Predicting Credit Card Default

BT2101 – Decision making tools and method - Group Project

Contents

Chapter 1: Project Introduction

BACKGROUND:

Credit card is considered as a flexible tool by which you can use bank's money for a short period of time. If you accept a credit card, you agree to pay your bills by the due date listed on your credit card statement. Otherwise, the credit card will be defaulted. When a customer is not able to pay back the loan by the due date and the bank is totally certain that they are not able to collect the payment, it will usually try to sell the loan. After that, if the bank recognizes that they are not able to sell it, they will write it off. This is called a charge-off. This results in significant financial losses to the bank on top of the damaged credit rating of the customer and thus it is an important problem to be tackled. [1]

Predicting accurately which customers are most probable to default represents significant business opportunity for all banks. Bank cards are widely used over the world, which emphasizes the impact of risk prediction to both consumers and banks. Hence, risk prediction is essential for predicting business performance or individual customers' credit risk and to reduce the damage and uncertainty. [1]

PROJECT DESIGN:

Motivation: The fundamental objective is to implement a model to predict default to help bank identify and act on customers with high probability of defaulting to improve their bottom line.

Even though plenty of solutions to the default prediction have been previously done, the scope of this project extends beyond that. Not only we aim to create an effective model to predict default, discussion between different models are emphasized and become one of the top goals in our project.

Dataset: The dataset is obtained from UCI repository.

It contains payment information of 30,000 credit card holders obtained from a bank in Taiwan. Each data sample is described by 23 feature attributes.

In this dataset, we aim to perform binary classification for target feature to be predicted is binary valued 0 (=not default) or 1 (=default).

Chapter 2: Literature review about Machine Learning models group by Tribe

Part 1: Tribes machine learning:

Models in Machine learning can be categorized into different tribes

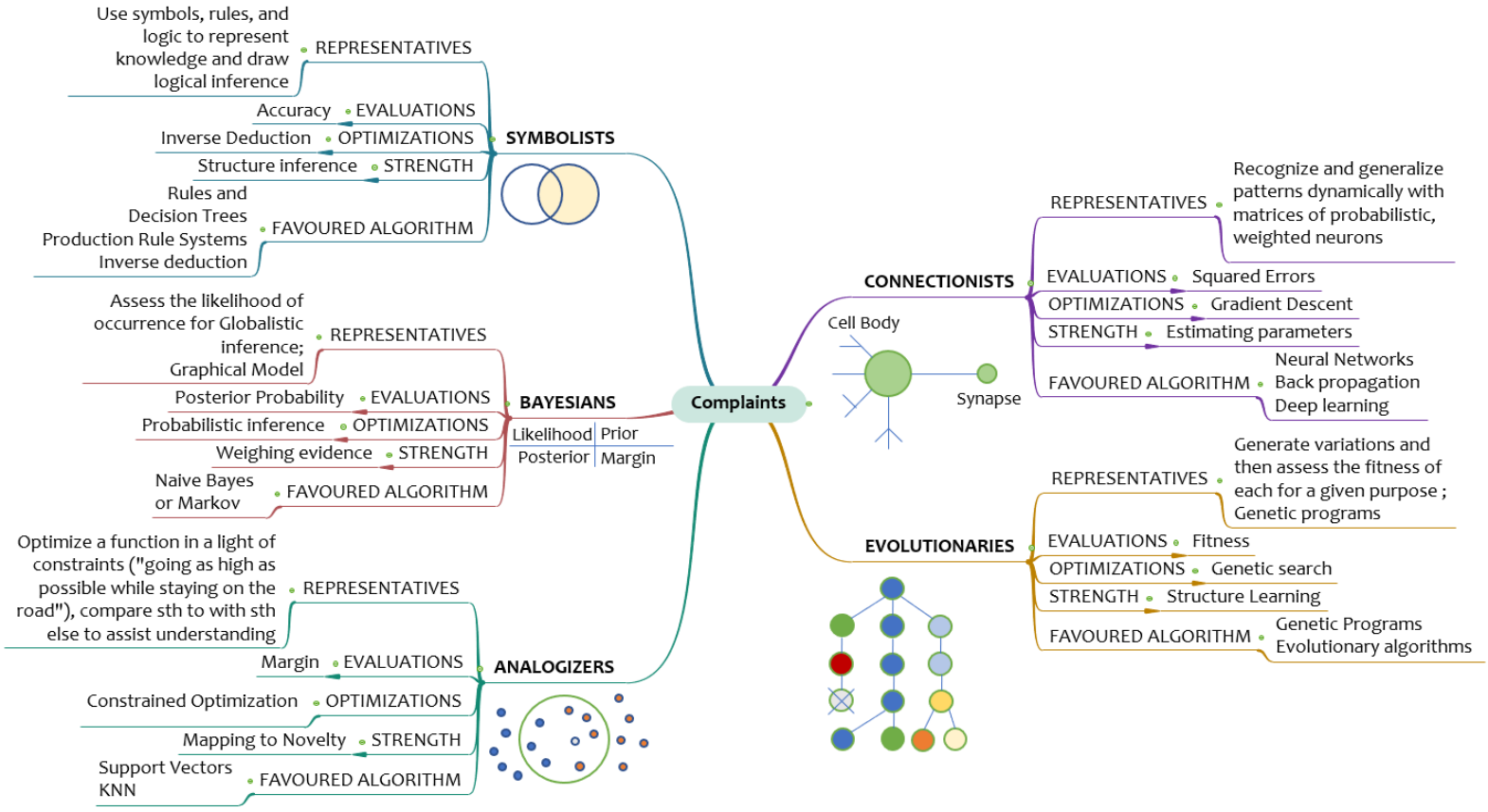


Figure 1: Five different tribes in Machine Learning

Part 2: Tribe #1: Symbolists

DECISION TREE (DT):

A tree can be "learned" by splitting training dataset into subsets based on an features value test.

—Each internal node represents a "test" on a feature resulting on split of current sample.

—At each step algorithm selects feature and a cut–off value that maximises a given metric.

Different metrics exist for regression tree (target is continuous) or classification tree (the target is qualitative).

—This process is repeated on each derived subset in a recursive manner called recursive partitioning. Recursion is completed when subset at a node has all same value of target variable, or when splitting no longer adds value to predictions. This general principle is implemented by many recursive partitioning tree algorithms.

—Decision trees are simple to understand and interpret however they tend to overfit data.

—However, decision trees tend to overfit training set. Leo Breiman propose random forest to deal with this issue.

Decision tree learning algorithm:

Aim: find a small tree minimize J_{train}

Idea: (recursively) choose most significant attribute as root of tree

function $DTL(examples, attribute, default)$ return decision tree

if $examples$ is **empty, return** $default$
else if $attributes$ is **empty, return** $MODE(examples)$
else
  $best \leftarrow CHOOSE.ATTRIBUTE(attributes, examples)$
  $tree \leftarrow$ a **new decision tree with root test** $best$
  for each value v_i of $best$ **do**:
    $examples_i \leftarrow$ {element of examples with $best = v_i$}
    $subtree \leftarrow DTL\begin{pmatrix} examples, attributes - best, \\ MODE(example) \end{pmatrix}$
    add a branch to $tree$ with label v_i **and subtree** $subtree$
  return $tree$

Picking an attribute via entropy ($CHOOSE.ATTRIBUTE$ function):

Idea: a good attribute split training set into subsets that are (ideally) "all positive" or "all negative", hence, purity of subproblems.

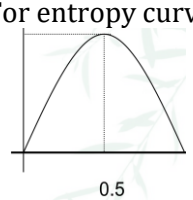
To implement $CHOOSE.ATTRIBUTE$ in decision tree learning enumerated feature sets with C possible values, we first need a concept of purity. We will use entropy:

$$H(X) = -\sum_{i=0}^{C} p \log_2(p)$$

For training set having p positive examples and n negative examples:

$$H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log_2\left(\frac{n}{p+n}\right)$$

For entropy curve: When there are 2 target classes



$$\frac{p}{p+n} = \begin{cases} 0 \text{ when } \frac{p}{p+n} = 0 \\ 1 \text{ when } \frac{p}{p+n} = 0.5 \\ 0 \text{ when } \frac{p}{p+n} = 1 \end{cases}$$

Monotonically increasing in [0; 0.5]

Monotonically decreasing in [0.5; 1]

RANDOM FOREST (RF):

It is a meta estimator that fits several decision tree learners on subsamples of dataset and use averaging to improve predictive accuracy and control over–fitting. A tree–based technique that uses a high number of decision trees built out of randomly selected set of features. Contrary to simple decision tree, it is highly uninterpretable but it's generally good performance. Strong learners composed of multiple trees can be called "forests". Trees that compose a forest can be chosen to be either shallow (few depths) or deep (lot of depths, if not fully grown). [2]

Shallow trees have less variance but higher bias, hence, better choice for sequential methods. Deep trees, on other side, have low bias but high variance, hence, bagging method that is mainly focused at reducing variance. [2]

The random forest approach is a bagging method where deep trees, fitted on bootstrap samples, are combined to produce an output with lower variance. Random forests also use another trick to make multiple fitted trees a bit less correlated with each other's: when growing each tree, instead of only sampling over observations in dataset to generate a bootstrap sample, we also sample over features and keep only a random subset of them to build tree. [2]

Algorithm: The steps are as follows:

Randomly select a subset of data from training set with replacement.

Generate a decision tree from selected data samples. At each node of tree:

a. Randomly select K features without replacement.
b. Split node by searching among K features best cut to maximize information gain.

Repeat steps 1 and 2 N times to obtain N trees.

Final prediction is achieved by majority voting.

GRADIENT BOOSTING:

In gradient boosting, ensemble model we try to build is a weighted sum of weak learners

$$s_L(.) = \sum_{l=1}^{L} c_l \times w_l(.)\ ;\ c_l \text{ are cofficient; } w_l \text{ weak learners}$$

Gradient boosting casts problem into a gradient descent one: at each iteration we fit a weak learner to opposite of gradient of current fitting error with respect to current ensemble model. [2]

First, theoretical gradient descent process over ensemble model can be written $\boldsymbol{s_l(.) = s_{l-1}(.) + c_l \times \nabla_{s_{l-1}} E(s_{l-1})(.)}$ where $\boldsymbol{E(.)}$ **is fitting error** of given model, $\boldsymbol{c_l}$ **is a coefficient** corresponding to **step size** and $\boldsymbol{-\nabla_{s_{l-1}} E(s_{l-1})(.)}$ is **opposite of gradient of fitting error** with respect to **ensemble model at step** $\boldsymbol{l-1}$. This (abstract) opposite of gradient is a function that can only be evaluated for observations in training dataset (for which we know inputs and outputs): these evaluations are called **pseudo-residuals** attached to each observation. [2]

Moreover, even if we know for observations values of these pseudo-residuals, we don't want to add to our ensemble model any kind of function: we only want to add a new instance of weak model. So, natural thing to do is to fit a weak learner to pseudo-residuals computed for each observation. [2]

Finally, **coefficient** $\boldsymbol{c_l}$ is computed following a one-dimensional optimisation process (line-search to obtain best step size c_l). [2]

Illustration of boosting process:

At very beginning of algorithm (first model of sequence), pseudo-residuals are set equal to observation values. Then, we repeat L times (for L models of sequence):

— Fit best possible weak learner to pseudo-residuals (approximate opposite of gradient with respect to current strong learner)

— Compute value of optimal step size that defines by how much we update ensemble model in direction of new weak learner

— Update ensemble model by adding new weak learner multiplied by step size (make a step of gradient descent)

— Compute new pseudo-residuals that indicate, for each observation, in which direction we would like to update next ensemble model predictions

Repeating these steps, we have then build sequentially our L models and aggregate them following a gradient descent approach.

STACKING:

Stacking mainly differ from boosting on two points.

—Stacking uses heterogeneous weak learners (different learning algorithms are combined) whereas bagging and boosting consider mainly homogeneous weak learners.

—Stacking learns to combine base models using a meta-model whereas bagging and boosting combine weak learners following deterministic algorithms. [2]

Idea of stacking is to learn several different weak learners and combine them by training a meta-model to output predictions based on multiple predictions returned by these weak models. So, we need to define two things in order to build our stacking model: L learners we want to fit and meta-model that combines them. [2]

Classification problem example: We can choose as weak learners a KNN classifier, a logistic regression and a SVM, and decide to learn a neural network as meta-model. Then, neural network will take as inputs outputs of our three weak learners and will learn to return final predictions based on it.

Illustration of stacking process: We want to fit a stacking ensemble composed of L weak learners. Then we must follow steps thereafter:

—split training data in two folds

—choose L weak learners and fit them to data of first fold

—for each of L weak learners, make predictions for observations in second fold fit meta-model on second fold, using predictions made by weak learners as inputs

In previous steps, we split dataset in two folds because predictions on data that have been used for training of weak learners are not relevant for training of meta-model.

Thus, an obvious drawback of this split of our dataset in two parts is that we only have half of data to train base models and half of data to train meta-model.

In order to overcome this limitation, we can however follow some kind of "k-fold cross-training" approach (similar to what is done in k-fold cross-validation) such that all observations can be used to train meta-model: for any observation, prediction of weak learners are done with instances of these weak learners trained on $k-1$ folds that do not contain considered observation.

