Machine Learning & Predictive Analytics

Duo Zhou

Agenda

1. Problem Statement
2. Exploratory Data Analysis & Feature Engineering
3. Analytic Approach
4. Model Building & Evaluation
5. Model Comparison
6. Conclusion

Problem Statement

- Many people are struggling to get loans due to insufficient credit histories
- Home Credit’s role is to ensure clients who is capable of repayment are not rejected from getting loans
- Make default prediction given certain characteristics of a credit applicant can help load companies to minimize risk.
- Home Credit provides positive and safe loan experience to clients
- Home Credit is exploring avenues to unlock the complete potential of the data and thereby increasing the correct prediction of their clients' repayment abilities

EDA & Feature Engineering

Data Overview

Application Dataset

Data Shape

- 121 columns, 307511 rows
- Each observation is an application

Data Type

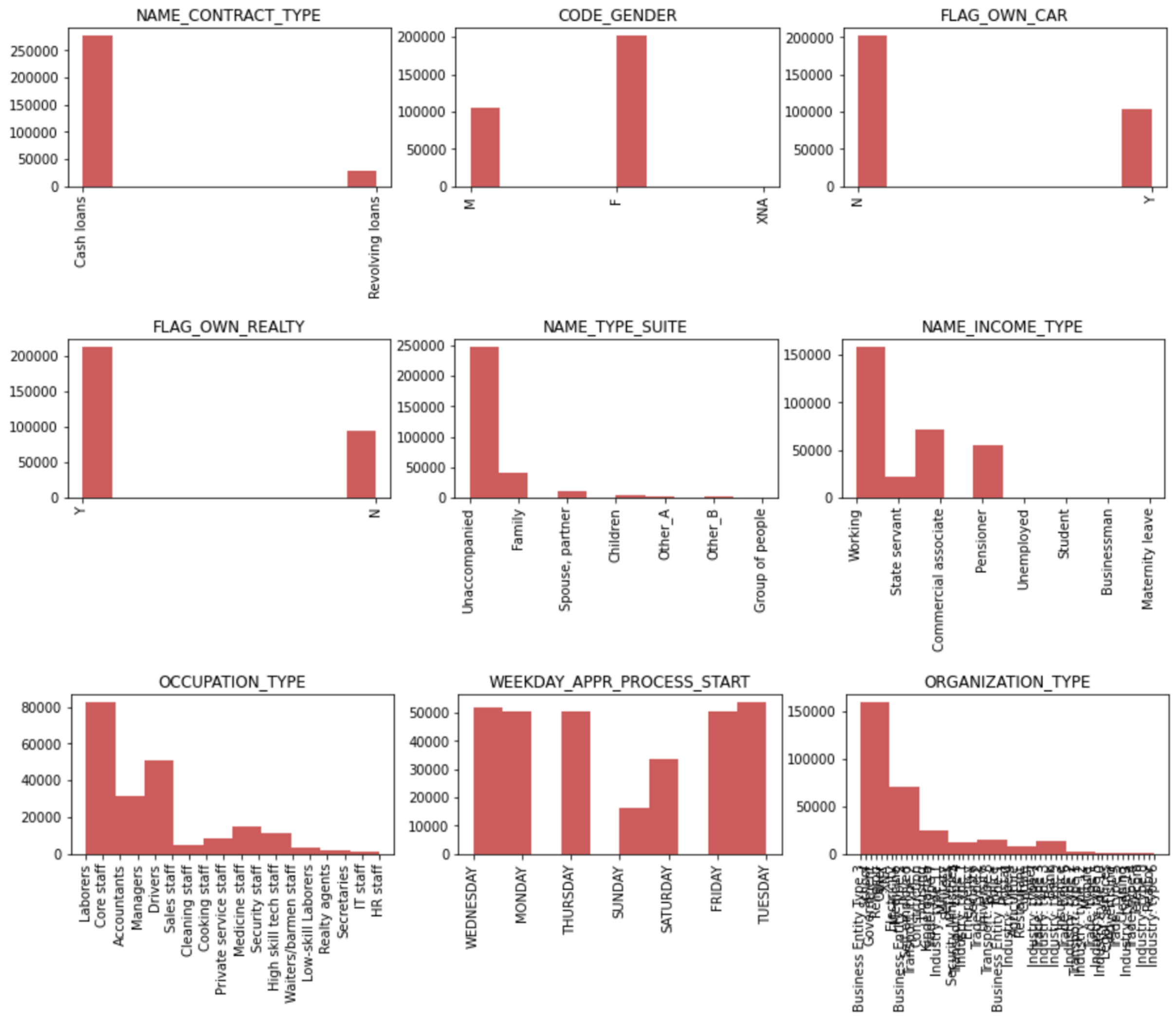
- 65 numerical variables
- 56 categorical variables

