# Bank Marketing Analysis

Group 2

Wang Yuanyuan | Xu Shaoqing | Zeng Xiangwei | Zhang Bojun | Zhang Shizhe

# Problem Statement & Data Description

# Problem Statement

A term deposit is a deposit that a bank offers with a fixed rate in which clients will get their money back at a specific maturity time. This is a common approach for banks to ensure their cash flow and invest for a higher rate of return (RoR).

This project looks at some data from direct marketing campaigns through phone calls of a Portuguese banking institution in order to predict if the client will subscribe to a term deposit and develop better strategies for future marketing campaigns.

# Data Description

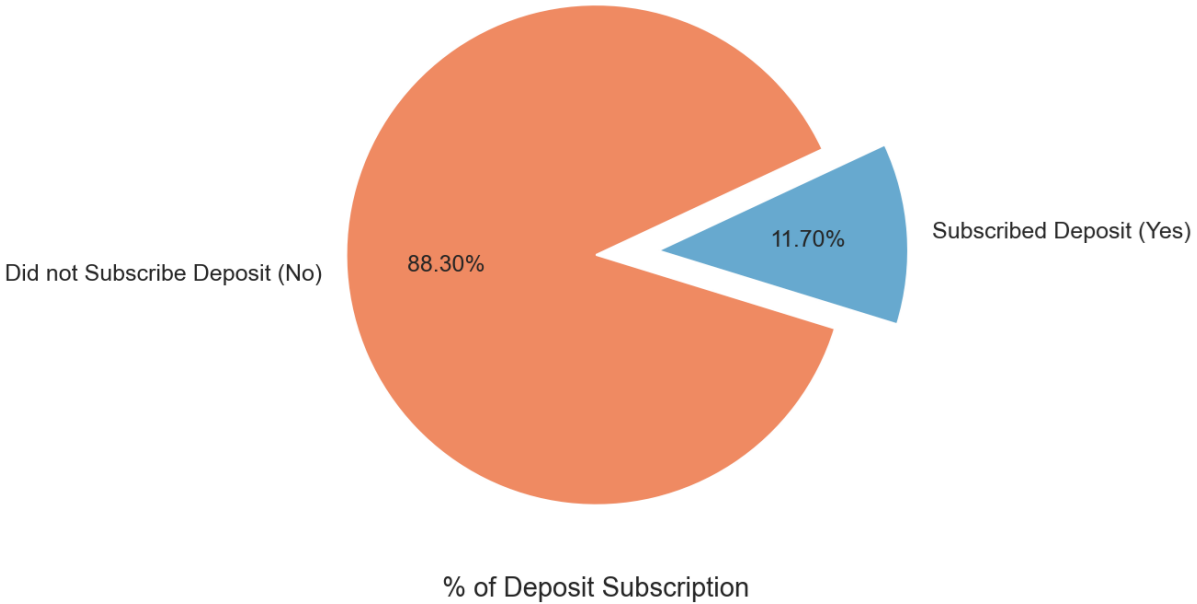
- 45211 instances and 17 columns (16 attributes and 1 target)
- 9 categorical attributes and 7 numerical attributes
- Each instance represents a client with their historical campaign activities
- The target attribute is ‘y’ with two class labels which represents whether a client subscribes to a term deposit

| Categorical Attribute |
|---|
| job |
| marital |
| education |
| default |
| housing |
| loan |
| contact |
| month |
| poutcome |

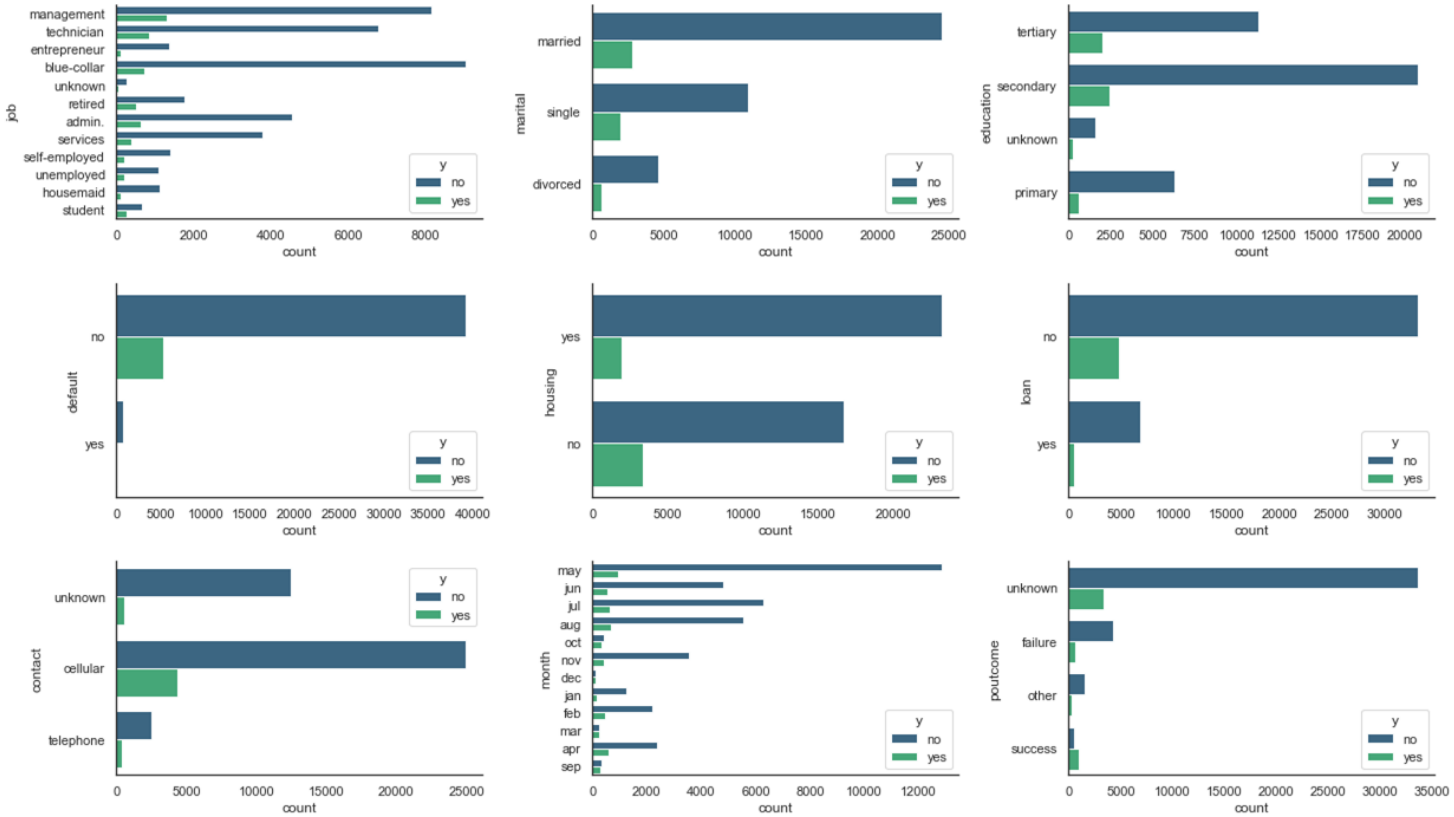
| Numerical Attribute |
|---|
| age |
| balance |
| day |
| duration |
| campaign |
| pdays |
| previous |
| |
| |

# Exploratory Data Analysis

# Imbalanced Classification Problem



# Categorical Attribute Visualization




Due to imbalanced class labels, the comparison between the count of yes and no labels within each category is not valid.