Part 3: Tribe #2: Bayesians

NAÏVE BAYES CLASSIFICATION:

Naive Bayes models are a group of extremely fast and simple classification algorithms that are often suitable for very high dimensional datasets. Because they are so fast and have so few tuneable parameters, they end up being very useful as a quick and dirty baseline for a classification problem. This section will focus on an intuitive explanation of how naive Bayes classifiers work, followed by a couple examples of them in action on some datasets. [3]

Naive Bayes classifiers are built on Bayesian classification methods. These rely on Bayes' theorem, which is an equation describing the relationship of conditional probabilities of statistical quantities. In Bayesian classification, we're interested in finding the probability of a label given some observed features, which we can write as $P(L|features)$.

Bayes' theorem tells us how to express this in terms of quantities we can compute more directly:

$$P(L|features) = \frac{P(features|L)P(L)}{P(features)}$$

If we are trying to decide between two labels— let's call them L_1 and L_2 then one way to make this decision is to compute the ratio of the posterior probabilities for each label:

$$\frac{P(L_1|feaures)}{P(L_1|features)} = \frac{P(features|L_1)P(L_1)}{P(features|L_2)P(L_2)}$$

Gaussian Naïve Bayes: in this classifier, the assumption is data from each label is drawn from a simple Gaussian distribution.

Multinomial Naïve Bayes: classifier where the features are assumed to be generated from a simple multinomial distribution. The multinomial distribution describes the probability of observing counts among a number of categories, and thus multinomial naive Bayes is most appropriate for features that represent counts or count rates.

The idea is precisely the same as before, except that instead of modeling the data distribution with the best-fit Gaussian, we model the data distribution with a best-fit multinomial distribution.

Special cases for Naïve Bayes well performance:

Naïve Bayes tends to perform especially well in one of the following situations:

— When the Naïve assumption actually match the data (very rare in practice)

—For very well-separated categories, when model complexity is less important

—For very high-dimensional data, when model complexity is less important

The last two points seem distinct, but they actually are related: as the dimension of a dataset grows, it is much less likely for any two points to be found close together (after all, they must be close in every single dimension to be close overall). This means that clusters in high dimensions tend to be more separated, on average, than clusters in low dimensions, assuming the new dimensions actually add information. For this reason, simplistic classifiers like naïve Bayes tend to work as well or better than more complicated classifiers as the dimensionality grows: once you have enough data, even a simple model can be very powerful. [3]

Part 4: Tribe #3: Analogizers

SUPPORT VECTOR MACHINE (LINEAR SUPPORT VECTOR CLASSIFIER)

Support vector machines (SVMs) are a particularly powerful and flexible class of supervised algorithms for both classification and regression

In classification problem, we will try to find a hyperplane to separate data samples.

In order to achieve that, we will find a hyperplane whose distances from points nearest to it have maximum values.

On the overall picture, we will find hyperplane that maximize the geometric margin to itself.

We can pose following optimization problem:

$$\max_{\gamma,w,b} \gamma \text{ such that}$$

$$y_i(w^T x_i + b) \geq \gamma;\ i = 1 \dots m;\ ||w|| = 1$$

❋We want to maximise γ subject to each training example having functional margin at least γ. The $||w|| = 1$ constraint moreover ensures that functional margin equals to geometric margin, so we guaranteed that all geometric margins are at least γ, hence, solving problem will result in (w, b) with largest possible geometric margin with respect to training set.

❋If we could solve optimization problem above, we'd be done. But "$||w|| = 1$" constraint is a nasty (nonconvex) 1 and this problem certainly isn't in any format that we can plug into standard optimization software to solve, hence, try transforming problem into a nicer one.:

$$\max_{\hat{\gamma},w,b} \frac{\hat{\gamma}}{||w||} \textit{ such that } y_i(w^T x_i + b)$$

❋We can add an arbitrary scaling constraint on w and b without changing anything. We will introduce scaling constraint that functional margin of w, b with respect to training set must be 1: $\hat{\gamma} = 1$.

❋Since multiplying w and b by some constant results in functional margin being multiplied by that same constant, this is indeed a **scaling constraint** and can be satisfied by rescaling w, b, hence, $\max \hat{\gamma}/||w|| = \max 1/||w||$

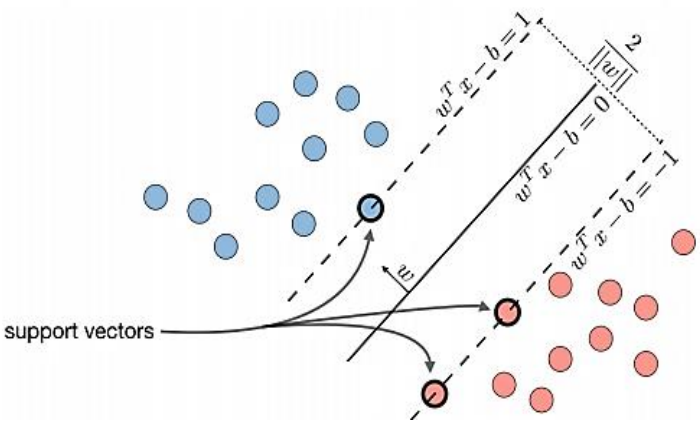
❋The **optimal margin classifier h is** such that:

$$h(x) = sign(w^T x - b)$$

where (w, b) is solution of above optimization problem:

$$\min_{\gamma,w,b} \frac{1}{2}||w||^2 \text{ such that } y_i(w^T x_i - b) \geq 1, i = 1..m$$

We can write as: $g_i(w) = -y_i(w^T x_i + b) + 1 \leq 0$



Remarks: **line is defined as** $(w^T x - b)$

To make algorithm work for **non–linearly separable** datasets as well as be less sensitive to outliers, we reformulate our optimization (ℓ_1 **regularization**) as follows:

$$\min_{\gamma,w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{m} \xi_i \textit{ such that}$$

$$y_i(w^T x_i + b) \geq 1 - \xi_i \textit{ and } \xi_i \geq 0;\ i = 1 \dots m;$$

Parameter C controls relative weighting between twin goals of making $||w||^2$ small and of ensuring that most examples have functional margin ≥ 1

As before we can form Lagrangian:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2}w^T w + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i [y_i(x^T w + b) - 1 + \xi_i] - \sum_{i=1}^{m} r_i \xi_i$$

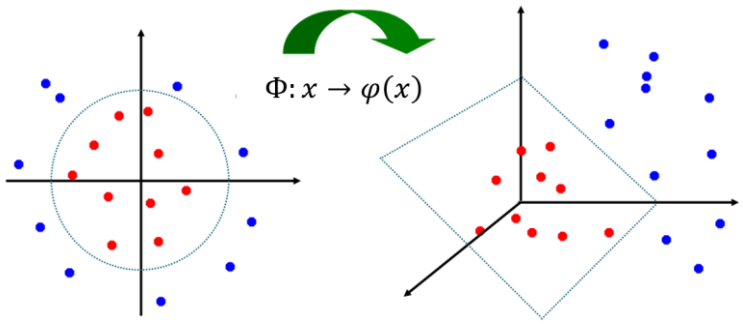
where $\alpha_i's$ and $r_i's$ are our Lagrange multipliers (constrained to be ≥ 0). After setting derivatives of dual with respect to *w and b* to zero as before, substituting them back in and simplifying we obtain

DUAL OPTIMIZATION PROBLEM:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

$$\textit{such that } 0 \leq \alpha_i \leq C, i = 1 \dots m \textit{ and } \sum_{i=1}^{m} \alpha_i y_i = 0$$

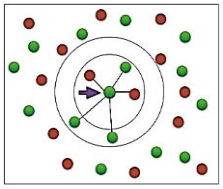
KERNEL TRICKS: General idea: original input space can be mapped to some higher dimensional feature space where training set is separable



Common types of inner product kernel for SVM $K(x^T x_i)$

Type	Kernel	Comment
Linear	$x^T x_i$	None
Polynomial learning machine	$(x^T x_i + 1)^p$	p is specified a priori by user.
Radial basis function network (**Gaussian** kernel)	$\exp\left(-\frac{1}{\sigma^2}\|x - x_i\|^2\right)$	the width σ^2 is specified a priori by user
Two–layer perceptron (sigmoid)	$\tanh(\beta_0 x^T x_i + \beta_1)$	only for some values of β_0 and β_1

K – NEAREST NEIGHBOURS (K-NN):



This model will look at 'k' closest labelled data points and then we apply majority vote

Finds records in database have similar numerical values of set of predictor variables

❋Measure Euclidean distance between records in training data set. Nearest neighbour to record in training data set is 1 that that has smallest distance from it.

❋If $k = 1$, hence, $1 - NN$ rule classifies record in same category as its nearest neighbour.

❋$k - NN$ rule finds k–Nearest Neighbours in training data set to each record we want to classify;, hence, assigns classification as classification of majority of k nearest neighbours.

❋Typically, various values of k are used; then results inspected to determine which is best.

MODEL COMPLEXITY: Small value of K means that noise will have higher influence on result → more complex model → can lead to *overfitting*

Large value of K makes it computationally expensive; defeats basic idea behind KNN (that points that are near might have similar classes) → Smoother decision boundary, hence, less complex model

HOW TO CHOOSE VALUE K?

Higher k, hence, higher bias, hence, Complex surface

Lower k, hence, lower variance, hence, Smoother surface

Selecting value of K in K–nearest neighbour is most critical prob.

$\Longrightarrow$ Simple approach to select k is $k = n^{1/2}$.

Part 5: Tribe #4: Connectionists

LOGISTIC REGRESSION

Logistic regression is a statistical model widely used in binary classification problem.

The output of the problem is a probability.

Assumptions made in Logistic regression: [4]

To make **inference** on model parameters. In other words, infer properties about unknown population coefficients β_0 and β_1 from sample $(X_1, Y_1), \dots (X_n, Y_n)$

—Linearity (of relationship between Y and X):

Residual vs. fitted – *Find straight horizontal line*

$$\mathbb{E}[\boldsymbol{Y}|\boldsymbol{X} = \boldsymbol{x}] = \boldsymbol{\beta_0} + \boldsymbol{\beta_1 x}$$

—Normality of errors = Errors (e; residuals) are normally distributed: Normal Q–Q plot – Look for linear relationship

$$\boldsymbol{\varepsilon_i} \sim \boldsymbol{\mathcal{N}}\left(\mathbf{0}, \boldsymbol{\sigma^2}\right) \textbf{ for } \boldsymbol{i} = \mathbf{1} \dots \boldsymbol{N}$$

—Homoscedasticity = Constant / Equal variance of errors (e) for all values of X = Impact of X on Y is same for all X values: Residual vs. fitted; Scale–location – Look for straight horizontal line

$$\boldsymbol{Var}[\boldsymbol{\varepsilon_i}] = \boldsymbol{\sigma^2} \textbf{ with } \boldsymbol{\sigma^2} \textbf{ constant for } \boldsymbol{i} = \mathbf{1} \dots \boldsymbol{n}$$

—Independence of errors = There is no correlation between errors (e) calculated from regression model – Need additional plot/test

• Panel/time–series data need to check

• Cross–sectional data – data is collected only once, from different individuals/entities

• Panel/time–series data – data is collected multiple times from each individual/entity

$\varepsilon_1, \varepsilon_2, \dots, \varepsilon_n$ are independent (uncorrelated)

$\mathbb{E}\left[\varepsilon_i \varepsilon_j\right] = 0 \; ; i \neq j \rightarrow$ assumed to be normal

Good-to-1-line summary of model (independence assume) $Y|X = x \sim \mathcal{N}(\beta_0 + \beta_1 x \; ; \sigma^2)$

Algorithm:

1. Initialise weights at $t = 0 \; to \; \theta(0)$

2. Do:

Compute gradient for cost function: $\nabla(t) = \nabla J\big(\theta(t)\big)$

$$= -\frac{1}{m} X^T (h_\theta(X) - y) \; = -\frac{1}{m} \sum_{j=1}^{m} \frac{y^{(j)} x^{(j)}}{1 + \exp(y^{(j)} \theta(t)^T x^{(j)})}$$

$\nabla J\big(\theta(t)\big)$ is gradient and should be $n-$ dimensional vector, where $i - th$ element of $\nabla J\big(\theta(t)\big)$ is partial derivative of los function $J(\theta) \; with \; respect \; to \; \theta_i$

// Move in direction $v(t) = -\nabla(t)$

Update weights $\boldsymbol{\theta(t+1)} = \boldsymbol{\theta(t)} - \boldsymbol{\alpha \nabla_{\theta(t)} J\big(\theta(t)\big)}$ where $\theta(t)$ represents weights in $t - th$ iteration and α denotes learning rate

Continue next iteration, until it is time to stop

3. Return final weights θ^*

STOCHASTIC GRADIENT DESCENT (SGD)

Gradient Descent models are used to reduce the loss, or residuals. However, it takes very slow when dataset is in huge size, the amount of computations is impressively high.

Say we have 10,000 data points and 10 features. The sum of squared residuals consists of as many terms as there are data points, so 10000 terms in our case. We need to compute the derivative of this function with respect to each of the features, so in effect we will be doing 10000 * 10 = 100,000 computations per iteration. It is common to take 1000 iterations; in effect we have 100,000 * 1000 = 100000000 computations to complete the algorithm. That is pretty much an overhead and hence gradient descent is slow on huge data.

"Stochastic" gradient descent model comes to rescue.

