HCMUS - International Training & Education Center

CS300

final lab project

Loans Prediction Problem

class: 19BIT by: Group 7
1959002 - Phạm Đình Chương
1959024 - Nguyễn Cao Nhân
1959035 - Lê Trần Bá Tân
1959040 - Hồ Ngọc Thảo Trang

TABLE OF CONTENTS

I. GROUP INFORMATION

For this Midterm project, our group consists of 4 people. All of us are members of Cycle 13, the ITEC program, and class 19BIT. Members of our group include:

- **1959002** – **Phạm Đình Chương**
- **1959024** – **Nguyễn Cao Nhân** (Group Leader)
- **1959035** – **Lê Trần Bá Tân**
- **1959040** – **Hồ Ngọc Thảo Trang**

II. JOB DIVISION AND WORKING PHASES

1. JOB DIVISION

ID	Name	Job	Contributed
1959002	**Phạm Đình Chương**	Decision Tree Random Forest Feature Scaling	100%
1959024	**Nguyễn Cao Nhân**	Support Vector Machine Collect codes and merge them Rerun the complete Notebook	100%
1959035	**Lê Trần Bá Tân**	One-Hot Encoding K-Nearest Neighbors Logistic Regression	100%
1959040	**Hồ Ngọc Thảo Trang**	Feature Selection Bernoulli Naïve Bayes Artificial Neural Network	100%

2. WORKING PHASES

The working course of our group is divided into main phases:

- **Phase 1:** Our team completed the two given notebooks together via an Online Zoom meeting, for all the team members to understand the requirements of the problem and basic steps for completion.
- **Phase 2:** We spent a few days for all team members to make some research and think of the steps for implementation of the project.
- **Phase 3:** Our team had a meeting to agree on the steps included, from the steps during Preprocessing to the choices of different classification models to compare.
- **Phase 4:** We divided the steps and algorithms among different members to complete and collect the full workable code.
- **Phase 5:** The team's leader merges the work of all people into a complete Notebook, reformats the code to follow the same pattern for easy implementation, and reruns the complete Notebook.
- **Phase 6:** Analyze the results, make comparisons and write the report for the project.

III. ABSTRACT

Determining whether a loan is a safe loan or a bad loan based on different properties and information is one of the challenges for lending companies. Using a dataset of loans provided by LendingClub company, this project applies the use of Machine Learning and Deep Learning techniques to train models that are capable of performing the prediction of safe or bad loans and makes comparisons between the models used. Generally speaking, among the seven methods used for the project, some of the classifiers seemed to be more suitable for the classification, giving better results in both accuracy and F1 score.

IV. PROBLEM APPROACH

This problem is a binary classification task, as there are only two values available for the output labels (0 for a safe loan and 1 for a bad loan).

The training set is quite big with more than 120.000 samples and nearly 70 features. Processing steps used for this project include Exploratory Data Analysis (to understand the data and look for some information to select the features to train), Feature Selection, Feature Scaling, Hyperparameters Tuning, and finally Model Training. The detailed implementation steps will be discussed in the later parts of the report.

As the requirement of the project besides building a decision tree to handle the classification is to compare the models, a total of 7 classifiers were used for this project. Those are classifiers that tend to be suitable and performed well for binary classification tasks [1]:

1. Decision Tree: a baseline for the project
2. K-Nearest Neighbors
3. Random Forest
4. Logistic Regression
5. Support Vector Machine
6. Naïve Bayes
7. Artificial Neural Network

The implementation in detail will be discussed in the later section of the report. Generally describing, some basic techniques in Machine Learning to fit the model to the training set and validate using the test set were applied. Then the 7 classifiers were compared in terms of some basic evaluation measurements: accuracy, F1-score, precision, and recall.

V. DECISION TREE

1. THEORY BASE

Decision Tree (DT) is a non-parametric supervised learning method used for classification and regression [2]. A decision tree can be visualized as a tree, where each node will contain a specific rule. Data passed into the Decision Tree will traverse from the root node, following the conditions of the rule until reaching one of the leaf nodes where one of the labels or outputs can be returned from the model.

Decision Tree can be learned by splitting the source set into subsets based on an attribute value test, recursively until a subset can meet a shared condition for directly determining the target label of data when reaching the current node [3].

In a visualization example of a simple decision tree below (Figure 1), each node is a condition of an attribute whether the value of the attribute will determine the traverse path for data from root to leaves. For example, data with rainy and weak wind values can end up being predicted with a Y label, meaning that it is suitable to play.

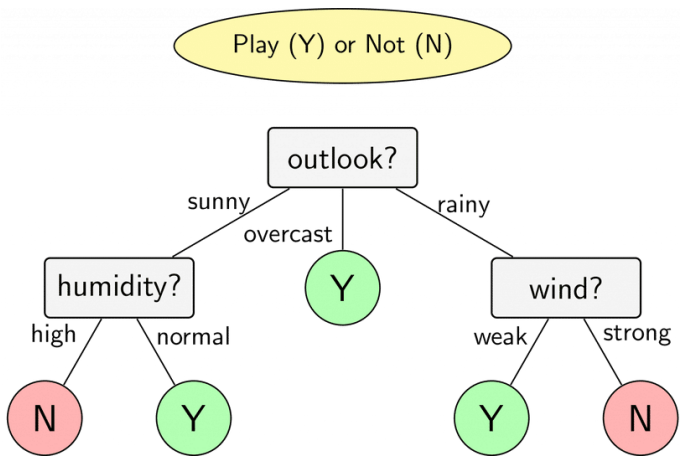

Figure 1 - Visualization of a simple Decision Tree [4]

Many different algorithms can be used for the implementation of Decision Trees. Some of the common algorithms are ID3, C4.5, CHAID, C&R, etc.

Some of the strengths of decision tree methods include [3]:

- The rules are understandable.
- Not requiring much computation.
- Can handle both classification and regression problems.
- Provide a clear indication of which fields are most important for prediction or classification.

The weaknesses of decision tree methods include:

- Can be computationally expensive to train.
- As the structure of the tree is built based on the training set, it can be the case that the Decision Tree overfits the training set and performs very badly on the validation set.
- The method can be prone to errors in complex classification tasks.

2. PREPROCESSING AND TRAINING

The instructions in the given Notebooks were followed to complete the two tasks:

- Implement a decision tree to perform the required task using a given Turicreate library (NB01.ipynb)
- Self-build a decision tree from scratch following the step-by-step instructions of the given Notebook (NB02.ipynb)

During the following preprocessing of the given notebook, there are some difficult points for this dataset and task identified, which are:

- An imbalanced dataset: Around 81% of the dataset are safe loans and only 19% are risky loans. This imbalance can cause the dataset to overfit the training set and keep predicting False for all data (meaning the loan is safe). This model still ends up with high accuracy but the overall performance is considered very bad as it cannot identify risky loans at all.
- The dataset contains many features, some of which can be meaningless to the task: The notebook shortlisted the list of features to 12 features (in a total set of 67 features), but some more features can be useful for the training process based on their meanings. This can be a hint for some further analysis of the features for our later task of models comparison.

The notebook resolved the imbalance problem of the dataset by undersampling and used two different decision trees with different complexity to train and validate.

For the second task of self-build a decision tree from scratch, the instructions in the given notebook (NB02.ipynb) were followed. The processes for building a binary decision tree include:

- Subsampling the dataset (resolving the imbalance issue): this was resolved in a similar approach to the first notebook.
- Transform categorical data into binary features: the process was similar to one-hot encoding the categorical features.
- Train-test split: in 80% / 20% ratio of train / test
- Implementing the decision tree: the implementation follows a bottom-up approach, from defining the functions for counting errors, determining the features to split, and finally structuring up the whole decision tree.

3. RESULTS

The decision trees built using the Turicreate library in the first notebook after completion received the following accuracies[1]:

- Small model (max-depth of 2): 61.93%
- Medium model (max-depth of 6): 63.68%
- Large model (max-depth of 10): 62.74%

For the decision tree self-built in the second notebook, the error rate was 0.3838, so the accuracy on the test set of the model is 61.62%

[1] Those are the accuracies performed on the validation set (or test set) of data

Those accuracies will be used as the baseline for our team to compare during our implementation of different types of classifiers in the next section of the project.

In the notebooks, some additional tasks were also performed, including visualizing out a subpart of the tree to see how each node containing a condition is structured and calculating different evaluation measurements such as False negatives (FN), False positives (FP), etc.

VI. IMPLEMENTATION PROCESSES

In this section, the complete pipeline that our team has done on the dataset to proceed from loading data into the notebook to the evaluation of the trained models will be briefly described. In general, there are 8 steps:

1. EXPLORATORY DATA ANALYSIS (EDA)

After loading the data into the notebook and performing the train/test split, some basic steps of Exploratory Data Analysis (EDA) were performed to observe and find some key points from the training set. This step is included because as mentioned in the previous section of the report, the dataset contains a very large number of features and even the shortlisted features list can contain some features that can make the classification task more difficult and less efficient.