Target Variable

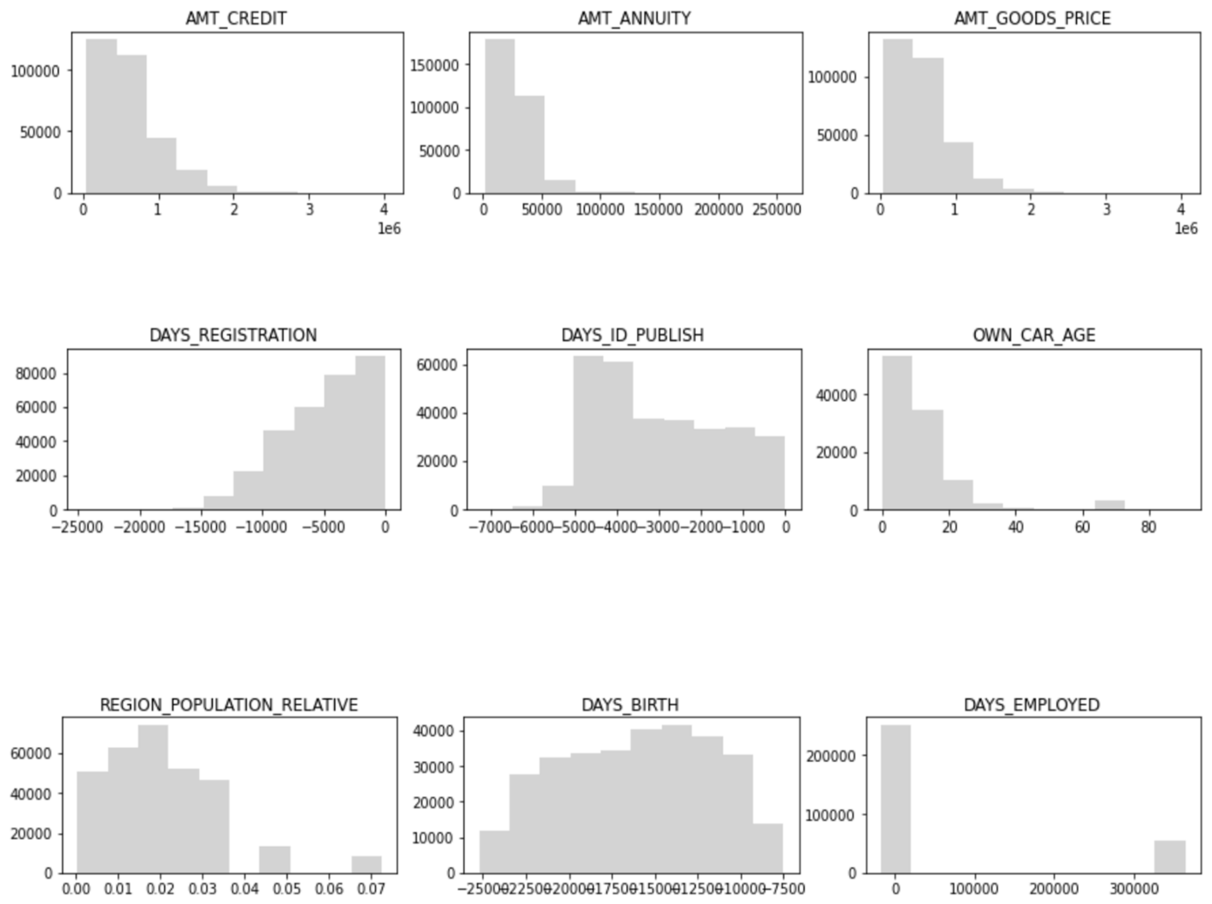
- 1 - client with payment difficulties (minor class)
- 0 - all other cases (major class)
- minor class : major class = 8% : 92%