It will randomly picks one data point from the whole data set at each iteration to reduce the computations enormously. [5]

Chapter 3: Exploratory data analysis

Part 1: Describing data

The data consists of 30,000 customers and 24 columns of variables. Each sample corresponds to a single customer. The columns consist of the following variables:

- ***DEFAULT*** (Yes or no) as a binary response variable – Categorical variable
- ***BALANCE LIMIT*** – Continuous variable
- ***SEX*** encoded as 1 or 2 – Categorical variable
- ***EDUCATION*** encoded as 0 → 6
- ***MARRIAGE*** shows Marital status encoded as 0 → 3 – Categorical variable
- ***AGE*** (Years) – Discrete variable
- ***PAY(PAY_0, PAY_1, …)*** shows Payment status – Categorical variable
- ***BILL_AMT(BILL_AMT1, BILL_AMT2, …)*** shows Bill amount – Continuous variable
- ***PAY_AMT(PAY_AMT1, PAY_AMT2, …)*** shows Payment amount

We renamed last column to "default" in short

The variables Balance limit, Age, Sex, Education, Marital status, Employer, and Location are defined as demographic variables, since they describe a demography of customers and are available for new customers.

	count	mean	std	min	25%	50%	75%	max
LIMIT_BAL	30000.0	167484.322667	129747.661567	10000.0	50000.00	140000.0	240000.00	1000000.0
SEX	30000.0	1.603733	0.489129	1.0	1.00	2.0	2.00	2.0
EDUCATION	30000.0	1.853133	0.790349	0.0	1.00	2.0	2.00	6.0
MARRIAGE	30000.0	1.551867	0.521970	0.0	1.00	2.0	2.00	3.0
AGE	30000.0	35.485500	9.217904	21.0	28.00	34.0	41.00	79.0
PAY_0	30000.0	-0.016700	1.123802	-2.0	-1.00	0.0	0.00	8.0
PAY_2	30000.0	-0.133767	1.197186	-2.0	-1.00	0.0	0.00	8.0
PAY_3	30000.0	-0.166200	1.196868	-2.0	-1.00	0.0	0.00	8.0
PAY_4	30000.0	-0.220667	1.169139	-2.0	-1.00	0.0	0.00	8.0
PAY_5	30000.0	-0.266200	1.133187	-2.0	-1.00	0.0	0.00	8.0
PAY_6	30000.0	-0.291100	1.149988	-2.0	-1.00	0.0	0.00	8.0
BILL_AMT1	30000.0	51223.330900	73635.860576	-165580.0	3558.75	22381.5	67091.00	964511.0
BILL_AMT2	30000.0	49179.075167	71173.768783	-69777.0	2984.75	21200.0	64006.25	983931.0
BILL_AMT3	30000.0	47013.154800	69349.387427	-157264.0	2666.25	20088.5	60164.75	1664089.0
BILL_AMT4	30000.0	43262.948967	64332.856134	-170000.0	2326.75	19052.0	54506.00	891586.0
BILL_AMT5	30000.0	40311.400967	60797.155770	-81334.0	1763.00	18104.5	50190.50	927171.0
BILL_AMT6	30000.0	38871.760400	59554.107537	-339603.0	1256.00	17071.0	49198.25	961664.0
PAY_AMT1	30000.0	5663.580500	16563.280354	0.0	1000.00	2100.0	5006.00	873552.0
PAY_AMT2	30000.0	5921.163500	23040.870402	0.0	833.00	2009.0	5000.00	1684259.0
PAY_AMT3	30000.0	5225.681500	17606.961470	0.0	390.00	1800.0	4505.00	896040.0
PAY_AMT4	30000.0	4826.076867	15666.159744	0.0	296.00	1500.0	4013.25	621000.0
PAY_AMT5	30000.0	4799.387633	15278.305679	0.0	252.50	1500.0	4031.50	426529.0
PAY_AMT6	30000.0	5215.502567	17777.465775	0.0	117.75	1500.0	4000.00	528666.0
default	30000.0	0.221200	0.415062	0.0	0.00	0.0	0.00	1.0

