Loan Default Prediction

Presented by: Zack Chen

Outline

- Introduction
- Objective
- Data
- Modelling
- Challenges
- Conclusion
- Next Steps

Objective

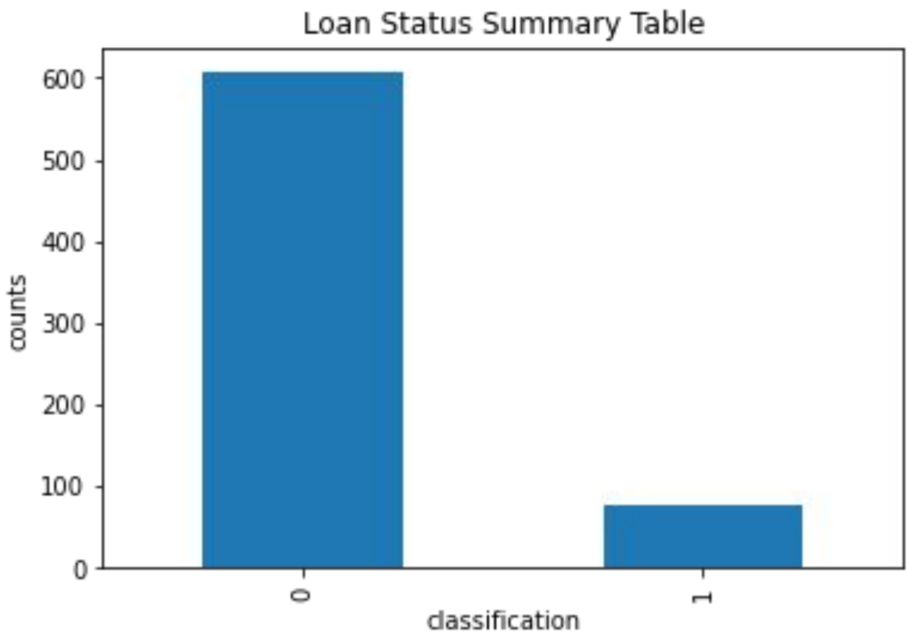
- Predict if clients apply for first loan will default or not
  - Loan Data between 1993 and 1998
    - Transactions before Loan Granted Date
- Label: Loan Status - Classification
  - **0**: A & C not likely to default on loan
  - **1**: B & D likely to default on loan
- Metrics:
  - Maximizing Recall Score for Class 1
  - Balance with f1 Score

Data

- Data Source: 8 Dataset from a bank
  - Load to python from SQL database
- Data Cleaning
  - Delete Columns with irrelevant values (e.g. ‘bank_to’, ‘account_to’)
  - Delete Columns with a Single value (e.g. disposition ‘type’)
  - Data type conversion (e.g. datetime)
  - Consider missing values (numerical and categorical)
    - Imputation strategies