Univariate Analysis

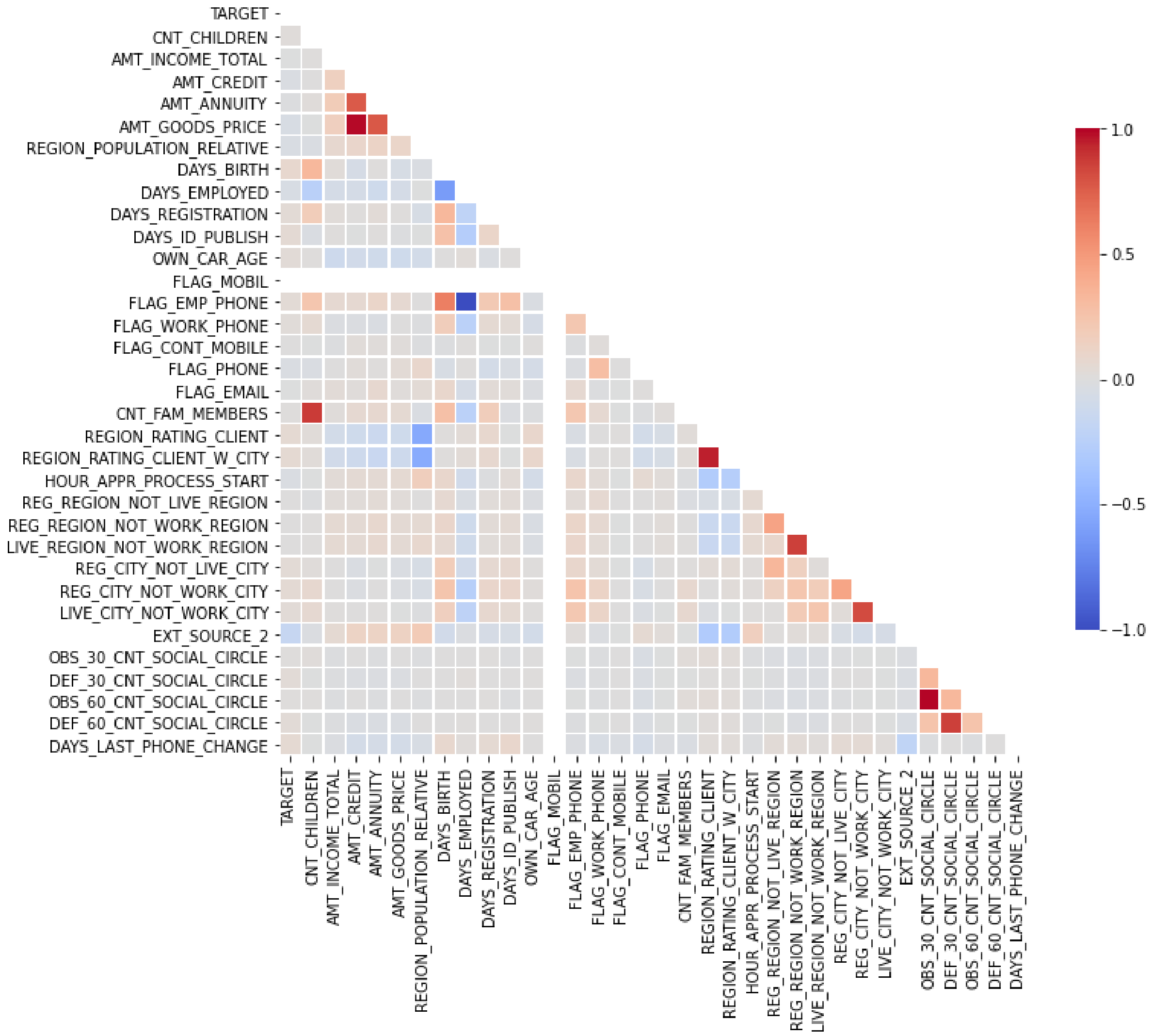
Categorical Variables



Numerical Variables



Bivariate Analysis



- **AMT_CREDIT → AMT_GOODS_PRICE** **corr: 0.98**
  Credit amount of the loan is highly correlated with the price of the goods for which the loan is given
- **REGION_RATING_CLIENT → REGION_RATING_CLIENT_W_CITY** **corr: 0.95**
  Rating of the region where client lives is highly correlated with rating of the region where client lives with taking city into account
- **CNT_FAM_MEMBERS → CNT_CHILDREN** **corr: 0.88**
  Number of family members does client have is highly correlated with number of children the client has
- **LIVE_REGION_NOT WORK_REGION → REG_REGION_NOT_WORK_REGION** **corr: 0.86**
  Client's contact address does not match work is highly correlated with client's permanent address does not match work address
- **LIVE_CITY_NOT_WORK_CITY → REG_CITY_NOT_WOTK_CITY** **corr: 0.82**
  Client's contact address does not match work address is highly correlated with / client's permanent address does not match work address
- **AMT_GOODS_PRICE → AMT_ANNUITY** **corr: 0.77**
  The price of the goods for which the loan is given is highly correlated with the loan annuity

Missing Value

	missing value rate
COMMONAREA_AVG	0.698723
COMMONAREA_MODE	0.698723
COMMONAREA_MEDI	0.698723
NONLIVINGAPARTMENTS_AVG	0.694330
NONLIVINGAPARTMENTS_MODE	0.694330
NONLIVINGAPARTMENTS_MEDI	0.694330
FONDKAPREMONT_MODE	0.683862
LIVINGAPARTMENTS_MEDI	0.683550
LIVINGAPARTMENTS_AVG	0.683550
LIVINGAPARTMENTS_MODE	0.683550
FLOORSMIN_AVG	0.678486