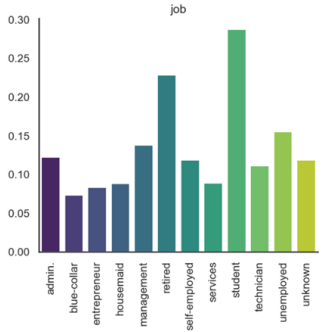
# Deposit Subscription Ratio

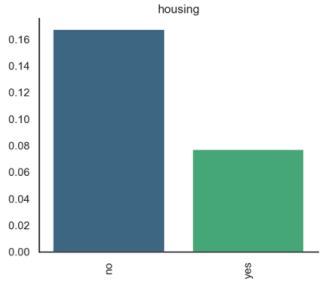
$$Ratio = \frac{COUNT(yes|category\ A)}{COUNT(category\ A)}$$

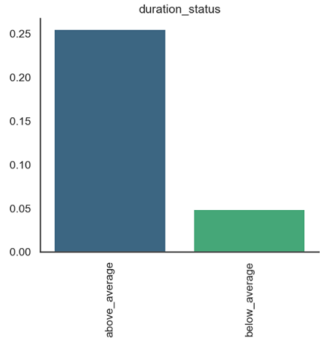
The **profile** of a client who is more likely to subscribe a deposit:

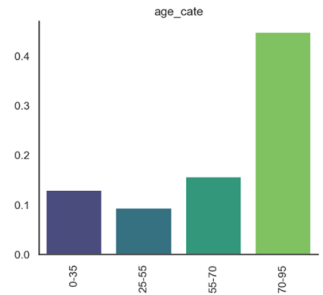
- Elderly (70-95)
- Without loan
- Previous campaign successful
- Last contact duration above average
- Retired/*Student**

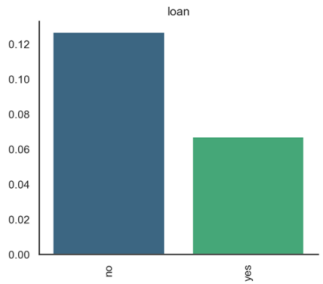
** Small sample size*

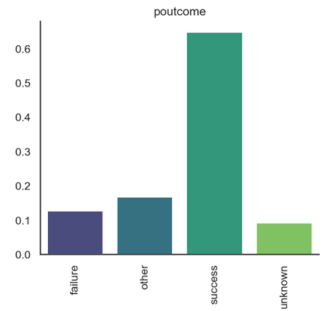


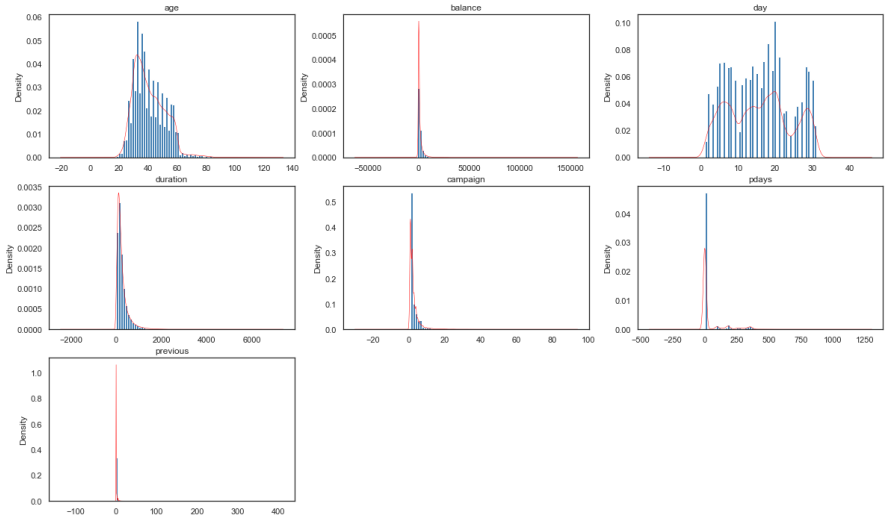



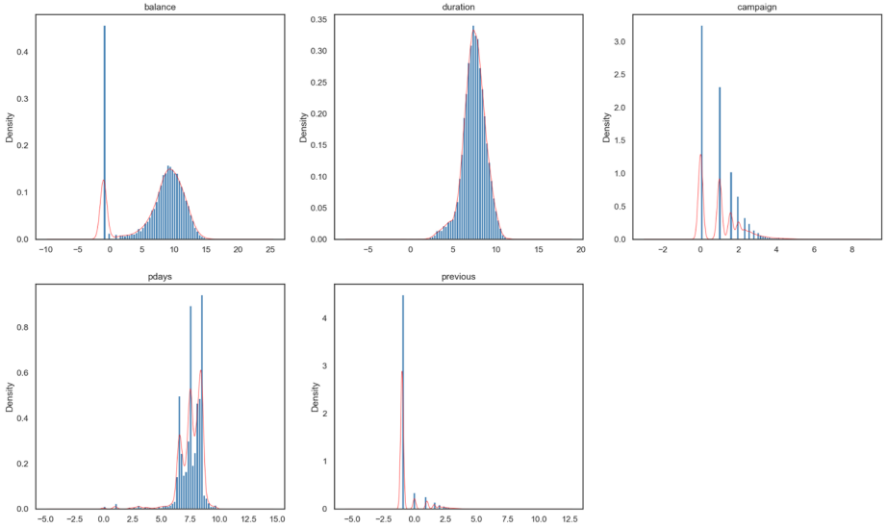
# Numerical Attribute Visualization

- The distribution of balance, duration, campaign, pdays, previous are right-skewed
- Apply log transformation to remove skewness

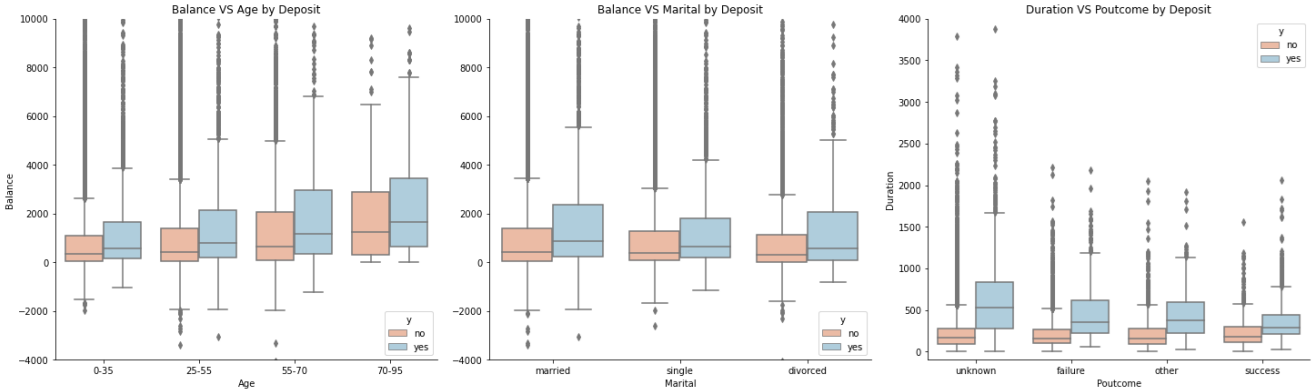
Before log transformation:




After log transformation:



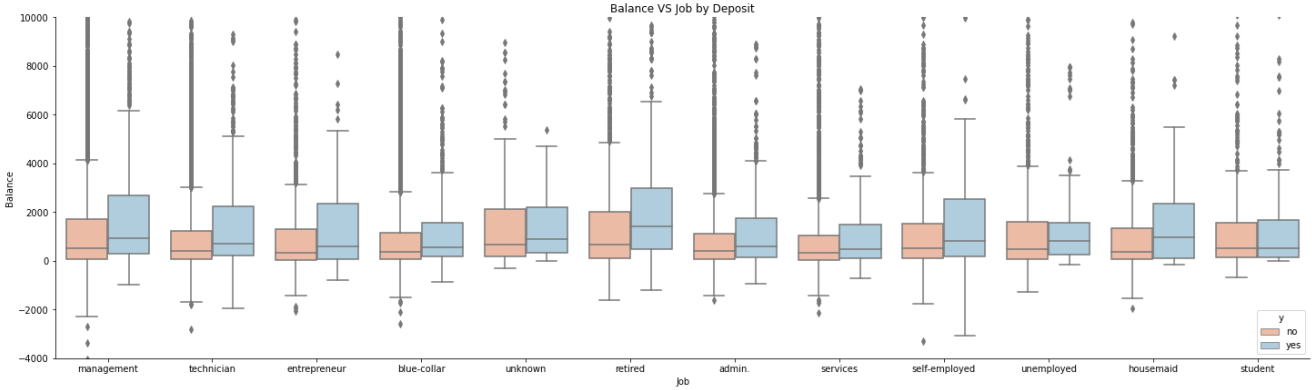
# Visualization on Balance and Duration










The **profile** of high bank balance clients:

- Elderly
- Married
- Retired

Clients who subscribed the deposit tend to have a higher balance and last contact duration compared to the clients who refused the deposit.

