Predicting Loan Applicant Risk Profile Using Machine Learning

Commerce 3FN3 Big Data in Finance

G-15

Table of Contents

Problem Background

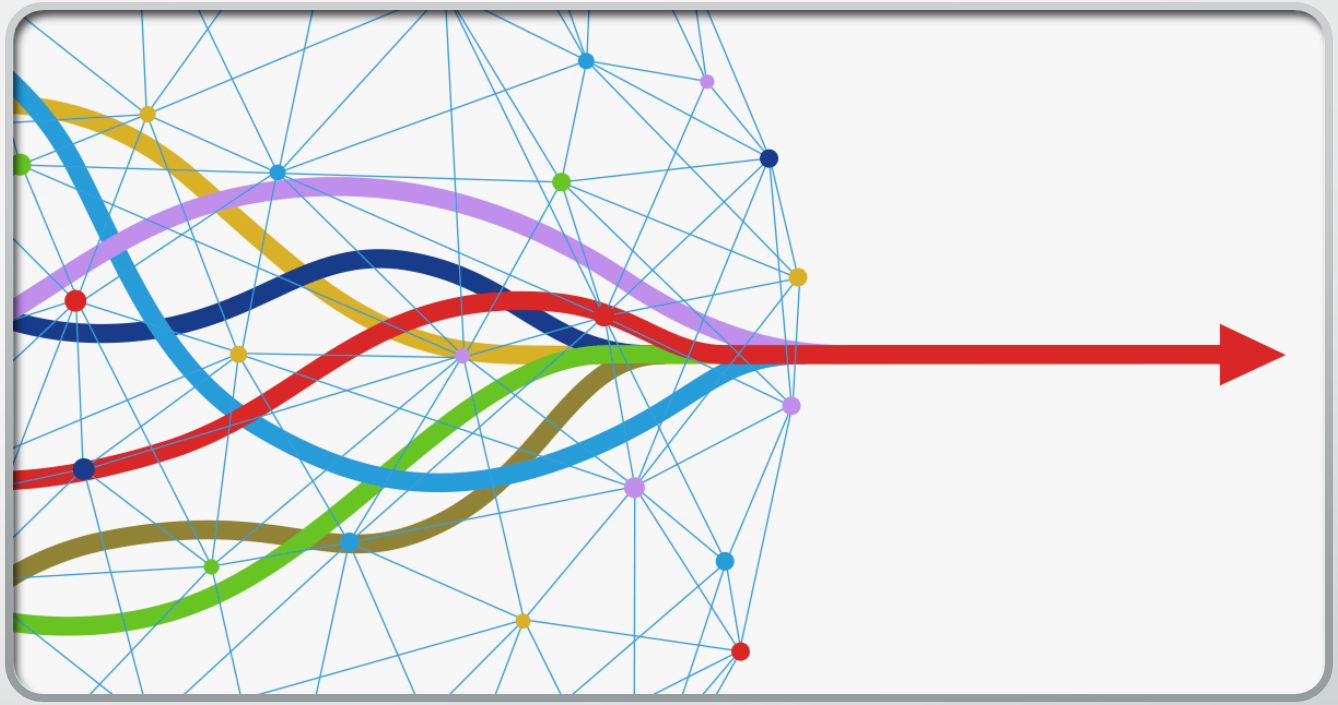
Every year banks receive thousands of loan applications each with unique circumstances that must be evaluated. This presents a variety of issues for banks to solve such as time consumption, inconsistencies, limited scalability, underutilization of data, and compliance risks. The first and most prominent issue, is how much time must be allocated to evaluating loan applications by employees of the firm. Traditionally, financial institutions have used teams of people to evaluate each applicant's information, and this is both expensive and time consuming. Human workers also make mistakes. During each evaluation process, errors can be made, and this can lead to unfair outcomes for loan applicants or increased risk for the institution. Another disadvantage of relying only on employees to process loan applications is that the demand for loans is not linear. When a surge of loan requests come in, having a limited team can cause bottlenecks in the workflow leading to delays. Generally, depending on the amount and complexity of the loan request, institutions are flooded with vast amounts of data about each candidate. Without the support of software and machine learning, interacting with all the information about each loan applicant is impossible for even a large team of employees. Finally, manual systems for evaluating loan applications may lack transparency and fairness in lending decisions. Regulations require institutions to carefully follow a set a guideline for fairness and transparency, and human employees working with manual systems may struggle to document and justify decisions, exposing institutions to compliance risks. Clearly, a digital solution is required.