FLOORSMIN_MODE	0.678486
FLOORSMIN_MEDI	0.678486
YEARS_BUILD_AVG	0.664978
YEARS_BUILD_MEDI	0.664978
YEARS_BUILD_MODE	0.664978
OWN_CAR_AGE	0.659908
OCCUPATION_TYPE	0.313455

...

- Identify variables with missing value rate above 15%
- Inspect the mechanism of missing values. Determine whether those missing values are MCAR, MAR or MNAR
- Except below two variables, no clear pattern is observed in other missing values. Therefore, we assume they are Missing Completely At Random
- **OWN_CAR_AGE** : Most missing values are because people don't have cars

```
Missing values: 202929
Missing values with N in the column FLAG_OWN_CAR: 202924
```

- **OCCUPATION_TYPE:** Most of the missing values in OCCUPATION_TYPE are people whose INCOME_TYPE is pensioner.

```
Pensioner               55357
Working                 24920
Commercial associate    12297
State servant            3787
Unemployed                 22
Student                     5
Businessman                 2
Maternity leave             1
```

HOME CREDIT

Feature Engineering Process

1 Drop Columns, Rows, Outliers

- Drop columns with missing value rate above 15%
- Drop columns that are highly correlated (corr. > 0.7)
- Drop rows with too many missing values given their target variable is 0
- Drop outliers

2 Train Test Split

- Divide train and test set following same distribution

3 Impute Missing Values

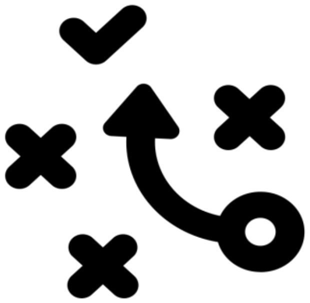
- Impute columns with structural deficiency
- Categorical variable:
  Most common value
- Numerical variable:
  Median for skewed variable,
  Mean for non-skewed variable

