	0.155628454	0.949236579	0.72972973	1	0	1	0	1	1	0	1	0	1	0	0
244	0	0.246398268	0	0.796954315	0.72972973	1	0	1	1	0	1	0	1	0	1	0	0
101	0	0.054829932	0	0.126903553	0.72972973	1	1	0	0	1	1	0	1	0	1	0	0
176	0	0.03834261	0	0.272419428	0.72972973	1	0	1	0	1	1	0	1	0	1	0	0
118	0.666666667	0.026357452	0	0.175972927	0.72972973	1	0	1	0	1	0	1	1	0	1	0	0
357	0	0.044007421	0.075922806	0.2749577	0.72972973	1	0	1	0	1	1	0	1	0	0	0	1
163	0	0.040420532	0	0.199661591	0.72972973	1	0	1	1	0	1	0	1	0	0	1	0
164	0	0.12183055	0	0.346869712	0.72972973	1	1	0	1	0	1	0	1	0	0	1	0
271	0	0.106369821	0.1283149215	0.535922166	0.72972973	1	0	1	1	0	1	0	1	0	1	0	0


Figure 4: False positive observations of benchmark, logistic regression

Index No.	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Female	Gender_Male	Married_No	Married_Yes	Education_Graduate	Education_Not Graduate	Self_Employed_No	Self_Employed_Yes	Property_Area_Rural	Property_Area_Semiurban	Property_Area_Urban
176	0	0.03834261	0	0.272419428	0.72972973	1	0	1	0	1	1	0	1	0	1	0	0
118	0.666666667	0.026357452	0	0.175972927	0.72972973	1	0	1	0	1	0	1	1	0	1	0	0
357	0	0.044007421	0.075922806	0.2749577	0.72972973	1	0	1	0	1	1	0	1	0	0	0	1
163	0	0.040420532	0	0.199661591	0.72972973	1	0	1	1	0	1	0	1	0	0	1	0
164	0	0.12183055	0	0.346869712	0.72972973	1	1	0	1	0	1	0	1	0	0	1	0

Figure 5: False positive observations of final model, ensemble random forest

Observations number 286, 244, 101 and 271 were correctly classified by the random forest model but not the benchmark, so here is where the RF overperformed the LR. However, the RF still failed, just like LR, to capture observations 176, 118, 357, 163, 164. By comparing the variables in both sets, it seems that the RF is able to capture the observations on the higher end but miss a lot of the lower end values in the applicant and co applicant incomes. The model is missing applicants who are married, non-graduates living in rural areas. This makes sense in business logic as these characteristics are not aligning very well with the dominant features of target segment shown in figure 1. However, if the bank chooses to continue targeting the same segment, then these applications will most likely continue to be missed by the model. The alternative is to synthesize more of these observations in the SMOTE sample, but this is for future improvements.

5.0. Production Testing

To ensure that the model will perform as expected in production, I synthesize eleven random observations, based on the distribution of each variable, to test the model for production use. The incoming data passes through two steps of data preprocessing. Firstly, it will be one hot encoded using get dummies method, then it will be scaled, through the transform method, using the MinMaxScaler the training data was fit to during development process. The artifacts of this scaler were pickled and stored in .txt file. The processed data is then passed into the model artifacts to predict the loan status. The resulting prediction is eight accepted loans and three rejected loans. This production test is simulating the real data in production environment. So to ensure the soundness, the way I populated the observations was through trial and error of observations similar but not exactly equal some observations in the training dataset. The model is verified to be working correctly and the first milestone is accomplished .

Milestone II: Web Development

After finishing the machine learning part and pickling the model artifacts to use it locally, the web development part comes next. I use Django, a very popular python web framework to develop the database, the routing and the backend logic of the application. The goal of this section is to build a progressive single page web application that interacts with the model artifacts directly to classify the loan application sent to it from the frontend user interface. In Udacity, the way I've learnt how to do this by deploying the model on sagemaker and building an api to connect to the model artifacts. However, being someone who likes challenges, I decide to build the API myself using Django rest framework. The main steps included in this milestone are the following:

1. Build the data schema in SQLite for inputs passed on from the UI, as html form
2. Build the API POST request in rest framework to connect to the model artifacts
3. Serialize the data using JSON serializer in python
4. Test the API using Postman
5. Set up page routing in routers (rest framework)
6. Build the views and business logic to accept data and send it to the ML model API
7. Use HTML and CSS to build the frontend UI
8. Connect the UI to the backend on the HTML
9. Test the web app on localhost

I follow the documentation on Django to build out these steps. The starter app on Django comes with a lot of prebuilt functionality to help people starting out in web development and it's the reason why this milestone was not very challenging. All the files are included in the MyAPI folder on github.

Milestone III: Reflections.

It's time for reflection. This project has been quite interesting as it involved more than one technology and had broader perspective than typical data science problems. There are a lot of improvements that can be done. For example, In the preprocessing step, I could use ADASYN to oversample data instead of SMOTE. SMOTE the very popular among the data science community, but what makes ADASYN usually better is that adds a "noise" factor to synthesized data which makes the data more realistic. In the algorithm development step, I can use containerized models like the ones I used in other Udacity projects. Also, I can build more complex neural network architecture and tune the models on more parameters using gridsearch. For web development, the possibilities are limitless. I can build multiple page website instead of a single page progressive web application. There's one additional feature I added in the end that's important to mention. The app is designed for micro to small loan amounts. The same process cannot be used for large-sum loans as it clearly requires a lot more verification and has a larger risk of default, which makes the problem very than the one we have just solved. So, in the views model, I add a \$5,000 cap on the loan amount that could be requested as a validation check before the client submits the form. I ensured that this amount is not outside the range of the original dataset.

Bibliography

Khandani, Amir & Kim, Adlar & Lo, Andrew. (2010). Consumer Credit-Risk Models Via Machine-Learning Algorithms. Journal of Banking & Finance. 34. 2767–2787.
10.1016/j.jbankfin.2010.06.001.

https://scikitlearn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html#sklearn.naive_bayes.ComplementNB

<https://towardsdatascience.com/random-forest-a-powerful-ensemble-learning-algorithm-2bf132ba639d>