Credit Card Approval Prediction

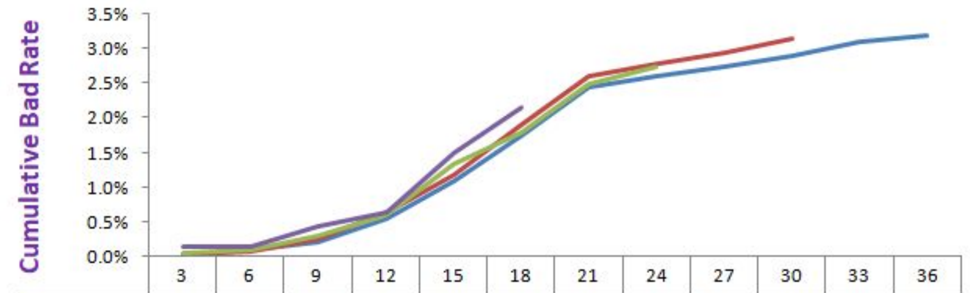
CS3244 PG7

Ang Lin Xuan
Han Weihang
Lim Tze Xin
Manish Seal
Teng Hao Earm
Yune Thiri Khin

Overview of Problem

Purpose of the Model

Related Works



	3	6	9	12	15	18	21	24	27	30	33	36
Q1-2016	0.03%	0.09%	0.20%	0.55%	1.10%	1.75%	2.45%	2.60%	2.75%	2.90%	3.10%	3.20%
Q3-2016	0.04%	0.08%	0.26%	0.65%	1.20%	1.90%	2.60%	2.80%	2.95%	3.15%		
Q1-2017	0.05%	0.09%	0.30%	0.60%	1.35%	1.80%	2.50%	2.75%				
Q3-2017	0.15%	0.15%	0.45%	0.65%	1.50%	2.15%						

Graph on Vintage Analysis taken from Credit Risk: Vintage Analysis on listendata.com

VINTAGE ANALYSIS

AIM: Other approaches to predict credit risk?

Pipeline of Model

Logistic Regression

credit_record.csv

Data Cleaning

Random Forest

Model Evaluation

Feature Engineering

application_record.csv

Neural Network

DATASETS

MODELLING

INSIGHTS

Credit Records Dataset

Credits Records Dataset

```
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   ID              1048575 non-null  int64
 1   MONTHS_BALANCE  1048575 non-null  int64
 2   STATUS          1048575 non-null  object
```

Overview of Dataset

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C
5	5001712	-1	C
6	5001712	-2	C

PURPOSE: Target variable for model

Credits Records Dataset

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C
5	5001712	-1	C
6	5001712	-2	C

ID: Client Number

MONTHS_BALANCE: Record Month

STATUS:
X: No loans
C: Loan paid off
0-5: Number of months loan is overdue

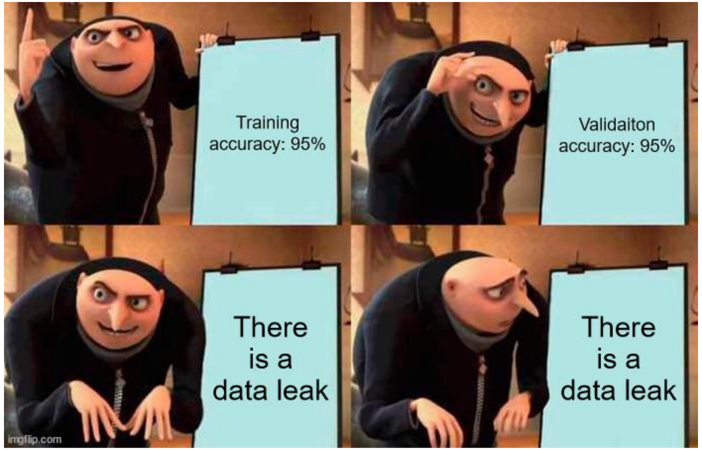
Training Set and Testing Set

Overview of Code

```python
# Split the data into training and testing sets, keeping customer IDs separate
train_ids, test_ids = train_test_split(df['ID'].unique(), test_size=0.2, random_state=42, stratify=None)

# Create the training and testing data subsets based on the selected customer IDs
train_data = df[df['ID'].isin(train_ids)]
test_data = df[df['ID'].isin(test_ids)]
```

Why?
Data Leakage!




```
train_data.shape
(838506, 3)

test_data.shape
(210069, 3)
```

Resulting Datasets

Feature Engineering

SUM(MONTHS_BALANCE) - 1

PERCENTAGE:
STATUS == X / STATUS == C

AVERAGE:
STATUS == 0/1/2/3/4/5

E.g.
(0+1+1+1)/4 = 0.75

	ID	Account_Length	X_Percentage	C_Percentage	Avg_Months_Overdue
0	5001711	3	0.25000	0.000000	0.000000
1	5001713	21	1.00000	0.000000	-1.000000
2	5001714	14	1.00000	0.000000	-1.000000
3	5001717	21	0.00000	0.227273	0.000000
4	5001718	38	0.25641	0.076923	0.076923