First of all, rechecking the distribution of training set data to ensure that the training set is balanced enough for the model to get rid of overfitting the major label. Based on the following bar graph, it is clear that the dataset is very imbalanced, as the number of samples with label ‘0’ is many times more than the number of samples with label ‘1’. This suggests that the dataset may need to be resampled to prevent the case that the models will predict label ‘0’ almost all the time and still have high accuracy on the training set.

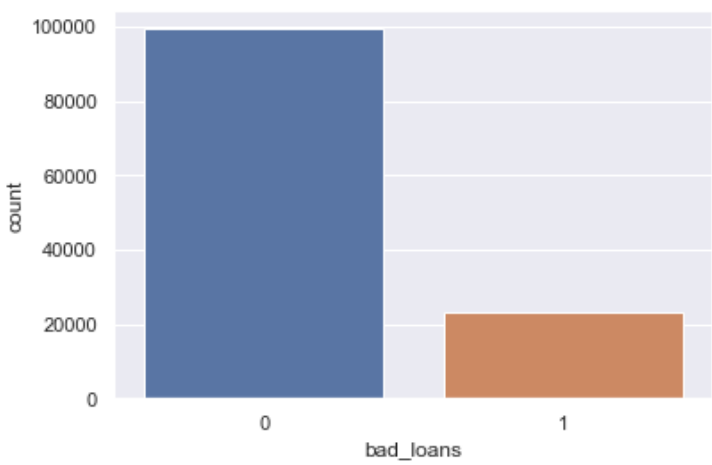

Figure 2 - Distribution of labels in the data set

Secondly, use the Seaborn library with the given pair plot option to plot out the distribution of samples for pairs of features (only numeric features can be plotted). The intention of this was to observe and see if the relationships of some features are too high that one of them should be eliminated to prevent the situation of being linearly dependent and can cause the training inaccurate [5].

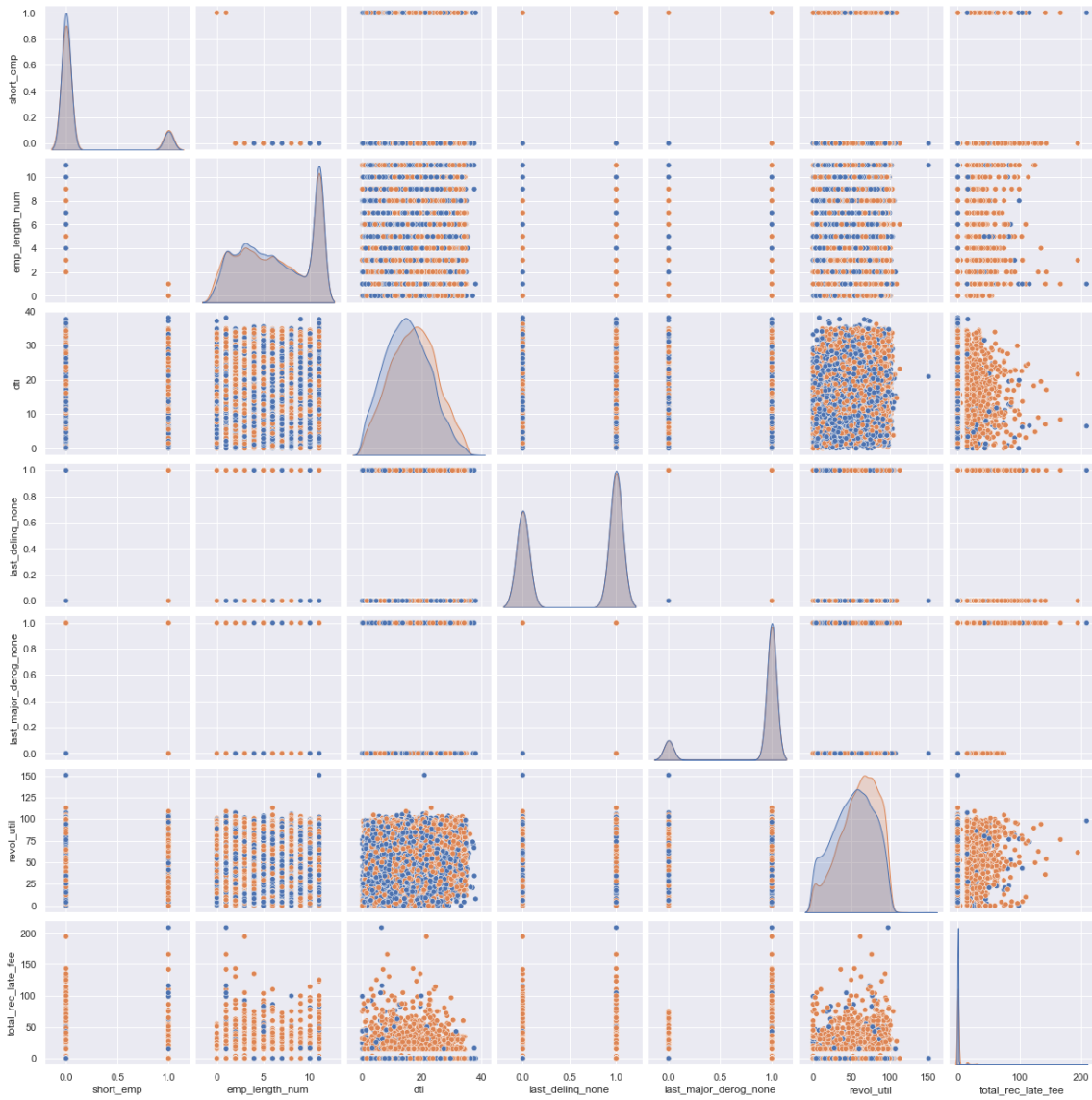

Figure 3 - Relationships between numeric features of the training set

The pair plot generally shows that some correlations can be figured out. For example, it can be observed that the features 'last_delinq_none' and 'last_major_derog_none' correlate were loans with both features holding values 0 tending to be bad_loans (label 0). However, this may not be true because the colors do not clearly illustrate the density. Using a scatterplot with opacity for each point, we can see that the results can be different.

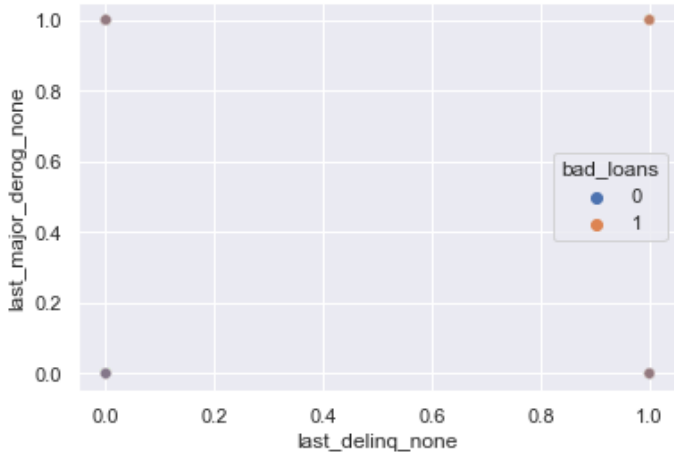

Figure 4 - Distribution of samples based on last_major_derog_none and last_delinq_none features

Clearly, good loans and bad loans varied with the values of both features. With a large number of features and not too high a correlation[2], it is hard to determine which attributes to keep or remove.

After examining, a different method to evaluate the features for selection besides correlation was decided to be used. As there are some of the categorical features that tend to have a good relationship with the target labels, one of the choices is to rate the features by chi-squared score[3]. However, there exists a problem in this step. After filtering the features that tend to be meaningful for the prediction, some of the attributes have very high chi-squared scores (many times more than the others). Researching on some of the sources, this is possible to happen but those features are not always good to keep. Because of that, both cases will be tested (keeping those features with extremely high chi-squared scores and removing them) to compare the results of training.

