# Predicting Acceptance of Personal Loans Using Machine Learning Algorithms

AJ Rossman(ajrossmn@stanford.edu), Conor Casey(conorc@stanford.edu)

## 1 Introduction

Banks spend millions of dollars on sales teams that try to find customers who will accept a personal loan if offered. The rate at which individuals accept these personal loans is very low (~10%). The low acceptance rate means that randomly choosing customers to market to is incredibly wasteful in terms of both time and money. Thus, it is in the interest of the banks to determine which customers are the most likely to accept this type of loan based on the customer's information. For our project, we aim to apply the K-Means, Support Vector Machine, Logistic Regression, Perceptron, and Multilayer Perceptron algorithms to predict whether bank customers will be willing to take on a personal loan. The input into our algorithms will be a customer's age, years of professional experience, annual income measured in thousands of dollars, home address, ZIP code, family size, average spending on credit cards per month measured in thousands, education level, and the value of a house mortgage (if the customer has any). Our algorithm will use this information to output a prediction on whether a customer will take a personal loan (1) or not (0). In practice, sales associates can then target these customers knowing that they are not wasting their time and effort on individuals who are unlikely to take a loan.

## 2 Related Work

This personal loan scenario can be categorized under the umbrella problem of classification for unbalanced datasets. In our example, this unbalance is due to the fact that there were more people that declined the personal loan than accepted one. While our situation deals with bank customers, there are many other unrelated situations where the dataset contains a disproportionate number of labels in a specific group. This characteristic can cause some difficulties in classification because, as Leihua Ye in his work states, "the ML algorithm learns more from the majority group and may easily misclassify the small data group".

One general solution to this problem is simply under-sampling the larger class in order to balance out the dataset. This was explored by Bao Lei and colleagues in their article about trying to optimize support vector machines utilizing under-sampling techniques.

Conversely, another common solution to this problem is going the other direction and over-sampling points from the minority class. One such work that successfully used this was by J.Piyasak, K. Wong, and C. Fung where they combined an oversampling algorithm (SMOTE) with Complementary Neural Network.

Interestingly, Chris Drummond and Robert Holte compared these two general methods and came to the conclusion in their paper that under-sampling was better than over-sampling. In short, they saw that "undersampling produces a reasonable sensitivity to changes in misclassification costs and class distribution. On the other hand, over-sampling is

surprisingly ineffective." While we did not implement either of these methods in our work, we are interested in exploring these solutions in future work to see if our results agree with their reasoning.

Among other strategies, many other works focus on fine-tuning the algorithm's parameters, such as efforts to optimize the parameters of support vector machines, as researched by Eitrich Tatjana and Bruno Lang in their paper. This is the strategy that we generally focused on in our work and the one that we believed made the most significant impact on the overall accuracy. Furthermore, it was relatively quick and easy to adjust parameters of our algorithms because we implemented the algorithms using scikit libraries, which provide simple ways to fine-tune and adjust the parameters.

## 3 Dataset and Features

Our dataset is titled "Personal Loan Modeling" from Kaggle. It contains a list of 5000 bank customer's data resulting from a marketing campaign. For each customer, there are 12 features and a [0,1] label of whether they accepted a personal loan offered in the bank's marketing campaign. The specific features for each customer are described in detail below:

| # Age | # Experience | # Income | # ZIP Code | # Family | # CCAvg |
|---|---|---|---|---|---|
| Customer's age in completed years | Number of years of professional experience | Annual income of the customer ($000) | Home Address ZIP code. | Family size of the customer | Avg. spending on credit cards per month ($000) |
| **# Education** | **# Mortgage** | **# Securities Account** | **# CD Account** | **# Online** | **# CreditCard** |
| Education Level. 1: Undergrad; 2: Graduate; 3: Advanced/Professional | Value of house mortgage if any. ($000) | Does the customer have a securities account with the bank? | Does the customer have a certificate of deposit (CD) account with the bank? | Does the customer use internet banking facilities? | Does the customer use a credit card issued by this Bank? |

We opted to use 80% of our dataset for training (4000 examples), and the remaining 20% (1000 examples) for testing. Thankfully, our dataset was very easy to use and not much preprocessing was needed. The data did however initially contain a column of indices labeled 1- 5000 which we removed before inputting into our algorithms as it was simply a numbering of data points. In terms of normalization, we augmented the dataset for our Multi-Layer Perceptron Classifier by scaling the values between [0,1] using the scikit function *MinMaxScalar()*. As mentioned above, we hope to explore under-sampling and over-sampling with this dataset in the future.

## 4 Methods

### 4.1 Perceptron

We initially implemented the Perceptron algorithm because it is a classic algorithm used for classification. The Perceptron algorithm takes in data labeled with either 0's or 1's and learns the weights of features that can then be used for prediction. The algorithm first randomly initializes the weights. Then, a data point is randomly selected. If the data point is labeled 1 and the dot product between the current weights and the data point is greater than 0, then the data point's values are added to the weights. If the data point is labeled 0 and the dot product of the weights and input is equal to or less than 0, then the input data is subtracted from the weights. The algorithm iterates this process until convergence. Prediction is then performed by taking the dot product of the weights and a new data point. If the dot

product is equal to or greater than 0, then the algorithm predicts 1. If the dot product is less than 0, then the algorithm predicts 0.