Null Values:
-1 for Avg_Months_Overdue, 0 for X_Percentage & C_Percentage

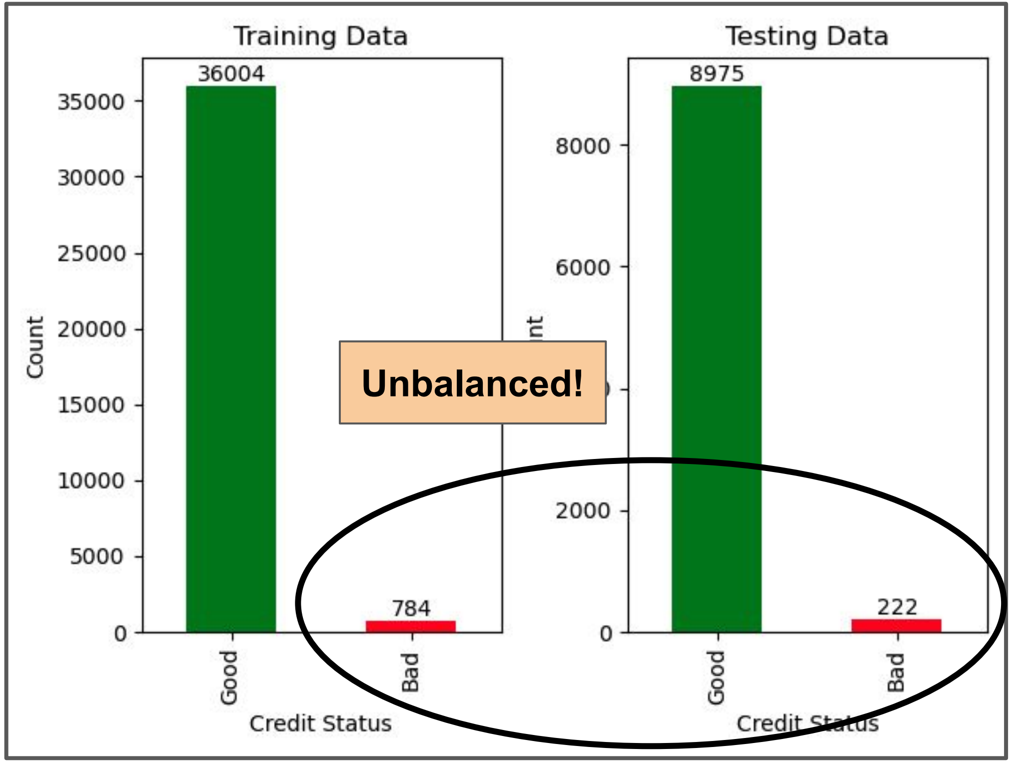
Splitting into “Good” / “Bad” Credit Records

CRITERIA FOR BAD

1) C_Percentage == 0
- Never paid off loans

2) Avg_Months_Overdue > 95th percentile

```
c_percentage_low = 0.0 # never paid off
avg_months_overdue_high = 0.195 # 95% of Avg_Months_Overdue
```



Application Records Dataset

Application Dataset

```
RangeIndex: 438557 entries, 0 to 438556
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ID                   438557 non-null  int64
 1   CODE_GENDER          438557 non-null  object
 2   FLAG_OWN_CAR         438557 non-null  object
 3   FLAG_OWN_REALTY      438557 non-null  object
 4   CNT_CHILDREN         438557 non-null  int64
 5   AMT_INCOME_TOTAL     438557 non-null  float64
 6   NAME_INCOME_TYPE     438557 non-null  object
 7   NAME_EDUCATION_TYPE  438557 non-null  object
 8   NAME_FAMILY_STATUS   438557 non-null  object
 9   NAME_HOUSING_TYPE    438557 non-null  object
 10  DAYS_BIRTH           438557 non-null  int64
 11  DAYS_EMPLOYED        438557 non-null  int64
 12  FLAG_MOBIL           438557 non-null  int64
 13  FLAG_WORK_PHONE      438557 non-null  int64
 14  FLAG_PHONE           438557 non-null  int64
 15  FLAG_EMAIL           438557 non-null  int64
 16  OCCUPATION_TYPE      304354 non-null  object
 17  CNT_FAM_MEMBERS      438557 non-null  float64
```

Overview of Dataset

Application Dataset

Reducing Dimensionality

```
df['FLAG_MOBIL'].describe()
```

```
count    438557.0
mean          1.0
std           0.0
min           1.0
25%           1.0
50%           1.0
75%           1.0
max           1.0
```

Removing Duplicates

```
df['ID'].duplicated().sum()
```

```
47
```

Application Dataset

Abnormal values

```
df['DAYS_EMPLOYED'].describe()

count    438557.000000
mean      60563.675328
std      138767.799647
min      -17531.000000
25%       -3103.000000
50%       -1467.000000
75%        -371.000000
max      365243.000000
```

Empty Values

```
df.isnull().sum()

ID                         0
CODE_GENDER                0
FLAG_OWN_CAR               0
FLAG_OWN_REALTY            0
CNT_CHILDREN               0
AMT_INCOME_TOTAL           0
NAME_INCOME_TYPE           0
NAME_EDUCATION_TYPE        0
NAME_FAMILY_STATUS         0
NAME_HOUSING_TYPE          0
DAYS_BIRTH                 0
DAYS_EMPLOYED              0
FLAG_WORK_PHONE            0
FLAG_PHONE                 0
FLAG_EMAIL                 0
OCCUPATION_TYPE        58868
CNT_FAM_MEMBERS            0
```

Application Dataset

Group1: Unemployed

Income Type

```
unemployed_df['NAME_INCOME_TYPE'].unique()
```

```
array(['Pensioner'], dtype=object)
```

Setting Occupation Type

```
unemployed_df = unemployed_df.fillna(value={'OCCUPATION_TYPE':'Pensioner'})
```

Application Dataset

Group1: Unemployed

Age Distribution

```
df['DAYS_BIRTH'].describe()
```

```
count    438510.000000
mean      15998.192778
std        4185.074780
min        7489.000000
25%       12514.000000
50%       15630.000000
75%       19484.000000
max       25201.000000
```

Employment Days Adjustment

```
unemployed_df['DAYS_EMPLOYED'] = unemployed_df.apply(lambda x: (x['DAYS_BIRTH']-retirement_age) \
                                      if (x['DAYS_BIRTH']>retirement_age) else ((x['DAYS_BIRTH'])), axis=1)
```

Application Dataset

Group1: Employed

Income Type

```
employed_df['NAME_INCOME_TYPE'].unique()
```

```
array(['Working', 'Commercial associate', 'State servant', 'Pensioner',
       'Student'], dtype=object)
```

Setting Occupation Type

```
employed_df = employed_df.fillna(value={'OCCUPATION_TYPE':'Unknown'})
```

Application Dataset (Feature Engineering)