# Data Preprocessing

# Label Encoding and One - Hot Encoding

| | Gender |
|---|---|
| 1 | Male |
| 2 | Female |
| 3 | Female |
| 4 | Male |

| | Gender |
|---|---|
| 1 | 1 |
| 2 | 0 |
| 3 | 0 |
| 4 | 1 |

**Label Encoding**: Assign each unique category in a categorical Feature with an integer, no new columns are created.

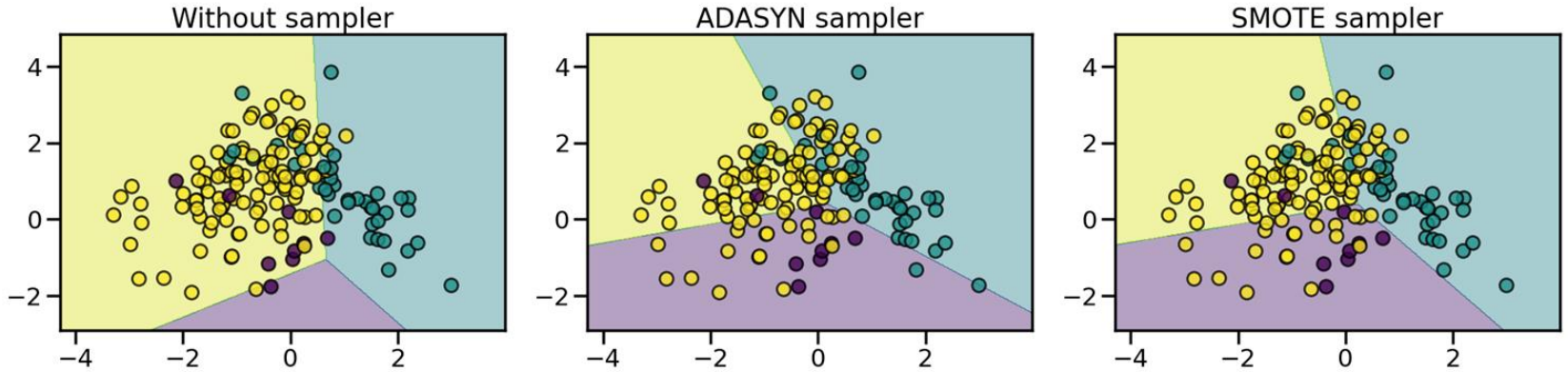
| | Job |
|---|---|
| 1 | Teacher |
| 2 | Worker |
| 3 | Driver |
| 4 | Engineer |

| | Job_Worker | Job_Driver | Job_Teacher | Job_Engineer |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 |

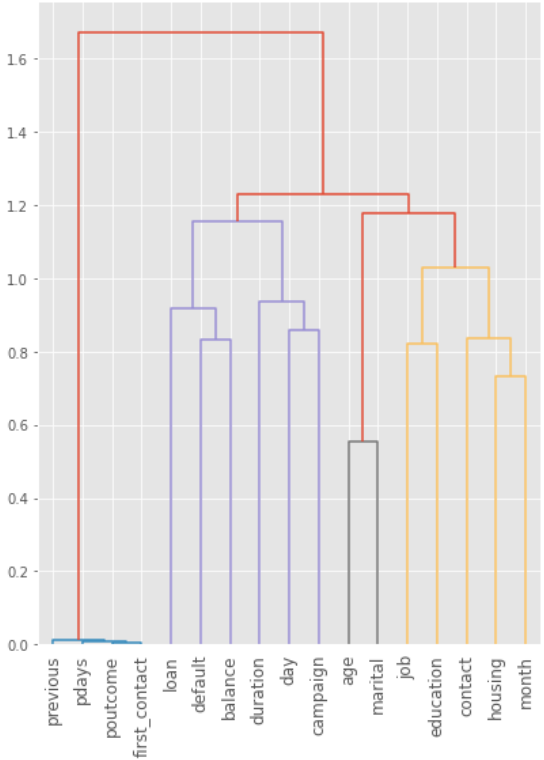
**One - Hot Encoding:** Create a new column for each unique category in a categorical feature and drop one column to avoid multicollinearity.

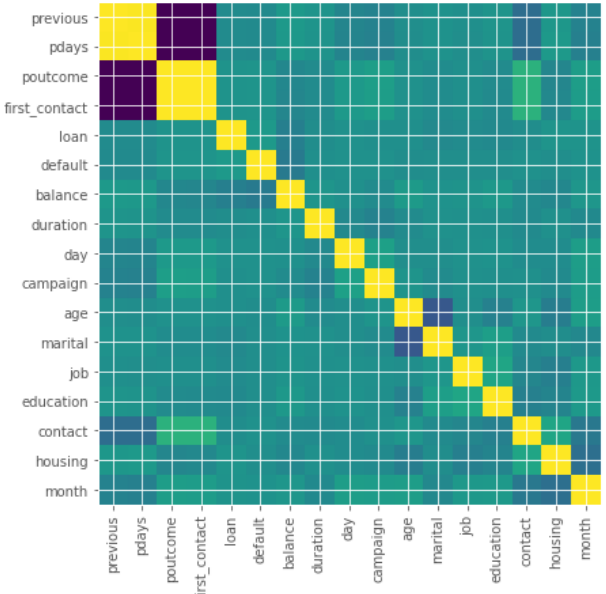
# Imbalanced Data Processing

1. Random oversampling
2. Random undersampling
3. Synthetic data: ADASYN sampler, SMOTE sampler



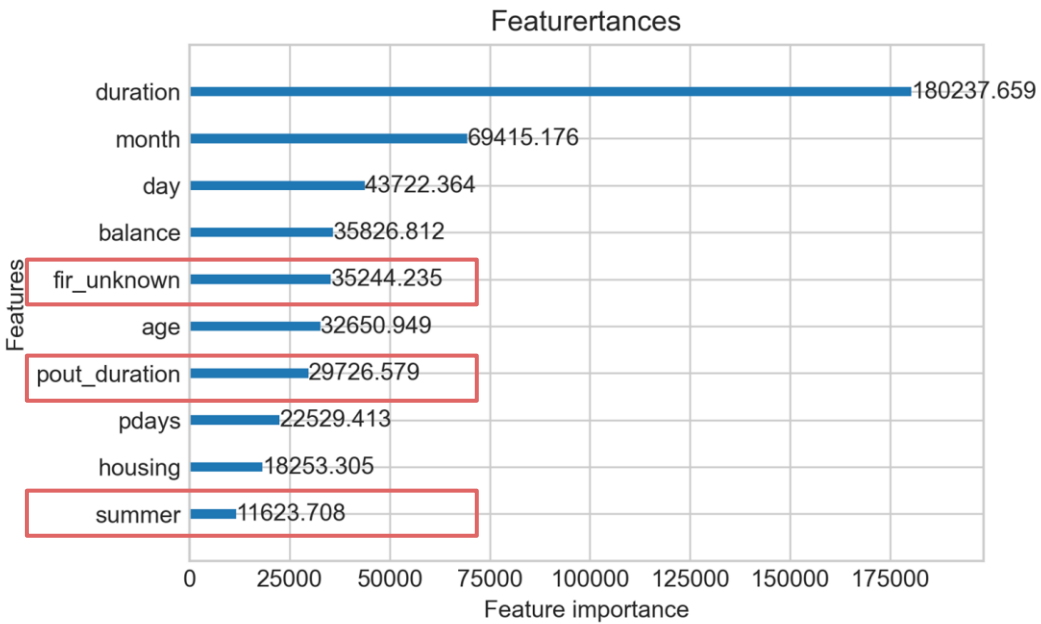
# Feature Selection




- Remove zero, near zero variance features with highly identical values.
- Remove one of the highly correlated feature sets with 0.8 cutoff to reduce redundancy.

# Feature Construction




Feature importance based on lightGBM

Construct new features by combining poutcome, duration, previous, contact and campaign ets.

Distinguish the willingness of purchase by combining marital, age and balance status.

**Rank 5th: fir_unknown**

**?** new customers with unknown contact type

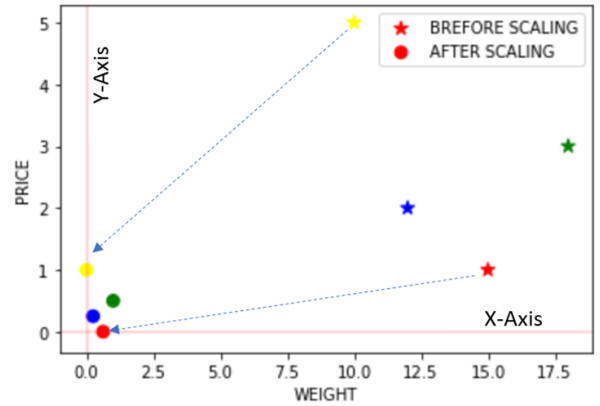
**Rank 7th: pout_duration**

**?** customers has a long contact duration with successful previous outcome

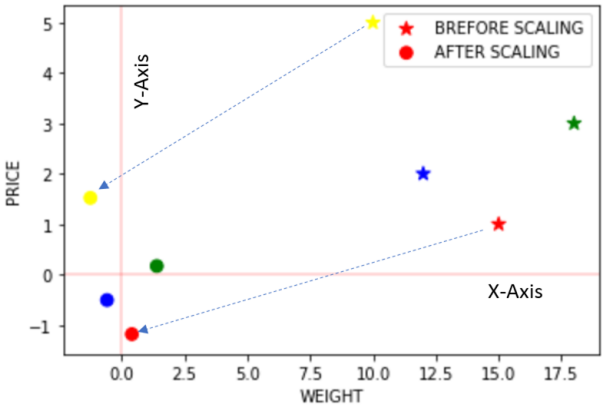
**Rank 10th: summer**

**?** last contact month in summer

# Feature Scaling

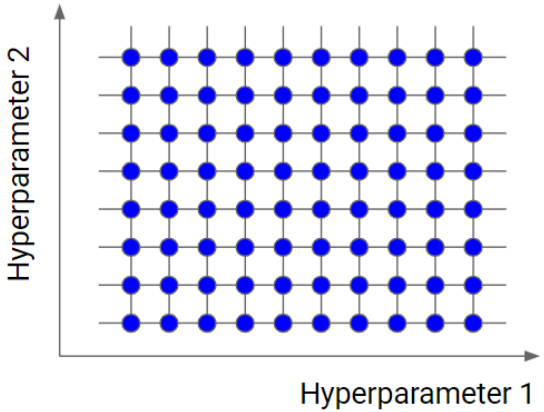



- **Normalization:** rescales the values of each feature within a range like [0,1] and [-1,1].