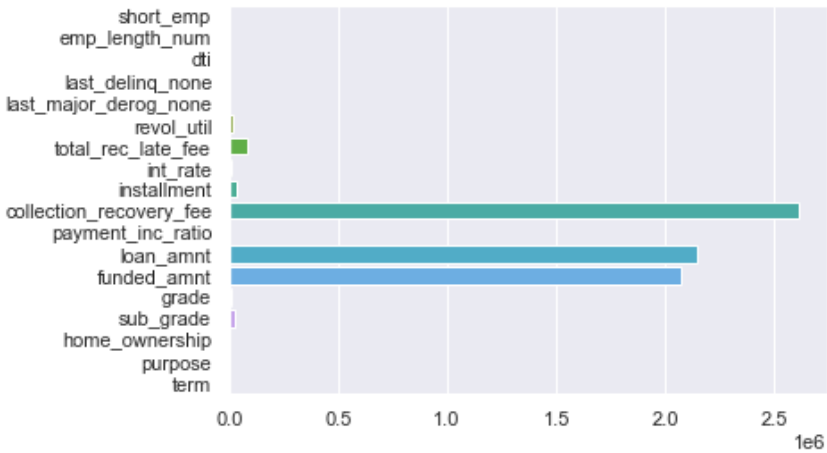

Figure 5 - Some of the features have extremely high chi-squared scores

[2] the correlations were only testable for features that are numerical

[3] chi-squared points measures dependence between stochastic variables, so using this function "weeds out" the features that are the most likely to be independent of class and therefore irrelevant for classification.

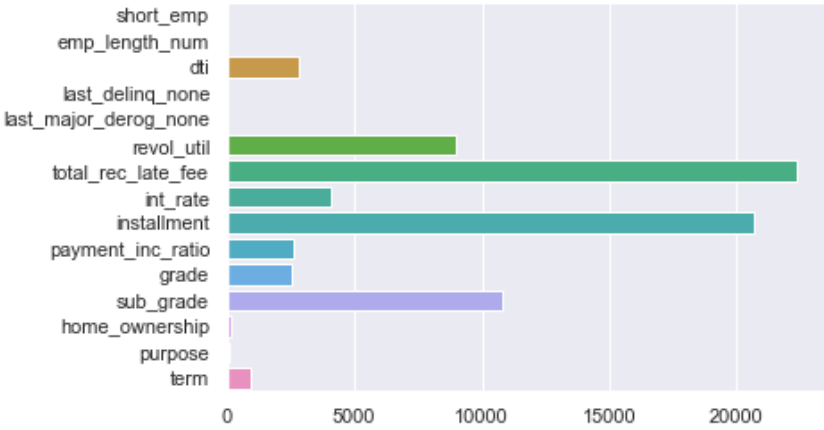

Figure 6 – The features' chi-squared scores after filtering the extreme ones

It is observable that some features were likely to not be independent of the target labels. Among those, **total_rec_late_fee**, **installment**, **sub_grade,** and **revol_util** are the notable ones. This information can be helpful for the next step of Feature Selection.

2. RESAMPLING

Having identified the problem of imbalance in the training set, the required process to perform is to resample the dataset. Two common techniques can be used for the resampling of the dataset, including:

- Oversampling: duplicate examples of the minority class to equal with those of the majority class
- Undersampling: delete examples of the majority class to equal to those of the minority class.

Undersampling can lose some data, but oversampling can lead to the case that some samples are involved in the training many times and have more impact on the training.

Examining the number of samples, the technique of random undersampling was selected to be used (perform the deletion randomly until the samples of 2 labels are equal). This is to ensure that the number of training samples is about 46,000, and the training time is not too slow compared to oversampling where the number of samples can be nearly 200,000.

3. FEATURE SELECTION

In the first step of feature selection, the features were examined by their meanings and some features that are sure to be meaningless for the training (such as id, member_id, url, description, etc.) were discarded.

Besides, recommendations from the 2 given notebooks were looked up, and agreed to reuse the set of features suggested in the notebook.

```
features = ['grade',                    # grade of the loan
            'sub_grade',                # sub-grade of the loan
            'short_emp',                # one year or less of employment
            'emp_length_num',           # number of years of employment
            'home_ownership',           # home_ownership status: own, mortgage or rent
            'dti',                      # debt to income ratio
            'purpose',                  # the purpose of the loan
            'term',                     # the term of the loan
            'last_delinq_none',         # has borrower had a delinquincy
            'last_major_derog_none',    # has borrower had 90 day or worse rating
            'revol_util',               # percent of available credit being used
            'total_rec_late_fee',       # total late fees received to day
           ]
```

Figure 7 - Features used in the given Notebook

Finally, some information on the results of the chi-squared score examination mentioned in the previous step was researched. As the approach to testing both cases of keeping the high-value chi-squared scores feature and removing them was agreed upon, some features decided to add to the dataset for training include:

- **int_rate**: the interest rate of the loan
- **installment**: the monthly payment owed by the borrower if the loan originates
- **collection_recovery_fee**: post charge off collection fee
- **payment_inc_ratio**: borrower's payment to income ratio
- **loan_amnt**: the listed amount of the loan applied for by the borrower
- **funded_amnt**: the total amount committed to that loan at that point in time

Training with the complete training set is a normal and basic approach, but it should be considered that not all of the features are useful for the classification task. For this task, it is also hard to know whether it is better to shortlist a group of features among the selected ones to be used for the training.

Because of that, the approach chosen for this step is to loop through and train the models on a different set of features, that will be selected from the complete set ordered by an evaluation method. For this task, two evaluation measurements were used, including:

- **chi-squared score**: the one discussed in the previous section that will be used to filter out the features that are likely to be independent and may provide not much value for the training.
- **mutual information**: this is a measurement that illustrates the dependency between two variables. Calculating the mutual information estimate between the features and the target labels can provide useful information for selecting the features that can better the training model.

Because there are no clear hints of which of those evaluation measurements are better, it is better to examine both of them. So basically, for each classifier, 10 training loops will be run and the results will be recorded to compare and select the best one later, including:

- For 5 loops, training the models with different choices of best features (eliminate at most 4 features from the training set, and the last loop is the training set will all features retained) ordered by the chi-squared score.
- For the remaining 5 loops, training the models with the same number of features, but ordered by the mutual information score.

4. ONE-HOT ENCODING

After selecting the appropriate features to train for each training loop, the categorical features of the training set will pass through a one-hot encoding.

The idea of one-hot encoding is to encode each possible value for each of the categorical features into a new feature holding a value of 0 (not of that column type) or 1 (of that column type).

For example, the dataset has an attribute “home_ownership” that can hold 4 possible values: MORTAGE, RENT, OWN, and OTHER. One-hot Encoding will discard the feature “home_ownership” and add to the dataset 4 new features: home_ownership.MORTAGE, home_ownership.RENT, home_ownership.OWN and home_ownership.OTHER where each of the features will hold a value of 0 or 1.

In that way, the data can better represent the state of having or not having a specific value of a feature, and not fall into the case that the model may find an inaccurately linear correlation between the feature and the target labels if we keep the old feature and simply encode them into numerical values from categorical values.

5. FEATURE SCALING

There are two techniques for Feature Scaling: Normalization and Standardization. Both techniques can bring value to the training process: Normalization will normalize the data points into a specific positive range while Standardization will standardize the data points into a specific area surrounding the center of the axis. Which technique works better depends on the dataset and the training algorithm used.

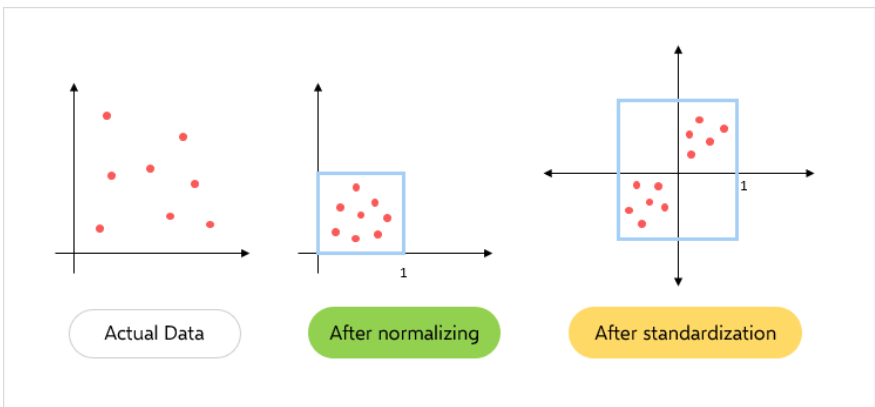

Figure 8 - Normalization vs Standardization [6]

To identify a better approach for feature scaling with the given dataset, some trials were performed with two techniques, as recommended in the reference [7].

Due to some technical difficulties during the examining step, the results for the comparison were not recorded. However, it was noticed that the Normalization technique gave a slightly better result, with a little higher accuracies for the models.

Therefore, it came to the decision to use the technique of Min-max Normalization, to normalize all the features into the range of [0, 1]. This step can be helpful to many learning algorithms, especially those that train based on the distance between the data points.

6. HYPERPARAMETERS TUNING

Before training the dataset with different classifiers and comparing the performance, it is better to ensure the model used for each type of classifier is the optimized one. Although the performance of models strongly depends on the set of important hyperparameters for the model, it is not easy to define those hyperparameters to ensure the best performance for the model.