Numerical Data: StandardScalar

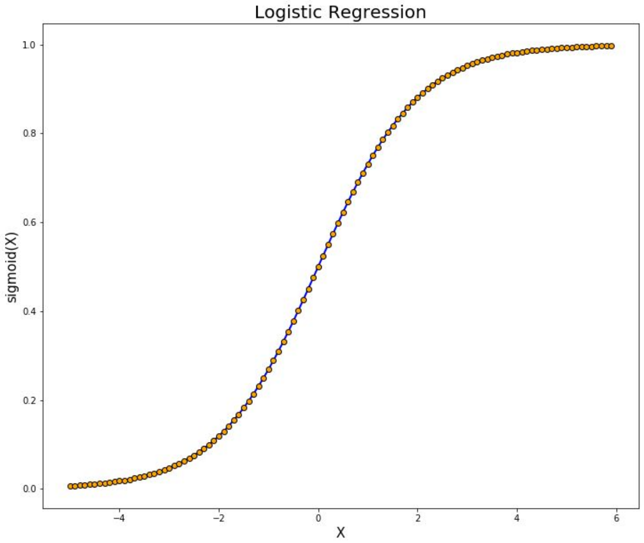
```
scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
```

Categorical Data: OneHotEncoder

```
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
encoded_columns = encoder.fit_transform(X_train[categorical_col])
```

Logistic Regression

Logistic Regression




```python
model = LogisticRegression(max_iter=1000)
model.fit(X_train_2label, y_train_2label)
y_pred = model.predict(X_test_2label)
```

Logistic Regression - Analysis of Features (Top 20)

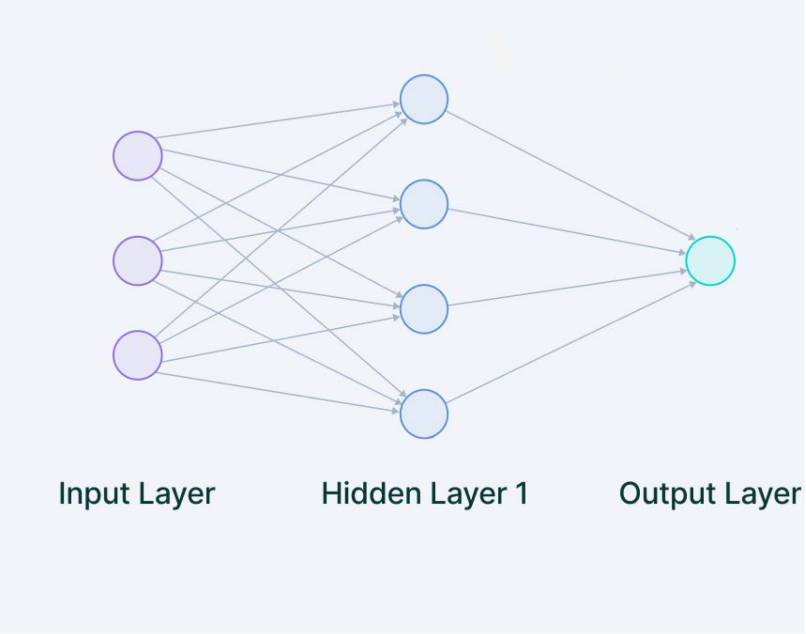
Feature	Coefficient
OCCUPATION_TYPE_Pensioners	1.827710
NAME_INCOME_TYPE_Pensioner	-1.357854
CNT_FAM_MEMBERS	-1.291707
CNT_CHILDREN	1.123387
OCCUPATION_TYPE_HR staff	-0.970123
NAME_FAMILY_STATUS_Married	0.901971
NAME_FAMILY_STATUS_Civil marriage	0.804473
NAME_FAMILY_STATUS_Separated	-0.767451
OCCUPATION_TYPE_Realty agents	0.717867
OCCUPATION_TYPE_Waiters/barmen staff	-0.692681
OCCUPATION_TYPE_IT staff	-0.677235
NAME_HOUSING_TYPE_Office apartment	-0.673166
OCCUPATION_TYPE_Cooking staff	-0.600913
NAME_FAMILY_STATUS_Single / not married	-0.583813
NAME_HOUSING_TYPE_Municipal apartment	0.535924
NAME_HOUSING_TYPE_Co-op apartment	0.484418
OCCUPATION_TYPE_Accountants	0.465875
OCCUPATION_TYPE_Private service staff	0.456994
NAME_INCOME_TYPE_Working	0.426057
NAME_INCOME_TYPE_State servant	0.374375