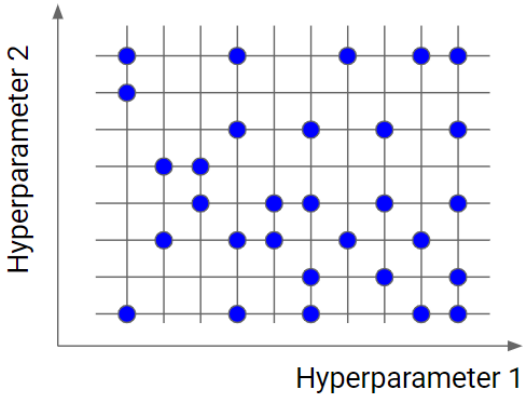



- **Standardization:** rescales the data to have a mean of 0 and a unit variance.

# Hyperparameter Tuning



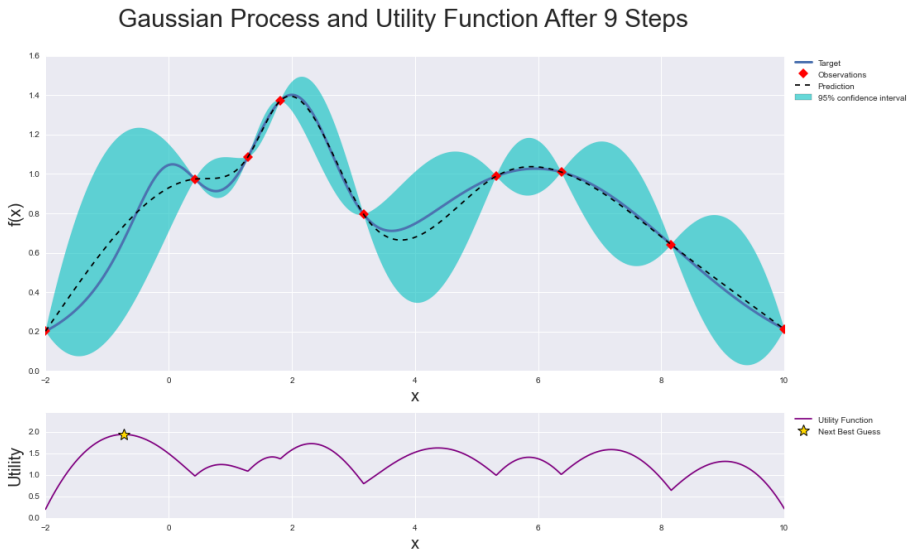

- **Grid Search:** Divide the domain of the hyperparameters into a discrete grid and try every combination of values of this grid, calculating some performance metrics using cross-validation to find the optimal combination of values.

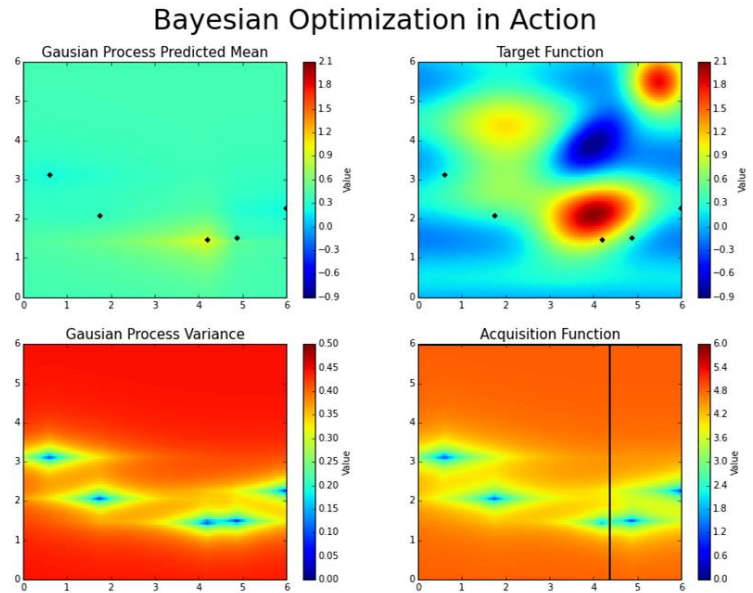



- **Random Search:** Similar to grid search, but instead of using all the points in the grid, it tests only a randomly selected subset of these points.

# Hyperparameter Tuning

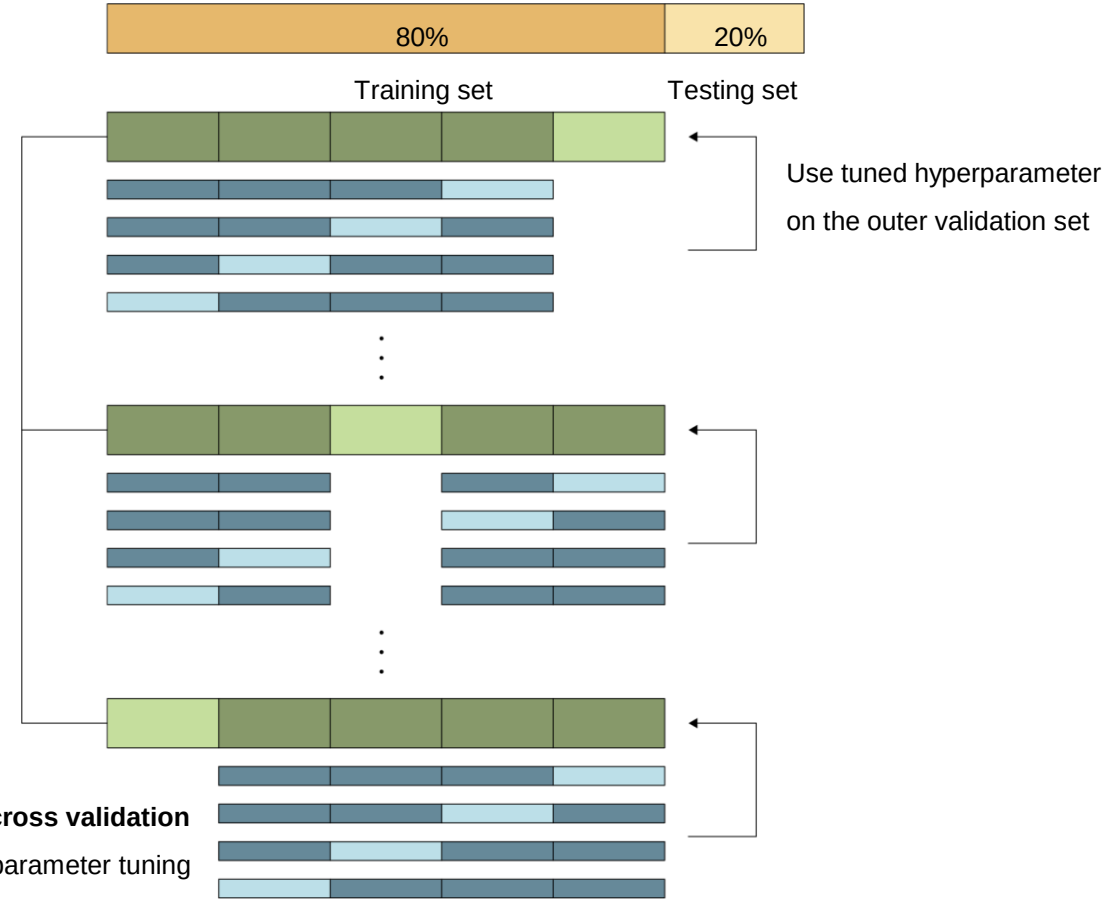
- **Bayesian Optimization**




Bayesian optimization works by constructing a posterior distribution of functions (gaussian process) that best describes the function you want to optimize. As the number of observations grows, the posterior distribution improves, and the algorithm becomes more certain of which regions in parameter space are worth exploring, as seen in the picture above. It ensures that next step has the largest probability to result a better model performance.