To tune the hyperparameters for each of the classifiers used, a technique provided by the sklearn library called RandomSearchCV was used. This process will input a set of range of possible values for some of the hyperparameters, initialize and train the model on the given dataset on a randomly selected subset of the combination of values for hyperparameters[4] and return the best-performed model of all.

The returned best set of hyperparameters will then be used to initialize the model and perform the training, validating the results to compare with the other models.

The selection for which hyperparameters to tune, and which set of values to search was mostly based on some of the online references (which will be included in the References section).

```
#Find the set of best parameters
params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'criterion': ["gini", "entropy"]
}
```

Figure 9 - Example of hyperparameters tuned for Decision Tree

7. TRAINING

After completing all of the previous processes, the training loops for a different set of features were performed.

During the training process, we record the result for each training loop of each classifier, along with some evaluation measurements like Accuracy, F1-score, Precision, Recall, and the set of tuned hyperparameters.

The images capturing the training process are included in the submission (inside the images folder).

[4] The GridSearchCV technique, which tested all possible combinations of values for hyperparameters due to time limit for training, was not used. For some classifier like SVM, the training time can be very long, so using this approach would require a huge amount of time.

```
In [32]: dnn = deepNeuralNetwork()
         create_json("deepNeuralNetwork.txt", dnn)

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 8)                 544
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 625
Trainable params: 625
Non-trainable params: 0
_________________________________________________________________
Epoch 1/40
WARNING:tensorflow:AutoGraph could not transform <bound method Dense.call of <keras.layers.core.Dense object at 0x0000027AE59B2
F70>> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: invalid syntax (tmpw5bplrqz.py, line 48)
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <bound method Dense.call of <keras.layers.core.Dense object at 0x0000027AE59B2F70>> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: invalid syntax (tmpw5bplrqz.py, line 48)
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
373/373 [==============================] - 11s 2ms/step - loss: 0.6911 - accuracy: 0.5720 - val_loss: 0.6407 - val_accuracy: 0.
6276
Epoch 2/40
373/373 [==============================] - 0s 965us/step - loss: 0.6369 - accuracy: 0.6335 - val_loss: 0.6331 - val_accuracy:
0.6392
Epoch 3/40
373/373 [==============================] - 0s 944us/step - loss: 0.6306 - accuracy: 0.6405 - val_loss: 0.6309 - val_accuracy:
0.6435
Epoch 4/40
373/373 [==============================] - 0s 976us/step - loss: 0.6277 - accuracy: 0.6496 - val_loss: 0.6304 - val_accuracy:
0.6415
Epoch 5/40
373/373 [==============================] - 0s 969us/step - loss: 0.6277 - accuracy: 0.6454 - val_loss: 0.6301 - val_accuracy:
0.6438
Epoch 6/40
```

Figure 10 - Example of the training process of Neural Network in our notebook

8. EVALUATION

To evaluate the performance of models, two main measurements decided to use are Accuracy and F1-score.

- **Accuracy**: a good measurement to overall evaluate how accurate can the model predict target labels of samples in the test set. However, it is not enough to only evaluate the accuracy because of the possibility of predicting a label for much more time than the remaining label (as discussed in the previous sections).
- **F1-score**: this measurement will take into account some additional measurements including Precision (the ratio of truly good loans over the samples predicted good loans by the model) and Recall (the ratio of samples predicted good loans by the model over the total number of truly good loans).

Using two evaluations measurements rather than one ensures that the model can be better evaluated and prevent the situation of a false result. Relying only on the accuracy of the model is not a good idea.

VII. MODELS

1. DECISION TREE

A. RESULT

Decision Tree was one of the classifiers that created a huge difference between keeping the features with too high chi-squared scores and eliminating them. The difference is significant in both accuracies and weighted F1 scores.

The best-recorded accuracy for the 1st approach (eliminating high chi-squared scores features) was **64.4%**, compared to the accuracy of **74.5%** for the 2nd approach.

13 features	{'min_samples_leaf': 100, 'max_depth': 10, 'criterion': 'gini'}	0.643952
14 features	{'min_samples_leaf': 100, 'max_depth': 20, 'criterion': 'entropy'}	0.745032

Figure 11 - Accuracies of Decision Tree for both runs

The detailed result for each training loop of different features used is available in the Appendices section of the report.

2. K-NEAREST NEIGHBORS

A. SUMMARIZED THEORY BASE

K-Nearest Neighbors (KNN) is one of the simplest supervised classifiers that can be applied to both classification and regression problems.

Categorized into the group of "lazy-loading" algorithms, the KNN algorithm does not study anything about the training set but uses the training set to make calculations for each prediction sample with the existing training set and returns an appropriate result [8].

KNN uses the concept of distance between data points. Simply describing, the algorithm will use a distance calculation method to calculate the "distance" between the predicting sample with every sample in the training set, and filtered out a group of k-nearest samples to the predicting one. The majority of target labels among those neighbors will be outputted as the predicted target label for the predicting sample.

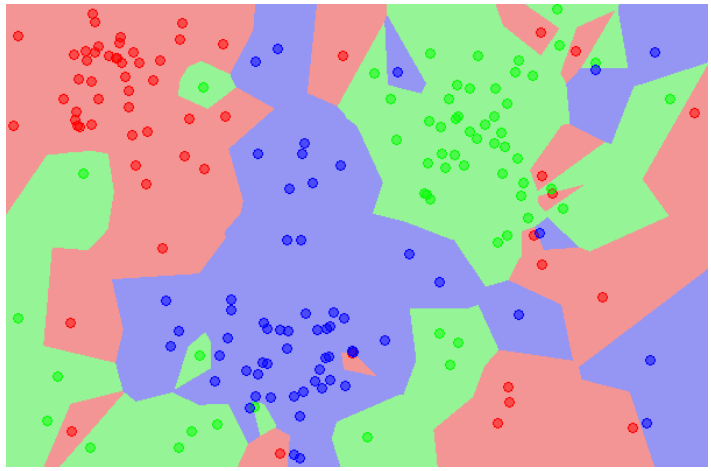

Figure 12 - Example of 1NN map, each data point is predicted based on the one nearest neighbor [8]

B. RESULT

For the K-Nearest Neighbor algorithm, the results for both runs were not different too much. The best-recorded accuracy for the 1st approach (eliminating high chi-squared scores features) was **61.46%**, compared to the accuracy of **61.65%** for the 2nd approach.

11 features	{'weights': 'distance', 'n_neighbors': 15, 'metric': 'manhattan'}	0.614579
14 features	{'weights': 'uniform', 'n_neighbors': 15, 'metric': 'manhattan'}	0.616523

Figure 13 - Accuracies of KNN for both runs

The detailed result for each training loop of different features used is available in the Appendices section of the report.

3. RANDOM FOREST

A. SUMMARIZED THEORY BASE

Random Forest is an algorithm that can be described as a combination of Decision Trees. Random Forest resolves one of the common problems of Decision Tree, overfitting into the training set and can perform very badly on validation.

The idea of Random Forest is that for each training iteration, a specific number of Decision Trees will be used that will train on a different subset of training data. Then the prediction result of the "forest" will be the label that is predicted by the most number of trees in the forest. This can get rid of the situation when there is a single tree and the tree is structured in a way that matches too much to the training set.

B. RESULT

Similar to the Decision Tree, the Random Forest classifier observed a large difference between the accuracies for both runs. Especially in the 2nd run when the Random Forest classifier was the one that gave the best accuracy and weighted F1 score.

The best-recorded accuracy for the 1st approach (eliminating high chi-squared scores features) was **65.62%**, compared to the accuracy of **75.67%** for the 2nd approach.

14 features	{'n_estimators': 90, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'auto'}	0.656156
17 features	{'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto'}	0.756695

Figure 14 - Accuracies of Random Forest for both runs

4. LOGISTIC REGRESSION

A. SUMMARIZED THEORY BASE

Logistic Regression is one of the most commonly used classifiers for binary classification problems. Despite being called "regression", the method's outcome is binary.

Logistic Regression is a special case of Linear Regression where the target variable is categorical. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function [9].

The main idea of Logistic Regression is similar to that of Linear Regression, to build a linear boundary that assumes a linear relationship between the variables of input with the target label. However, Logistic Regression uses the sigmoid function (also called logistic function) that gives an 'S-shaped curve that can take any real-valued number and map it into a value between 0 and 1 [9]. In that way, the output can be either 0 or 1, and thus be suitable for binary classification tasks.