Figure 2: Description of Dataset

Part 2: Distribution plot for each column according to "default" values

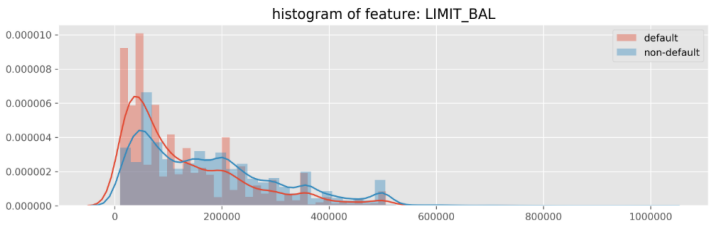


Figure 3: Distribution plot of LIMIT_BAL

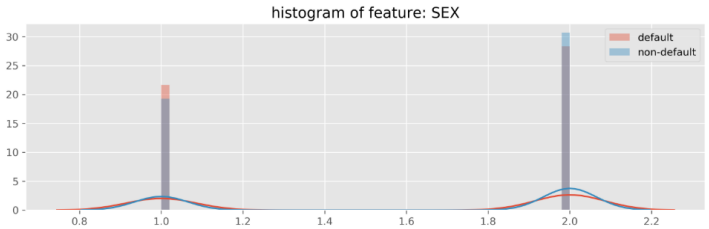


Figure 4: Distribution plot of SEX

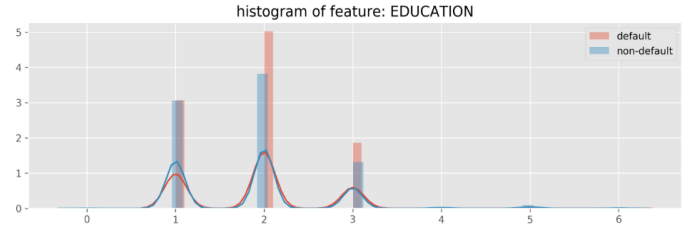


Figure 5: Distribution plot of EDUCATION

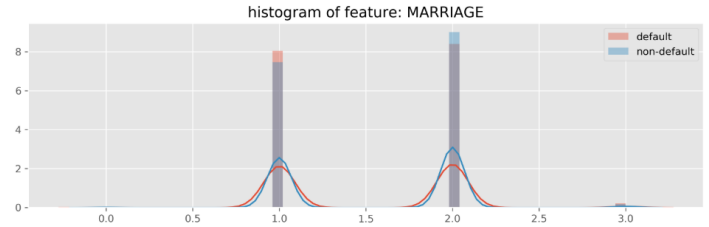


Figure 6: Distribution plot of MARRIAGE

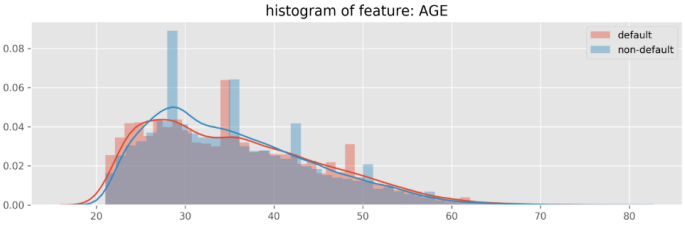


Figure 7: Distribution plot of AGE

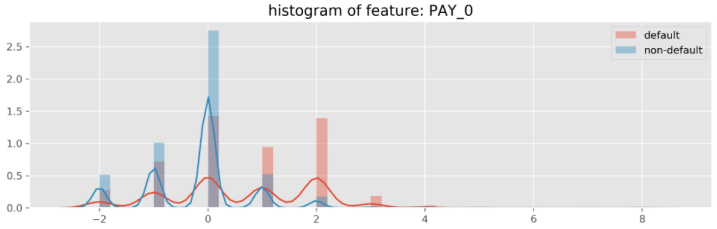


Figure 8: Distribution plot of PAY_0

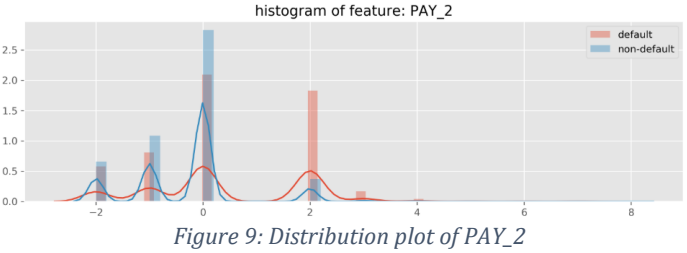


Figure 9: Distribution plot of PAY_2

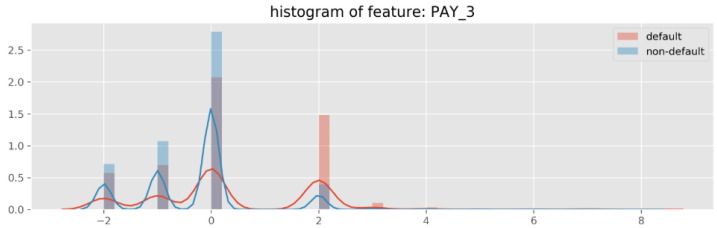


Figure 10: Distribution plot of PAY_5

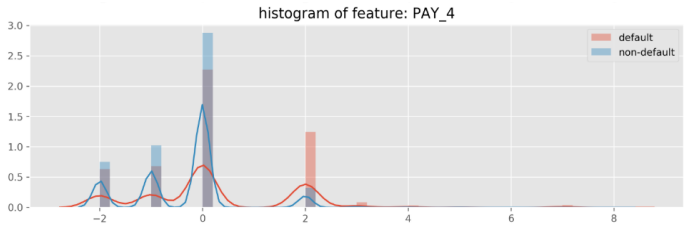


Figure 11: Distribution plot of PAY_4

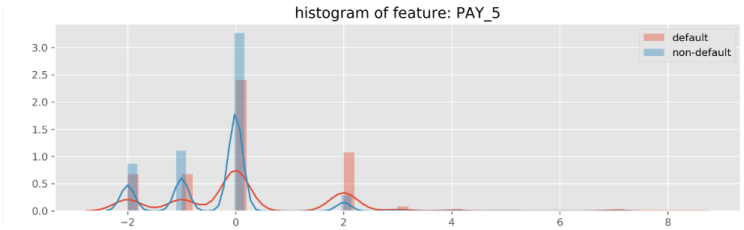


Figure 12: Distribution plot of PAY_5

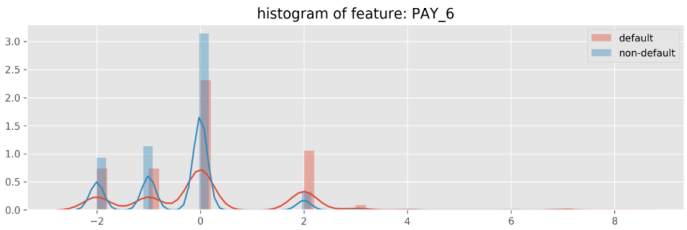


Figure 13: Distribution plot of PAY_6

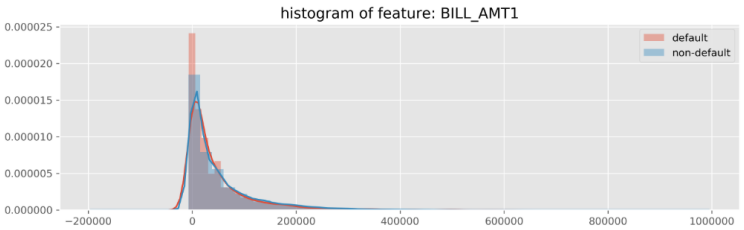


Figure 14: Distribution plot of BILL_AMT1

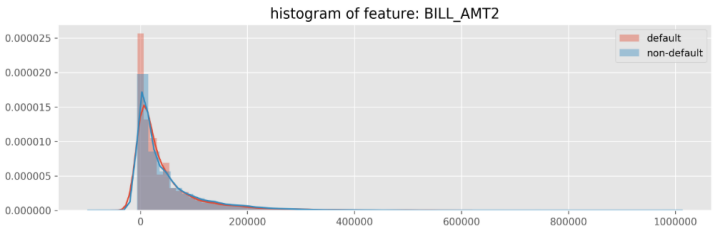


Figure 15: Distribution plot of BILL_AMT2

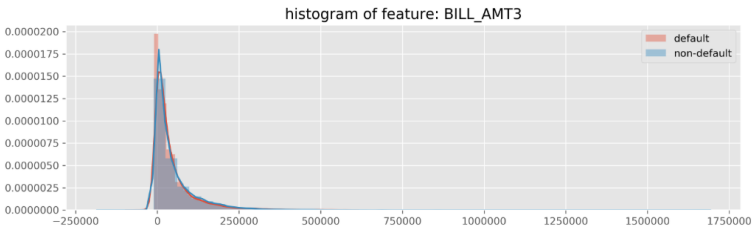


Figure 16: Distribution plot of BILL_AMT3

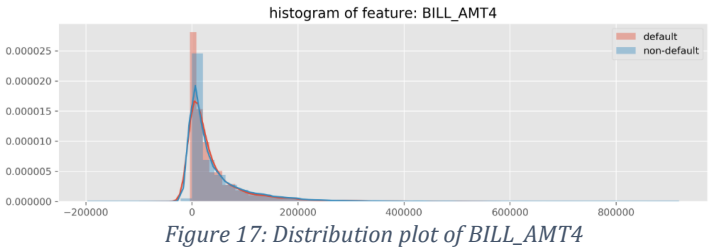


Figure 17: Distribution plot of BILL_AMT4

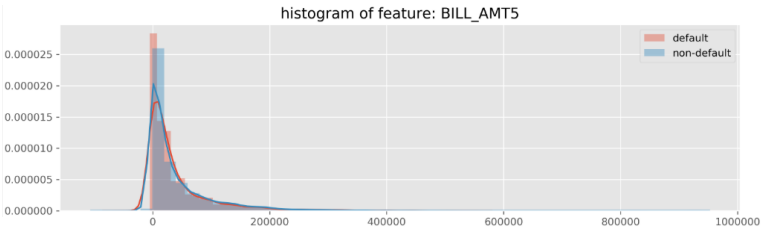


Figure 18: Distribution plot of PAY_AMT1

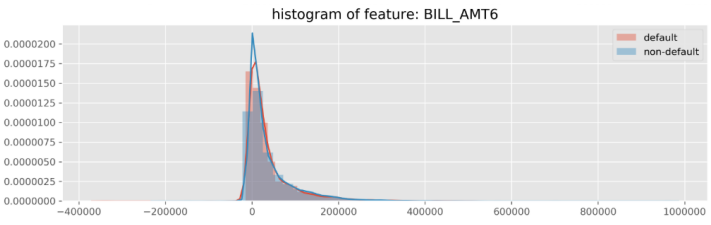


Figure 19: Distribution plot of BILL_AMT6

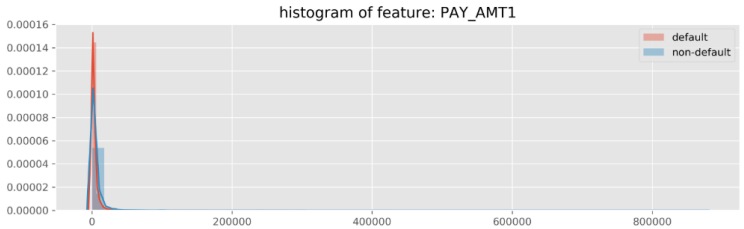


Figure 20: Distribution plot of PAY_AMT1

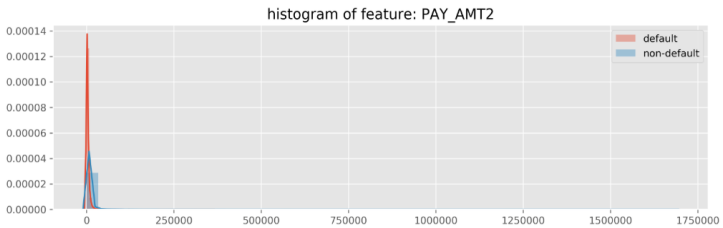


Figure 21: Distribution plot of PAY_AMT2

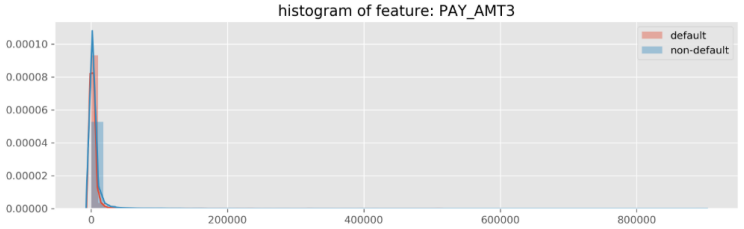


Figure 22: Distribution plot of PAY_AMT3

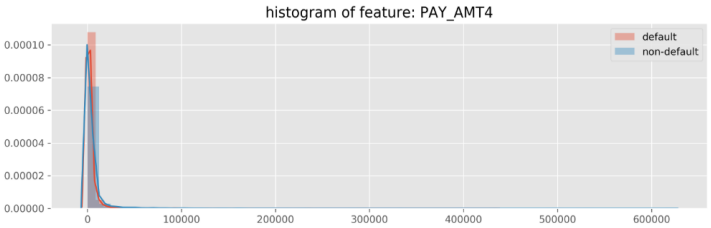


Figure 23: Distribution plot of PAY_AMT4

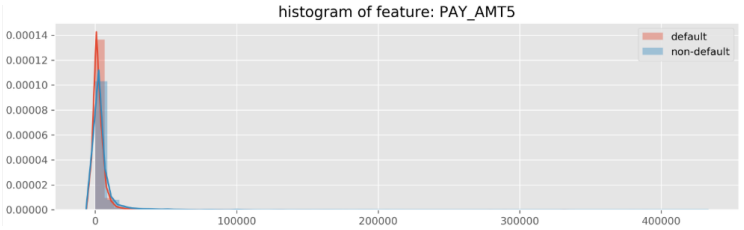


Figure 24: Distribution plot of PAY_AMT5

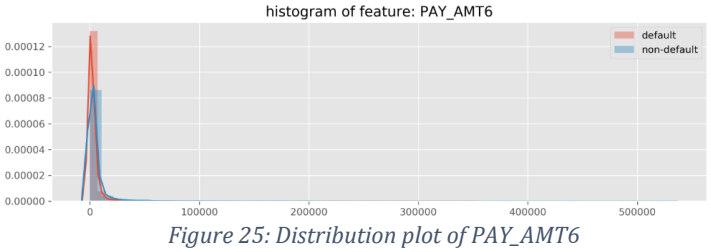


Figure 25: Distribution plot of PAY_AMT6

Part 3: Correlation analysis

F1	AGE	Bill Amt1	Bill Amt2	Bill Amt3	Bill Amt4	Bill Amt5	Bill Amt6	Education	Limit Bal	Marriage	Pay 0	Pay 2	Pay 3	Pay 4	Pay 5	Pay 6	Pay Amt1	Pay Amt2	Pay Amt3	Pay Amt4	Pay Amt5	Pay Amt6	SEX	Default
AGE	1.000	0.056	0.054	0.054	0.051	0.049	0.048	0.175	0.145	-0.414	-0.039	-0.050	-0.053	-0.050	-0.054	-0.049	0.026	0.022	0.029	0.021	0.023	0.019	-0.091	0.014
BILL_AMT1	0.056	1.000	0.951	0.892	0.860	0.830	0.803	0.024	0.285	-0.023	0.187	0.235	0.208	0.203	0.207	0.207	0.140	0.099	0.157	0.158	0.167	0.179	-0.034	-0.020
BILL_AMT2	0.054	0.951	1.000	0.928	0.892	0.860	0.832	0.019	0.278	-0.022	0.190	0.235	0.237	0.226	0.227	0.227	0.280	0.101	0.151	0.147	0.158	0.174	-0.031	-0.014
BILL_AMT3	0.054	0.892	0.928	1.000	0.924	0.884	0.853	0.013	0.283	-0.025	0.180	0.224	0.227	0.245	0.243	0.241	0.244	0.317	0.130	0.143	0.180	0.182	-0.025	-0.014
BILL_AMT4	0.051	0.860	0.892	0.924	1.000	0.940	0.901	0.000	0.294	-0.023	0.179	0.222	0.227	0.246	0.272	0.266	0.233	0.208	0.300	0.130	0.160	0.178	-0.022	-0.010
BILL_AMT5	0.049	0.830	0.860	0.884	0.940	1.000	0.946	-0.008	0.296	-0.025	0.181	0.221	0.225	0.243	0.270	0.291	0.217	0.181	0.252	0.293	0.142	0.164	-0.017	-0.007
BILL_AMT6	0.048	0.803	0.832	0.853	0.901	0.946	1.000	-0.009	0.290	-0.021	0.177	0.219	0.222	0.239	0.263	0.285	0.200	0.173	0.234	0.250	0.308	0.115	-0.017	-0.005
EDUCATION	0.175	0.024	0.019	0.013	0.000	-0.008	-0.009	1.000	-0.219	-0.143	0.105	0.122	0.114	0.109	0.098	0.082	-0.037	-0.030	-0.040	-0.038	-0.040	-0.037	0.014	0.028
LIMIT_BAL	0.145	0.285	0.278	0.283	0.294	0.296	0.290	-0.219	1.000	-0.108	-0.271	-0.296	-0.286	-0.267	-0.249	-0.235	0.195	0.178	0.210	0.203	0.217	0.220	0.025	-0.154
MARRIAGE	-0.414	-0.023	-0.022	-0.025	-0.023	-0.025	-0.021	-0.143	-0.108	1.000	0.020	0.024	0.033	0.033	0.036	0.034	-0.006	-0.008	-0.004	-0.013	-0.001	-0.007	-0.031	-0.024
PAY_0	-0.039	0.187	0.190	0.180	0.179	0.181	0.177	0.105	-0.271	0.020	1.000	0.672	0.574	0.539	0.509	0.475	-0.079	-0.070	-0.071	-0.064	-0.058	-0.059	-0.058	0.325
PAY_2	-0.050	0.235	0.235	0.224	0.222	0.221	0.219	0.122	-0.296	0.024	0.672	1.000	0.767	0.662	0.623	0.576	-0.081	-0.059	-0.056	-0.047	-0.037	-0.037	-0.071	0.264
PAY_3	-0.053	0.208	0.237	0.227	0.227	0.225	0.222	0.114	-0.286	0.033	0.574	0.767	1.000	0.777	0.687	0.633	0.001	-0.067	-0.053	-0.046	-0.036	-0.036	-0.066	0.235
PAY_4	-0.050	0.203	0.226	0.245	0.246	0.243	0.239	0.109	-0.267	0.033	0.539	0.662	0.777	1.000	0.820	0.716	-0.009	-0.002	-0.069	-0.043	-0.034	-0.027	-0.060	0.217
PAY_5	-0.054	0.207	0.227	0.243	0.272	0.270	0.263	0.098	-0.249	0.036	0.509	0.623	0.687	0.820	1.000	0.817	-0.006	-0.003	0.009	-0.058	-0.033	-0.023	-0.055	0.204
PAY_6	-0.049	0.207	0.227	0.241	0.266	0.291	0.285	0.082	-0.235	0.034	0.475	0.576	0.633	0.716	0.817	1.000	-0.001	-0.005	0.006	0.019	-0.046	-0.025	-0.044	0.187
PAY_AMT1	0.026	0.140	0.280	0.244	0.233	0.217	0.200	-0.037	0.195	-0.006	-0.079	-0.081	0.001	-0.009	-0.006	-0.001	1.000	0.286	0.252	0.200	0.148	0.186	0.000	-0.073
PAY_AMT2	0.022	0.099	0.101	0.317	0.208	0.181	0.173	-0.030	0.178	-0.008	-0.070	-0.059	-0.067	-0.002	-0.003	-0.005	0.286	1.000	0.245	0.180	0.181	0.158	-0.001	-0.059
PAY_AMT3	0.029	0.157	0.151	0.130	0.300	0.252	0.234	-0.040	0.210	-0.004	-0.071	-0.056	-0.053	-0.069	0.009	0.006	0.252	0.245	1.000	0.216	0.159	0.163	-0.009	-0.056
PAY_AMT4	0.021	0.158	0.147	0.143	0.130	0.293	0.250	-0.038	0.203	-0.013	-0.064	-0.047	-0.046	-0.043	-0.058	0.019	0.200	0.180	0.216	1.000	0.152	0.158	-0.002	-0.057
PAY_AMT5	0.023	0.167	0.158	0.180	0.160	0.142	0.308	-0.040	0.217	-0.001	-0.058	-0.037	-0.036	-0.034	-0.033	-0.046	0.148	0.181	0.159	0.152	1.000	0.155	-0.002	-0.055
PAY_AMT6	0.019	0.179	0.174	0.182	0.178	0.164	0.115	-0.037	0.220	-0.007	-0.059	-0.037	-0.036	-0.027	-0.023	-0.025	0.186	0.158	0.163	0.158	0.155	1.000	-0.003	-0.053
SEX	-0.091	-0.034	-0.031	-0.025	-0.022	-0.017	-0.017	0.014	0.025	-0.031	-0.058	-0.071	-0.066	-0.060	-0.055	-0.044	0.000	-0.001	-0.009	-0.002	-0.002	-0.003	1.000	-0.040
default	0.014	-0.020	-0.014	-0.014	-0.010	-0.007	-0.005	0.028	-0.154	-0.024	0.325	0.264	0.235	0.217	0.204	0.187	-0.073	-0.059	-0.056	-0.057	-0.055	-0.053	-0.040	1.000

Figure 26: Correlation matrix of the entire dataset featuring all variables.

Second one shows the correlation between columns by showing the correlation between 2 columns is the size of square.

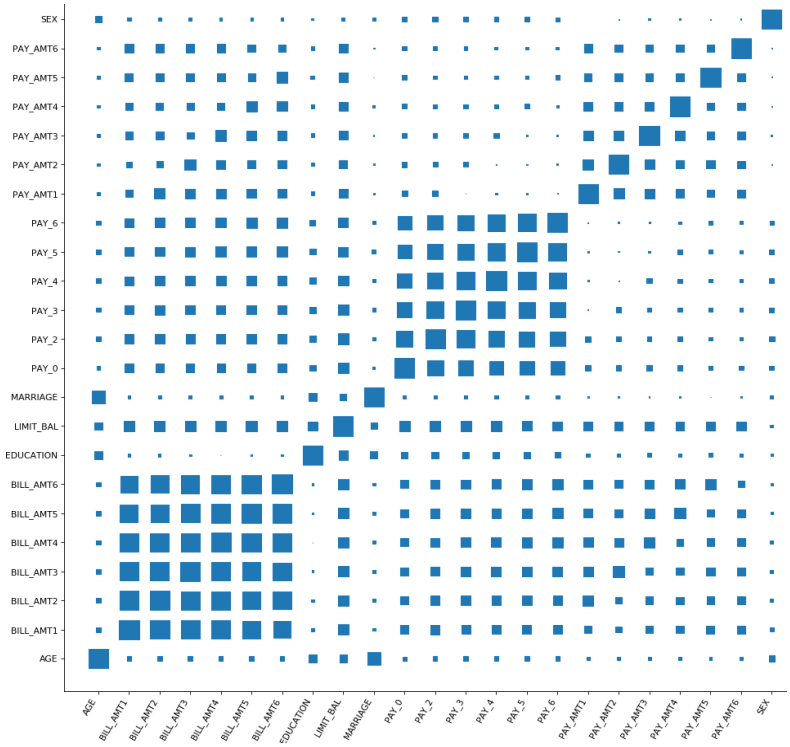


Figure 27: Correlation matrix shown by size of square cells

Most importantly, the only variable with a notable correlation with default is payment status (PAY columns)

The next highest correlation with default occurs with balance limit at -0.15, which itself is somewhat negatively correlated with payment statuses. These measures indicate that customers with lower balance limits have more delays in payments and are more likely to default. This argument is also supported by Figure 3, which you can see defaulting customers have low balance limits, and payment status (PAY columns) are correlated with default.