Current Applications

Banks such as Deutsche Bank, Bank of America, and JP Morgan Chase are on the right track. These banks are adopting machine learning and robotic process automation to improve their loan processing systems and address the plethora of issues. Some examples of software platforms that can streamline the loan processing workflow are nCino, LendingPad, Finastra, and Fiserv. nCino is a cloud-based platform that automates application intake, loan origination, underwriting, and compliance tracking. LendingPad is designed for mortgage lenders, and it offers tools for document management and risk assessment. Finastra and Finserv are major players in the financial software space, and they offer solutions for loan lifecycle management, with advanced analytics and machine learning that optimizes decision making.

Solution Framing

The next step for a definite solution to automating banks loans is combining the features of the existing systems into a streamlined and accurate program. This is what we hope to accomplish with our project. Our code will analyze large sets of data and will be supervised learning built off risk profiles. This project will provide an accurate, transparent, and simple system that highlights the risk of each loan applicant to assist institutions of all sizes in the decision-making process.



Conceptual Overview

The solution takes a large dataset in the form of client financial profiles, cleans the data and then trains a machine learning algorithm. The algorithm can then predict a new applicant's risk profile and determine if they should be approved for a loan or not.

Results and Output of the Solution

The algorithm itself, given client metrics, will evaluate the applicant based on previous loan applications from other clients and their outcomes. This evaluation will determine if the client is likely or unlikely to successfully pay off the loan and subsequently if they should be approved or declined for the loan.

Potential Savings and Improvements

- The solution has broad implications for the financial sector. Since loan applications are not spaced out linearly, a backlog of applications can often exist leading to overworked staff who are prone to making mistakes in the approval or denial of loan applications. This solution can be scaled very easily to meet the demand at a given time by utilizing more compute power which can otherwise be held in reserve for minimal costs. The direct savings for the financial institutions come from the reduced staff requirement for loan processing. A quantitative amount of savings cannot be determined at this stage of the solution however, the solution does have the potential to eliminate the need for human intervention in the loan processing system. There are also indirect savings created through a reduction in loan acceptance errors which could otherwise result in a loan not being repaid.
- Another important improvement for the financial institutions, is the ability to scale the solution to meet demand relatively quickly and easily when compared to hiring more employees to meet demand. This almost instantaneous ability to scale means that the institutions can process loans faster, leaving clients waiting for a shorter period of time resulting in a better customer experience which could increase product demand and subsequently revenue.

Required Tools

- Python
- Libraries: pandas, numpy, sklearn
- Machine Learning Algorithms: Random Forests, K-Nearest Neighbours, Gradient Boosting, Naïve Bayes, Logistic Regression, SVC

Implementation Plan

- For this solution to be fully implemented and tested to be reliable enough for banks and lenders to implement to its fullest extent would take a significant amount of time.
- Key Steps:
  - Data gathering, cleanup, and normalization from all past applications
    - May be difficult if many applications are stored physically
    - Must determine important metrics to be considered and categorized
  - Selection of optimal machine learning algorithm
    - Perform subset tests to determine which algorithm or combination of algorithms result in the most accurate results
  - Model Training
    - Train the optimal algorithm with all the cleaned and normalized client data
  - Algorithm testing on past applications
    - Test the algorithm with previous applications with known outcomes, and adjust the algorithm parameters until a satisfactory accuracy is achieved
  - Algorithm testing on new applications
    - Test the algorithm with applications it has not seen before and compare the algorithm output to that of current employees to ensure that the solution is still accurate
  - Creating a software implementation for the algorithm which acts as a front end to collect the data for new applications
    - Once the algorithm is tested for accuracy, a way to cleanly and easily have the algorithm process loan applications and provide a verdict for the next step in the loan journey must be created
  - Provide training to employees or other users of the solution on how to use it properly
  - Once the tool is created, the employees would need to be trained to use the tool properly, ensuring the most accurate results for the financial institutions.