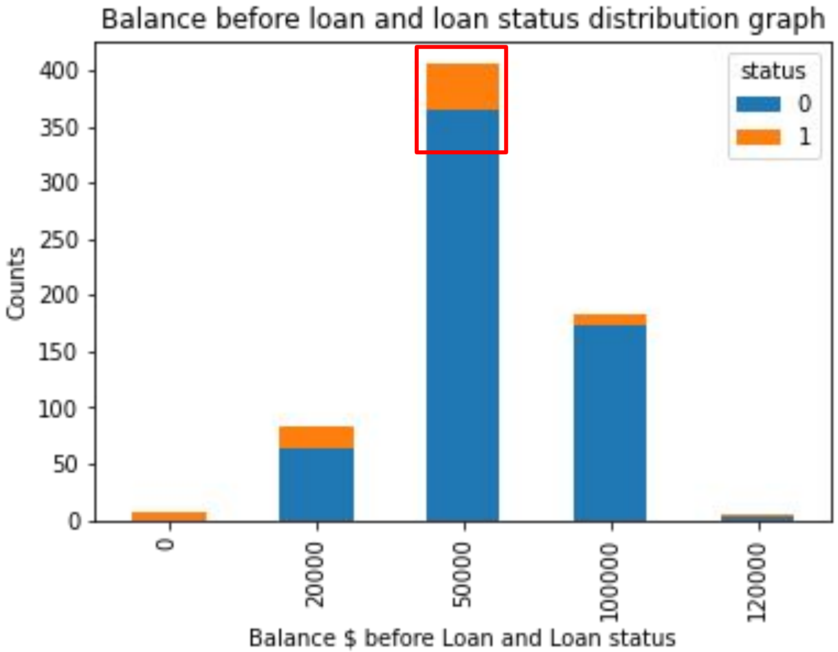
Data

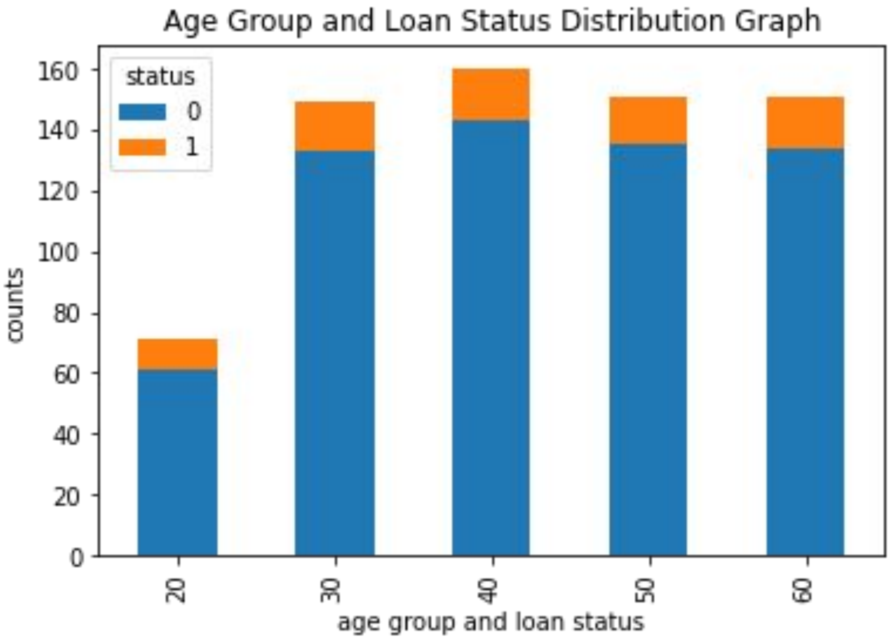
- Data Exploration
    - Imbalanced Data
    - Oversampling



Data

- Data Distribution





Data

- Feature Engineering
  - Age of client loan granted date
  - Balance before loan, min balance before loan
  - Salary
  - Total/mean credit amount
  - Total/mean withdraw amount
  - Transactions in # months before loan
  - Age of account
  - Etc.
- Feature Selection
  - Correlation Matrix with Heatmap
  - Feature Importances

Check for Model Stability

Splitted data into ‘Train_full’ and ‘test’ sets (with stratify = y)

Ran K-fold Cross Validation on the ‘train_full’ data sets.

Then compared with y_test using the trained model.

```
CV scores: [0.94610778 0.93491124 0.94117647 0.95294118 0.94117647]
Accuracy score: 0.911504424778761
```

Random Forest

- We have decided to use the ‘Random Forest’ model to train and predict our data as it gave us the highest overall scores.
  - Higher scores
  - Less influence of outliers
  - Feature selection
- We will use ‘GridSearchCV’ to help us finding the best hyperparameters for the model.

Pipeline, Feature Scaling and Imputation

SimpleImputer: replaced the missing values with the mean

StandardScaler: performed feature scaling

Bundled all preprocessing steps in a pipeline to avoid data leakage.