There is a positive correlation coefficient of 0.14 between limit balance and age. And there is a negative −0.23 correlation between limit balance and education. Age seems do not have much correlation with default, but based on figure 7, defaulting customer groups tends to grow with age.

Education is negatively correlated with Balance limit.

Bills amount and payment amounts are moderately positively correlation with balance limit, as expected, since customer have higher credit are more likely to spend more.

Chapter 4: Data pre-processing

1. SPECIAL TREATMENT FOR AGE COLUMN:

For the age columns, we will replace the Age column with $\log_{10}(age + 1)$

2. DATA PROCESSING USING MIN-MAX SCALER WITH RANGE $(0, 1)$:

Before we apply any method, such as PCA, or feature selection, data should be scaled and non-negative:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4
0	0.010101	1.0	0.333333	0.333333	0.051724	0.4	0.4	0.1	0.1	0.0	0.0	0.149982	0.069164	0.086723	0.160138
1	0.111111	1.0	0.333333	0.666667	0.086207	0.1	0.4	0.2	0.2	0.2	0.4	0.148892	0.067858	0.087817	0.163220
2	0.080808	1.0	0.333333	0.666667	0.224138	0.2	0.2	0.2	0.2	0.2	0.2	0.172392	0.079532	0.093789	0.173637
3	0.040404	1.0	0.333333	0.333333	0.275862	0.2	0.2	0.2	0.2	0.2	0.2	0.188100	0.111995	0.113407	0.186809
4	0.040404	0.0	0.333333	0.333333	0.620690	0.1	0.2	0.1	0.2	0.2	0.2	0.154144	0.071601	0.106020	0.179863

BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6
0.080648	0.260979	0.000000	0.000409	0.000000	0.000000	0.000000	0.000000
0.084074	0.263485	0.000000	0.000594	0.001116	0.001610	0.000000	0.003783
0.095470	0.272928	0.001738	0.000891	0.001116	0.001610	0.002345	0.009458
0.109363	0.283685	0.002290	0.001199	0.001339	0.001771	0.002506	0.001892
0.099633	0.275681	0.002290	0.021779	0.011160	0.014493	0.001615	0.001284

3. DATA PROCESSING WITH PRINCIPLE COMPONENT ANALYSIS: PCA (Principal Component Analysis) mainly using to reduce the size of the feature space while retaining as much of the information as possible. In order to decide how many components, we should reduce to, we will plot a graph for Cumulative Summation of the Explained Variance

Choosing $n\_components$ when performing PCA (model did not exclude any columns):

```
n_component = 1 → 0.5528528428073621
n_component = 2 → 0.6964800622705274
n_component = 3 → 0.7904022739803037
n_component = 4 → 0.8455618207700792
n_component = 5 → 0.8839247959378522
n_component = 6 → 0.9166721899058589
n_component = 7 → 0.9380700545764724
n_component = 8 → 0.9562071457419538
n_component = 9 → 0.9687710243399839
n_component = 10 → 0.9771386310509677
n_component = 11 → 0.9833384310033771
n_component = 12 → 0.9875115316576303
n_component = 13 → 0.9906108232655144
n_component = 14 → 0.9931429460461323
n_component = 15 → 0.9954486759841078
n_component = 16 → 0.9968243940331725
n_component = 17 → 0.9978175525224455
n_component = 18 → 0.9986091155659746
n_component = 19 → 0.9991613442997643
n_component = 20 → 0.9994913120632744
n_component = 21 → 0.9997417197558053
n_component = 22 → 0.999912543541501
n_component = 23 → 1.0
```

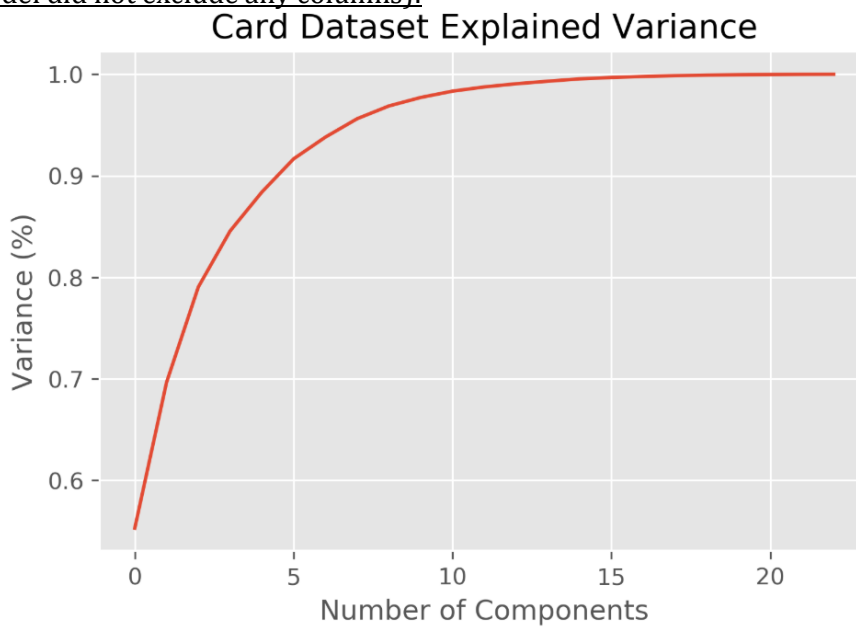


Figure 28: Number of components in PCA vs. Explained Variance of the dataset

Using this graph, we can decide to set $n\_components = 13$ in PCA

After applying PCA with $n\_components = 13$, below are dimensional reduction for first few samples:

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	default
0	-0.404232	-0.008005	-0.002597	-0.212566	-0.211858	0.142288	-0.353981	-0.055854	-0.037792	0.047794	0.098949	0.061357	0.008969	NaN
1	-0.391889	0.209224	-0.196988	-0.063890	-0.009020	0.000373	0.104670	-0.040404	0.036161	0.249256	0.028769	0.093556	-0.011375	1.0
2	-0.392642	0.084095	-0.126567	-0.077123	0.080733	-0.050136	0.021148	0.006008	0.000249	0.006144	0.001706	0.000031	-0.002720	1.0
3	-0.394038	0.068885	0.154030	-0.108658	-0.085688	0.014440	0.002314	0.077498	0.005925	0.001265	0.001589	-0.000635	-0.005089	0.0
4	0.610813	-0.089679	0.334445	-0.141113	0.074264	-0.165212	0.070620	0.116729	0.010839	0.046725	0.077771	0.028073	0.007249	0.0

Chapter 5: Feature selection

Some popular feature selection techniques such as univariate selection using Chi-squared test of independence, feature importance with tree classifiers and variance thresholds were ran to understand whether a feature is important or noise. These techniques were chosen to represent different aspects of the dataset.

Chi-squared test of independence is a statistical tool used to understand the level of dependence between a feature and the target variable. If the target variable is highly dependent on a feature, it means that changing the feature is likely to induce a change in the target variable thus making it desirable as a predictor.

$$Entropy, H(X) = -\sum_{i=0}^{C} p_i \log_2(p_i) \; ; \; Information\ gain, IG(S, x_i) = H(S) - \sum_{j=1}^{c_i} \frac{|S_j|}{|S|} H(S_j)$$

Decision tree classifier was also used to compute feature importance as a cross reference to support our decision. Decision tree classifiers are good feature selectors as at every node in the tree, a feature is selected for that node and the information gain for that node is calculated based on entropy reduction. Entropy is a metric for purity of a sample. The feature with the most information gain is then used for the node. Above formula shows the mathematical formula of calculating entropy and the information gain from entropy. As shown in the formula, maximising information gain also maximises the loss of entropy which makes it a good metric for feature importance.

METHOD 1: UNIVARIATE SELECTION:

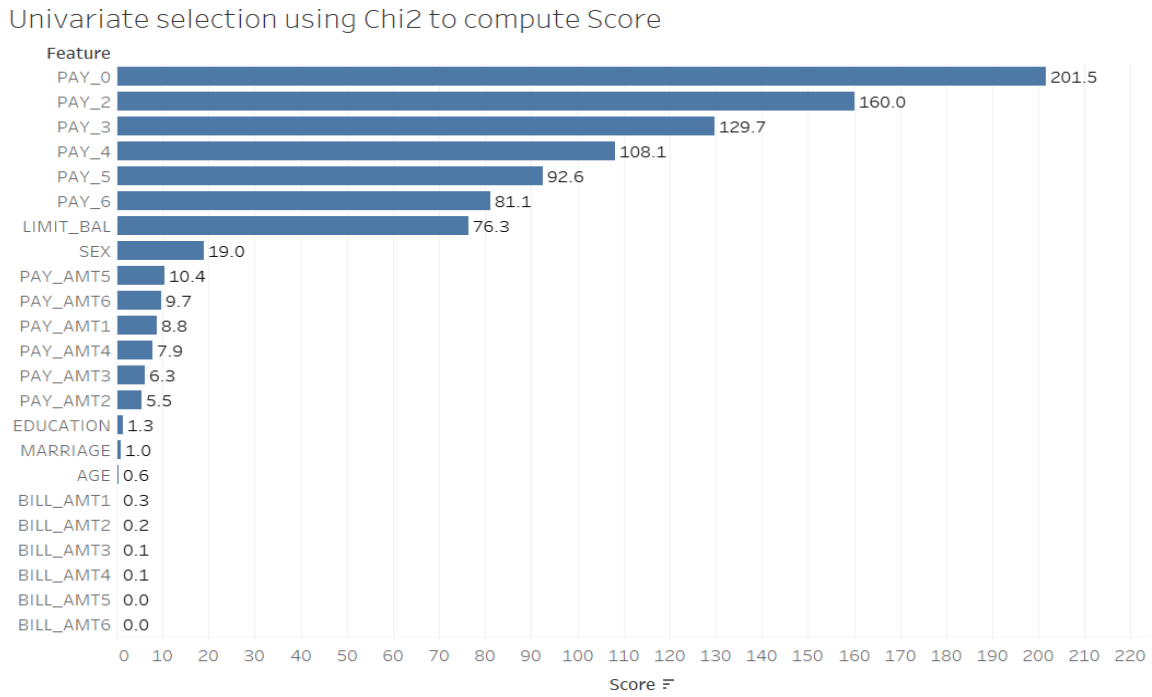


Figure 29: Univariate Selection using Chi2 to compute score

METHOD 2: FEATURE IMPORTANCE BY *ExtraTreesClassifier*:

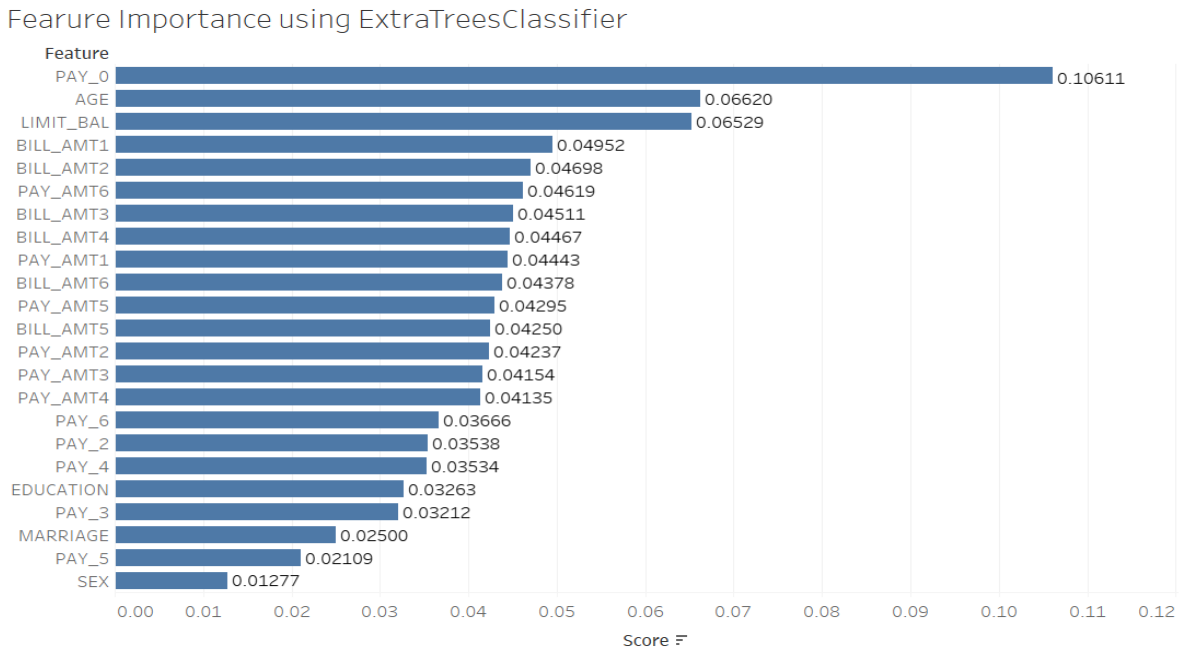


Figure 30: Feature importance using Extra Tree Classifier

Chapter 6: Splitting data and Imbalanced Class treatment

Part 1: Data snooping prevention:

Before doing any further research or training models, to avoid data snooping, we must split the dataset into 2 sub parts, naming training and testing dataset with ratio: 7: 3

```
Number transactions training dataset: 21000
Number transactions testing dataset: 9000
Total number of transactions: 30000
```

Part 2: Imbalanced class treatment:

As you can see, there are many records have "default payment next month" = 0 while less in 1 (there are 22.12 % of no-default and 77.88 % default next month), hence, this phenomenon is called "Imbalanced class" The reason behind why we should find a treatment for Imbalanced class is because conventional methods such as Decision Tree and Logistic Regression have a bias towards classes which have number of instances. They tend to only predict the majority class data. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class.