Neural Network

Neural Network

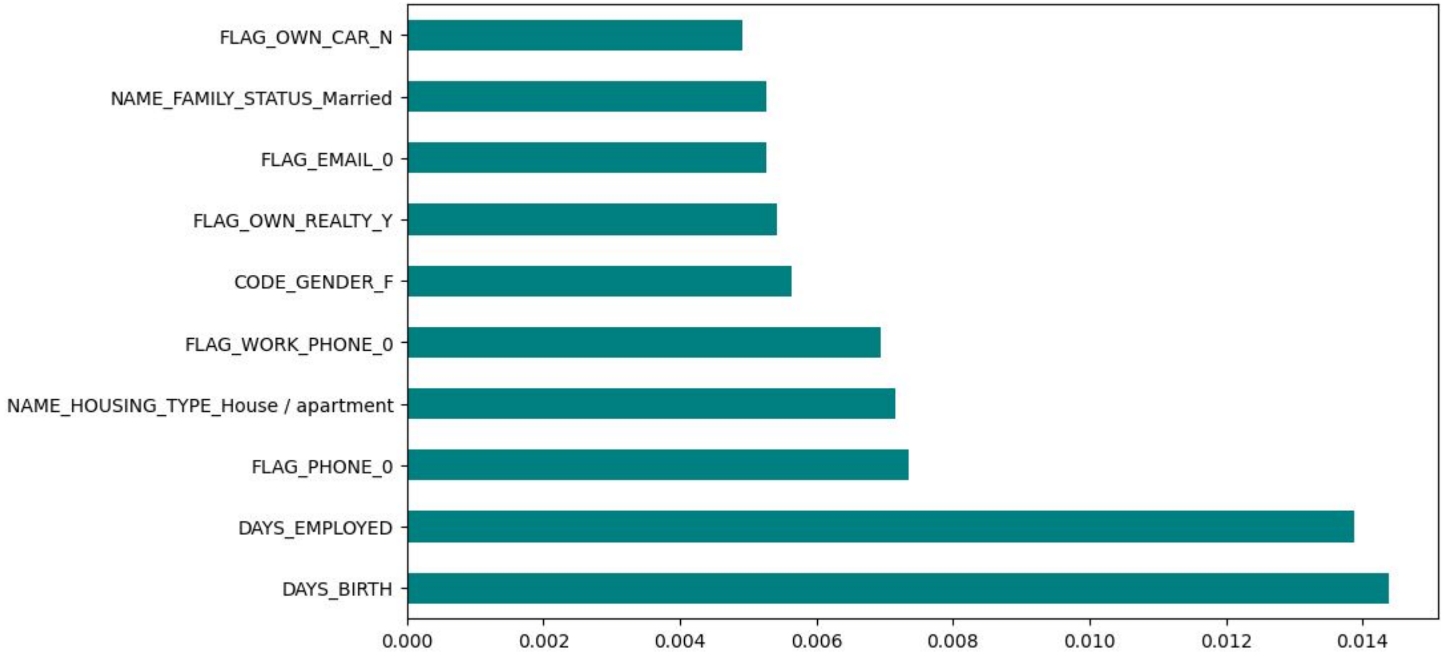
TensorFlow

Keras



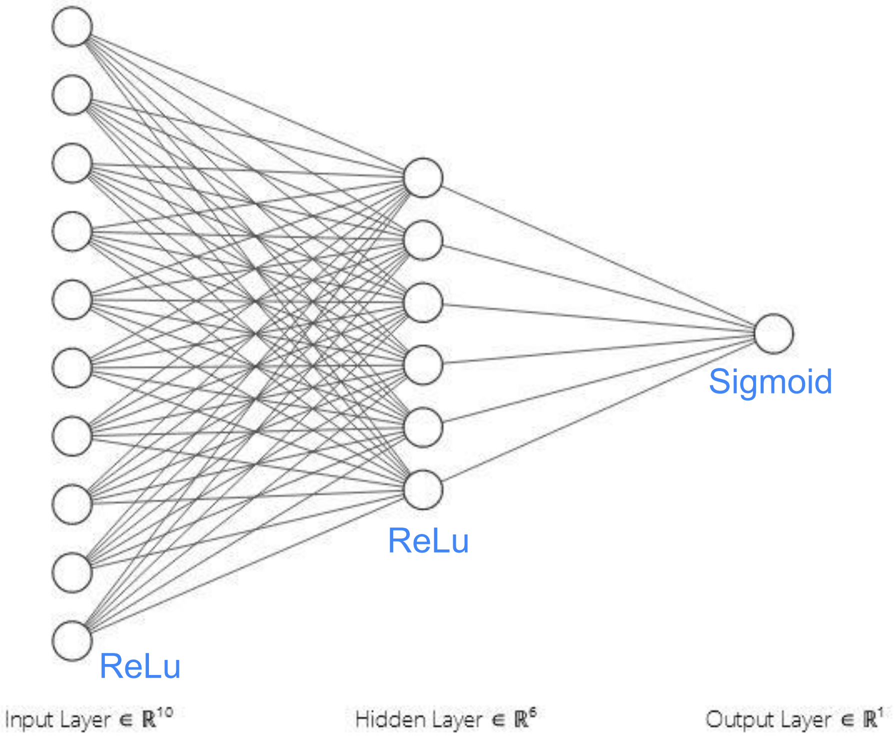
Using a Neural Network model for predictions

Neural Network - 1. Feature Selection



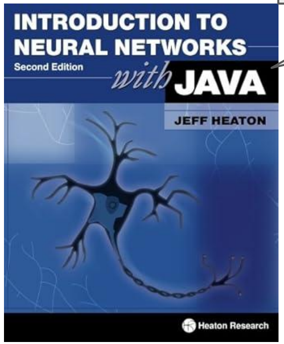
Selecting the 10 most useful features by information gain, and using them to train the neural network model

Neural Network - 2 Training the Model (2-label)



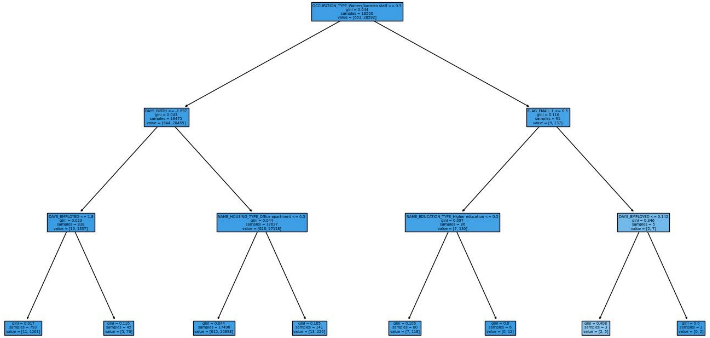
"In fact, for many practical problems, there is no reason to use any more than one hidden layer."

"The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer."