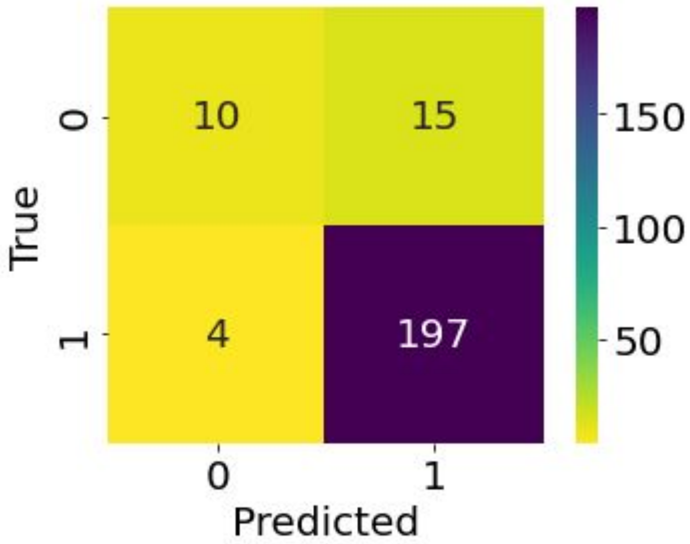
```
pipe_rf = Pipeline([('imputer',SimpleImputer(strategy='mean')),
                    ('scaling', StandardScaler()),
                    ('rf', RandomForestClassifier(random_state=41))])
```

GridSearchCV: Hyperparameter tuning

Random Forest Result

Now, what can we do to improve our model?

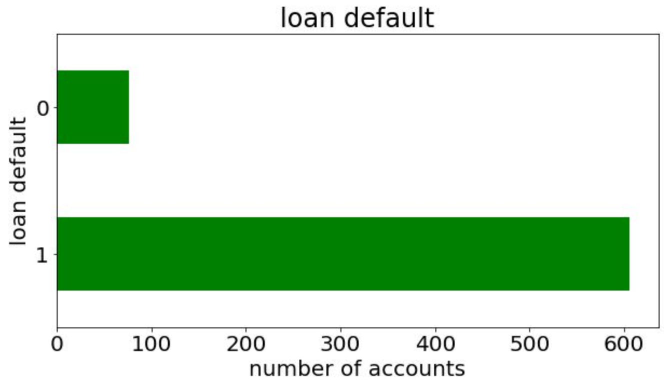
	precision	recall	f1-score	support
0	0.71	0.40	0.51	25
1	0.93	0.98	0.95	201
accuracy			0.92	226
macro avg	0.82	0.69	0.73	226
weighted avg	0.91	0.92	0.91	226



Handling imbalanced data

606 vs. 76

We will use SMOTE to over sample the minorities.

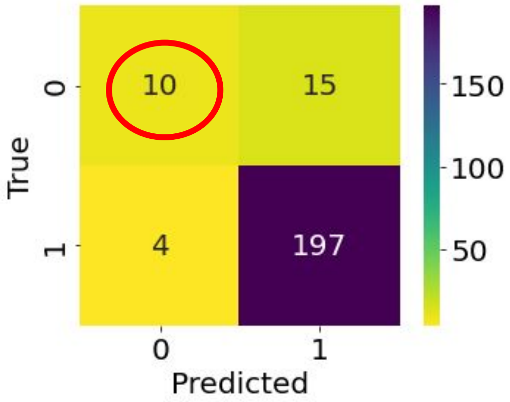



```
from imblearn.over_sampling import SMOTE
```

```
imba_pipeline = imblearn.pipeline.Pipeline([('imputer',SimpleImputer(strategy='mean')),
                                             ('scaling', StandardScaler()),
                                             ('smt',smt),
                                             ('rf', RandomForestClassifier(random_state=41))])
```