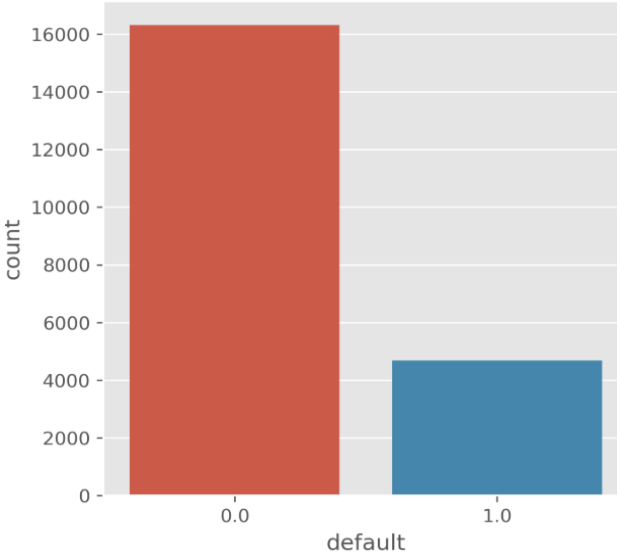


Figure 31: Count-plot for target column: "default payment next month"

RESAMPLING TECHNIQUES:

Under sampling	Over sampling
When you randomly eliminate instances from the majority class of a dataset and assign it to the minority class (without filling out the void created in majority class). The void that gets created in the majority dataset for this makes the process *random*.	Taking any help from the majority class, you increase the instances corresponding to the minority class by replicating them up to a constant degree. In this case, you do not decrease the number of instances assigned to the majority class.
It can help improve the runtime of the model and solve the memory problems by reducing the number of training data samples when the training data set is enormous.	Unlike undersampling, this method leads to no information loss.
It can discard useful information about the data itself which could be necessary for building rule-based classifiers such as Random Forests. The sample chosen by random undersampling may be a biased sample. And it will not be an accurate representation of the population in that case. Therefore, it can cause the classifier to perform poorly on real unseen data.	It increases the likelihood of overfitting since it replicates the minority class events.

INBUILT OVERSAMPLING AND UNDERSAMPLING METHODS:

Synthetic Minority Over-sampling Technique ($SMOTe$)
$MOTe$ is a technique based on nearest neighbor judged by Euclidean Distance between data points in feature space.
There is a percentage of Over-Sampling which indicates the number of synthetic samples to be created and this percentage parameter of Over-sampling is always a multiple of 100. If the percentage of Over-sampling is 100, then for each instance, a new sample will be created. Hence, the number of minority class instances will get doubled. Similarly, if the percentage of Over-sampling is 200, then the total number of minority class samples will get tripled.
In $SMOTe$, for each minority instance, k number of nearest neighbours are found such that they also belong to the same class where, $k = (SMOTe\%)/100$
The difference between the feature vector of the considered instance and the feature vectors of the k nearest neighbor are found. So, k number of difference vectors are obtained.
The k difference vectors are each multiplied by a random number between 0 and 1 (excluding 0 and 1).
Now, the difference vectors, after being multiplied by random numbers, are added to the feature vector of the considered instance (original minority instance) at each iteration. [6]

Adaptive synthetic sampling method (ADASYN) is an improved version of $SMOTe$. There is a small improvement. After creating some, it adds a small value to the points thus making it more realistic. In order words, instead of all sampling being linearly corelated to the parent, they have a little more variance in them, more scattered. [7]

Random under sample: perform random under sampling by undersampling majority class by randomly picking samples with or without replacement [8]

Combination of undersampling and oversampling into a hybrid strategy SMOTE + ENN: Combination we will focus on this project is SMOTE and Edited Nearest Neighbours (ENN). [9]

Chapter 7: Model Selection and evaluation

Part 1: Models and datasets

MODELS USING ORIGIN DATASET WITH OPTIONS "CLASS_WEIGHT" IN SKLEARN MODELS

In order to overcome Imbalanced class, there is an option we can use while avoiding spending time to resample dataset. In `sklearn` models, the option `class_weight` is available for several models such as:

- Random Forest classifier
- Logistic regression
- Linear Support Vector Classifier
- Stochastic gradient descent classifier
- Decision tree classifier,…

The default value for class_weight is None. When we want to use it, define it as "balanced" value:

- Weights associated with classes in the form `{class_label: weight}`. If not given, all classes are supposed to have weight one.
- The "`balanced`" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`.
- Note that these weights will be multiplied with `sample_weight` (passed through the fit method) if `sample_weight` is specified.

For binary classification problem, if class 0 samples are much less than class 1, more weight will be given to class 0 samples. Following this approach, we will achieve a balanced weight model eventually. Therefore, this is also an effective method to overcome Imbalanced class.

MODEL SYSTEM STRUCTURE:

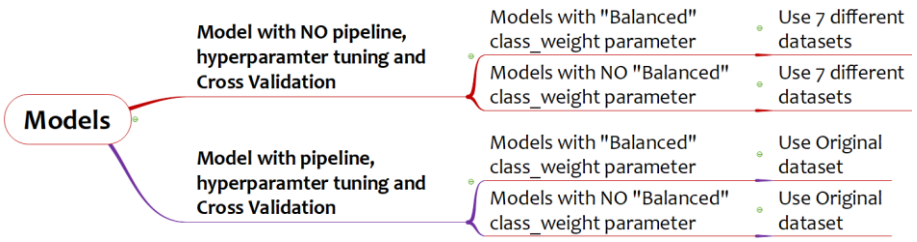


Figure 32: Model system structure

We have tried several models and tested them both with their default setting or with cross validated versions.

Also, for each model created, we will test them with different datasets, these datasets are created with resampling treatment methods shown above.

There are **7 total datasets,** namely: Original dataset (no resampling treatment), SMOTE dataset, ADASYN dataset, EMOTE + ENN dataset, Random Under sample dataset, Oversampling and Undersampling dataset (we did design our own oversample, under sample methods)

There are different models, such as: Naïve Bayes, Support Vector Machine, Logistic regression, Multilayer perceptron, Random Forest, Xtreme Gradient Boosting, Stochastic gradient descent, K-nearest neighbors, …

ENSEMBLE MODEL: STACKING

We did try to improve model performance by using different pipeline or scaler and hyperparameter tuning. Every model has it owns strength and weaknesses. Eventually, we came up with an idea of combining different models together is to cover one model's weakness by using another model's strength. There is a term to define this process, it is called Stacking. The explanation of Stacking is shown in Chapter 2, part 2. Below is our stacking model:

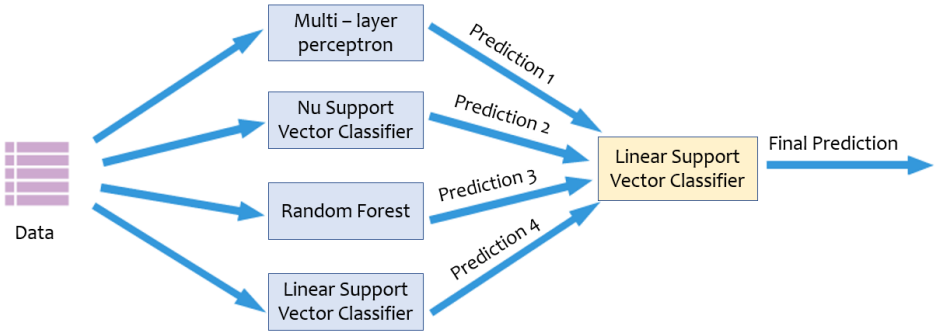


Figure 33: Our project Stacking model structure

MODELS EVALUATION BENCHMARK

Various models along with different pre-processing methods were tested and the performance of each classifier based on their F1 score, balanced accuracy, precision, recall, sensitivity, specificity. Balanced accuracy or F1 score can be a benchmark for the evaluation as they are more robust to imbalanced dataset. The strategy used was to start from simple models and gradually increase their complexity. Thus, Logistic Regression was chosen as a baseline since it is a relatively simple linear model and thus easier to understand compared to other learning models. Also, training time was factored in when making this decision. After tuning hyperparameters of Logistic Regression models, a F1 score of **0.4733** was achieved, which became our baseline of comparison.

Part 2: Model evaluation results

Details of all models with all available training datasets:

Explanation for model type:

- *Original means model with NO pipeline or Cross validation.*
- *Original Stack means original model used as a classifier in Stacking, but now is it evaluated solely.*
- *GSCV means model only use Grid Search Cross Validation, no pipeline*
- *Pipeline_GSCV means model with pipeline use Grid Search Cross Validation*
- *Pipeline_RSCV means model with pipeline use Randomized Search Cross Validation*
- *Multiple SVM means a Support Vector Machine model with different kernels as Hyperparamter.*

Explanation of Model Name:

- *NB stands for Naïve Bayes*
- *SVM stands for Support Vector Machine*
- *SVC stands for Support Vector classifier*
- *RBF stands for Radial Basis Function (a kernel of Support Vector Machine)*
- *SVC stands for Support Vector Classifier (a linear kernel Support Vector Machine)*
- *SGD stands for Stochastic Gradient Descent*
- *XGB stands for Xtreme Gradient Boosting*
- *RF stands for Random Forest*
- *MLP stands for Multi-layer perceptron*

Name	Data set	Type	Class weight	Balanced Accuracy	F1 score	Accuracy	Sensitivity	Specificity	Precision	Recall
Bernoulli NB	ADASYN	Original	None	0.6936	0.5007	0.7649	0.5413	0.8459	0.4658	0.5413
	Origin	Original	None	0.6780	0.4827	0.7680	0.4969	0.8591	0.4692	0.4969
	Over sample	Original	None	0.7018	0.4868	0.7369	0.5730	0.8307	0.4231	0.5730
	Random US	Original	None	0.7022	0.4849	0.7340	0.5750	0.8294	0.4193	0.5750
	SMOTE	Original	None	0.6686	0.4787	0.7779	0.4684	0.8689	0.4896	0.4684
	SMOTE + ENN	Original	None	0.7217	0.5076	0.7336	0.6306	0.8128	0.4247	0.6306
	Under sample	Original	None	0.6998	0.4909	0.7449	0.5648	0.8349	0.4341	0.5648
Decision Tree	ADASYN	Original	None	0.6635	0.3947	0.6783	0.4816	0.8454	0.3344	0.4816
	Origin	Original	Balanced	0.6361	0.3867	0.7324	0.3872	0.8849	0.3861	0.3872
			None	0.6424	0.3932	0.7267	0.4066	0.8781	0.3806	0.4066
	Over sample	Original	None	0.5788	0.2318	0.6604	0.2352	0.9224	0.2284	0.2352
	Random US	Original	None	0.7023	0.4171	0.6204	0.6235	0.7812	0.3133	0.6235
	SMOTE	Original	None	0.6610	0.3990	0.6918	0.4699	0.8521	0.3468	0.4699
	SMOTE + ENN	Original	None	0.7064	0.4560	0.6856	0.6051	0.8078	0.3658	0.6051
	Under sample	Original	None	0.6400	0.3022	0.5038	0.4934	0.7867	0.2178	0.4934
Gaussian NB	ADASYN	Original	None	0.6782	0.3821	0.3572	0.9128	0.4435	0.2417	0.9128
	Origin	Original	None	0.7251	0.4815	0.6918	0.6571	0.7931	0.3799	0.6571
	Over sample	Original	None	0.6143	0.3648	0.2879	0.9388	0.2898	0.2264	0.9388
	Random US	Original	None	0.7368	0.4339	0.5592	0.7755	0.6980	0.3012	0.7755
	SMOTE	Original	None	0.6992	0.3908	0.3927	0.8944	0.5040	0.2500	0.8944
	SMOTE + ENN	Original	None	0.7022	0.3920	0.3996	0.8888	0.5156	0.2514	0.8888
	Under sample	Original	None	0.6874	0.3723	0.4057	0.8092	0.5656	0.2417	0.8092
K-Nearest Neighbors	ADASYN	Original	None	0.7074	0.4271	0.6296	0.6342	0.7806	0.3220	0.6342
	Origin	Original	None	0.6255	0.4211	0.7929	0.3459	0.9050	0.5381	0.3459
	Over sample	Original	None	0.6290	0.2924	0.5493	0.4276	0.8305	0.2222	0.4276
	Random US	Original	None	0.7192	0.4585	0.6646	0.6520	0.7863	0.3535	0.6520
	SMOTE	Original	None	0.7018	0.4377	0.6640	0.6005	0.8030	0.3444	0.6005
	SMOTE + ENN	Original	None	0.7224	0.4377	0.6147	0.6888	0.7560	0.3208	0.6888
	Under sample	Original	None	0.6426	0.3056	0.4996	0.5056	0.7796	0.2190	0.5056
Multi Layer Perceptron	ADASYN	Original Stack	None	0.6865	0.4968	0.7707	0.5199	0.8531	0.4757	0.5199
	Origin	Original Stack	None	0.6187	0.4381	0.8193	0.3235	0.9140	0.6788	0.3235
	Over sample	Original Stack	None	0.6229	0.3739	0.7470	0.3469	0.8989	0.4055	0.3469
	Random US	Original Stack	None	0.6842	0.5061	0.7831	0.5102	0.8581	0.5020	0.5102
	SMOTE	Original Stack	None	0.6811	0.5026	0.7838	0.5015	0.8606	0.5036	0.5015
	SMOTE + ENN	Original Stack	None	0.7115	0.4782	0.7100	0.6102	0.8128	0.3932	0.6102
	Under sample	Original Stack	None	0.6761	0.3777	0.6053	0.5500	0.8021	0.2876	0.5500

