
Home Credit Default Risk

Group 03

Zhengyuan Ding , Chenqin Yang

Jiayao Liu, Ziyu Lei, Zian Chen



Agenda

- Background
- Problem Statement
- Dataset
- Preprocessing
- Methodology
- Evaluation
- Optimization
- Future Work

Background

Home Credit tries to provide loans to underprivileged people who do not have sufficient credit histories. They used a variety of variables including telco and transactional information to predict the repayment ability of their clients.

The logo for Home Credit, featuring the words "HOME" and "CREDIT" in a bold, red, sans-serif font, stacked vertically. The letter "O" in "HOME" is stylized with a white circle inside it.

HOME
CREDIT

This project will use the statistical and machine learning methods to help Home Credit fully unlock the potential of the data it collects from the clients.

Problem Statement:



Can you predict how capable each applicant is of repaying a loan?

Dataset

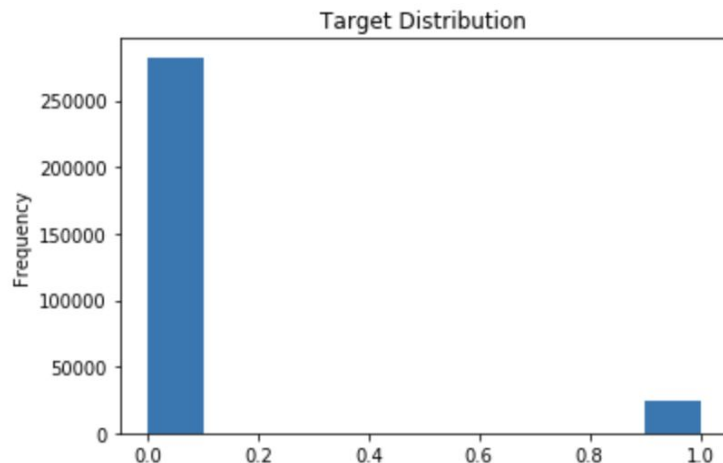
- Application.csv
 - Main table
 - One row represents one loan in our data sample.
- Bureau.csv
 - All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample)
- Bureau_balance.csv
 - Monthly balances of previous credits in Credit Bureau

Dataset (Cont'd)

- Credit_card_balance.csv
 - Monthly balance snapshots of previous credit cards that the applicant has with Home Credit
- Previous_application.csv
 - All previous applications for Home Credit loans of clients who have loans in our sample
- Installments_payments.csv
 - Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample

Target Distribution

- 0: Not Default vs. 1: Default
- Roughly 11:1
- Imbalanced Dataset
 - Choose different evaluation metrics
 - Cannot use Accuracy; Instead using AUC, Recall
 - Resampling
 - Smote