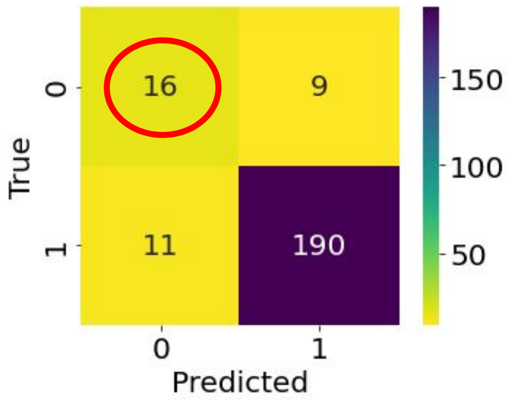
Before oversampling

	precision	recall	f1-score	support
0	0.71	0.40	0.51	25
1	0.93	0.98	0.95	201
accuracy			0.92	226
macro avg	0.82	0.69	0.73	226
weighted avg	0.91	0.92	0.91	226



After oversampling

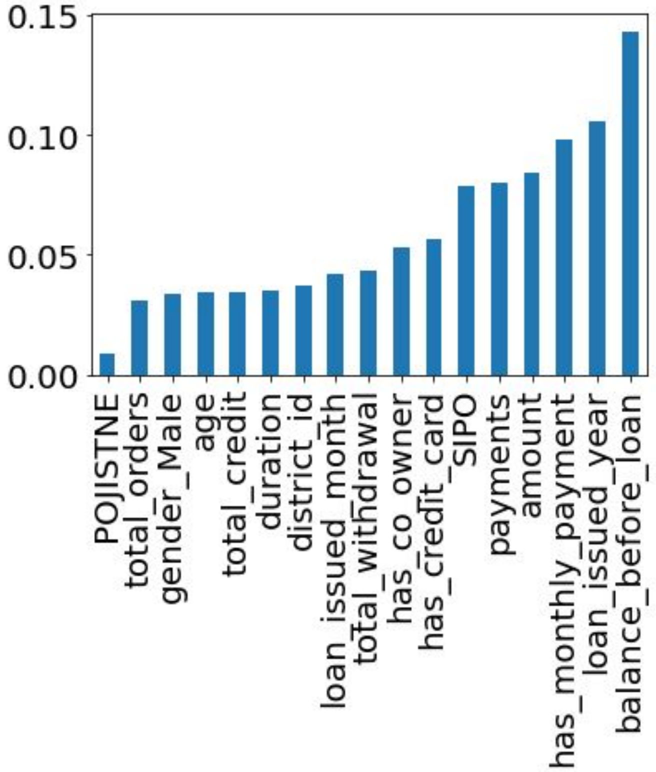
	precision	recall	f1-score	support
0	0.59	0.64	0.62	25
1	0.95	0.95	0.95	201
accuracy			0.91	226
macro avg	0.77	0.79	0.78	226
weighted avg	0.91	0.91	0.91	226



Feature Importance Chart

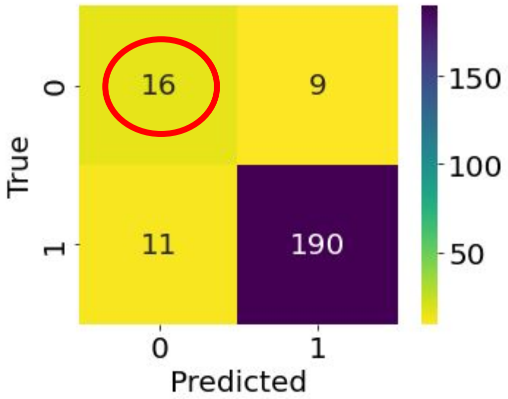
What else can we do to improve our model?

Can we get rid of some features with low importance?