Logistic Regression	ADASYN	Original	None	0.7261	0.4518	0.6368	0.6872	0.7650	0.3365	0.6872
		Pipeline GSCV	Balanced	0.7297	0.4531	0.6312	0.7015	0.7580	0.3346	0.7015
			None	0.7278	0.4550	0.6396	0.6908	0.7648	0.3392	0.6908
	Origin	Original	Balanced	0.7159	0.4696	0.6896	0.6311	0.8007	0.3739	0.6311
			None	0.5867	0.3525	0.8102	0.2372	0.9362	0.6858	0.2372
		Pipeline GSCV	Balanced	0.7173	0.4733	0.6929	0.6337	0.8008	0.3777	0.6337
			None	0.5850	0.3470	0.8093	0.2327	0.9374	0.6826	0.2327
	Over sample	Original	None	0.6885	0.4194	0.6610	0.5622	0.8148	0.3344	0.5622
		Pipeline GSCV	None	0.5000	0.3577	0.2178	1.0000	0.0000	0.2178	1.0000
	Random US	Original	None	0.7132	0.4695	0.6941	0.6214	0.8050	0.3772	0.6214
		Pipeline GSCV	Balanced	0.7189	0.4687	0.6826	0.6429	0.7949	0.3687	0.6429
			None	0.7176	0.4728	0.6916	0.6352	0.8000	0.3766	0.6352
	SMOTE	Original	None	0.7157	0.4685	0.6881	0.6311	0.8003	0.3725	0.6311
		Pipeline GSCV	Balanced	0.7199	0.4677	0.6791	0.6474	0.7924	0.3661	0.6474
			None	0.7186	0.4679	0.6818	0.6423	0.7948	0.3679	0.6423
	SMOTE + ENN	Original	None	0.7286	0.4297	0.5741	0.7367	0.7205	0.3033	0.7367
		Pipeline GSCV	Balanced	0.7148	0.4706	0.6930	0.6265	0.8031	0.3768	0.6265
			None	0.7313	0.4399	0.5957	0.7291	0.7334	0.3150	0.7291
	Under sample	Original	None	0.6192	0.3156	0.6650	0.3546	0.8839	0.2843	0.3546
Multinomial NB	ADASYN	Original	None	0.7018	0.4469	0.6799	0.5939	0.8098	0.3583	0.5939
	Over sample	Original	None	0.5018	0.0101	0.7820	0.0051	0.9986	0.4545	0.0051
	Random US	Original	None	0.7092	0.4391	0.6500	0.6291	0.7892	0.3373	0.6291
	SMOTE	Original	None	0.7082	0.4386	0.6513	0.6255	0.7909	0.3377	0.6255
	SMOTE + ENN	Original	None	0.6758	0.3767	0.3607	0.8872	0.4643	0.2391	0.8872
	Under sample	Original	None	0.6236	0.4174	0.7928	0.3408	0.9064	0.5383	0.3408
Multiple SVM	ADASYN	Pipeline GSCV	None	0.6398	0.3643	0.6894	0.4087	0.8709	0.3287	0.4087
	Origin	Pipeline GSCV	None	0.5960	0.3698	0.8046	0.2633	0.9287	0.6209	0.2633
	Over sample	Pipeline GSCV	None	0.5913	0.2415	0.6127	0.2832	0.8993	0.2105	0.2832
	Random US	Pipeline GSCV	None	0.6909	0.5034	0.7713	0.5321	0.8498	0.4776	0.5321
	SMOTE	Pipeline GSCV	None	0.6429	0.3753	0.6989	0.4153	0.8706	0.3423	0.4153
	SMOTE + ENN	Pipeline GSCV	None	0.6972	0.4336	0.6668	0.5857	0.8087	0.3442	0.5857
	Under sample	Pipeline GSCV	None	0.6390	0.3043	0.5306	0.4714	0.8065	0.2247	0.4714
Nu SVC	ADASYN	Original Stack	None	0.6745	0.3612	0.5602	0.5709	0.7781	0.2642	0.5709
	Origin	Original Stack	None	0.6040	0.2491	0.5452	0.3464	0.8616	0.1945	0.3464
	Over sample	Original Stack	None	0.6020	0.2477	0.5567	0.3352	0.8689	0.1965	0.3352
	Random US	Original Stack	None	0.6342	0.2943	0.5129	0.4663	0.8020	0.2150	0.4663
	SMOTE	Original Stack	None	0.6436	0.3116	0.5321	0.4862	0.8010	0.2293	0.4862
	SMOTE + ENN	Original Stack	None	0.6996	0.3871	0.5367	0.6719	0.7273	0.2719	0.6719
	Under sample	Original Stack	None	0.6014	0.2442	0.5439	0.3383	0.8646	0.1910	0.3383
SVM (RBF)	ADASYN	Pipeline GSCV	None	0.7132	0.4463	0.6548	0.6388	0.7875	0.3429	0.6388
	Origin	Pipeline GSCV	Balanced	0.6597	0.4781	0.7904	0.4408	0.8785	0.5224	0.4408
	Random US	Pipeline GSCV	None	0.6907	0.4990	0.7671	0.5327	0.8488	0.4694	0.5327
	SMOTE	Pipeline GSCV	None	0.6965	0.4609	0.7114	0.5663	0.8266	0.3885	0.5663
	SMOTE + ENN	Pipeline GSCV	None	0.7186	0.4437	0.6372	0.6643	0.7730	0.3331	0.6643
	Under sample	Pipeline GSCV	None	0.6402	0.3320	0.6217	0.4316	0.8488	0.2697	0.4316
Stacking	Origin	Pipeline GSCV	None	0.6165	0.4341	0.8198	0.3173	0.9157	0.6865	0.3173
	Random US	Pipeline GSCV	None	0.6866	0.5125	0.7861	0.5163	0.8570	0.5088	0.5163
	SMOTE + ENN	Pipeline GSCV	None	0.7291	0.4936	0.7033	0.6638	0.7945	0.3928	0.6638
XGB	ADASYN	Pipeline GSCV	None	0.6417	0.4686	0.8084	0.3878	0.8955	0.5919	0.3878
	Origin	Original	None	0.6237	0.4499	0.8207	0.3367	0.9106	0.6776	0.3367
		Pipeline GSCV	None	0.6320	0.4664	0.8210	0.3592	0.9047	0.6648	0.3592
	Over sample	Original	None	0.6386	0.3192	0.5929	0.4383	0.8390	0.2510	0.4383
	Random US	Pipeline GSCV	None	0.7241	0.5190	0.7442	0.6337	0.8146	0.4395	0.6337
	SMOTE	Pipeline GSCV	None	0.6420	0.4685	0.8079	0.3888	0.8952	0.5893	0.3888
	SMOTE + ENN	Pipeline GSCV	None	0.6954	0.5148	0.7771	0.5429	0.8479	0.4894	0.5429
	Under sample	Original	None	0.6403	0.3055	0.5251	0.4796	0.8011	0.2241	0.4796
		Pipeline GSCV	None	0.6387	0.2982	0.4618	0.5250	0.7524	0.2082	0.5250

Random Forest	ADASYN	Original	None	0.6645	0.4615	0.7663	0.4597	0.8694	0.4632	0.4597
		Original Stack	None	0.7137	0.5021	0.7382	0.6061	0.8212	0.4286	0.6061
		Pipeline GSCV	Balanced	0.7219	0.5017	0.7257	0.6342	0.8097	0.4150	0.6342
			None	0.7183	0.4997	0.7283	0.6230	0.8137	0.4172	0.6230
		Pipeline RSCV	Balanced	0.7201	0.4906	0.7134	0.6337	0.8066	0.4003	0.6337
			None	0.7320	0.5037	0.7133	0.6679	0.7961	0.4043	0.6679
	Origin	Original	Balanced	0.6089	0.3977	0.8028	0.2990	0.9189	0.5937	0.2990
			None	0.6120	0.4228	0.8186	0.3051	0.9188	0.6881	0.3051
		Pipeline GSCV	Balanced	0.6973	0.5125	0.7723	0.5495	0.8451	0.4802	0.5495
			None	0.6171	0.4355	0.8200	0.3189	0.9153	0.6868	0.3189
		Pipeline RSCV	Balanced	0.6856	0.5039	0.7790	0.5153	0.8559	0.4929	0.5153
	Over sample	Original	None	0.5248	0.1074	0.7469	0.0699	0.9796	0.2314	0.0699
		Original Stack	None	0.6119	0.2865	0.6287	0.3423	0.8814	0.2463	0.3423
	Random US	Original	None	0.7175	0.5157	0.7491	0.6133	0.8217	0.4449	0.6133
		Original Stack	None	0.7203	0.5160	0.7457	0.6224	0.8182	0.4406	0.6224
		Pipeline GSCV	Balanced	0.7088	0.5220	0.7674	0.5832	0.8345	0.4725	0.5832
			None	0.7171	0.5147	0.7486	0.6122	0.8219	0.4440	0.6122
		Pipeline RSCV	Balanced	0.7151	0.5237	0.7613	0.6026	0.8276	0.4631	0.6026
			None	0.7252	0.5211	0.7453	0.6362	0.8141	0.4413	0.6362
	SMOTE	Original	None	0.6579	0.4615	0.7770	0.4388	0.8770	0.4867	0.4388
		Original Stack	None	0.7030	0.5044	0.7559	0.5704	0.8357	0.4521	0.5704
		Pipeline GSCV	Balanced	0.7184	0.5158	0.7480	0.6163	0.8206	0.4435	0.6163
			None	0.7152	0.5171	0.7539	0.6051	0.8252	0.4515	0.6051
		Pipeline RSCV	Balanced	0.7141	0.5217	0.7604	0.6000	0.8282	0.4615	0.6000
			None	0.7177	0.5198	0.7537	0.6122	0.8231	0.4516	0.6122
	SMOTE + ENN	Original	None	0.7090	0.5055	0.7489	0.5893	0.8286	0.4425	0.5893
		Original Stack	None	0.7286	0.4885	0.6966	0.6653	0.7920	0.3859	0.6653
		Pipeline GSCV	Balanced	0.7121	0.5134	0.7538	0.5964	0.8277	0.4507	0.5964
			None	0.7178	0.5147	0.7476	0.6148	0.8209	0.4427	0.6148
		Pipeline RSCV	Balanced	0.7119	0.5137	0.7543	0.5959	0.8280	0.4515	0.5959
			None	0.7418	0.4980	0.6898	0.7066	0.7769	0.3845	0.7066
	Under sample	Original	None	0.6134	0.2679	0.5568	0.3724	0.8543	0.2092	0.3724
		Original Stack	None	0.6418	0.3057	0.5110	0.4944	0.7893	0.2213	0.4944
SGD	ADASYN	Original	None	0.6952	0.4803	0.7391	0.5536	0.8369	0.4242	0.5536
		Pipeline GSCV	Balanced	0.7299	0.4324	0.5781	0.7378	0.7221	0.3058	0.7378
			None	0.7152	0.4694	0.6906	0.6286	0.8018	0.3746	0.6286
	Origin	Original	Balanced	0.6885	0.4925	0.7632	0.5276	0.8495	0.4618	0.5276
			None	0.5011	0.0061	0.7824	0.0031	0.9991	0.6000	0.0031
		Pipeline GSCV	Balanced	0.6045	0.3919	0.8066	0.2862	0.9227	0.6213	0.2862
			None	0.5686	0.2941	0.8032	0.1883	0.9490	0.6721	0.1883
	Over sample	Original	None	0.5287	0.1385	0.7844	0.0796	0.9779	0.5342	0.0796
		Pipeline GSCV	None	0.5838	0.3642	0.2633	0.9689	0.1987	0.2243	0.9689
	Oversample	Pipeline GSCV	Balanced	0.6792	0.3763	0.3693	0.8735	0.4850	0.2398	0.8735
	Random US	Original	None	0.6899	0.4900	0.7586	0.5327	0.8471	0.4537	0.5327
		Pipeline GSCV	Balanced	0.6902	0.4971	0.7658	0.5316	0.8488	0.4668	0.5316
			None	0.6885	0.4973	0.7684	0.5260	0.8509	0.4716	0.5260
	SMOTE	Original	None	0.6763	0.4959	0.7838	0.4883	0.8643	0.5037	0.4883
		Pipeline GSCV	Balanced	0.6922	0.4979	0.7638	0.5378	0.8467	0.4635	0.5378
			None	0.6904	0.4974	0.7658	0.5321	0.8487	0.4669	0.5321
	SMOTE + ENN	Original	None	0.7284	0.4504	0.6286	0.6990	0.7578	0.3323	0.6990
		Pipeline GSCV	Balanced	0.7117	0.4702	0.6980	0.6153	0.8080	0.3804	0.6153
			None	0.7218	0.4549	0.6526	0.6658	0.7778	0.3455	0.6658
	Under sample	Pipeline GSCV	None	0.5918	0.3644	0.2691	0.9622	0.2213	0.2248	0.9622
	Undersample	Pipeline GSCV	Balanced	0.5906	0.3647	0.2681	0.9648	0.2163	0.2249	0.9648