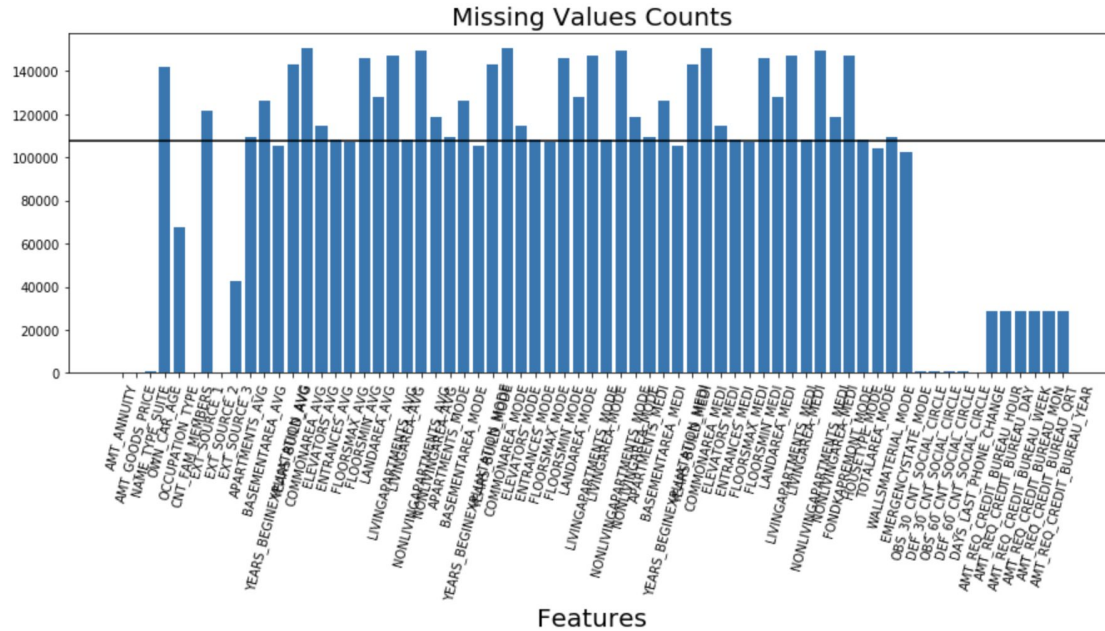


Preprocessing Stage

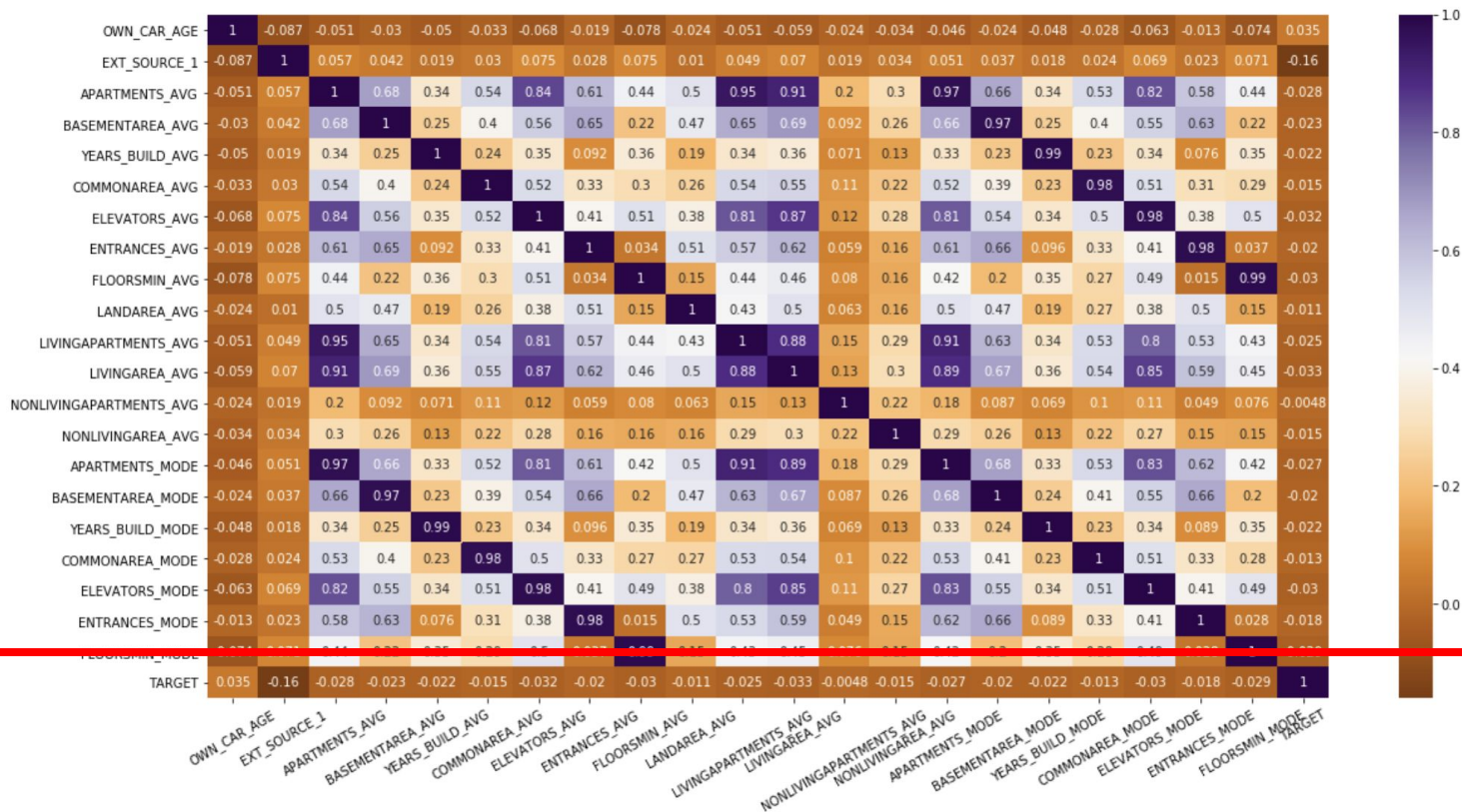
- **Deal with missing values**
- Create dummy variables for categorical features
- Anomaly Detection
- Domain Knowledge Feature Extractions
 - Credit Income Percentage, Previous Late_Payment Counts, Credit Utilization ...
 - Interaction Terms: External Source 1, External Source 2, Data_Birth...
 - Numerical variables: calculate aggregation statistics (min, max, mean)
 - Categorical variables: counts, normed counts
- Feature Normalization