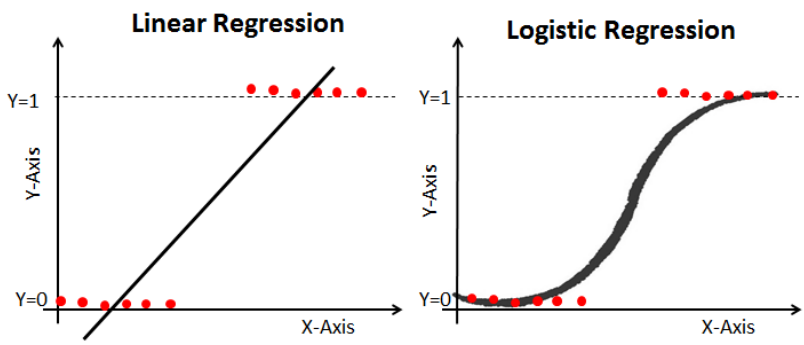

Figure 15 - Linear Regression vs Logistic Regression [9]

B. RESULT

For Linear Regression, the best-recorded accuracy for the 1st approach (eliminating high chi-squared scores features) was **65.93%**, compared to the accuracy of **69.83%** for the 2nd approach.

15 features	{'solver': 'lbfgs', 'penalty': 'l2', 'C': 100}	0.659287
18 features	{'solver': 'newton-cg', 'penalty': 'l2', 'C': 100}	0.698272

Figure 16 - Accuracies of Linear Regression for both runs

The detailed result for each training loop of different features used is available in the Appendices section of the report.

5. BERNOULLI NAÏVE BAYES

A. SUMMARIZED THEORY BASE

Naïve Bayes classifiers are a collection of classification algorithms based on the Bayes' Theorem in statistics. Naïve Bayes is not a single algorithm but a family of algorithms that shares a common principle: every pair of features being classified is independent of the other [10].

Naïve Bayes works based on two elements:

- Bayes' Theorem: the probability of an event occurring given the probability of another event that has already occurred. For the case of classification, the model uses the mathematical equation of $P(y|X)$, which is equivalent to "The probability of reaching a label y given the set of attributes X".
- Naïve Assumption: additional assumption of independence between the variables is put along with Bayes's Theorem. Instead of using the set of attributes X as a whole, naïve assumption suggests splitting the "evidence" into independent parts in assumption.

Using the above elements, Naïve Bayes works by calculating the probability of a specific target label for the set of input features as independent variables and will output the target label with the highest probability.

There are some of the algorithms in the family of Naïve Bayes, such as:

- Gaussian Naïve Bayes: normally used for classification tasks where the data's features are assumed to be distributed according to a Gaussian distribution[5].
- Bernoulli Naïve Bayes: normally used when the features are binary (contain the value of 0 or 1).
- Multinomial Naïve Bayes: normally used for multinomially distributed data, such as text classification.

For this dataset, our team observed that there exist both binary features and numeric features that tend to be distributed according to a Gaussian distribution, so there are two options available. However, after one-hot encoding, the number of binary features compared to numeric features are much more. Because of that, our team decided to use Bernoulli Naïve Bayes for this dataset. Some experiments were conducted by our team also pointed out that Bernoulli Naïve Bayes performed better for this dataset compared to Gaussian Naïve Bayes.

B. RESULT

For Bernoulli Naïve Bayes, the best-recorded accuracy for the 1st approach (eliminating high chi-squared scores features) was **62.68%**, compared to a rapid higher accuracy of **73.22%** for the 2nd approach.

13 features	{'alpha': 0.1}	0.626782
14 features	{'alpha': 10.0}	0.732181

Figure 17 - Accuracies of Bernoulli Naïve Bayes for both runs

[5] also called Normal distribution. Basically describing, when plotted, the normal distribution dataset gives a bell-shaped curve which is symmetric about the mean of the features [10].

The detailed result for each training loop of different features used is available in the Appendices section of the report.

6. SUPPORT VECTOR MACHINE

A. SUMMARIZED THEORY BASE

Support Vector Machine (SVM) is a supervised learning algorithm that applies to both classification and regression problems.

The basic idea of the algorithm is to find hyperplane(s) that will distinguish the data points are in an n-dimensional space (where n is the number of features available).

For example, with data points containing 2 features, the algorithm will find a line that best split the data points into 2 areas, each area is assumed to correspond to a target label.

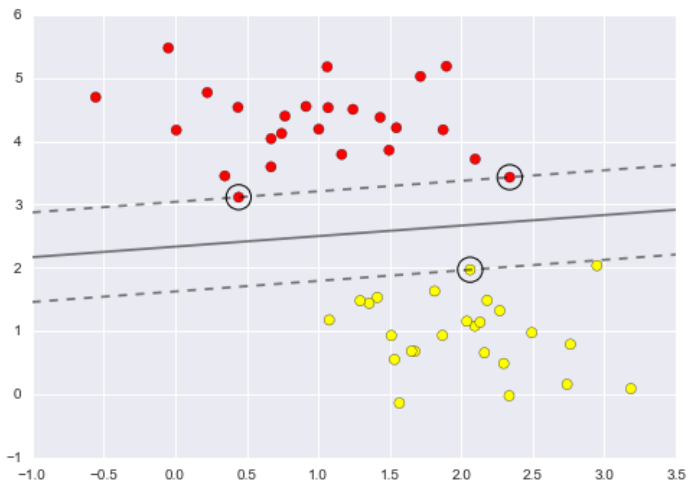

Figure 18 - Example of a hyperplane founded by SVM in 2D data space [11]

The mathematical computations for the algorithm are very complex so in this summary of the theory base for the algorithm, our team will just mention the idea and assumption of the algorithm rather than the mathematical aspects of the algorithm.

The target of the SVM algorithm is to find a hyperplane that maximizes the margin, which is the distance between the hyperplane with the closest data points for each target label. The iterations of SVM algorithm contain complex mathematical computations that maximize the margin, reducing the possibility of misclassification for new data points during prediction.

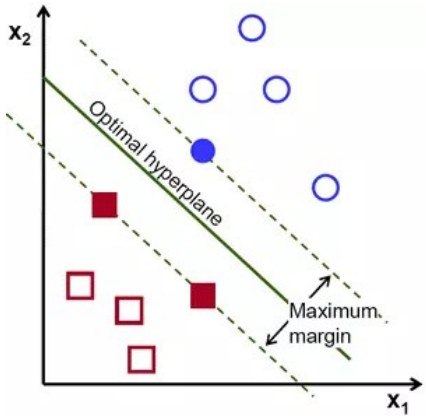

Figure 19 - Maximum margin in an optimal hyperplane found by SVM in 2D data space [12]

SVM fits very well for data with multiple features (even though the SVM performs badly for those classification tasks where the number of features exceeds the number of training samples) and is a good choice for memory optimization. One of the difficult points our team encountered with the algorithm was the training time was too long, compared to the remaining classifiers.

B. RESULT

For Support Vector Machine, the difference between the accuracies of both runs was very small. The best-recorded accuracy for the 1st approach (eliminating high chi-squared scores features) was **64.43%,** compared to the accuracy of **65.12%** for the 2nd approach.

11 features	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.644276
14 features	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.651188

Figure 20 - Accuracies of SVM for both runs

The detailed result for each training loop of different features used is available in the Appendices section of the report.

7. ARTIFICIAL NEURAL NETWORK

A. SUMMARIZED THEORY BASE

The final choice for the classifier model was using a deep neural network, chosen also as a chance to practice the theory lectures on this subject about the Artificial Neural Network.

An Artificial Neural Network (ANN) is a Deep Learning structure, of which the design was inspired by the structure and activation of the human brain.

Artificial Neural Networks are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer.

- Each node (neuron) connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network [13]

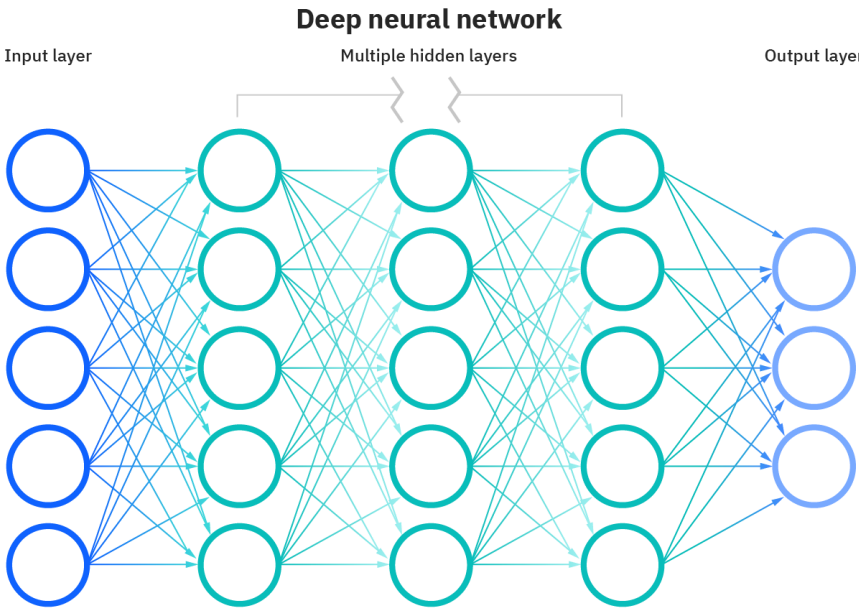

Figure 21 - Basic structure of a deep neural network [13]

Each node can be thought of as its linear regression model, composed of input, data, weights, threshold, and an output.