4 Deal with categorical variable

- Reduce levels in some categorical variables
- One-Hot Encoding

5 Normalize the data

Analysis Approaches



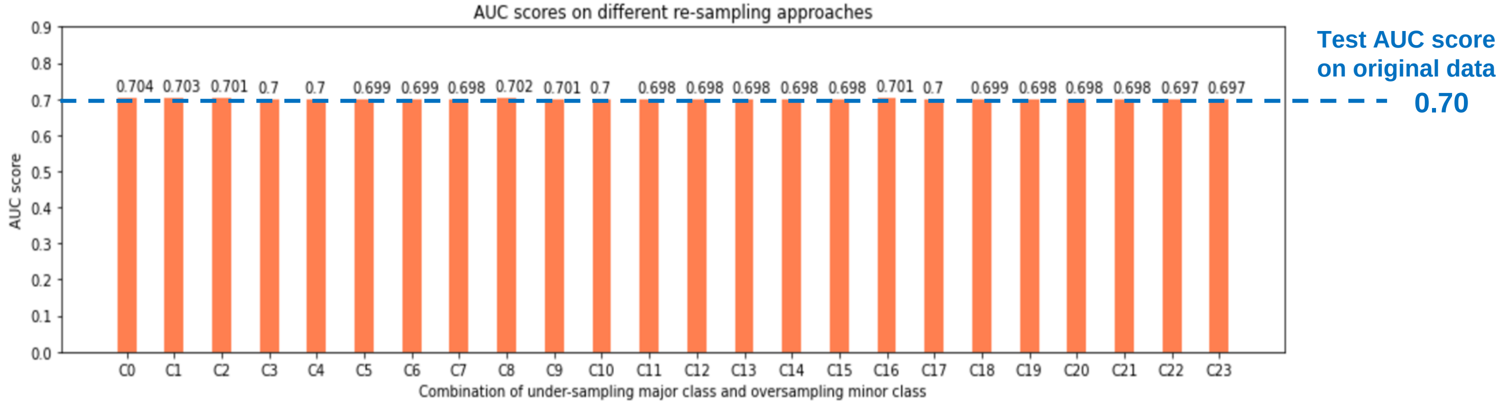
Approaches for Imbalanced Data

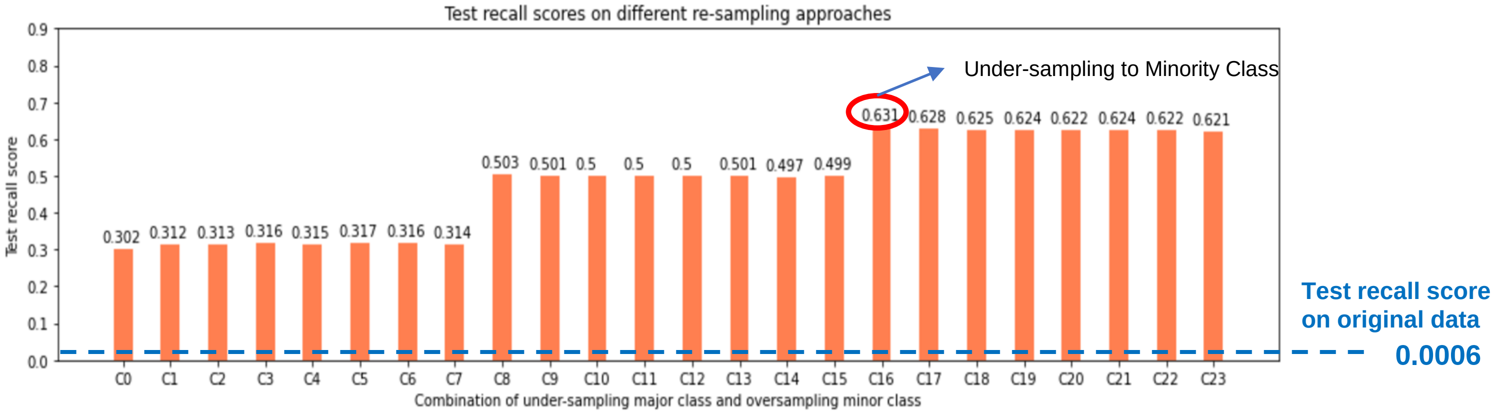
- **Key challenge**
  - The target class is imbalanced
  - This makes the optimization of classification accuracy meaningless
- **Adaptive re-sampling**
  - The data are re-sampled in order to magnify the relative proportion of the minor class
  - Oversample minor class (Synthetic Minority Over-sampling Technique - SMOTE)
  - Under-sample major class (Random Sampling)
  - The classification algorithm is learned on re-sampled data
- **Cost-sensitive learning**
  - Off-the-shelf classification algorithms are used
  - The loss function of the classification algorithm is modified to weight the classification errors differently for major and minor class
  - The misclassification cost is added to the loss function

HOME CREDIT

Adaptive re-sampling – Logistic Regression

Take Logistic Regression as an example:





- Identify re-sampling strategy using Logistic Regression
- Oversampling minor class using SMOTE; under-sampling major class using random sampling
- The results of the two algorithms indicate that under-sampling major class to the sample size of minor class is the best strategy
- Re-sampling doesn't affect the AUC score, but increases the recall drastically
- In our case, since correctly identifying a client is likely to default is more important, we will use recall as the evaluation metric.