Projected Timeline for Implementation

Small Institutions

For smaller institutions, the solution can be implemented more quickly as fewer large changes need to be made, in these cases an implementation period of 12-16 months could be realistic.

Large Institutions

For larger institutions such as big banks, where applications come in from hundreds of different cities and large corporate structure makes new solutions difficult to implement as they must integrate with all of the existing technologies and policies, here it may take as long as 5 years to implement this solution.

Solution Assumptions

The data is readily available and accessible

The data does not need to be cleanly organized but there must be a digital copy of everything, and the data should hopefully have some consistent data points across all applicants to be able to determine trends

The financial institution can pivot their process towards one of automation

Some Institutions may have limitations due to policy or lack of internal expertise

There are typical trends and rules for how loans are approved or denied

There must be a correlation between client data and risk rating for the machine learning algorithm to be accurate and useful

Data Gathering and Import

- The data would be collected from the financial institution being serviced. Ideally the raw client data is organized into a table or database of sorts which includes columns for each financial parameter of the client.
- For this solution implementation, the dataset was in a table format with 19 columns of client information including their risk rating

```
import pandas as pd
import numpy as np
```

```
df = pd.read_csv('financial_data.csv')
```

```
df.head(10)
```

Sampled Raw Data

	Age	Gender	Education Level	Marital Status	Income	Credit Score	Loan Amount	Loan Purpose	Employment Status	Years at Current Job	Payment History	Debt-to-Income Ratio	Assets Value	Number of Dependents	City	State	Country	Previous Defaults	Marital Status Change	Risk Rating
0	49	Male	PhD	Divorced	72799	688	45713	Business	Unemployed	19	Poor	0.154313	120228	0	Port Elizabeth	AS	Cyprus	2	2	Low
1	57	Female	Bachelor's	Widowed	NaN	690	33835	Auto	Employed	6	Fair	0.14892	55849	0	North Catherine	OH	Turkmenistan	3	2	Medium
2	21	Non-binary	Master's	Single	55687	600	36623	Home	Employed	8	Fair	0.362398	180700	3	South Scott	OK	Luxembourg	3	2	Medium
3	59	Male	Bachelor's	Single	26508	622	26541	Personal	Unemployed	2	Excellent	0.454964	157319	3	Robinhaven	PR	Uganda	4	2	Medium
4	25	Non-binary	Bachelor's	Widowed	49427	766	36528	Personal	Unemployed	10	Fair	0.143242	287140	NaN	New Heather	IL	Namibia	3	1	Low
5	30	Non-binary	PhD	Divorced	NaN	717	15613	Business	Unemployed	5	Fair	0.295984	NaN	4	Brianland	TN	Iceland	3	1	Medium
6	31	Non-binary	Master's	Widowed	45280	672	6553	Personal	Self-employed	1	Good	0.37889	NaN	NaN	West Lindaview	MD	Bouvet Island (Bouvetoya)	0	1	Low
7	18	Male	Bachelor's	Widowed	93678	NaN	NaN	Business	Unemployed	10	Poor	0.396636	246597	1	Melissahaven	MA	Honduras	1	1	Low
8	32	Non-binary	Bachelor's	Widowed	20205	710	NaN	Auto	Unemployed	4	Fair	0.335965	227599	0	North Beverly	DC	Pitcairn Islands	4	2	Low
9	55	Male	Bachelor's	Married	32190	600	29918	Personal	Self-employed	5	Excellent	0.484333	130507	4	Davidstad	VT	Thailand	NaN	2	Low