**4.2 Logistic Regression**

Next, we implemented the Logistic Regression algorithm for classification. The Logistic Regression algorithm works by using the dot product of the weight vector with a data point as the input into the sigmoid function as seen here:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

If we let

$$\begin{aligned} P(y = 1 \mid x; \theta) &= h_\theta(x) \\ P(y = 0 \mid x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

Then we are able to maximize this log likelihood function using gradient descent to learn our weights

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{n} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \end{aligned}$$

Once these weights are found on the training data, we can dot the weights with new data to predict which class a data point is in.

**4.3 Multi-layer Perceptron**

We also implemented a Multi-layer Perceptron model. This model is made up of many singular nodes, all of which are their own perceptron model. The neurons on the input layer receive the data, put it through an activation function and input it into another neuron. These internal layers are called hidden layers. The various hidden layers eventually feed into an output layer, which consists of a single neuron and outputs the value of interest, in our case a 1 or 0. The algorithm learns through a process called backpropagation. This method runs the training data through the neural network, calculates the loss, and then propagates backwards through the network and updates the weights of each neuron according to their corresponding error. Prediction is then performed by passing data through each neuron of the network forwards and the output is the prediction.

**4.4 K-Means**

Another algorithm we experimented with was the KMeans Algorithm. With this algorithm, classification is decided based on a point's distance to any number of centroids. In our case, we only used two centroids because an individual can either accept the personal loan, or reject it. To start, the centroids will be randomly initialized to two of the data points in the dataset. Then, data points will be assigned to a centroid depending on which one is closer. Next, the centroids will re-center to be the mean of all points in their respective clusters. This process repeats until convergence is reached. For KMeans, convergence is reached when the two clusters created are the same as the clusters in the last iteration of the algorithm.

### 4.5 Support Vector Machines

Support Vector Machines work by finding a hyperplane that best separates the two groups of interest. The best hyperplane in this case is the one that maximizes the distance between the two groups. The distance is measured from the main dividing hyperplane to support vectors (hence the name of the algorithm). The maximum margin between the two groups is found through iterative steps of gradient descent as with many other algorithms. Finally, the prediction is made by simply checking which side of the hyperplane a data point lies.

## 5 Experiments/Results/Discussion

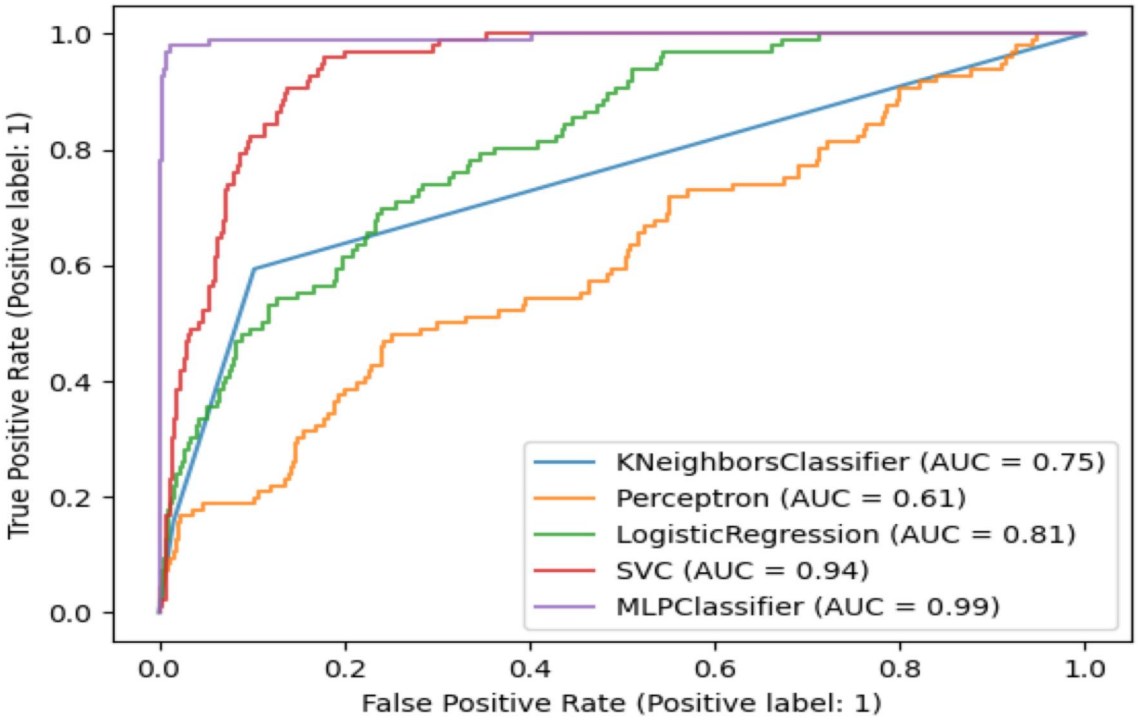
We initially experimented with a single perceptron model. The perceptron algorithm performed poorly on our data. Everytime we ran the algorithm, it predicted all zeros for our testing data. In context, this means that the algorithm predicted that no customers would take personal loans. If our entire motivation is to better target individuals who are likely to accept a personal loan, an algorithm that predicts that no one will accept a loan is not helpful. We believe that the reason for this outcome of the perceptron algorithm is due to the imbalanced nature of our data. Since the portion of customers that did take a loan is around 10%, we think that the algorithm found that all of the customers refusing a loan was the most likely scenario.