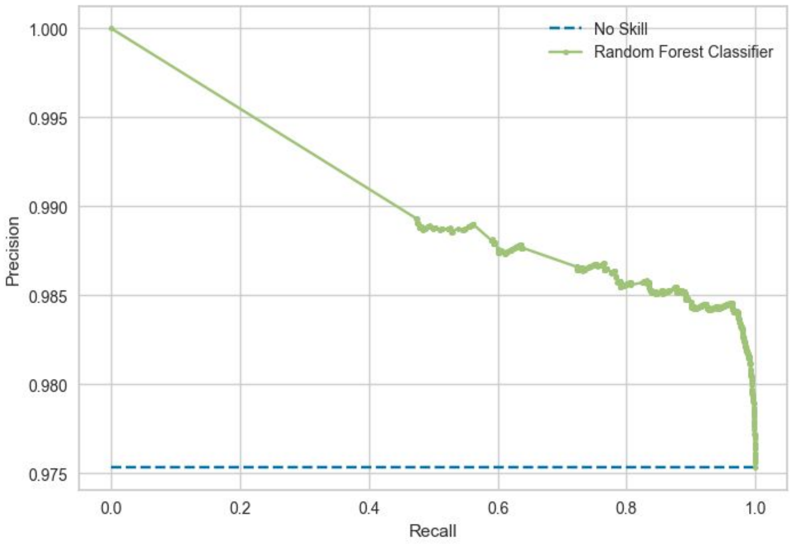
Tuning the Hidden Layer to avoid overfitting and long training time

Random Forest


OCCUPATION_TYPE_Waiters/barmen staff <= 0.5
gini = 0.044
samples = 18566
value = [653, 28592]
DAYS_BIRTH <= -1.697
gini = 0.043
samples = 18475
value = [644, 28455]
FLAG_EMAIL_1 <= 0.5
gini = 0.116
samples = 91
value = [9, 137]
DAYS_EMPLOYED <= 1.8
gini = 0.023
samples = 838
value = [16, 1337]
NAME_HOUSING_TYPE_Office apartment <= 0.5
gini = 0.044
samples = 17637
value = [628, 27118]
NAME_EDUCATION_TYPE_Higher education <= 0.5
gini = 0.097
samples = 86
value = [7, 130]
DAYS_EMPLOYED <= 0.142
gini = 0.346
samples = 5
value = [2, 7]
gini = 0.017
samples = 793
value = [11, 1261]
gini = 0.116
samples = 45
value = [5, 76]
gini = 0.044
samples = 17496
value = [615, 26898]
gini = 0.105
samples = 141
value = [13, 220]
gini = 0.106
samples = 80
value = [7, 118]
gini = 0.0
samples = 6
value = [0, 12]
gini = 0.408
samples = 3
value = [2, 5]
gini = 0.0
samples = 2
value = [0, 2]


LIFE IS JUST A
RANDOM FOREST
memegenerator.net

PR Curve



The precision and recall at any thresholds are higher than if there was no skill involved.

If we take good credit as positive, ideally, we are looking to reduce the False Negative as much as possible.

This lowers the recall but comes at a high precision cost.

Model Evaluation

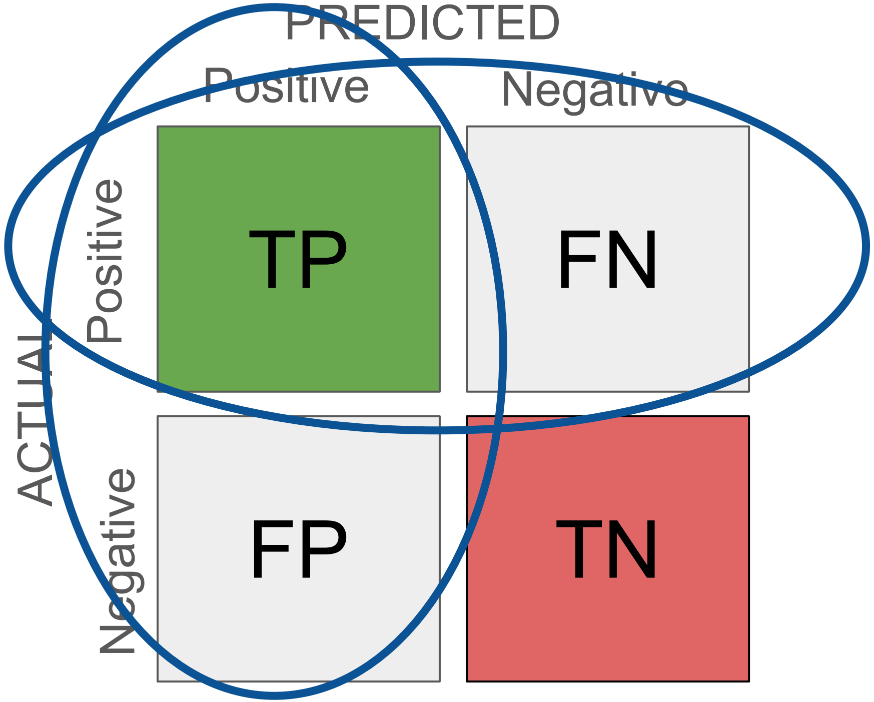
Positive Classes

Customers with **GOOD** credit status
Approval for credit card application.

Negative Classes

Customers with **BAD** credit status
Rejection for credit card application.