# Train Test Split & Nested Cross Validation




Nested cross validation estimates an unbiased generalization performance

# Modeling & Experimental Results

# Metric Selection

## Selected Metrics

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

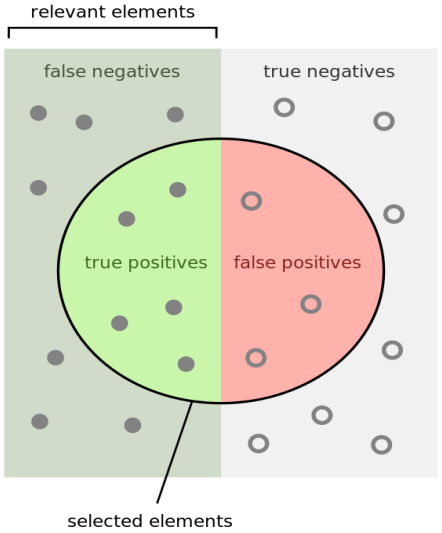
$$Precision = \frac{TP}{TP + FP}$$

$$F1\ Score = \frac{2TP}{2TP + FP + FN}$$

$$AUC = area\ under\ the\ ROC\ curve$$

$$Kappa = \frac{total\ accuracy - random\ accuracy}{1 - random\ accuracy}$$

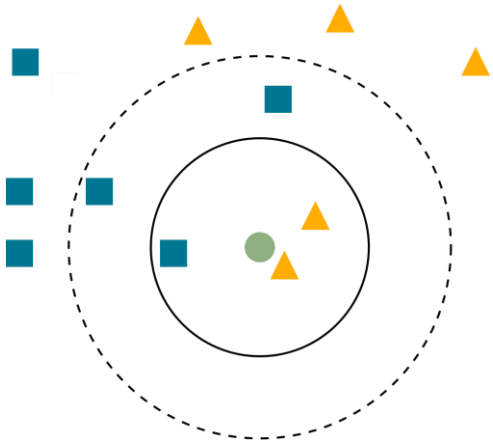
$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

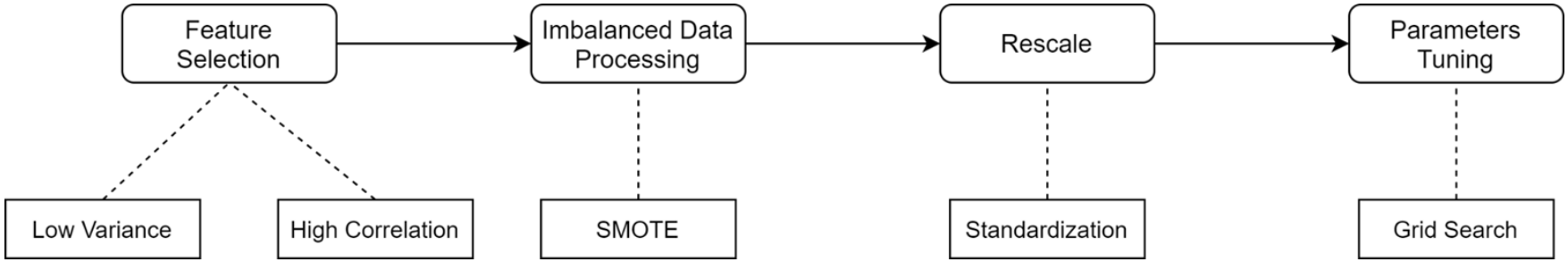



*Accuracy is invalid due to imbalanced data distribution.

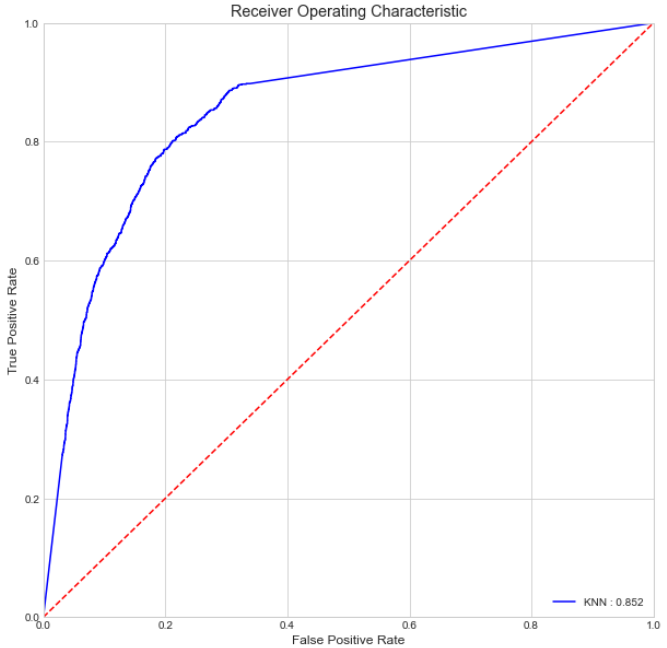
# K Nearest Neighbour

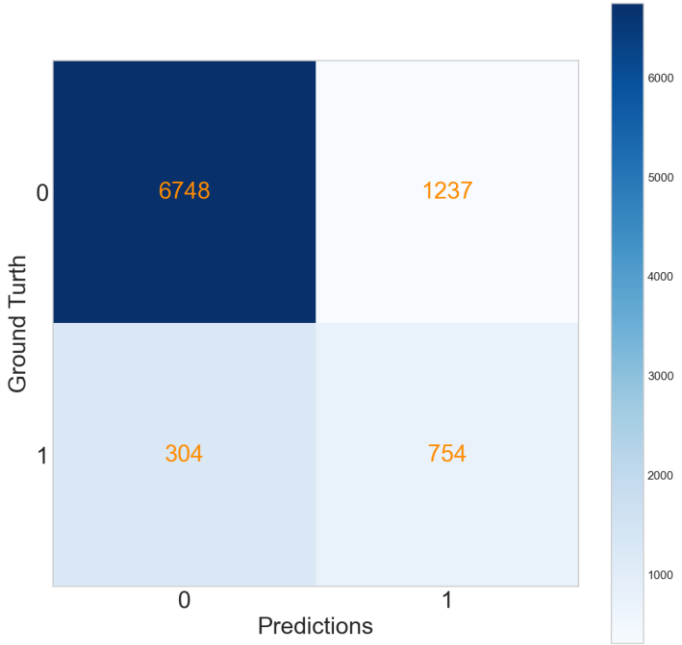
**K Nearest Neighbour is a simple algorithm that classifies the new data based on a similarity measure. It is mostly used to classifies a data point based on how its neighbours are classified.**





# K Nearest Neighbour

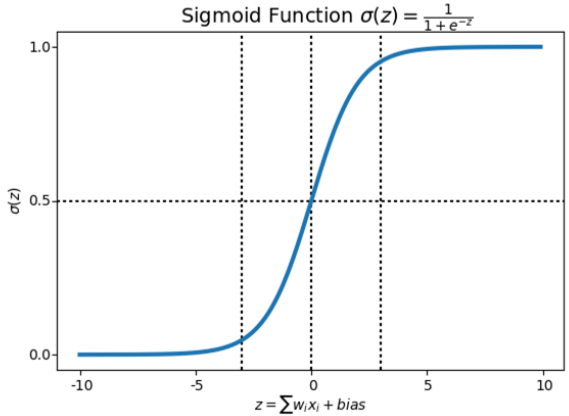


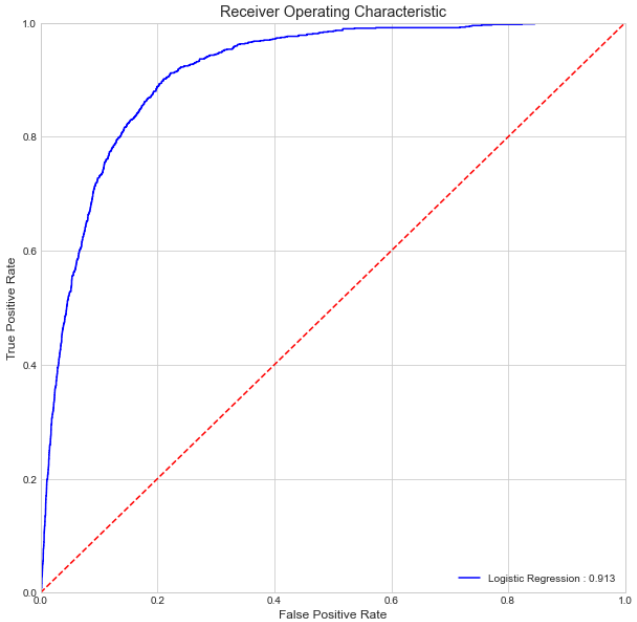

| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| K Nearest Neighbour | 0.713 | 0.379 | 0.495 | 0.852 | 0.403 | 0.433 |