SVM (Linear)	ADASYN	Original	None	0.7267	0.4511	0.6339	0.6908	0.7627	0.3349	0.6908
		Original Stack	None	0.6855	0.4931	0.7681	0.5179	0.8532	0.4706	0.5179
		Pipeline GSCV	Balanced	0.7299	0.4536	0.6320	0.7015	0.7583	0.3352	0.7015
			None	0.7276	0.4553	0.6406	0.6898	0.7655	0.3398	0.6898
	Origin	Original	Balanced	0.7149	0.4724	0.6958	0.6255	0.8042	0.3796	0.6255
			None	0.5417	0.1951	0.7938	0.1148	0.9685	0.6503	0.1148
		Original Stack	None	0.6065	0.4086	0.8170	0.2903	0.9226	0.6897	0.2903
		Pipeline GSCV	Balanced	0.7156	0.4762	0.7001	0.6260	0.8053	0.3843	0.6260
			None	0.5602	0.2660	0.8013	0.1653	0.9551	0.6807	0.1653
	Over sample	Original	None	0.6925	0.4141	0.6407	0.5832	0.8018	0.3211	0.5832
		Original Stack	None	0.6979	0.3782	0.4450	0.7750	0.6207	0.2501	0.7750
	Random US	Original	None	0.7106	0.4724	0.7032	0.6102	0.8110	0.3854	0.6102
		Original Stack	None	0.6804	0.5000	0.7822	0.5000	0.8608	0.5000	0.5000
		Pipeline GSCV	Balanced	0.7168	0.4706	0.6896	0.6337	0.7999	0.3743	0.6337
			None	0.7149	0.4734	0.6972	0.6250	0.8048	0.3810	0.6250
	SMOTE	Original	None	0.7150	0.4700	0.6918	0.6276	0.8024	0.3757	0.6276
		Original Stack	None	0.6693	0.5001	0.7972	0.4658	0.8728	0.5399	0.4658
		Pipeline GSCV	Balanced	0.7182	0.4701	0.6861	0.6393	0.7971	0.3717	0.6393
			None	0.7181	0.4714	0.6882	0.6383	0.7980	0.3737	0.6383
	SMOTE + ENN	Original	None	0.7278	0.4303	0.5787	0.7306	0.7250	0.3049	0.7306
		Original Stack	None	0.7272	0.4697	0.6686	0.6740	0.7805	0.3604	0.6740
		Pipeline GSCV	Balanced	0.7119	0.4751	0.7049	0.6133	0.8105	0.3877	0.6133
			None	0.7306	0.4410	0.6002	0.7240	0.7373	0.3170	0.7240
	Under sample	Original	None	0.6125	0.2897	0.6338	0.3429	0.8822	0.2507	0.3429
		Original Stack	None	0.6978	0.3905	0.5598	0.6474	0.7481	0.2795	0.6474

Figure 34: Evaluation metrics of all models for all dataset.

It is observed that most of the models using Pipeline have F1 score improved (F1 Score). Hence, we do have better model.

Best performance in each model (sorted by F1 Score):

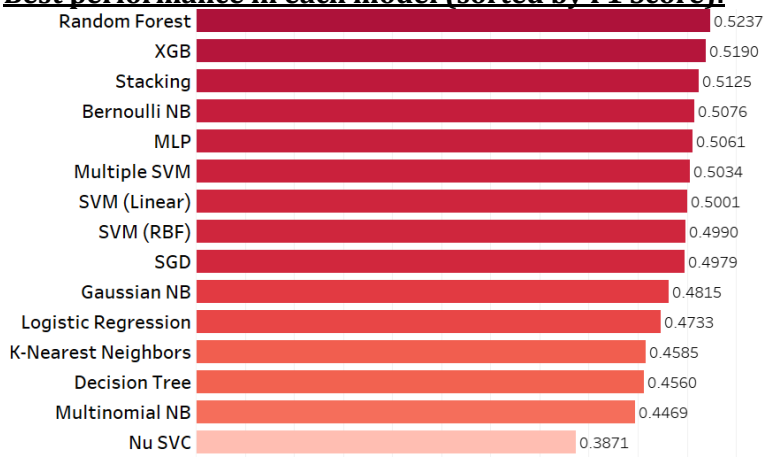


Figure 35: Maximum F1 Score for each model

The difference between F1 score of top models are insignificant.

The most 3 impressive ones are Random Forest, Xtreme Gradient Boosting and Stacking.

Although there is not much different in performance of these 3 models. Stacking seems to have vey big potential that can explored in future.

Worth note taking potential of Stacking model:

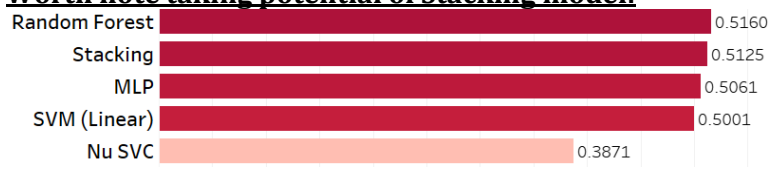


Figure 36: Maximum F1 Score in each classifier component of Stacking model

Stacking is the combination of four different models, namely Random Forest, Multi-layer perceptron, Linear Support Vector Machine and Nu SVC (shown in figure 33). Although the performance of Nu SVC is not high. And three others are high, the performance of Stacking model is not affected much. This proves how powerful of this model.

Part 3: Experiential lessons in this project:

For **linear models**, the performances of the models converge around the F1 score of around 0.47 with no sign of improvement no matter how much time is spent to tune the hyper-parameters of these models. This is likely caused by the lack of robustness of the models as they are not designed to accurately this non-linear dataset.

For **Naive Bayes**, the training time and the testing time of this model is the fastest out of all the models implemented in this project. And the F1 score of Bernoulli Naïve Bayes turns out to be impressive. This is a rare case as the dataset matches to Bernoulli distribution. However, Naïve Bayes are not on our preferable model lists as this performance are not consistent if we have a new set of data that does not follow Bernoulli Distribution, then our performance is very bad.

For **KNN**, the model is trained very quickly, but it takes a very long time for it to make its prediction. The more data used for training this model, the longer it takes for it to find the nearest neighbours of the data that it would like to classify. This model learn by memorising the data, which caused it to be easily overfitted. The F1 score converges to 0.46

For **SVM**, training of the data was not able converge easily as a high number of iterations is observed even though features have been normalized and the weights of the classes have been balanced. When number of iterations is limited, the F1 score observed during cross validation was not ideal. This shows that data was not able to fit well as it was hard to find a suitable separating hyperplane and computationally intensive to tune the hyperparameters. [10]

Extreme Gradient Boosting trees are an efficient implementation of traditional Gradient Boosting trees, having clever penalisation of trees, proportional shrinking of leaf nodes, extra randomisation parameters which helped increase both the performance and the speed of the classifiers [11]. The intuition behind gradient boosting trees and the other ensemble methods used in this paper is the idea that crowds generally perform better than individuals.

In **Random Forests classifiers**, multiple weak classifiers are trained in parallel using techniques such as bootstrapping on both data points and features, before aggregating them to reduce the variance. On the other hand, gradient boosting trees allow the results of each weak classifier to affect the next weak classifier. Since each individual classifier is made deliberately weak, it is unable to explore the full parameter space, thus making it unlikely to overfit. Also, with the aggregation of multiple weak learners, the resulting model can also prevent overfitting, which explains why Extreme Gradient Boosting trees, an advanced implementation with heuristics performed much better than the other models.

Similar to the idea of deep networks, **stacking** classifiers follow the idea that hierarchical approach to learning might be better in terms of variance and bias as it is hard for models to learn multiple things at once without being too large and resulting in overfitting from the curse of dimensionality phenomenon [12]. In stacking, by splitting models into 2 levels in this paper (the lower level classifiers with an additional layer as meta-classifiers), lower level models can explore different parts of the problem space and higher-level meta-classifiers can combine their predictions in meaningful ways. This means that when these 4 classifiers are combined together, with their predictions as input to the higher-level meta-classifier, the meta-classifier can prioritise certain classifiers when presented with certain types of data points, thus achieving a slightly better overall result. To achieve a better result, the weights of the meta-classifier must also be adjusted which presents a new set of hyperparameters to tune. As such, the complexity of the model has increased not only in the number of models but also the number of hyperparameters, increasing the difficulty of tuning such models. In this paper, only 4 models were used for stacking, but the number of models could be increased significantly, by not only varying the model type, but also the pre-processing steps or hyperparameters for model. This presents a tradeoff between model performance and understandability of the model, which must be duly considered when deciding to use similar techniques.

Chapter 8: Future plan for Model Improvement

Part 1: Overall picture

Predicting credit card default is a complex problem with many features and the exploratory description analysis shows this complexity in heatmap and distribution plots.

Although there are multiple reports showing the solution for this problem. In this project, we aim for a higher goal than that. We have explored a stacking model, which has not be implemented in articles we have read. The performance turns out to be optimistic and we would like to improve this model further.

Regretfully, some of the features in this dataset was obfuscated, possibly for privacy reasons and the contextual meaning was lost. In the future, new datasets could be created to include more features as well as replace some of the obfuscated features in this dataset with more relevant ones. This could help improve the exploratory description analysis part, so that enable us to improve model for higher performance.

Also, the focus in this paper was only on predicting whether a customer will default next month or not, but the goal can be extended into predict a customer will default within a range of period, not only one month.

Part 2: Improvement on existing models

Feature selection: In models, we have chosen some columns that can be removed in data-preprocessing step based on Feature Importance using Extra Tree Classifier, such as:

```
"SEX", "PAY_5", "MARRIAGE", "PAY_3", "EDUCATION", "PAY_2", "PAY_4", "PAY_6"
```

It is suggesting that we should try other feature selection method as well, namely Chi2 Univariate selection.

Potential models: In our opinion, one of the top potential models are **Ensemble cross validated Stacking**. First, it can combine different strong learners models. If we do in the right way, the Stacking model is expected to outperform its sub-models. However, the choice for meta-learner models has to be as simple as possible. Otherwise, overfitting will likely occur, resulting in worse performance. Another criterion to take note when we do Stacking is timing. In this project, we did put Support Vector Machine model as a meta-classifier in the end, and hyperparameter tuning for the desire of achieving higher score. However, efficiency is not achieved as it takes very long to train a dataset (more than 5 hours).

In the future, we planned to consider Logistic regression as our meta-classifier instead as it is faster to train, and it will prevent overfitting by combining with regularizers.

Part 3: Potential models have not been implemented

Neural network model is highly anticipated to have a potential outperform our existing models. However, if we want to tune Neural Network from scratch to build an efficient model will consume a lot of effort and energy. Therefore, we will recommend utilizing pre-built model.

Chapter 9: References

[1] M. Merikoski, “Predicting and Preventing Credit Card Default,” 2018.

[2] J. Rocca, “Ensemble methods: bagging, boosting and stacking,” 23 04 2019. [Online]. Available: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205.

[3] J. VanderPlas, Python Data Science Handbook.

[4] Book Down, “Lab notes for Statistics for Social Sciences II: Multivariate Techniques,” [Online]. Available: https://bookdown.org/egarpor/SSS2-UC3M/.

[5] A. V. Srinivasan, “Stochastic Gradient Descent — Clearly Explained,” 07 09 2019. [Online]. Available: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31.

[6] S. A. Rahim, “SMOTE AND NEAR MISS IN PYTHON: MACHINE LEARNING IN IMBALANCED DATASETS,” 4 06 2018. [Online]. Available: https://medium.com/@saeedAR/smote-and-near-miss-in-python-machine-learning-in-imbalanced-datasets-b7976d9a7a79.

[7] K. Vala, “ADASYN: Adaptive Synthetic Sampling Method for Imbalanced Data,” 10 09 2019. [Online]. Available: https://towardsdatascience.com/adasyn-adaptive-synthetic-sampling-method-for-imbalanced-data-602a3673ba16.

[8] imbalanced-learn.readthedocs, “Documentation for imblearn.under_sampling,” [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.RandomUnderSampler.html.

[9] imbalanced-learn.readthedocs, “Documentation for imbalanced-learn SMOTE ENN,” [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.combine.SMOTEENN.html.

[10] V. M. Y. Cherkassky, “Practical selection of SVM parameters and noise estimation for SVM regression.,” 2004.

[11] R. Gandhi, “Gradient Boosting and XGBoost,” 2018. [Online]. Available: https://hackernoon.com/gradient-boosting-and-xgboost-90862daa6c77.

[12] D. H. Wolpert., “Stacked generalization,” 1992. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0893608005800231.