Evaluation Metrics



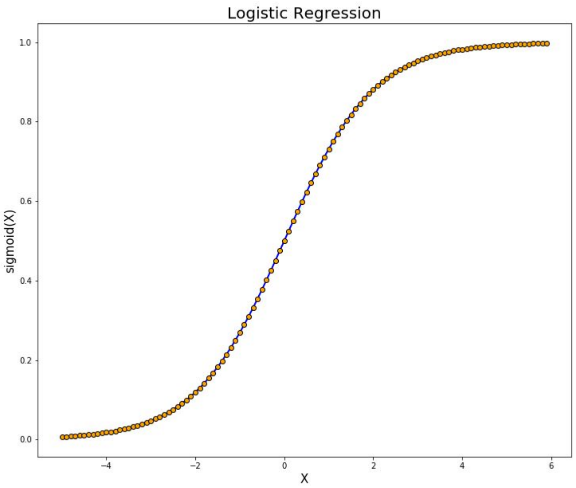
Accuracy

Recall

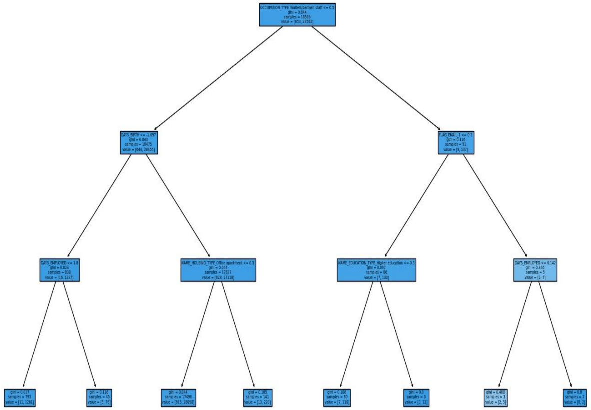
F1-Score

Precision

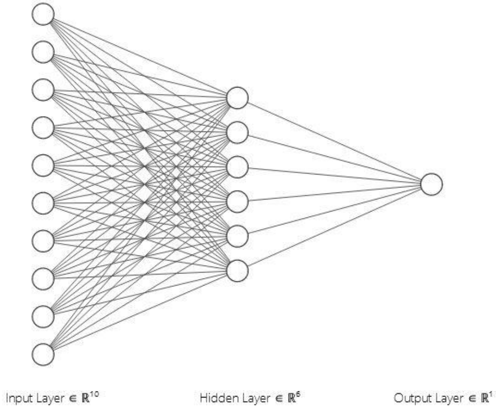
Accuracy



97.5%



97.7%

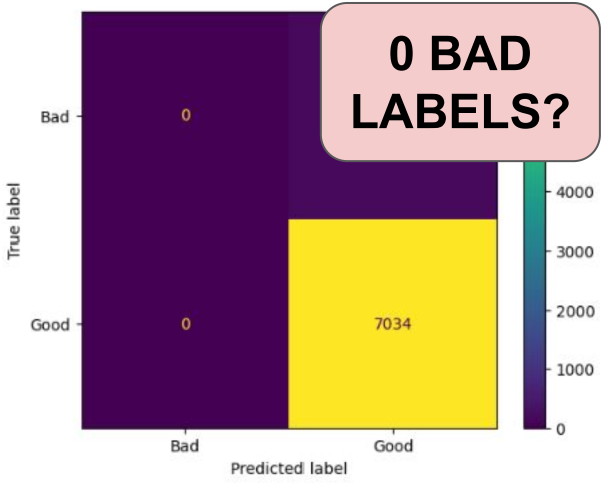


97.5%

Confusion Matrix

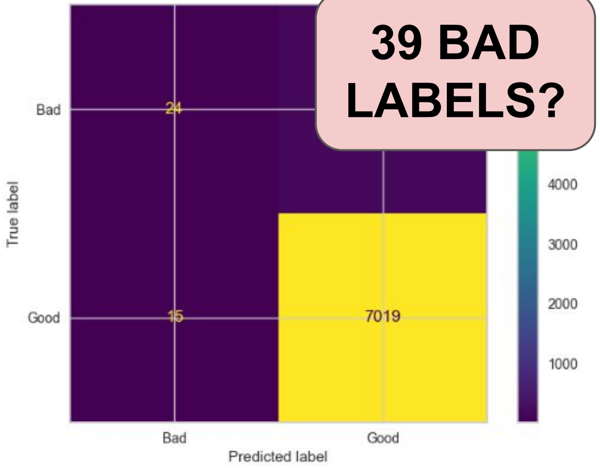
Logistic Regression

0 BAD LABELS?



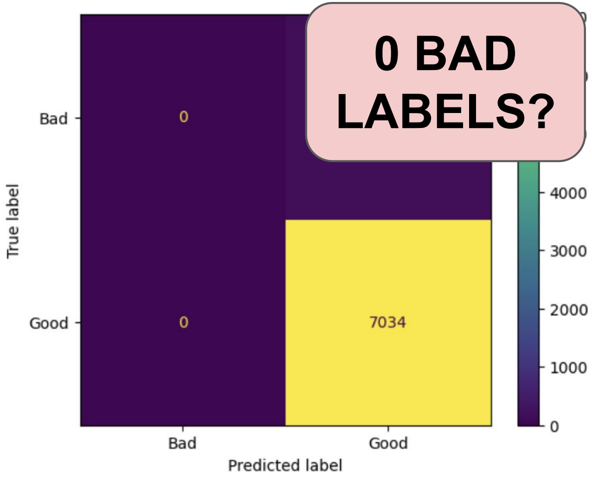
Random Forest

39 BAD LABELS?



Neural Network

0 BAD LABELS?

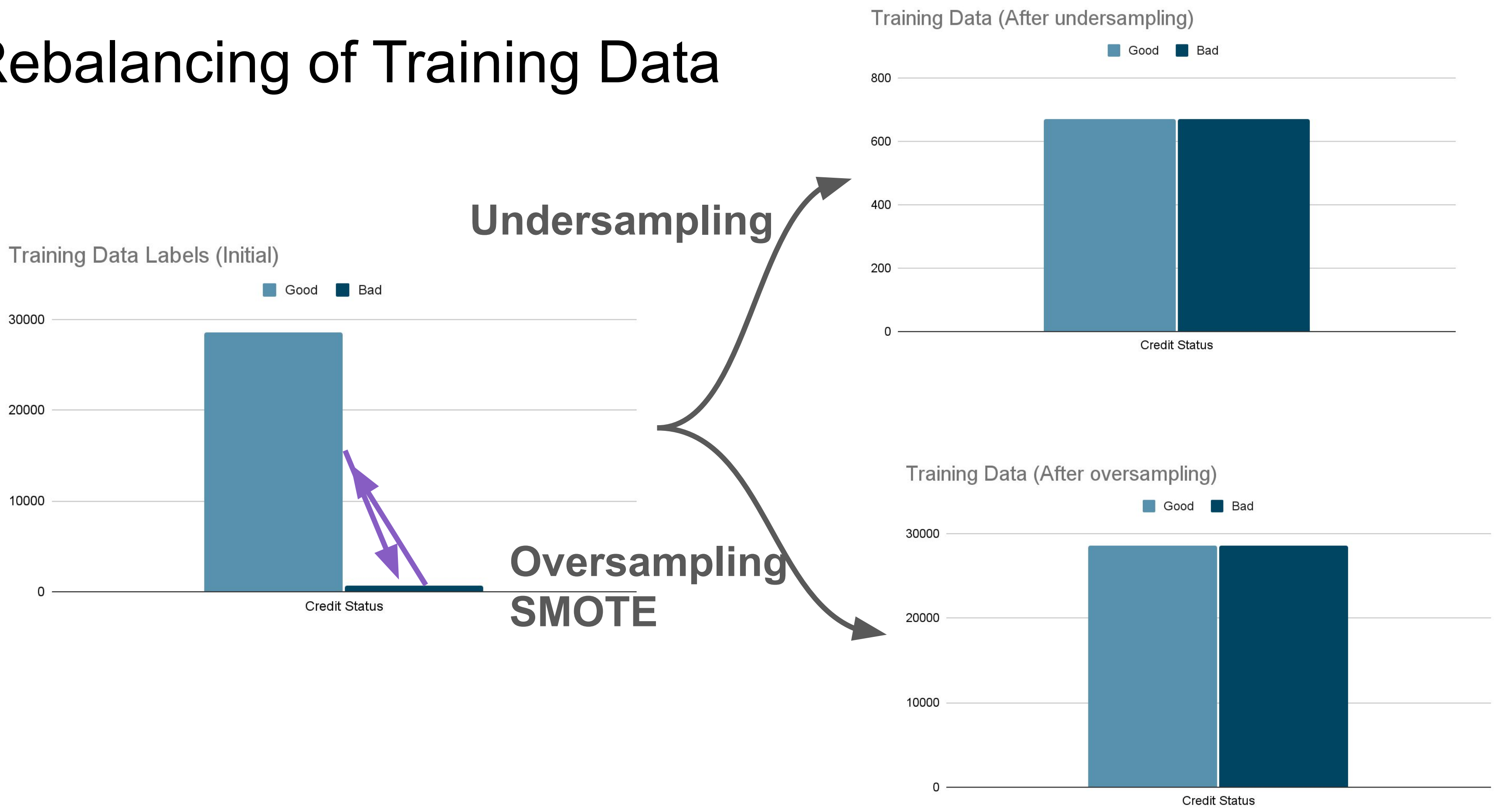


Accuracy Paradox

Precision, Recall and F1-score

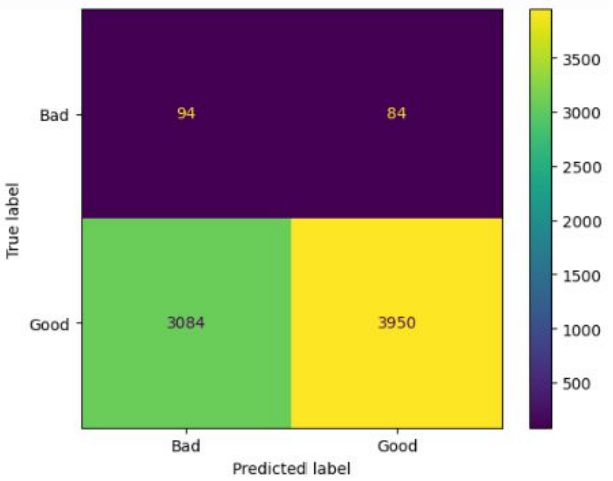