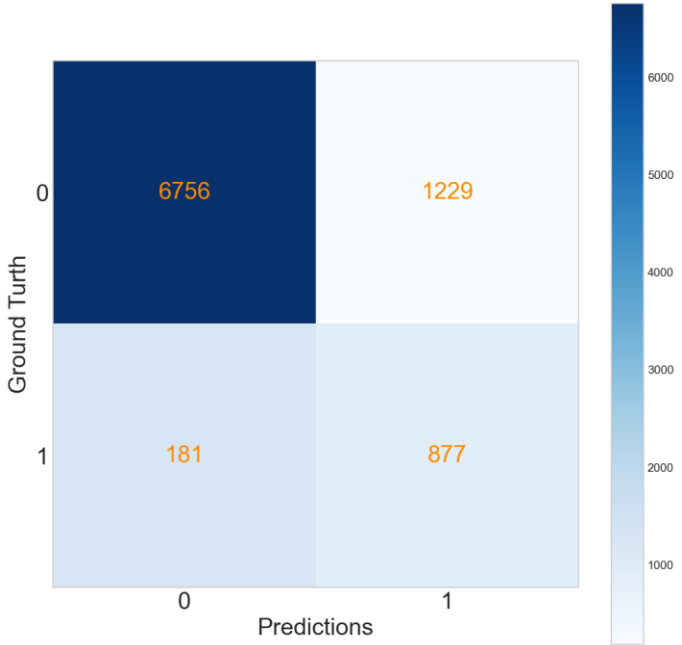
# Logistic Regression

**Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.**




# Logistic Regression

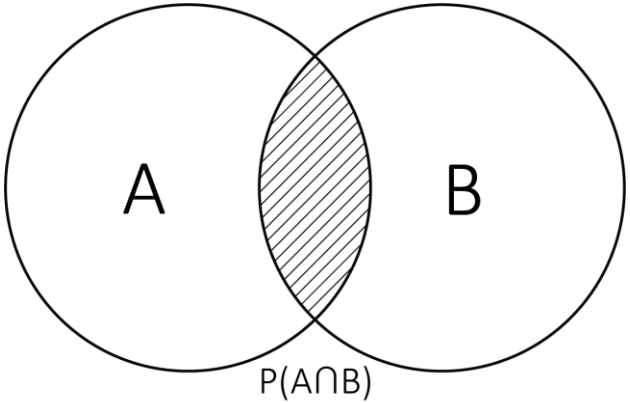


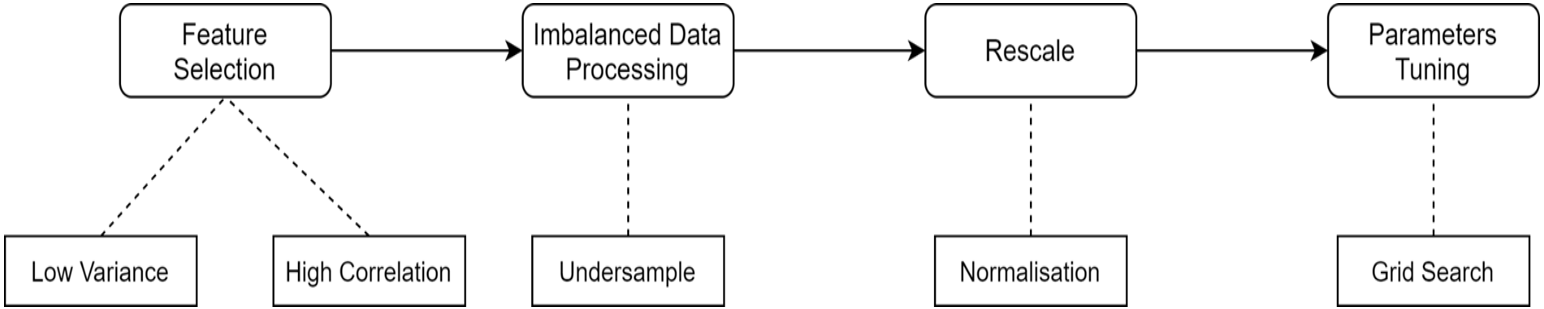




| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| Logistic Regression | 0.829 | 0.416 | 0.554 | 0.913 | 0.472 | 0.513 |

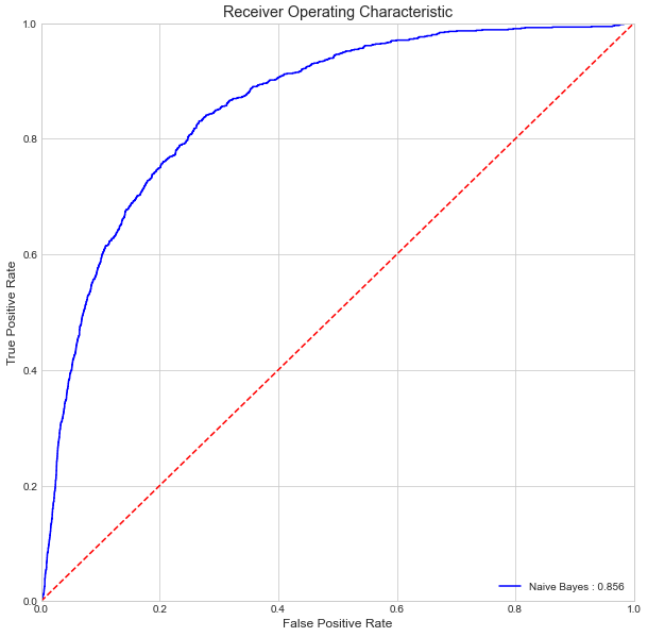
# Naive Bayes

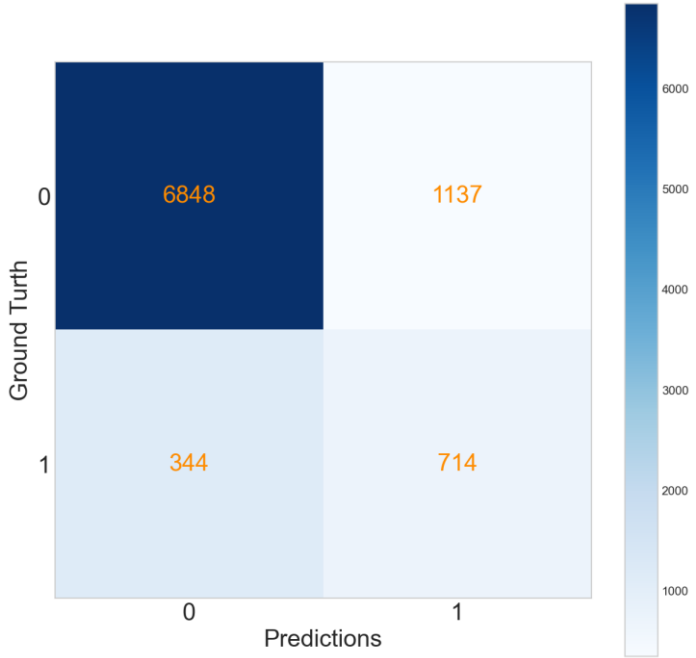
$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$



# Naive Bayes

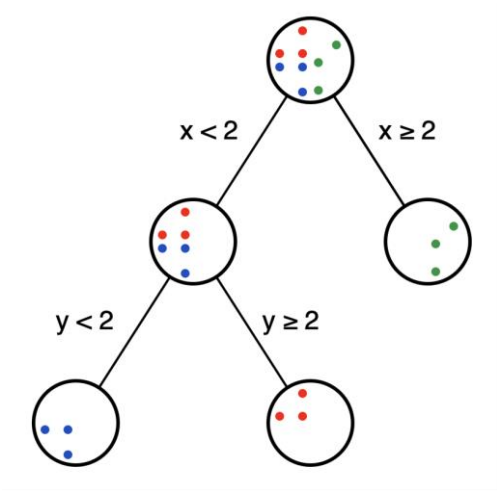


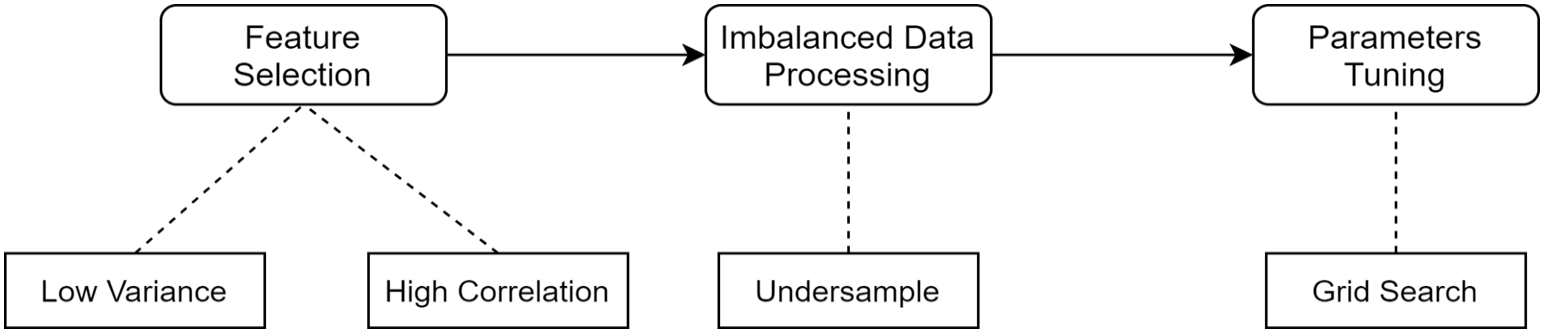




| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| Naive Bayes | 0.675 | 0.386 | 0.491 | 0.856 | 0.402 | 0.424 |