Missing Values

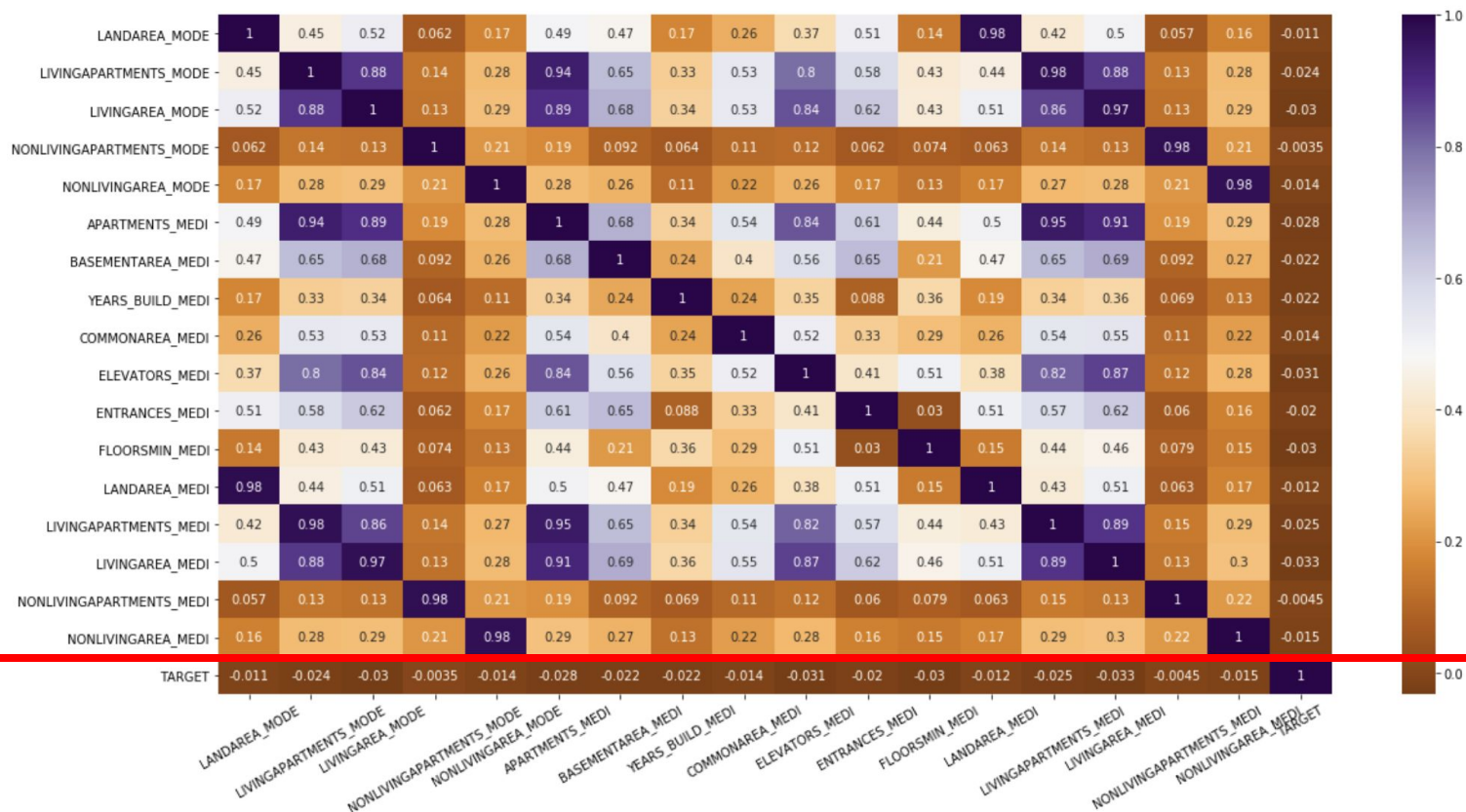
- Pick out # Missing Values > threshold (60%)



Check Correlations with Target



Check Correlations with Target (Cont'd)




Missing Values — Categorical features

- NAME_TYPE_SUITE (904 Missing)
 - Group with “Other_A”, “Other_B”, “Group of People” as “Others”
- OCCUPATION_TYPE (67553 Missing)
 - A large portion is missing
 - Flag as “Unknown”
- EMERGENCYSTATE (102265 Missing)
 - Low Correlation with Target
 - Drop the column

Unaccompanied	173927
Family	28148
Spouse, partner	7943
Children	2272
Other_B	1269
Other_A	606
Group of people	188

Name: NAME_TYPE_SUITE, dtype: int64



Unaccompanied	173927
Family	28148
Spouse, partner	7943
Others	2967
Children	2272

Name: NAME_TYPE_SUITE, dtype: int64

Missing Values —— Numeric features

- Housing_Information_Avg/Median/Mode
 - Only keep _Median columns
 - Fill _Median with column median respectively

	Missing Values	% of Total Values
YEARS_BEGINEXPLUATATION_AVG	105173	48.859271
FLOORSMAX_AVG	107261	49.829274
YEARS_BEGINEXPLUATATION_MODE	105173	48.859271
FLOORSMAX_MODE	107261	49.829274
APARTMENTS_MEDI	109407	50.826222
BASEMENTAREA_MEDI	126106	58.583925
YEARS_BEGINEXPLUATATION_MEDI	105173	48.859271

Preprocessing Stage

- Deal with missing values
- **Create dummy variables for categorical features**
- Anomaly Detection
- Domain Knowledge Feature Extractions
 - Credit Income Percentage, Previous Late_Payment Counts, Credit Utilization ...
 - Interaction Terms: External Source 1, External Source 2, Data_Birth...
 - Numerical variables: calculate aggregation statistics (min, max, mean)
 - Categorical variables: counts, normed counts
- Feature Normalization