The features of the dataset correspond to the single nodes in the input layer. The nodes between layers are connected via channels, that contain a specific value of weight illustrating the importance of the nodes that channels are connecting.

In the first iteration, the weights are normally generated randomly. The input data is passed into the input layer, triggering the activation functions, and propagating through the layers until reaching the output layer, where an appropriate output label will be determined by the result of the layer. This process is called forward propagation.

In the next iterations of the training process, an error function[6] is used to calculate whether each node in the output layers is making minor or significant errors. Using this information, the network will adjust the weights of the channels (the more significant errors made, the more modification to the weight is required) to gradually improve the performance of the whole Neural Network. This process is called backward propagation.

[6] a function that calculates the magnitudes of error, that is, how large is the gap between the output result with the correct result.

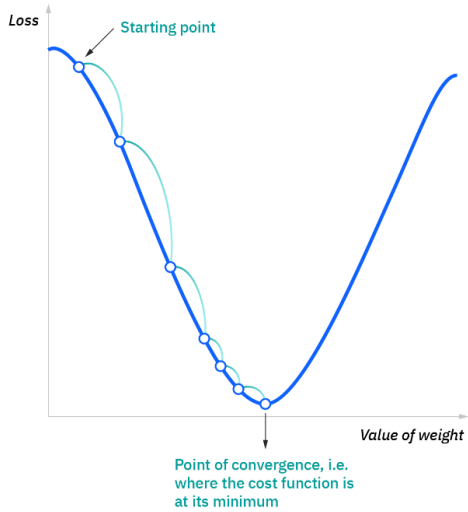

Figure 22 - The main iteration of Neural Network is to find the convergence point for each channel, to minimize the loss [13]

For this dataset, different structures of neural network layers were examined to pick up the most suitable one to be used. Those are some of the following options used (input layer not included):

- 3 layers:
  - 128 neurons, using ReLU activation function
  - 128 neurons, using ReLU activation function
  - 1 neuron, using sigmoid function (output layer)
- 3 layers:
  - 512 neurons, using ReLU activation function
  - 512 neurons, using ReLU activation function
  - 1 neuron, using sigmoid function (output layer)
- 3 layers:
  - 8 neurons, using ReLU activation function
  - 8 neurons, using ReLU activation function
  - 1 neuron, using sigmoid function (output layer)
- 3 layers:
  - 64 neurons, using ReLU activation function
  - 32 neurons, using ReLU activation function
  - 1 neuron, using sigmoid function (output layer)

The last networks outperform the accuracies of all models. Because of that, the structure of the last network was chosen to be used for this classification task.

Also applying different sets of features, 40 epochs were run for each training loop. Besides, the technique of early stopping[7] was used to prevent overfitting the training set.

[7] a technique that is commonly used in neural network training, to stop the training at an appropriate point when the model tends to begin overfitting the training set, performing better on the training set but worse on the validation test.

B. RESULT

For Artificial Neural Network, the best-recorded accuracy for the 1st approach (eliminating high chi-squared scores features) was **65.94%**, compared to the accuracy of **71.70%** for the 2nd approach.

After adding the features with high chi-squared scores features, although the accuracy of the neural network increased, it did not gain too much information compared to some other classifiers, losing its placement as the best-performed model in our team's first run.

14 features	{}	0.659395
17 features	{}	0.716955

Figure 23 - Accuracies of Neural Network for both runs

The detailed result for each training loop of different features used is available in the Appendices section of the report.

VIII. MODELS COMPARISON

In this section, the comparison between the accuracies and weighted F1 scores of the models for both runs with different approaches was conducted.

1. FIRST RUN

In the first run, after eliminating some features with extremely high chi-squared scores, the result is as follows:

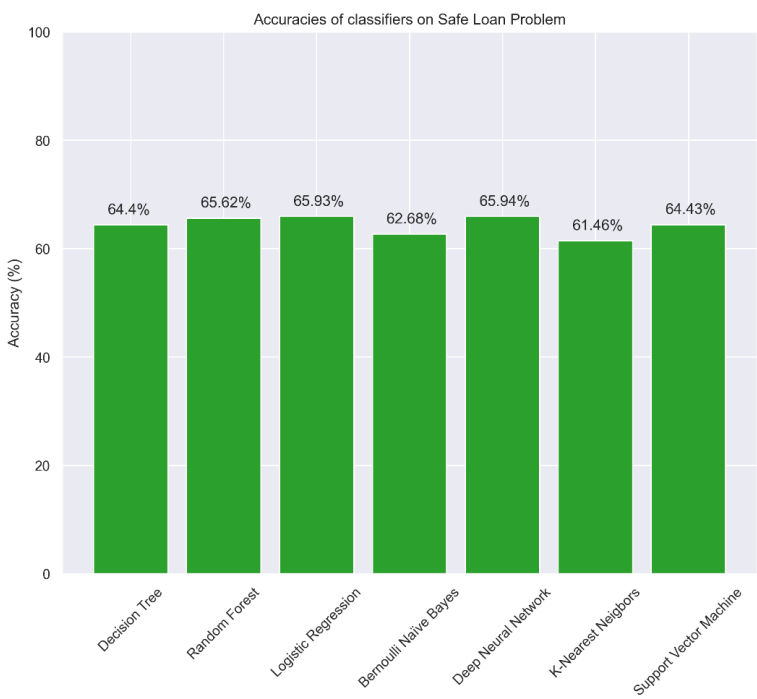

Figure 24 - Accuracies of models on the first run

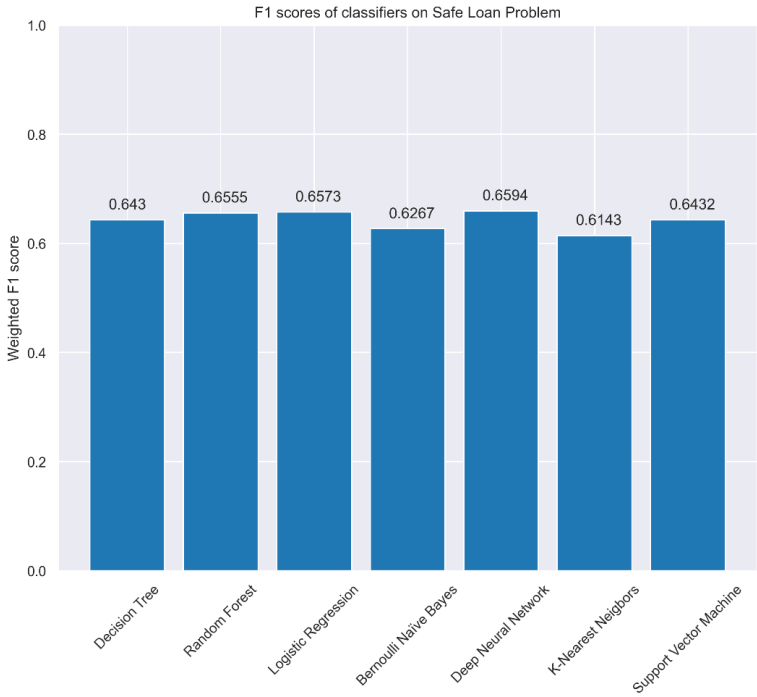

Figure 25 - Weighted F1 scores of models on the first run

The result showed that Deep Neural Network outperforms the remaining algorithms in both terms of accuracies and weighted F1 scores.

Generally, it is observable that the dataset seems to be suitable for training for some classifiers, reaching an accuracy of more than 65%.

Although still having accuracies of more than 61% (compared to the baseline in the given notebook, those accuracies are quite good), Bernoulli Naïve Bayes and K-Nearest Neighbors did not perform as well as the remaining ones. This will be noticed to examine the result of those 2 models in the second run with a different approach.

Finally, the difference between the accuracies and the weighted F1 scores for each model was not too different, meaning that the models seemed to make equal predictions for both labels, not too many predictions for one of them.