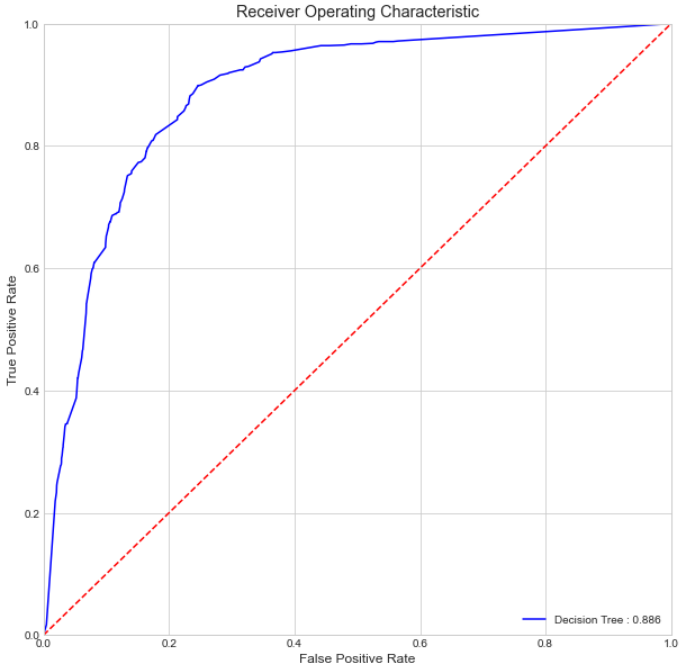
# Decision Tree

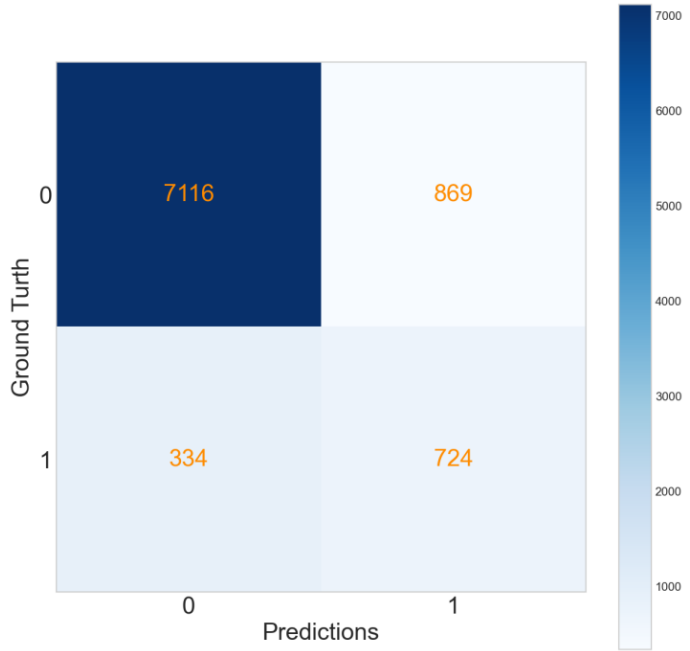
**Decision tree model predicts the value of a target variable by learning simple decision rules inferred from the data features.**



# Decision Tree

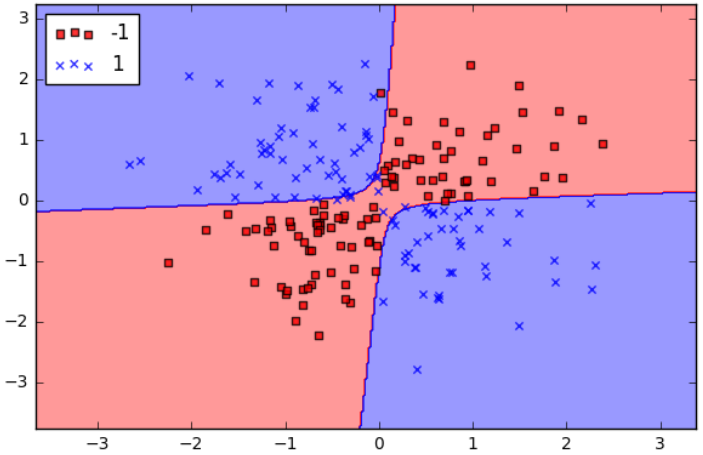


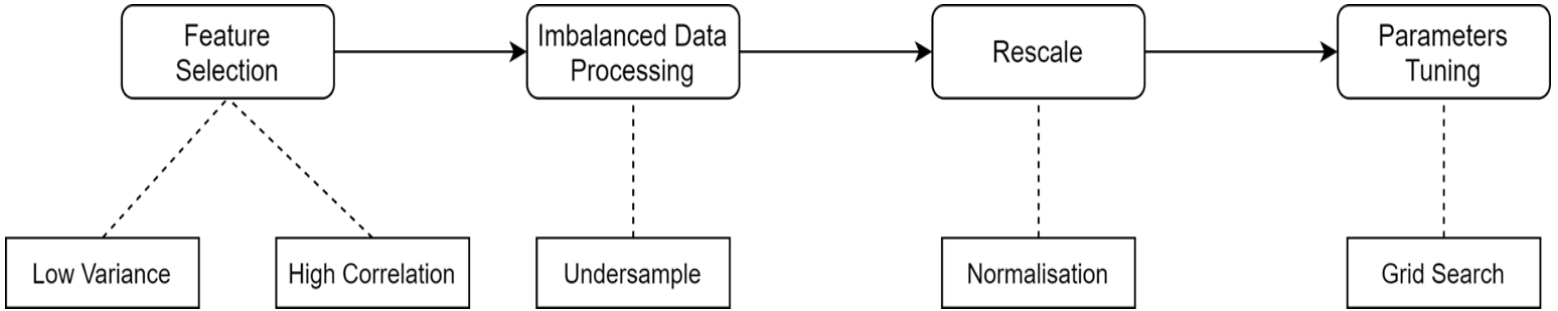




| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| Decision Tree | 0.684 | 0.454 | 0.546 | 0.886 | 0.472 | 0.486 |

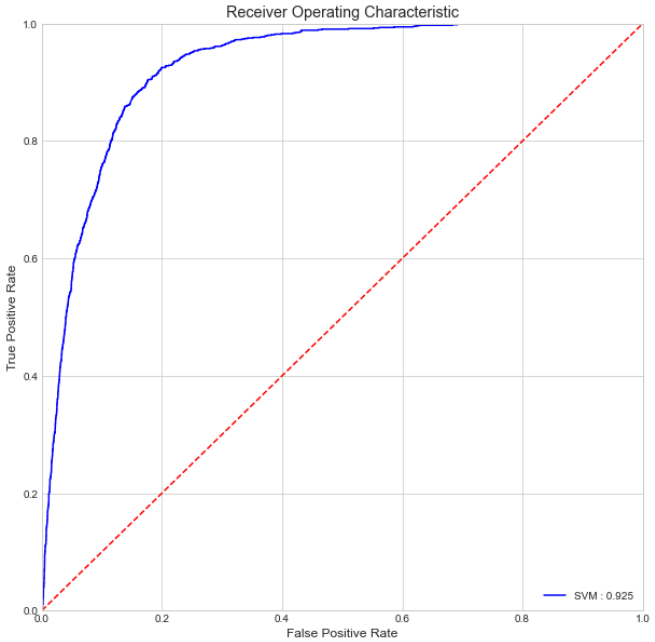
# Support Vector Machine

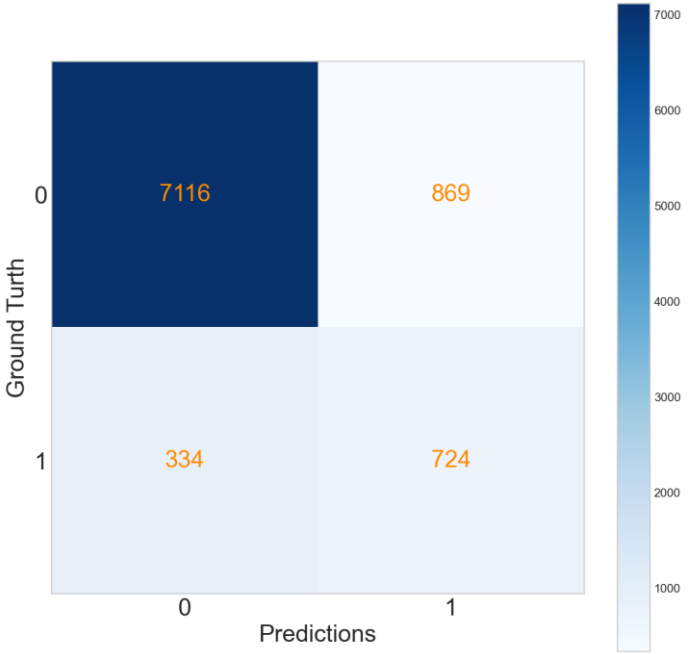
**A svm constructs a hyperplane or set of hyperplanes to split the data of different classes.**