Preprocessing Stage

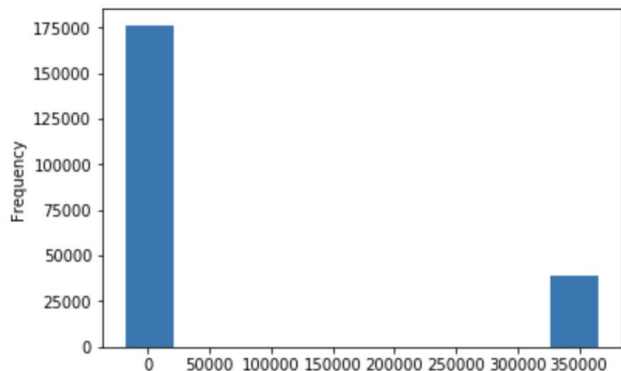
- Deal with missing values
- Create dummy variables for categorical features
- **Anomaly Detection**
- Domain Knowledge Feature Extractions
 - Credit Income Percentage, Previous Late_Payment Counts, Credit Utilization ...
 - Interaction Terms: External Source 1, External Source 2, Data_Birth...
 - Numerical variables: calculate aggregation statistics (min, max, mean)
 - Categorical variables: counts, normed counts
- Feature Normalization

Anomaly Detection (Example)

- Days_Employed
 - The maximum doesn't seem to be right
 - Fill as NaN and then fill with Median