Cost-sensitive learning

- Misclassification cost is added to loss function. Take log loss as an example:

$$Loss(\hat{y}_i, y_i) = \sum_{i=1}^{n}(-y_i log(\hat{y}_i) - (1 - y_i)\log(1 - \hat{y}_i))$$

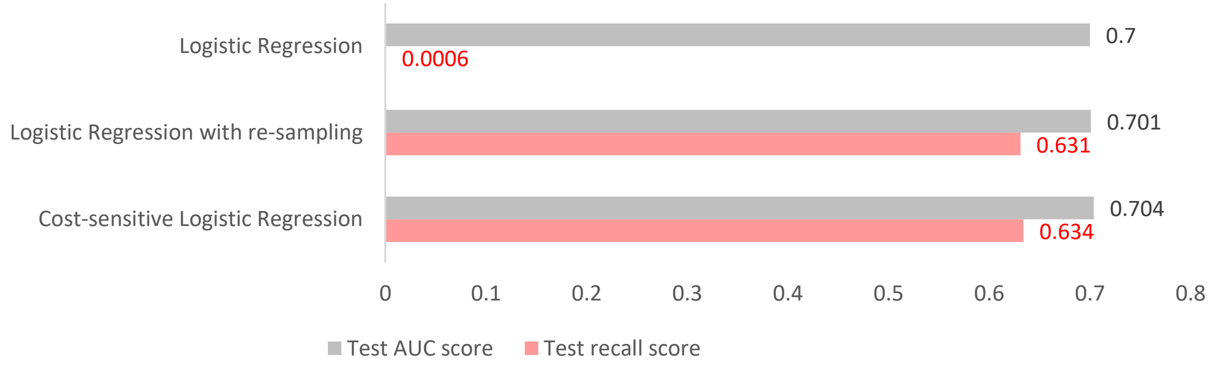
$$Weighted\ Loss(\hat{y}_i, y_i) = \sum_{i=1}^{n}(-w_0 y_i log(\hat{y}_i) - w_1(1 - y_i)\log(1 - \hat{y}_i))$$

- A smaller weight is assigned to majority class while a larger weight is assigned to minority class
- Conduct grid/random search weighs and use following weights:

  Major class: 1

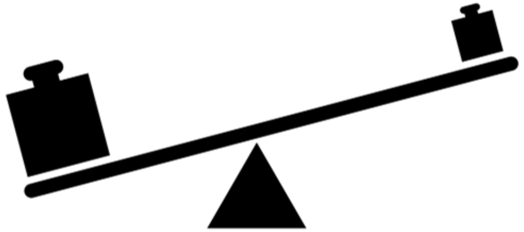
  Minor class: Size of Major Class / Size of Minor Class

- We can see that the **BEST c**ost sensitive learning produces very similar results as the best resampling strategy

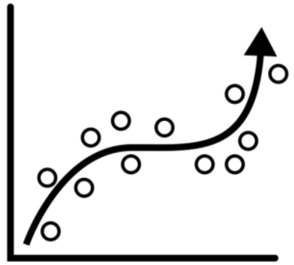


Imbalanced Data Approach Conclusion

- Both under-sampling of majority class and cost-sensitive learning produce similar results with logistic regression
- Three different scenarios will be applied to 4 different ML models(SVM, Random Forest, Boosting, and ANN):
  - Original Data without either re-sampling or cost-sensitive learning
  - Original Data with under sampling of major class to the same size of minor class. No cost-sensitive learning
  - Original Data without re-sampling, but different cost-sensitive learning rates applied for random/grid search