Data Clean up and Normalization

- The data was cleaned up by removing parameters which are difficult to develop a correlation with. In the case of this solution and dataset, removing the columns pertaining to city and state made the solution run much faster. Since there are multiple hundreds of potential cities and many states, finding a correlation between data points may become difficult unless the dataset is massive and includes many datapoints from each individual city.
- The data was normalized by filling in missing data points with median values from the whole dataset. The median value is used to prevent any changes to data averages while weighing each application consistently for the given information. For example, if payment history was an empty point, filling it in with poor could create a discrepancy in the overall correlation between risk profile and payment history from reality.

```
: #missing values per column
missing_vals = df.isnull().sum()
print(missing_vals)
```

```
Age                          0
Gender                       0
Education Level              0
Marital Status               0
Income                    2250
Credit Score              2250
Loan Amount               2250
Loan Purpose                 0
Employment Status            0
Years at Current Job         0
Payment History              0
Debt-to-Income Ratio         0
Assets Value              2250
Number of Dependents      2250
City                         0
State                        0
Country                      0
Previous Defaults         2250
Marital Status Change        0
Risk Rating                  0
dtype: int64
```

```
# get rid of location information, there are too many categories here
df = df.drop(columns=['City', 'State', 'Country'])
```

Sampled Cleaned and Normalized Data

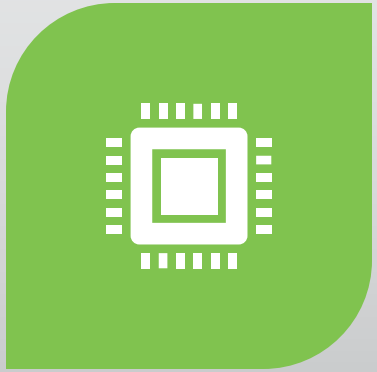
```
# store the columns that have null values
cols_missing_vals = ['Income', 'Credit Score', 'Loan Amount', 'Assets Value', 'Number of Dependents', 'Previous Defaults']

# figure out the medians for those columns
medians = df[cols_missing_vals].median()

# reassign null values within those columns to their respective medians
df[cols_missing_vals] = df[cols_missing_vals].fillna(medians)
```

	Age	Gender	Education Level	Marital Status	Income	Credit Score	Loan Amount	Loan Purpose	Employment Status	Years at Current Job	Payment History	Debt-to-Income Ratio	Assets Value	Number of Dependents	Previous Defaults	Marital Status Change	Risk Rating
0	49	Male	PhD	Divorced	72799	688	45713	Business	Unemployed	19	Poor	0.154313	120228	0	2	2	Low
1	57	Female	Bachelor's	Widowed	69773	690	33835	Auto	Employed	6	Fair	0.14892	55849	0	3	2	Medium
2	21	Non-binary	Master's	Single	55687	600	36623	Home	Employed	8	Fair	0.362398	180700	3	3	2	Medium
3	59	Male	Bachelor's	Single	26508	622	26541	Personal	Unemployed	2	Excellent	0.454964	157319	3	4	2	Medium
4	25	Non-binary	Bachelor's	Widowed	49427	766	36528	Personal	Unemployed	10	Fair	0.143242	287140	2	3	1	Low
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
14995	23	Non-binary	Bachelor's	Widowed	48088	609	26187	Home	Self-employed	2	Fair	0.317633	159362	4	2	0	Low
14996	56	Male	PhD	Single	107193	700	35111	Auto	Self-employed	10	Fair	0.155126	79102	2	0	0	Medium
14997	29	Non-binary	PhD	Married	46250	642	44369	Home	Unemployed	19	Excellent	0.593999	196930	4	2	1	High
14998	53	Non-binary	PhD	Divorced	40180	638	32752	Home	Self-employed	12	Excellent	0.478035	276060	2	0	2	High
14999	24	Non-binary	Bachelor's	Widowed	69773	765	27544	Personal	Self-employed	18	Excellent	0.116083	71699	3	3	2	Low