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

```
count    215257.000000
mean      63906.197099
std       141353.827451
min       -17583.000000
25%       -2760.000000
50%       -1217.000000
75%       -290.000000
max       365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```



Preprocessing Stage

- Deal with missing values
- Create dummy variables for categorical features
- Anomaly Detection
- **Domain Knowledge Feature Extractions**
 - Interaction Terms: External Source 1, External Source 2, Date_Birth...
 - Aggregation statistics: group by loan ID (min, max, mean) ; counts, normed counts
 - Late_Payment, Credit Utilization...
- Feature Normalization
 - Income ~ 10,000 ; Age 30 ; Credit Amount ~ 1,000

Domain Knowledge Features

- Late Payment
 - Data source: installment_payments.csv
 - DAYS_INSTALLMENT: When the installment of previous credit was supposed to be paid (relative to application date of current loan)
 - DAYS_ENTRY_PAYMENT: When was the installments of previous credit paid actually (relative to application date of current loan)
 - Delayed payment days = $\text{DAYS_INSTALLMENT} - \text{DAYS_ENTRY_PAYMENT}$
- Credit Utilization

Domain Knowledge Features

- Late Payment
- Credit Utilization
 - A strong indicator for a risky customer
 - AMT_BALANCE: Balance during the month of previous credit
 - AMT_CREDIT_LIMIT_ACTUAL: Credit card limit during the month of the previous credit
 - Credit Utilization = Credit card balance/(1+ credit card limit)
 - filter data in recent 2 month

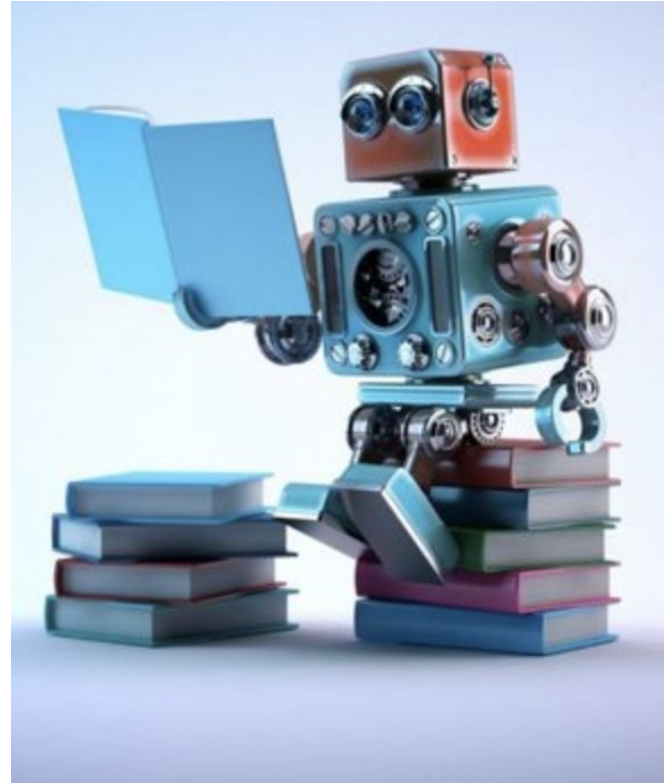
Preprocessing Stage

- Deal with missing values
- Create dummy variables for categorical features
- Anomaly Detection
- **Domain Knowledge Feature Extractions**
 - Credit Income Percentage, Previous Late_Payment Counts, Credit Utilization ...
 - Interaction Terms: External Source 1, External Source 2, Data_Birth...
 - Numerical variables: calculate aggregation statistics group by loan ID (min, max, mean)
 - Categorical variables: counts, normed counts group by loan ID
- **Feature Normalization**

Methodology

Binary Classification Models

- Baseline model: Naive Bayes
- SVM
- Random Forest
- Gradient Boosting Method



Evaluation Metrics

- **AUC** — the area under the ROC curve
 - ROC curve: TPR vs FPR at various thresholds
 - Range from [0,1]: the higher, the better
 - Scale-invariant: relative ranking of probabilities
 - Evaluation metric selected by Kaggle

- **Recall**
 - How many of the total positives out there does the model classify as positive?
 - Range from [0,1]: the higher, the better
 - **False Negative** may bring disasters and huge loss.
 - Predict “Not Default” but actually “Default”

$$Recall = \frac{TP}{P} = \frac{TP}{TP + FN}$$

Baseline Model

- Naive Bayes

- Assumption of conditional independence between every pair of features given the value of the class variable

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i \mid y)$$

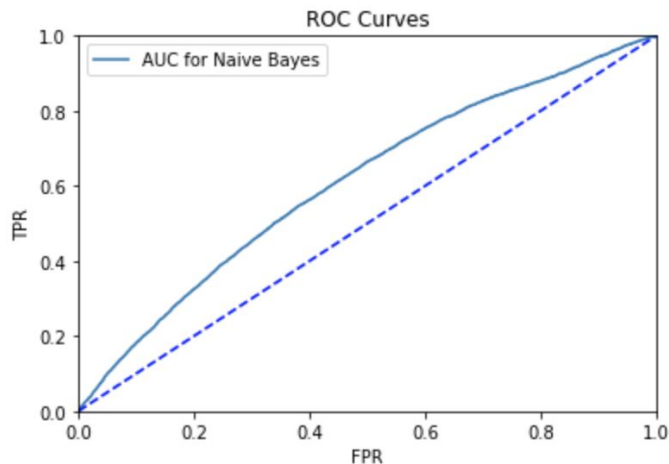
\Downarrow

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i \mid y),$$

Baseline Model

- Naive Bayes Result
 - AUC Score: 0.61
 - Not so good
 - Predictors are not completely independent

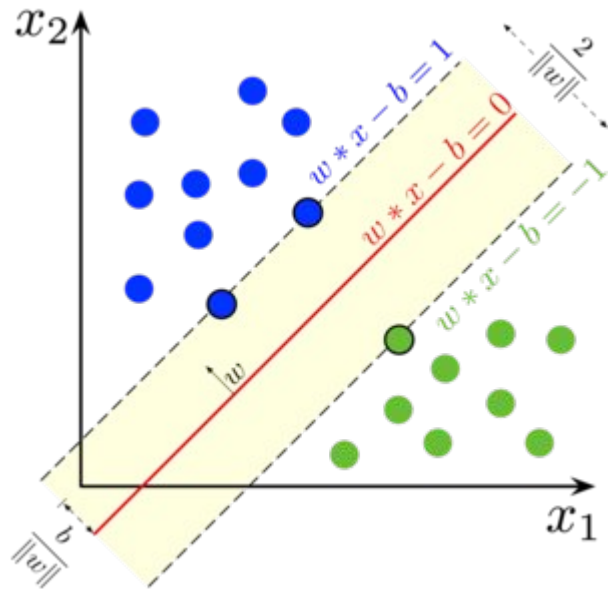
AUC score for Naive Bayes is: 0.6100048677235163



Thomas Bayes
1702 - 1761

Support Vector Machine

- An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible
- Too slow on local machine
- Use PySpark instead
 - Only SVM classifier with linear kernel on PySpark now
 - Other Kernels are under development

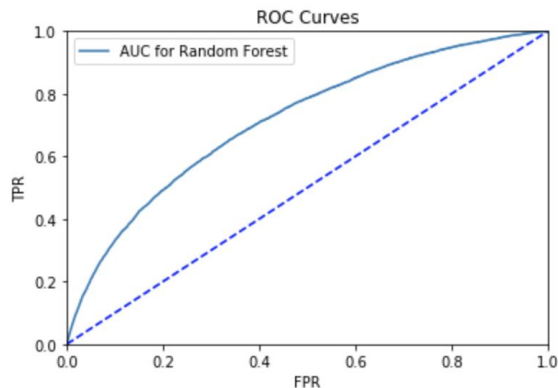