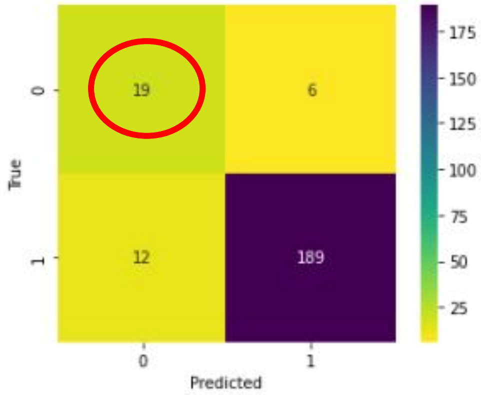


Final Model

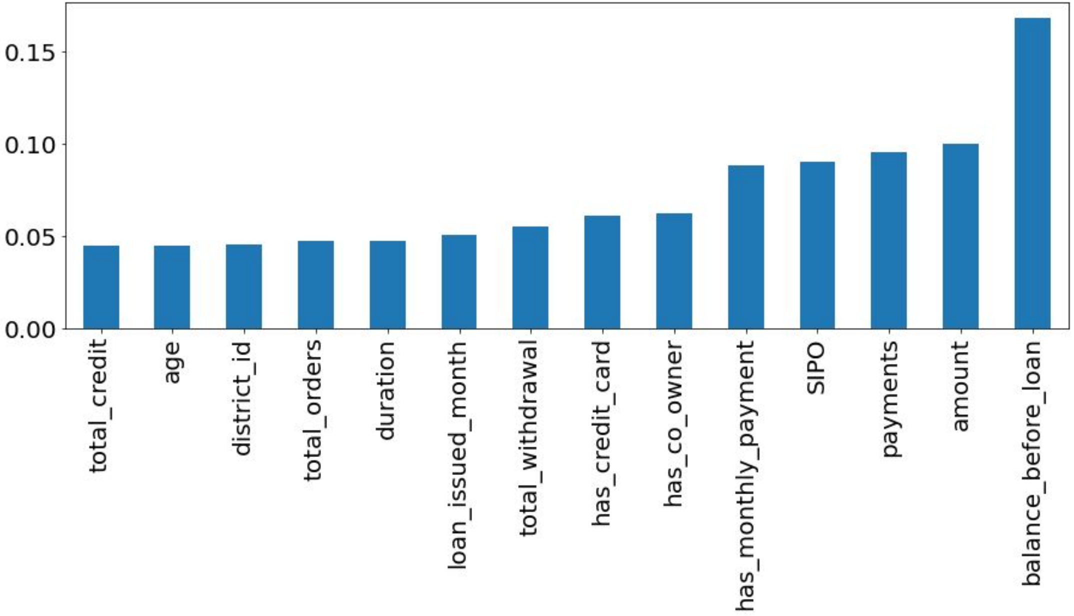
	precision	recall	f1-score	support
0	0.59	0.64	0.62	25
1	0.95	0.95	0.95	201
accuracy			0.91	226
macro avg	0.77	0.79	0.78	226
weighted avg	0.91	0.91	0.91	226



0	0.61	0.76	0.68	25
1	0.97	0.94	0.95	201
accuracy			0.92	226
macro avg	0.79	0.85	0.82	226
weighted avg	0.93	0.92	0.92	226



Final Model


0.15
0.10
0.05
0.00
total_credit
age
district_id
total_orders
duration
loan_issued_month
total_withdrawal
has_credit_card
has_co_owner
has_monthly_payment
SIPO
payments
amount
balance_before_loan

Challenges

- Feature engineering
- Feature selection
- Hyperparameter tuning
- Domain challenges

Conclusion

1. Random Forest performed better than Logistic Regression and KNN

2. Good loan scores > Bad loan scores

   This is due to having imbalanced data

   Oversampling improves accuracy for bad loans

3. Features with strong importance:
   - Min balance before loan
   - Amount borrowed
   - Payments
   - Year account was created (account age)

Next Steps

- Continue to feature engineer
- More domain knowledge search
- Try “threshold moving” by hyperparameter tuning the threshold point (default is 0.5)
- Plot ROC curve & Precision-Recall curve
- Continue to tune hyperparameters
- Experiment with more advanced models and NN

Thank you.