Model Building and Evaluation



Support Vector Machine

Train	Accuracy	Recall	AUC Score
Original Data with Cost Function Weight 1:1	0.918320	0.000655	0.698785
Under-sampling with Cost Function Weight 1:1	0.659225	0.629909	0.699494
Cost-sensitive with Cost Function Weight 1:4	0.910275	0.030916	0.629627

Test	Accuracy	Recall	AUC Score
Original Data with Cost Function Weight 1:1	0.918265	0.000201	0.703843
Under-sampling with Cost Function Weight 1:1	0.663740	0.633837	0.704300
Cost-sensitive with Cost Function Weight 1:4	0.910044	0.027190	0.635207

- Several kernel options are used, and the linear kernel performs better
- There is no sign of overfitting since the fitting scores for train and test are very close
- The recall for original data is very close to zero
- The result of cost-sensitive learning has good accuracy, but the recall is low
- The best model is using under-sampling method. It has the best recall and auc score

Random Forest

- Based on predetermined sampling techniques we've implemented Random Forest to test for both Recall and AUC metrics

- Under-sampling and Weighted Learning methods perform significantly better compared to the other sampling strategies

- Although, there is a significant increase in recall value, in scenarios like fraud detection it's below industry standards

Train	Accuracy	Recall	AUC Score
Original Data with Cost Function Weight 1:1	0.999	0.999	1.00
Under-sampling with Cost Function Weight 1:1	0.768	0.775	0.855
Cost-sensitive with Cost Function Weight 1:12	0.675	0.700	0.756

Test	Accuracy	Recall	AUC Score
Original Data with Cost Function Weight 1:1	0.918	0.001	0.676
Under-sampling with Cost Function Weight 1:1	0.651	0.641	0.702
Cost-sensitive with Cost Function Weight 1:12	0.666	0.622	0.697

XGBoost

Train	Accuracy	Recall	AUC Score
Original Data with Cost Function Weight 1:1	0.9186	0.0050	0.7122
Under-sampling Data with Cost Function Weight 1:1	0.6556	0.6489	0.7127
Original Data with Cost Function Weight 1:12	0.6730	0.6595	0.7271

Test	Accuracy	Recall	AUC Score
Original Data with Cost Function Weight 1:1	0.9184	0.0028	0.7104
Under-sampling Data with Cost Function Weight 1:1	0.6559	0.6483	0.7080
Original Data with Cost Function Weight 1:12	0.6727	0.6415	0.7146

- Different weights have been searched for cost-sensitive learning. The weight of 1:12 generates a higher recall than other weights
- After applying under-sampling or cost-sensitive learning:
  - A significant drop in accuracy
  - A great boost in recall
  - AUC scores remain similar
- Under-sampling approach produces the best recall result for test data
- From the metrics of training data and test data, we can see there is no sign of overfitting

Artificial Neural Network

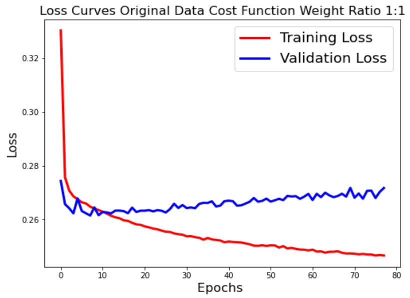
- An ANN model was built with dropouts and batch normalization.