Machine Learning Setup



THE MACHINE LEARNING ALGORITHM IS SUPERVISED AS IT IS TRAINED OFF OF A KNOWN AND LABELED DATASET WITH DETERMINED OUTCOMES

DEFINE A TRAINING AND TESTING SPLIT OF 30% TEST AND 70% TRAINING

IMPORTED ALL THE RELEVANT AND REQUIRED LIBRARIES TO RUN THE SEVERAL DIFFERENT MACHINE LEARNING ALGORITHMS

```
from sklearn.model_selection import train_test_split

Y = df.iloc[:, 16]
X = df.iloc[:, 0:16]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 100)
```

Machine Learning Algorithms

- Random Forests
- K-Nearest Neighbours
- Gradient Boosting
- Naïve Bayes
- Logistic Regression
- SVC

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import linear_model
from sklearn.decomposition import PCA
from sklearn.svm import SVC
```

Random Forests

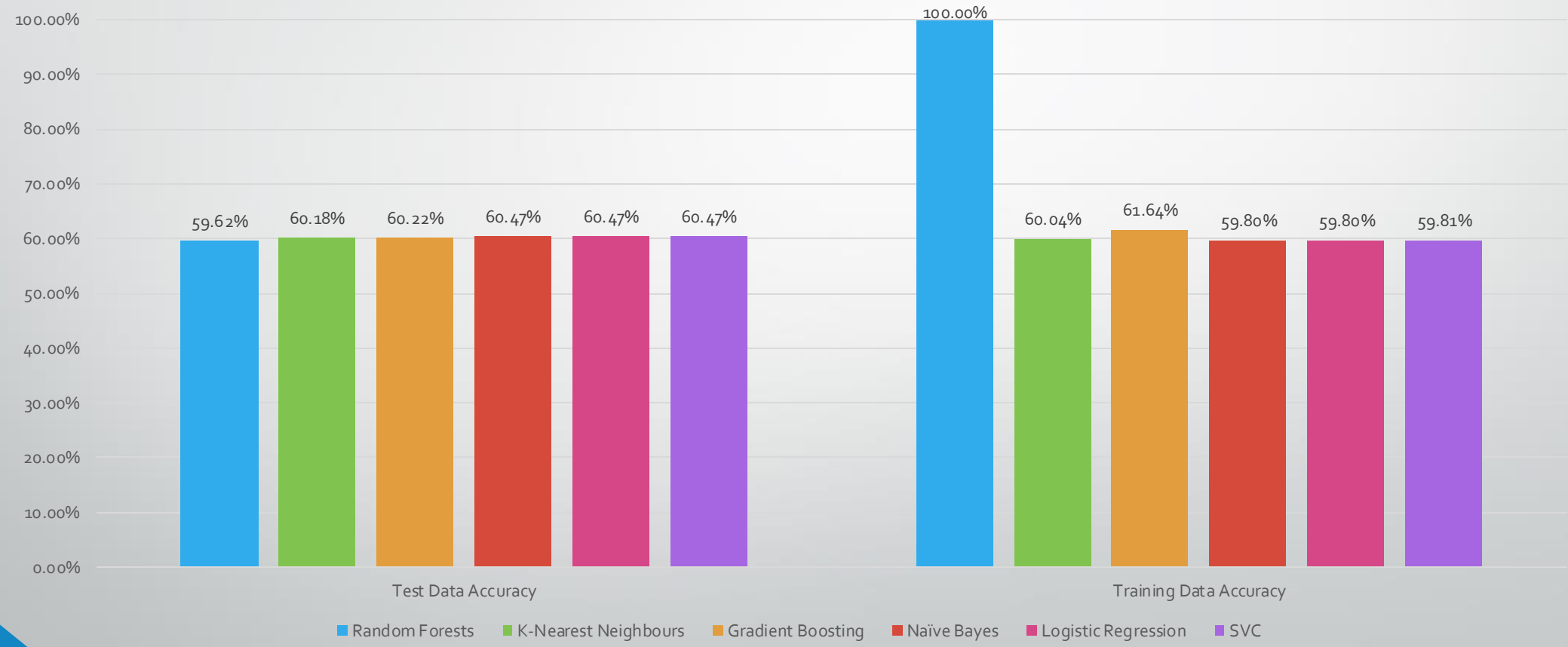
```
random_forests_pipe = Pipeline(
 [
 ('transform_columns', ColumnTransformation),
 ('pca', PCA(n_components = 32)),
 ('rf', RandomForestClassifier(random_state = 100))
 ]
)