2. SECOND RUN

In the second run, keeping the features with extremely high chi-squared scores, the result is as follows:

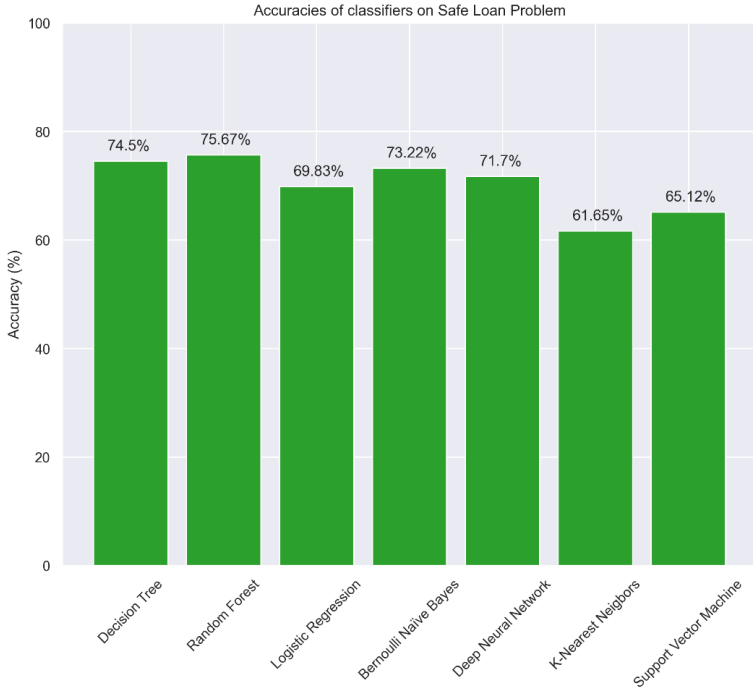

Figure 26 - Accuracies of models on the second run

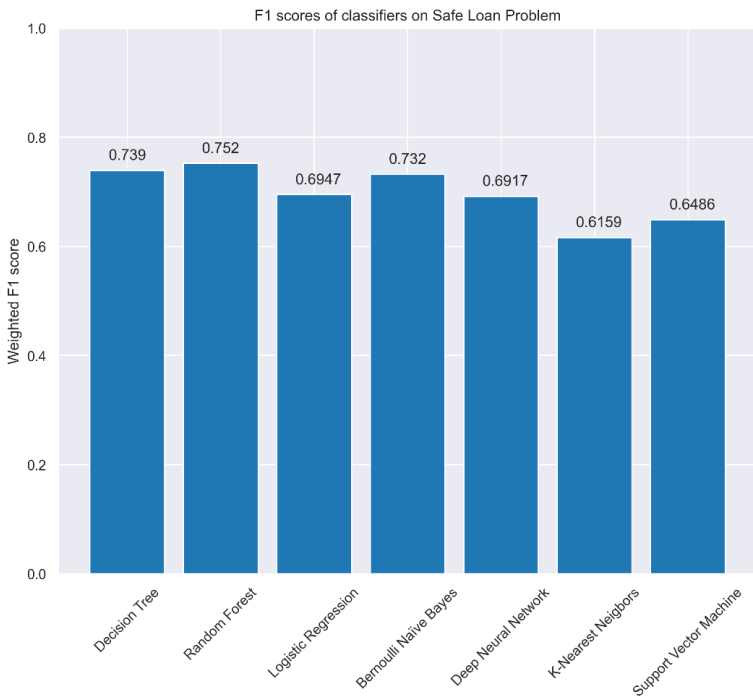

Figure 27 - Weighted F1 scores of models on the second run

Random Forest clearly gained much information after adding more features, became the model with the best performance in our second run, with an accuracy of 75.67% and a weighted F1 score of 0.752. Decision Tree also recorded a significant difference.

Notably was the change of Bernoulli Naïve Bayes. Being in the group of two classifiers that performed worse than the remaining ones in our first run, this classifier outperformed even Neural Network in the second run, being the 3rd best-performed classifier among all.

IX. CONCLUSION AND RECOMMENDATIONS

1. CONCLUSION

In conclusion, the performance of different types of classifiers for this classification task depends on multiple factors. With a small number of features, the models performed with quite similar evaluation results in both terms of accuracies and F1 scores, besides K-Nearest Neighbors and Bernoulli Naïve Bayes.

However, adding more features brought much information for the tree-based algorithms (Decision Tree and Random Forest) and probability-based algorithms (Bernoulli Naïve Bayes), even outperforming the neural network. The distance-based algorithms seemed not to have gained much information from the expansion of the feature set.

K-Nearest Neighbors seemed to be the algorithm that is not suitable for the task in all cases, as it performed the worst among all in both runs.

2. RECOMMENDATIONS

Some of the recommendations that can be pointed out after conducting the classification task on different classifiers and comparing the results include:

- To better the performance of the classifiers, it may depend largely on the dataset. So it would be better to first examine the dataset and features to see any optimization can be made before performing the training.
- The dataset and classification tasks may be more suitable for a specific group of algorithms. For example, our team observed that the task of this report worked well with tree-based algorithms and probability-based algorithms, but not the distance-based ones. Choosing which algorithm to use to receive the best performance may require a long process of testing, evaluating, and comparing.

X. CONVENIENCE AND DIFFICULTIES

1. CONVENIENCE

- The existing libraries such as sklearn, tensorflow, keras, etc. provided almost support functions for all of the chosen algorithms and preprocessing steps required for the project. Because of that, more than half the time of the project was spent researching and finding appropriate approaches instead of having to code from scratch every required step, which is very complex.
- Along with the dataset, the two given example notebooks were efficiently used as a referencing source to complete the requirements of the project. This was also one of the conveniences of this project.

2. DIFFICULTIES

- Platform: there were difficulties running the example notebook on our local machine, due to the conflict between the supporting operating systems for the library and the team's operating systems (Windows 10 and Windows 11). Some additional tools had to be used to run a virtual machine for the task.
- Time: Although the given time for the project was 3 weeks, there were difficulties in the first 2 weeks spent exploring the concepts and steps for implementing the project. That is because the project was handed out before the concepts of Machine Learning and Neural Network were taught in the theory part of the subject. And it was also challenging to research and understand how to apply and use the libraries, as well as how to make a complete pipeline for a classification task.
- Dataset: A dataset with nearly 70 features was also a challenge for our team. Too many features required spending an amount of time understanding the key features and deciding which features to keep for training or to remove.
- Training machine: At first, Google Colab was intended to be used for training, especially "heavily computed" tasks like Hyperparameters Tuning. However, due to the memory limit of Google Colab, it was not able to complete the training on that platform. Finally, own devices had been used to train the models, and it cost quite an amount of time.
- Knowledge: Besides lacking knowledge of implementation steps for a classification problem or platform, deep knowledge about some aspects of mathematics to clearly understand all of the machine learning algorithms was also lacking. Because of that, in some of the classifiers used, it was only able to stop at the step of understanding generally how the models are trained, not the deep formulas, and thus cannot optimize the models.

XI. APPENDICES

Table 1 - Validation results of the models on the first run

Algorithm		Model Params	Accuracy	Weighted F1
Decision Tree	**11 features**	{'min_samples_leaf': 100, 'max_depth': 10, 'criterion': 'entropy'}	0.640173	0.636758
	12 features	{'min_samples_leaf': 10, 'max_depth': 5, 'criterion': 'gini'}	0.637797	0.636373
	13 features	{'min_samples_leaf': 100, 'max_depth': 10, 'criterion': 'gini'}	0.643952	0.642992
	14 features	{'min_samples_leaf': 100, 'max_depth': 20, 'criterion': 'entropy'}	0.641253	0.639629
	15 features	{'min_samples_leaf': 100, 'max_depth': 5, 'criterion': 'gini'}	0.637797	0.636373
Random Forest	**11 features**	{'n_estimators': 90, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt'}	0.650108	0.649245
	12 features	{'n_estimators': 90, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'auto'}	0.653348	0.652566
	13 features	{'n_estimators': 70, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto'}	0.653132	0.652302
	14 features	{'n_estimators': 90, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'auto'}	0.656156	0.655502
	15 features	{'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'sqrt'}	0.655184	0.654423
Logistic Regression	**11 features**	{'solver': 'lbfgs', 'penalty': 'l2', 'C': 100}	0.657883	0.655839
	12 features	{'solver': 'newton-cg', 'penalty': 'l2', 'C': 100}	0.658639	0.65673
	13 features	{'solver': 'newton-cg', 'penalty': 'l2', 'C': 100}	0.658423	0.656513
	14 features	{'solver': 'lbfgs', 'penalty': 'l2', 'C': 100}	0.658747	0.656833
	15 features	{'solver': 'lbfgs', 'penalty': 'l2', 'C': 100}	0.659287	0.657303
Bernoulli Naïve Bayes	**11 features**	{'alpha': 0.5}	0.624406	0.623859
	12 features	{'alpha': 1.0}	0.625702	0.625612
	13 features	{'alpha': 0.1}	0.626782	0.626709
	14 features	{'alpha': 0.01}	0.626674	0.626602
	15 features	{'alpha': 0.01}	0.625702	0.625537
Deep Neural Network	**11 features**	{}	0.653456	0.651617
	12 features	{}	0.647192	0.64376
	13 features	{}	0.651944	0.651533
	14 features	{}	0.659395	0.659381
	15 features	{}	0.65486	0.654402
K-Nearest Neigbors	**11 features**	{'weights': 'distance', 'n_neighbors': 15, 'metric': 'manhattan'}	0.614579	0.614247
	12 features	{'weights': 'distance', 'n_neighbors': 15, 'metric': 'manhattan'}	0.614579	0.614275
	13 features	{'weights': 'distance', 'n_neighbors': 15, 'metric': 'manhattan'}	0.608315	0.608074
	14 features	{'weights': 'distance', 'n_neighbors': 15, 'metric': 'manhattan'}	0.608963	0.608923
	15 features	{'weights': 'distance', 'n_neighbors': 15, 'metric': 'manhattan'}	0.607235	0.607176
Support Vector Machine	**11 features**	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.644276	0.643197
	12 features	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.643521	0.642417
	13 features	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.642657	0.641371
	14 features	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.641253	0.640264
	15 features	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.640605	0.639488