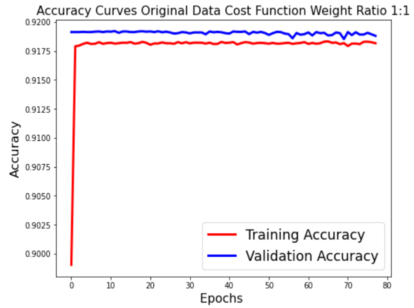
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                5184
_________________________________________________________________
batch_normalization (BatchNo (None, 64)                256
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               8320
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               33024
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               65792
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 514
=================================================================
Total params: 136,322
Trainable params: 134,914
Non-trainable params: 1,408
_________________________________________________________________
```

- Both under sampling and original data showed clear sign of overfitting.
- Cost-Sensitive Learning with the performed significantly better compared to the other scenarios without overfitting.
- Test accuracy and test AUC scores are very similar. Cost-Sensitive Learning with Weight ratio of 1:12 gives the best recall scores of 0.622

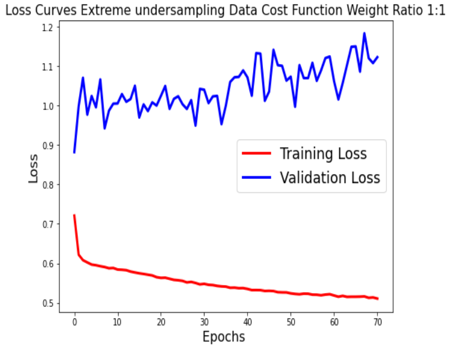
	Model	Test_Accuracy	Test_Precision	Test_Recall	Test_AUC
0	Original Data Cost Function Weight 1:1	0.918479	0.421053	0.00161128	0.680043
1	Undersampling Data Cost Function Weight Ratio 1:1	0.753713	0.152889	0.445519	0.66806
2	Original Data Cost Function Weight 1:8	0.238706	0.0584298	0.552064	0.669933
3	Original Data Cost Function Weight 1:10	0.713641	0.148519	0.531319	0.676267
4	Original Data Cost Function Weight 1:12	0.638995	0.133104	0.622356	0.679911

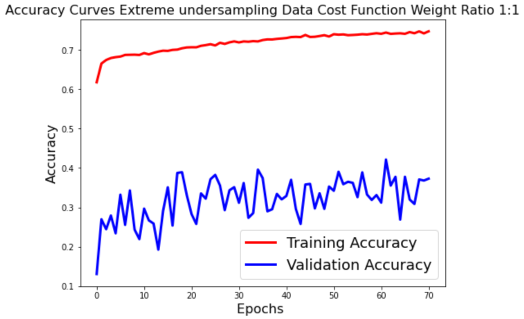
No resampling & No Cost Sensitive Learning



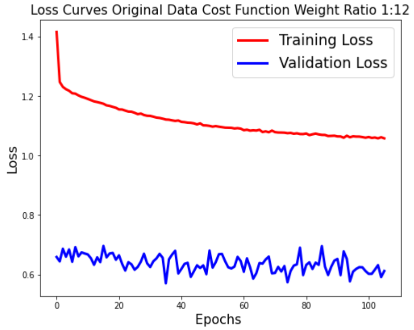


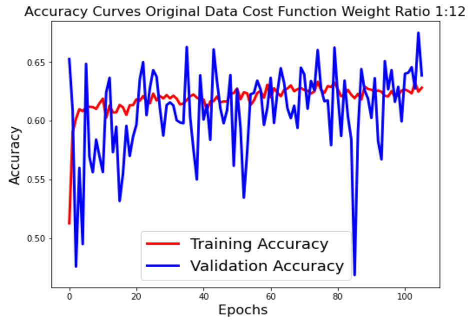
Under-sampling & No Cost Sensitive Learning





No resampling & Cost Sensitive Learning weight ratio 1:12





Model Comparison

- Based on the Test Recall and AUC values XGBoost has performed the best

Model	AUC	Recall
Random Forest (Under-sampling Data with Cost Function Weight 1:1)	0.7019	0.6418
ANN (Original Data with Cost Function Weight 1:12)	0.6799	0.6222
SVM (Under-sampling Data with Cost Function Weight 1:1)	0.7043	0.6338
XGBoost **(Under-sampling Data with Cost Function Weight 1:1)**	0.7080	0.6483

Conclusion & Future Work

Conclusion

- For this project, applying different resampling strategy or cost sensitive learning weights on different models do not significantly change test AUC scores
- However, Recall increases significantly by applying those techniques to offset the imbalanced data effect
- The model that gives the best test recall result is XGBoost with majority class under-sampled to the size of minority class and no cost sensitive learning

Future Work

- Inclusion of additional features may increase the recall percentage; additional feature engineering analysis is recommended
- Instead of simple mean and median imputation, other interpolative or regressive methods can be used for imputation
- Other selection criteria can be used to split the trees in Random Forest, i.e., binary cross-entropy
- Auto encoder can be employed to conduct outlier detection
- GAN can be used to generate additional minority class observations
- Different optimizers and learning rates can employed for ANN given sufficient computational resources and time

Thank You

Q & A