random_forests_pipe.fit(X_train, Y_train)
print("Random Forests Accuracy(test data):", random_forests_pipe.score(X_test, Y_test) * 100, "%")
print("Random Forests Accuracy(training data):", random_forests_pipe.score(X_train, Y_train) * 100, "%")
```

Deep Learning

- To implement a deep learning algorithm, the dataset which the model is trained on would need to be very large. This likely limits the application of this solution to very large banks which hundreds of thousands of loan applications. Smaller firms would not have sufficient data to train a deep learning algorithm
- Since the dataset is structured into specific categories a solution utilizing deep learning may not be needed to achieve a desirable outcome
- Deep learning, utilizing a neural network, can be implemented for firms where there may not be consistent parameters across all applications.

Accuracy Results by Algorithm



Recommendations

- Implement a machine learning algorithm to assist in loan application processing
  - Will allow for more flexible reaction time and capacity for the processing of new applications.
  - No longer need to rely on employee capacity for tasks which can be automated
  - The tested algorithms had relatively low scores across the board, however, this is most likely due to the limited range of data upon which it was trained. By utilizing larger datasets, accuracy will likely improve. Furthermore, adjustments can be made to the algorithms themselves to increase accuracy by adjusting specific parameters such as the number of PCA components.

Business Impacts

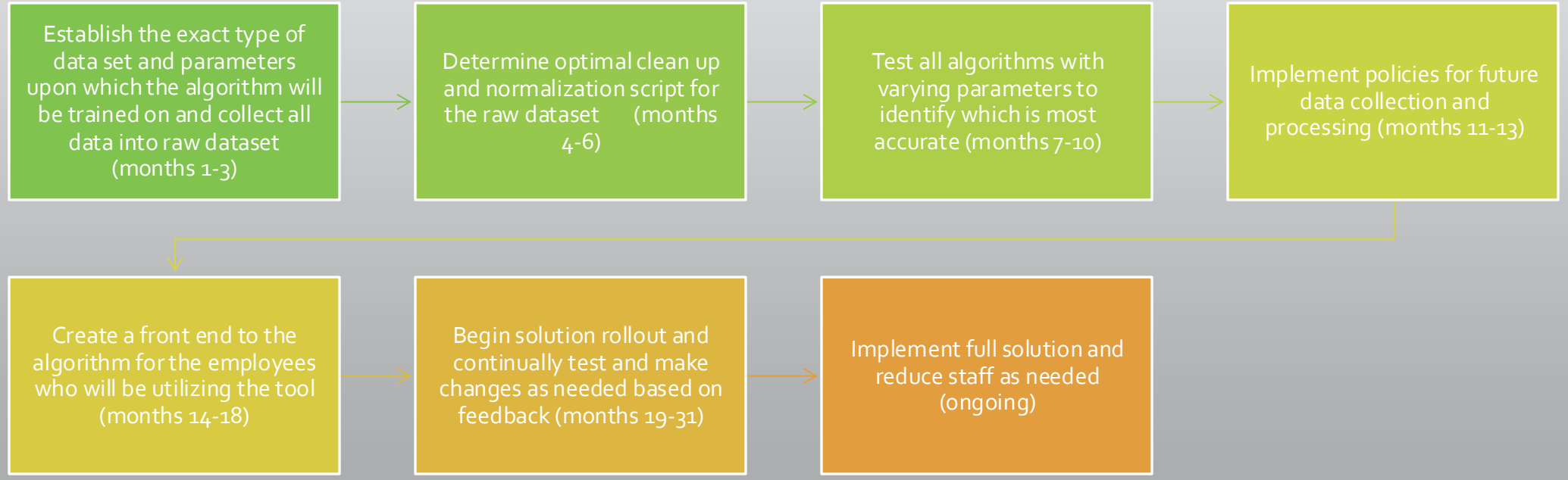
Employee Reduction

With the machine learning algorithm implemented, fewer man hours to process loan applications will be required

Faster Loan application turn around times

Algorithm can operate around the clock or be scaled easily as demand fluctuates, application backlog can be reduced or eliminated completely

Next Steps and Timeline



Summary of Learnings

Developed a deeper understanding of how to implement machine learning to real life applications

Learned of the limitations of machine learning and the importance of large datasets to train the algorithms upon

Became more familiar with each different machine learning model and their parameters, developing an understanding of the importance of fine tuning algorithms based on their application

Appendix

K-Nearest Neighbours