```
>>> from pyspark.ml.feature import StandardScaler
>>> standardscaler=StandardScaler().setInputCol("features").setOutputCol("Scaled_features")
>>> df=standardscaler.fit(df).transform(df)
>>> from pyspark.mllib.evaluation import BinaryClassificationMetrics
>>> predictionAndLabels= df2.rdd
>>> metrics = BinaryClassificationMetrics(predictionAndLabels)
>>> metrics.areaUnderROC
0.58759968368235428
```

Tree-Based Methods

- Random Forest
 - “Random” comes from two sides:
 - Bootstrap: sub-samples of the dataset with replacement
 - Subset of features
 - Reduce the variance without making bias worse: average the predictions
- Best Hyper-parameter Combination
 - n_estimators = [50, 100, 150, 200, 250]
 - max_depth = [5, 10, 20, 30]
 - min_samples_split = [10, 20, 30, 40]
 - AUC: 0.7156

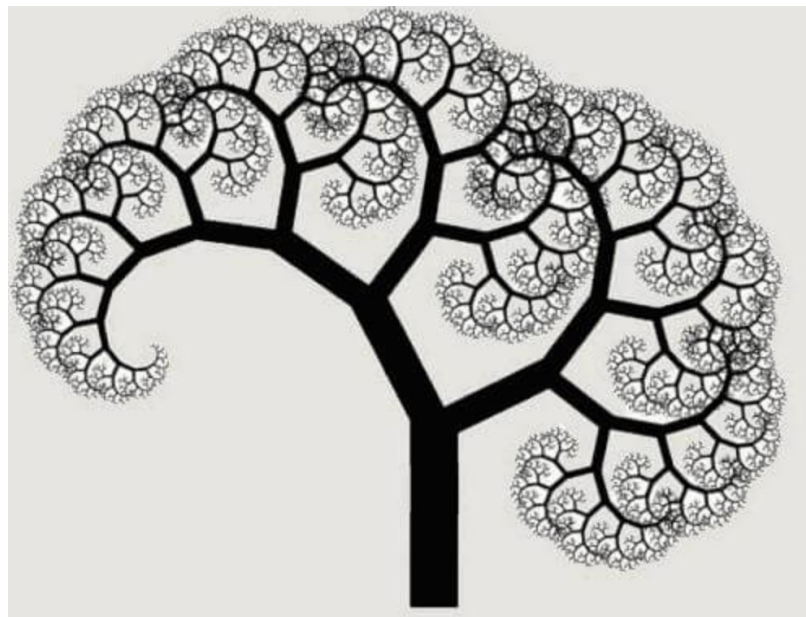
AUC score for Random Forest is: 0.7156391040680428





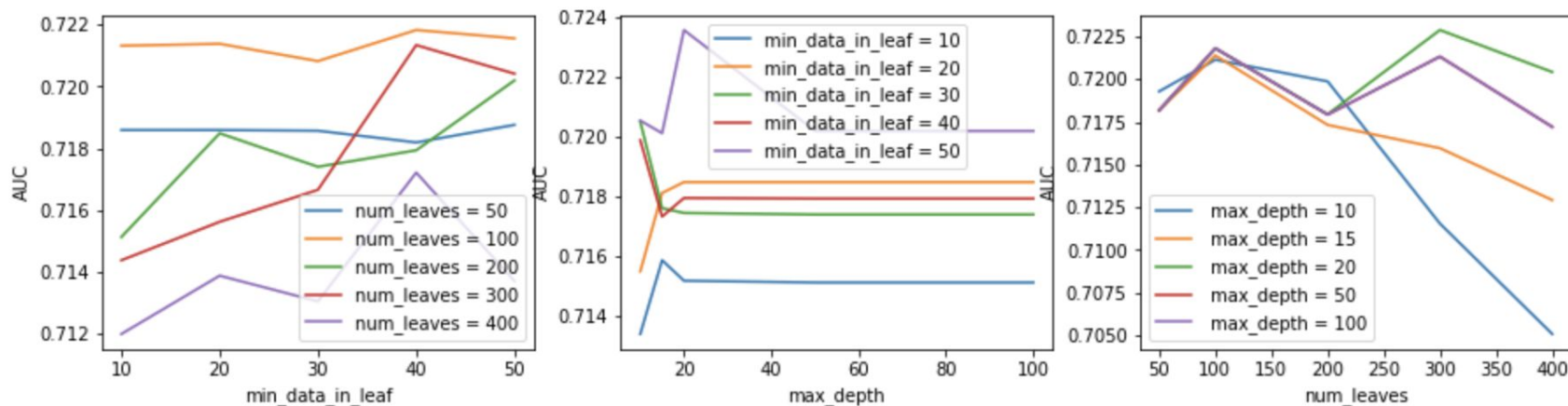
Gradient-Boosting Methods

- LightGBM by Microsoft
- A gradient boosting framework that uses tree based learning algorithms.
- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Support of parallel and GPU learning
- Capable of handling large-scale data



Hyper-parameter Tuning

- LightGBM
 - num_leaves = [50, 100, 200, 300, 400]
 - min_data_in_leaf = [10, 20, 30, 40, 50]
 - max_depth = [10, 15, 20, 50, 100]

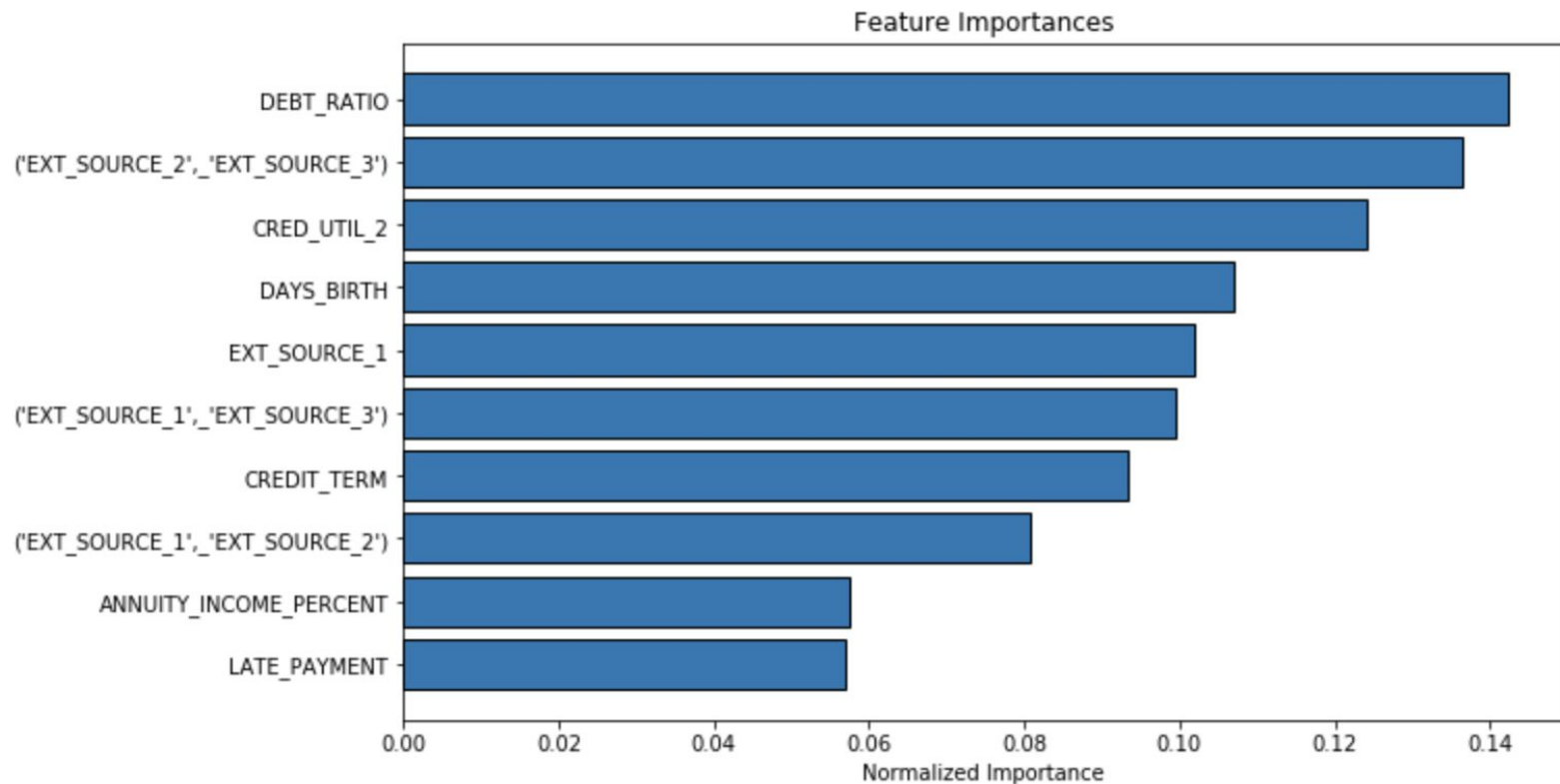


The current best hyper-parameter is: (Max_depth = 20, Min_data_in_leaf = 50, Num_leaves = 100)