	Logistic Regression	**Random Forest**	**Neural Network**
Precision TP/(TP+FP)	0.98	0.98	0.98
Recall TP/(TP+FN)	1.00	1.00	1.00
F1-score	0.988	0.988	0.988

Rebalancing of Training Data



Logistic Regression

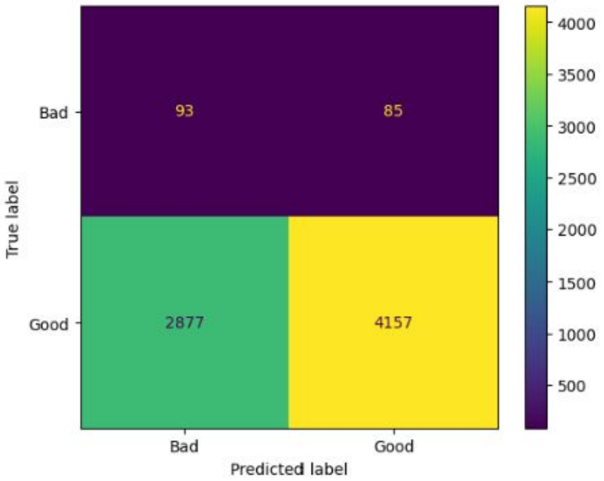
Undersampling



Accuracy: 56.1%

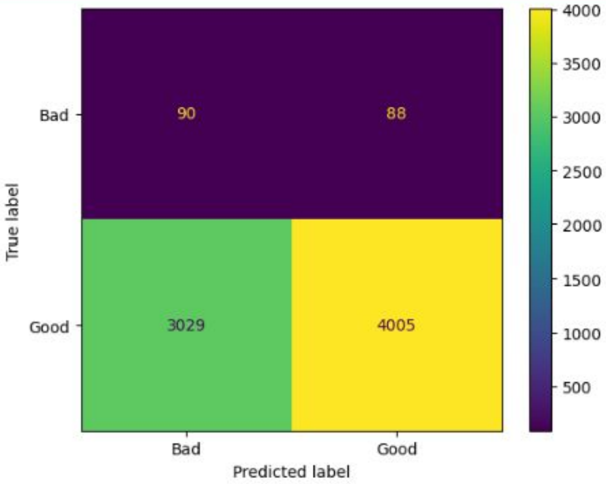
Oversampling

Random Oversampling



Accuracy: 58.9%

SMOTE



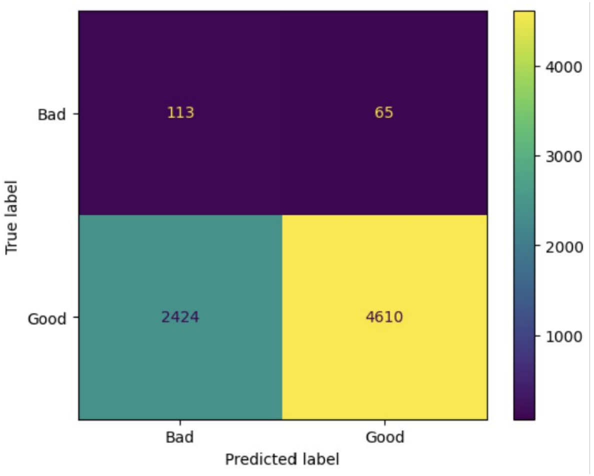
Accuracy: 56.8%

Logistic Regression

	With random undersampling	**With random oversampling**	**With SMOTE**
Precision TP/(TP + FP)	0.98	0.98	0.98
Recall TP/(TP + FN)	0.56	0.59	0.57
F1-score	0.714	0.737	0.720