```
KNN_pipe = Pipeline(
 [
 ('transform_columns', ColumnTransformation),
 ('pca', PCA(n_components = 32)),
 ('knn', KNeighborsClassifier(n_neighbors=5))
 ]
)

KNN_pipe.fit(X_train, Y_train)
print("K-Nearest Neighbours Accuracy(test data):", KNN_pipe.score(X_test, Y_test) * 100, "%")
print("K-Nearest Neighbours Accuracy(training data):",KNN_pipe.score(X_train, Y_train) * 100, "%")
```

Gradient Boosting

```
gradient_boosting_pipe = Pipeline(
 [
 ('transform_columns', ColumnTransformation),
 ('pca', PCA(n_components = 32)),
 ('gb', GradientBoostingClassifier())
 ]
)

gradient_boosting_pipe.fit(X_train, Y_train)
print("Gradient Boosting Accuracy(test data):", gradient_boosting_pipe.score(X_test, Y_test) * 100, "%")
print("Gradient Boosting Accuracy(training data):", gradient_boosting_pipe.score(X_train, Y_train) * 100, "%")
```

Logistic Regression

```
logistic_regression_pipe = Pipeline(
 [
 ('transform_columns', ColumnTransformation),
 ('pca', PCA(n_components = 32)),
 ('logReg', linear_model.LogisticRegression())
 ]
)

logistic_regression_pipe.fit(X_train, Y_train)
print("Logistic Regression Accuracy(test data):", logistic_regression_pipe.score(X_test, Y_test) * 100, "%")
print("Logistic Regression Accuracy(training data):", logistic_regression_pipe.score(X_train, Y_train) * 100, "%")
```

Naive Bayes

```
naive_bayes_pipe = Pipeline(
 [
 ('transform_columns', ColumnTransformation),
 ('pca', PCA(n_components = 32)),
 ('nb', GaussianNB())
 ]
)

naive_bayes_pipe.fit(X_train, Y_train)
print("Naive Bayes Accuracy(test data):", naive_bayes_pipe.score(X_test, Y_test) * 100, "%")
print("Naive Bayes Accuracy(training data): ", naive_bayes_pipe.score(X_train, Y_train) * 100, "%")
```

SVC

```
svc_pipe = Pipeline(
 [
 ('transform_columns', ColumnTransformation),
 ('pca', PCA(n_components = 32)),
 ('svc', SVC())
 ]
)

svc_pipe.fit(X_train, Y_train)
print("SVC Accuracy(test data):", svc_pipe.score(X_test, Y_test) * 100, "%")
print("SVC Accuracy(training data):", svc_pipe.score(X_train, Y_train) * 100, "%")
```