AUC score after hyper-parameter tuning for LightGBM model is: 0.7215612234439602

Recall score for LightGBM model is: 0.8228767515012868

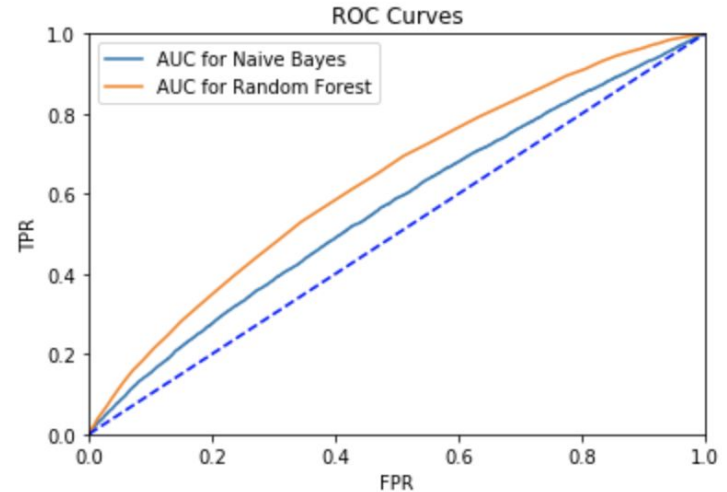
Feature Importance



SMOTE (Synthetic Minority Over-sampling Technique)

- The new instances are not just copies of existing minority cases
- Takes samples of the feature space for each target class and its nearest neighbors
- Generate new examples that combine features of the target case with features of its neighbors.
- This approach increases the samples available to each class and makes the samples more general.
- Advantages: Alleviates overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances.
- Disadvantages: SMOTE is not very practical for high dimensional data

AUC score for Naive Bayes is: 0.562100401117489
AUC score for Random Forest is: 0.630095041182555



Performance Improvements

- Parallel programming (MPI)
- Performance tuning: Line_profiler
- Concurrency (Pools)
- Vectorization in computation
- Itertools & Numba (vectorize)
- PySpark

MPI

- One program to exploit multiple processors
- Facilitate communication between parallel processors
- Accelerate hyperparameter tuning for random forest compared to for-loop
 - Each rank is given a group of hyper-parameter combinations

	For-loop	MPI (n=8)
Time	7502.6633841991425 s ~ greater than 2 hours	2696.423863172531 s ~ less than 1 hour

Performance Tuning

- line_profiler

Use line profiler to detect which lines should be optimized