We ran into the same problem with logistic regression. The algorithm predicted all zeros, or that no one would accept a loan. Since our dataset is unbalanced, this still meant that the algorithms had about 90% accuracy. This made it clear that the algorithm's accuracy was not an indicator of success. So, we looked for different measurements. After doing some additional research, we landed on using the Receiver Operating Characteristic (ROC) Curve. This curve is a "graphical representation showing how a classification model performs at all classification thresholds" (Ye, Leihua). Additionally, the Area Under the Curve (AUC) measurement is often used to quantify the evaluation of a curve. The closer to one, the better.

Afterwards, we decided to move away from probabilistic models and try the K-means algorithm. The K-means algorithm did manage to predict some ones and had an accuracy of around 90%, with 7 false positives and 72 false negatives. While the ROC curve for the K-means algorithm did not have as high of an AUC score as logistic regression (.75 vs .81), we felt that K-means was more useful than logistic regression and the perceptron as it was able to correctly predict some customers who would take a loan. Next, we implemented the Support Vector Machines algorithm with a linear kernel. This clearly outperformed all of the aforementioned approaches with an AUC of .94 and 27 false positives and 56 false negatives. We believe that SVM and K-Means worked well because they do not rely on probabilistic likelihoods which can overshadow minority data when predicting. Instead, they create a more geometric representation of the two classification groups allowing for predictions based on a data points location in multidimensional space.

Lastly, we implemented a Multi-Layer Perceptron (MLP) network to classify our data. Our neural network had 100 hidden layers, used the Relu activation function, a constant learning rate, the "adam" solver, and an L2 regularization penalty of .0001. Initially the neural network performed similarly to SVM with around 93% accuracy and an AUC score of .93. Occasionally, the neural network was worse than this since the loss function is not convex

and only local optimums are found. We then chose to scale our data as mentioned previously using the *MinMaxScalar* function. With this as the only change, our neural network then achieved a consistent accuracy of 98% and an AUC score of .99. The MLP network was by far the best approach with 9 false positive classifications and 11 false negative classifications. This was the most accurate overall and therefore the best at detecting customers that would take a loan.




## 6 Conclusion/future works

In this paper, we set out to predict a customer's acceptance of a bank's marketing campaign for a personal loan. Through experimenting with several algorithms, we showed that using Multi-Layer Perceptron Classification was the clear winner in terms of both accuracy and ROC AUC score. Closely behind, the Support Vector Classifier proved to be reliable as well. We believe these algorithms outperformed the rest of the group because the Perceptron and Logistic Regression algorithms probabilistic nature could not handle the intricacy of the unbalanced dataset. Furthermore, the K-means algorithm is perhaps too simplistic in its approach and therefore were unable to accurately classify the minority of people who would accept the personal loan with these methods.

For future work, we would like to explore and research more solutions for classification problems with unbalanced datasets. As mentioned above, a specific area of improvement we would like to pursue is under-sampling and/or over-sampling in our dataset. Additionally, we believe that finding more ways to fine-tune our hyperparameters could be an area of large improvement because it could help the algorithms distinguish the majority class from the minority data group in our dataset.

## 7 Contributions

Thankfully, both team members are on campus and were able to work together on this project in person. As such, we have contributed equally on all aspects of the project: planning, research, coding, and write-ups.

## References


Bao, Lei, et al. "Boosted near-miss under-sampling on SVM ensembles for concept detection in large-scale imbalanced datasets." *Neurocomputing* 172 (2016): 198-206.

Data, Anirban. “Personal Loan Modeling.” *Kaggle*, 2020. https://www.kaggle.com/teertha/personal-loan-modeling/activity

Drummond, Chris, and Robert C. Holte. "C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling." *Workshop on learning from imbalanced datasets*. Vol. 11. Washington DC: Citeseer, 2003.

Eitrich, Tatjana, and Bruno Lang. "Efficient optimization of support vector machine learning parameters for unbalanced datasets." *Journal of computational and applied mathematics* 196.2 (2006): 425-436.

Jeatrakul, Piyasak, Kok Wai Wong, and Chun Che Fung. "Classification of imbalanced data by combining the complementary neural network and SMOTE algorithm." *International Conference on Neural Information Processing*. Springer, Berlin, Heidelberg, 2010.

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

Ye, Leihua. “Classify A Rare Event Using 5 Machine Learning Algorithms.” *Towards Data Science*, 19 Oct. 2019, towardsdatascience.com/classifying-rare-events-using-five-machine-learning-techniques-fab464573233.