Random Forest

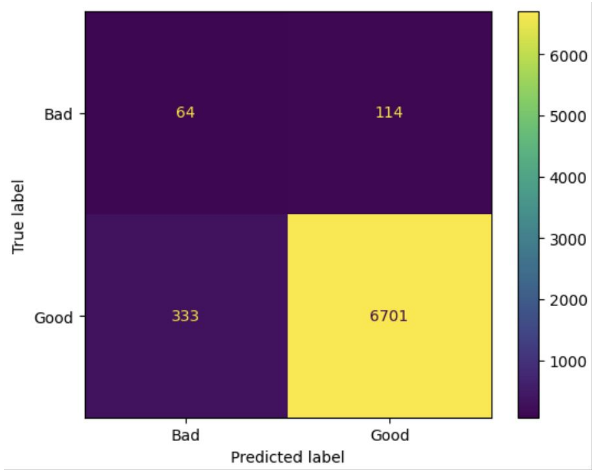
Undersampling



Accuracy: 65%

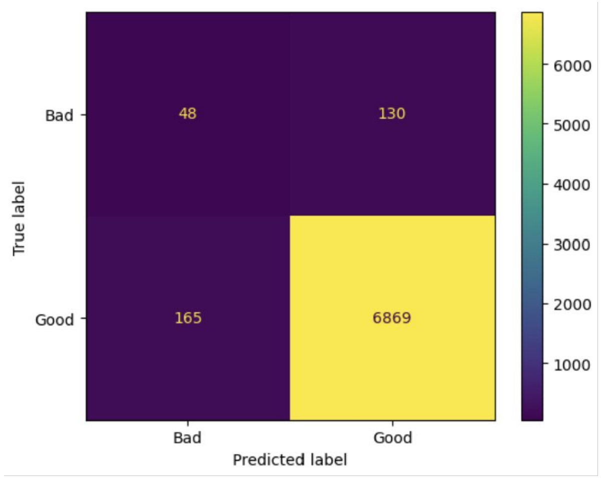
Oversampling

Random Oversampling



Accuracy: 94%

SMOTE

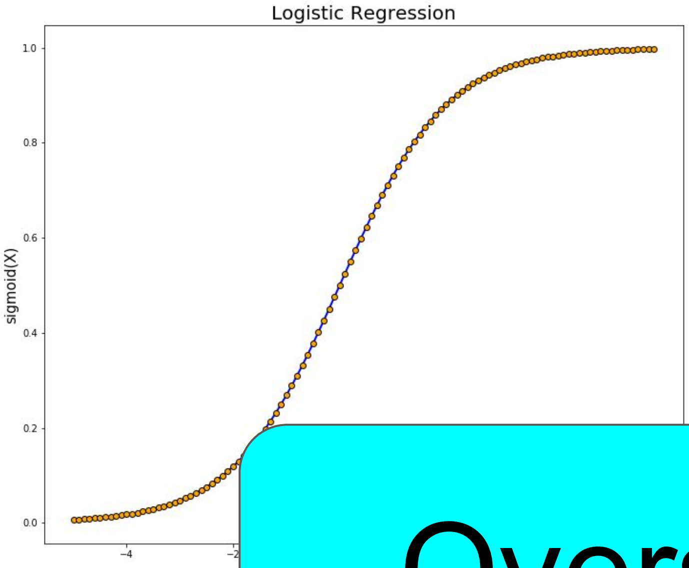


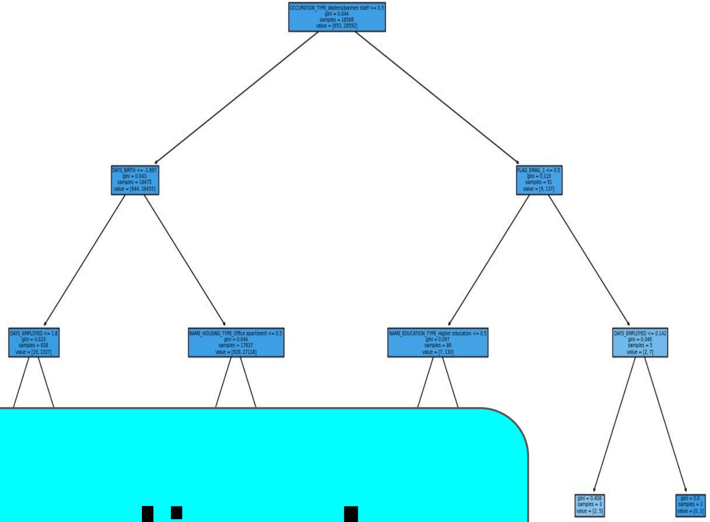
Accuracy: 96%

Random Forest

	With random undersampling	With random oversampling	With SMOTE
Precision TP/(TP + FP)	0.99	0.98	0.98
Recall TP/(TP + FN)	0.66	0.95	0.98
F1-score	0.787	0.968	0.979

Conclusion





Oversampling!

Future Improvements

Training neural network with the rebalanced data

Hyperparameter tuning for the models

Combining of random oversampling and random undersampling

- To prevent loss of information (undersampling) and overfitting (oversampling)

K-fold Cross Validation

Thank YOU