```
%load_ext line_profiler
```

```
%lprun -f agg_categorical agg_categorical(previous, 'SK_ID_CURR', 'previous')
```

Timer unit: 1e-06 s

Total time: 34.1753 s File: Function: agg_categorical at line 1

32					# Groupby the group var and calculate the sum and mean
33	1	26241780.0	26241780.0	76.8	categorical = categorical.groupby(parent_var).agg(['sum', 'mean'])

Concurrency

- Pools

- Multiprocessing for aggregation

```
%%timeit -n 1 -r 1
```

(Before)

```
previous_counts = agg_categorical(previous, 'SK_ID_CURR', 'previous') # aggregate categorical features
```

34.8 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

```
import multiprocessing as mp
```

```
p = mp.Pool(processes=8)
```

```
df_pool_results = p.map(agg, split_categoricals)
```

```
p.close()
```

```
%%timeit -n 1 -r 1
```

```
opt_previous_counts = opt_agg_categorical(previous, 'SK_ID_CURR', 'previous') (After)
```

20.1 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

42.24% ↑

Column Vectorization

- for loop vs vectorization

- Debt_ratio

```
bureau['DEBT_RATIO'] = bureau.apply(lambda row: row['AMT_CREDIT_SUM_DEBT']/(1+row['AMT_CREDIT_SUM']),  
                                   axis = 1)
```

27.5 s ± 503 ms per loop

```
bureau['DEBT_RATIO'] = bureau['AMT_CREDIT_SUM_DEBT']/(1+bureau['AMT_CREDIT_SUM'])
```

6.67 ms ± 109 µs per loop

99% ↑

Itertools & Numba (vectorize)

- Itertools: product, combinations
 - product: hyper-parameters tuning
 - combination: interaction terms
- Time before: 0.05254673957824707 s
- Time after: 0.037714738845825195 s
- Improvement: 28.22% ↑

```
from numba import vectorize, float64
from itertools import combinations_with_replacement
columns_name = ['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH']
c = list(combinations_with_replacement(columns_name, r=2))
@vectorize([float64(float64, float64)])
def vec_mul(a, b):
    return a*b
for i in c:
    X_train[i] = vec_mul(X_train[i[0]], X_train[i[1]])
    X_test[i] = vec_mul(X_test[i[0]], X_test[i[1]])
```

```
param = [(x, y, z) for x, y, z in itertools.product(num_leaves, min_data_in_leaf, max_depth)]
```

PySpark

- Translate Dataframe operations into RDD transformations.
- RDD (Resilient Distributed Dataset)
 - Lazy evaluation, Caching, Pipelining, Partition layout

```
>>> from pyspark.ml.feature import StandardScaler
>>> standardscaler=StandardScaler().setInputCol("features").setOutputCol("Scaled_features")
>>> df=standardscaler.fit(df).transform(df)
>>> from pyspark.mllib.evaluation import BinaryClassificationMetrics
>>> predictionAndLabels= df2.rdd
>>> metrics = BinaryClassificationMetrics(predictionAndLabels)
>>> metrics.areaUnderROC
0.58759968368235428
```

Reference:

- Lecture Slide from DSGA-1001 “Introduction to Data Science”
- Precision vs Recall (<https://towardsdatascience.com/precision-vs-recall-386cf9f89488>)
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems (pp. 3146-3154)
- Meng, Q., Ke, G., Wang, T., Chen, W., Ye, Q., Ma, Z. M., & Liu, T. Y. (2016). A communication-efficient parallel algorithm for decision tree. In Advances in Neural Information Processing Systems (pp. 1279-1287)
- LightGBM Documentation: <https://lightgbm.readthedocs.io/en/latest/>
- How to handle Imbalanced Classification Problems in machine learning?
(<https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/>)



Q&A

Thank You!