Table 2 - Validation results of the models on the second run

Algorithm		Model Params	Accuracy	Weighted F1
Decision Tree	**14 features**	{'min_samples_leaf': 100, 'max_depth': 20, 'criterion': 'entropy'}	0.745032	0.738968
	15 features	{'min_samples_leaf': 100, 'max_depth': 5, 'criterion': 'gini'}	0.741253	0.735917
	16 features	{'min_samples_leaf': 100, 'max_depth': 5, 'criterion': 'gini'}	0.741253	0.735917
	17 features	{'min_samples_leaf': 10, 'max_depth': 5, 'criterion': 'gini'}	0.740929	0.735545
	18 features	{'min_samples_leaf': 100, 'max_depth': 5, 'criterion': 'gini'}	0.741253	0.735917
Random Forest	**14 features**	{'n_estimators': 70, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto'}	0.754104	0.749072
	15 features	{'n_estimators': 90, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt'}	0.751728	0.747021
	16 features	{'n_estimators': 80, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt'}	0.75486	0.750485
	17 features	{'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto'}	0.756695	0.752008
	18 features	{'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt'}	0.755292	0.75091
Logistic Regression	**14 features**	{'solver': 'liblinear', 'penalty': 'l2', 'C': 100}	0.698272	0.694419
	15 features	{'solver': 'liblinear', 'penalty': 'l2', 'C': 100}	0.698164	0.694569
	16 features	{'solver': 'newton-cg', 'penalty': 'l2', 'C': 100}	0.69622	0.692558
	17 features	{'solver': 'newton-cg', 'penalty': 'l2', 'C': 100}	0.69784	0.694277
	18 features	{'solver': 'newton-cg', 'penalty': 'l2', 'C': 100}	0.698272	0.694656
Bernoulli Naïve Bayes	**14 features**	{'alpha': 10.0}	0.732181	0.732038
	15 features	{'alpha': 10.0}	0.731857	0.731715
	16 features	{'alpha': 0.01}	0.731965	0.731874
	17 features	{'alpha': 10.0}	0.732073	0.731974
	18 features	{'alpha': 1.0}	0.730886	0.730804
Deep Neural Network	**14 features**	{}	0.685529	0.649851
	15 features	{}	0.688553	0.687618
	16 features	{}	0.712203	0.691692
	17 features	{}	0.716955	0.691413
	18 features	{}	0.714579	0.688356
K-Nearest Neigbors	**14 features**	{'weights': 'uniform', 'n_neighbors': 15, 'metric': 'manhattan'}	0.616523	0.615946
	15 features	{'weights': 'uniform', 'n_neighbors': 15, 'metric': 'manhattan'}	0.613607	0.613158
	16 features	{'weights': 'uniform', 'n_neighbors': 15, 'metric': 'manhattan'}	0.606911	0.606579
	17 features	{'weights': 'uniform', 'n_neighbors': 15, 'metric': 'manhattan'}	0.606479	0.60626
	18 features	{'weights': 'uniform', 'n_neighbors': 15, 'metric': 'manhattan'}	0.611339	0.611144
Support Vector Machine	**14 features**	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.651188	0.648622
	15 features	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.648272	0.646487
	16 features	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.647948	0.64621
	17 features	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.649352	0.647278
	18 features	{'kernel': 'poly', 'gamma': 'scale', 'C': 10}	0.644384	0.641971

XII. REFERENCES

[1] J. Brownlee, "4 Types of Classification Tasks in Machine Learning," 8 April 2020. [Online]. Available: https://machinelearningmastery.com/types-of-classification-in-machine-learning/.

[2] ScikitLearn, "Decision Trees," n.d.. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html.

[3] "GeeksforGeeks," 22 June 2021. [Online]. Available: https://www.geeksforgeeks.org/decision-tree/.

[4] "Cây Quyết Định (Decision Tree)," 6 June 2019. [Online]. Available: https://trituenhantao.io/kien-thuc/decision-tree/.

[5] V. R, "Feature selection — Correlation and P-value," 11 September 2018. [Online]. Available: https://towardsdatascience.com/feature-selection-correlation-and-p-value-da8921bfb3cf.

[6] A. S. Chadha, "What Do Normalization and Standardization Mean? When to Normalize Data and When to Standardize Data?," 12 July 2021. [Online]. Available: https://becominghuman.ai/what-does-feature-scaling-mean-when-to-normalize-data-and-when-to-standardize-data-c3de654405ed.

[7] aniruddha, "Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization," 3 April 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

[8] T. Vu, "K-nearest neighbors," 18 January 2017. [Online]. Available: https://machinelearningcoban.com/2017/01/08/knn/.

[9] A. Navlani, "Understanding Logistic Regression in Python Tutorial," 17 December 2019. [Online]. Available: https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python.

[10] N. Kumar, "Naive Bayes Classifiers," 02 Feb 2022. [Online]. Available: https://www.geeksforgeeks.org/naive-bayes-classifiers/.

[11] J. VanderPlas, "In-Depth: Support Vector Machines," n.d.. [Online]. Available: https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html.

[12] T. Huynh, "Giới thiệu về Support Vector Machine (SVM)," 20 August 2020. [Online]. Available: https://viblo.asia/p/gioi-thieu-ve-support-vector-machine-svm-6J3ZgPVElmB.

[13] I. C. Education, "Neural Networks," 17 August 2020. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks.

[14] C. Grupman, "Data Cleaning and Preparation for Machine Learning," 19 June 2019. [Online]. Available: https://www.dataquest.io/blog/machine-learning-preparing-data/.

[15] A. Gupta, "2020," 10 October. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/.

[16] J. Brownlee, "How to Perform Feature Selection with Categorical Data," 25 November 2019. [Online]. Available: https://machinelearningmastery.com/feature-selection-with-categorical-data/.

[17] G. Dutta, "Hyperparameter Tuning in Decision Trees," 25 September 2020. [Online]. Available: https://www.kaggle.com/code/gauravduttakiit/hyperparameter-tuning-in-decision-trees/notebook.

[18] A. Samudra, "k-NN with Hyperparameter Tuning," 25 April 2020. [Online]. Available: https://www.kaggle.com/code/arunimsamudra/k-nn-with-hyperparameter-tuning/notebook.

[19] K. Naik, "P2 : Logistic Regression - hyperparameter tuning," 12 September 2021. [Online]. Available: https://www.kaggle.com/code/funxexcel/p2-logistic-regression-hyperparameter-tuning.

[20] A. Sharma, "Naive Bayes with Hyperpameter Tuning," 26 January 2021. [Online]. Available: https://www.kaggle.com/code/akshaysharma001/naive-bayes-with-hyperpameter-tuning/notebook.

[21] J. Brownlee, "Random Oversampling and Undersampling for Imbalanced Classification," 15 January 2020. [Online]. Available: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/.

[22] H. Bowne-Anderson, "Kaggle Tutorial: EDA & Machine Learning," 21 December 2017. [Online]. Available: https://www.datacamp.com/community/tutorials/kaggle-machine-learning-eda.

[23] Nguyễn Ngọc Thảo, Nguyễn Hải Minh, "CS300 Lecture Slides - Introduction to Neural Networks," Ho Chi Minh City, n.d..

[24] Nguyễn Ngọc Thảo, Nguyễn Hải Minh, "CS300 Lecture Slides - Machine Learning Basic Algorithms," Ho Chi Minh City, n.d..

[25] J. Brownlee, "Tune Hyperparameters for Classification Machine Learning Algorithms," 13 December 2019. [Online]. Available: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/.