# Support Vector Machine

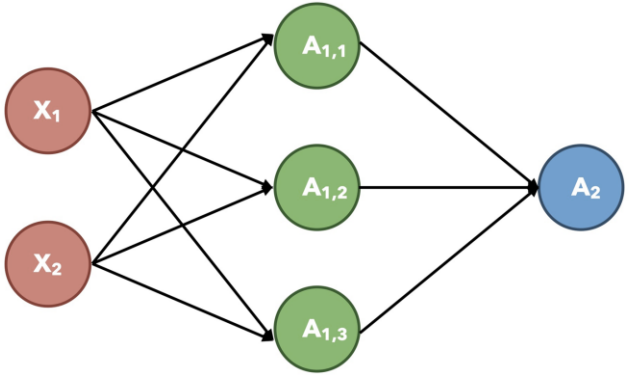


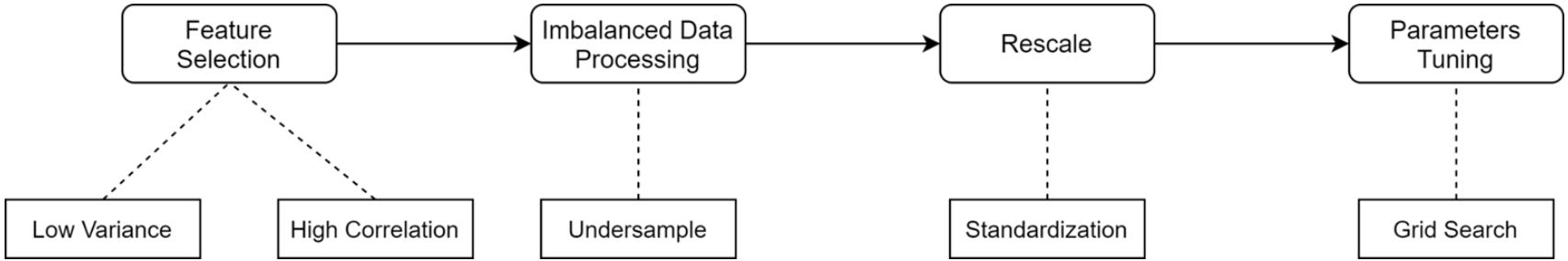




| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| Support Vector Machine | 0.880 | 0.425 | 0.573 | 0.925 | 0.493 | 0.542 |

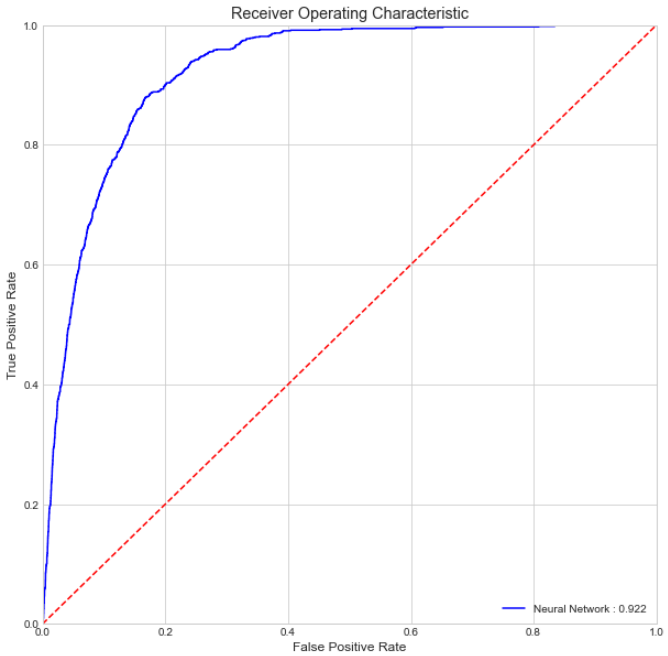
# Artificial Neural Network

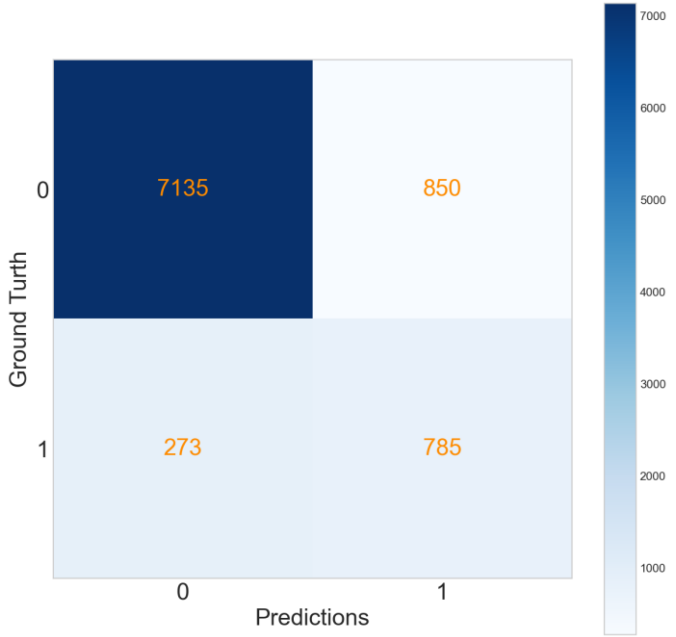
**An artificial neuron network is a computational model that mimics the way nerve cells work in the human brain.**



# Artificial Neural Network

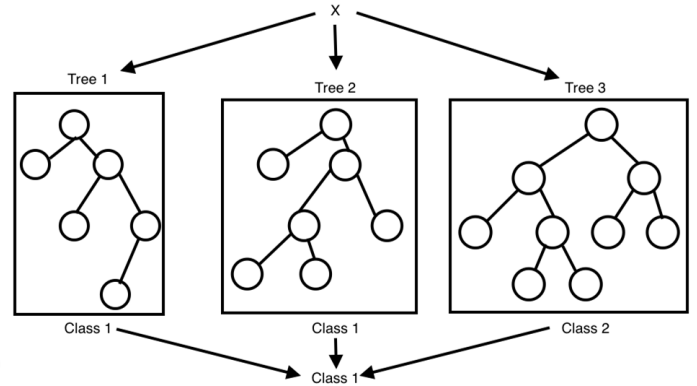


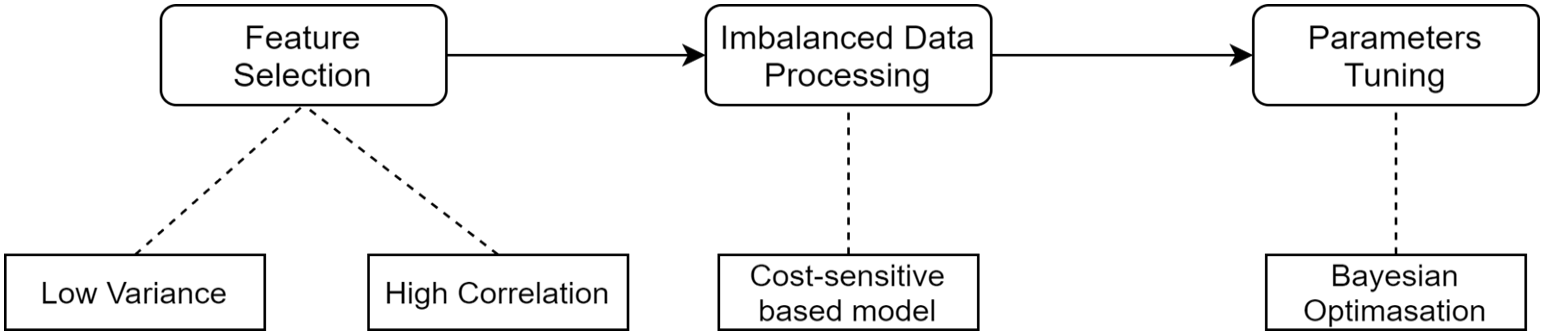




| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| Artificial Neural Network | 0.767 | 0.475 | 0.587 | 0.922 | 0.517 | 0.538 |

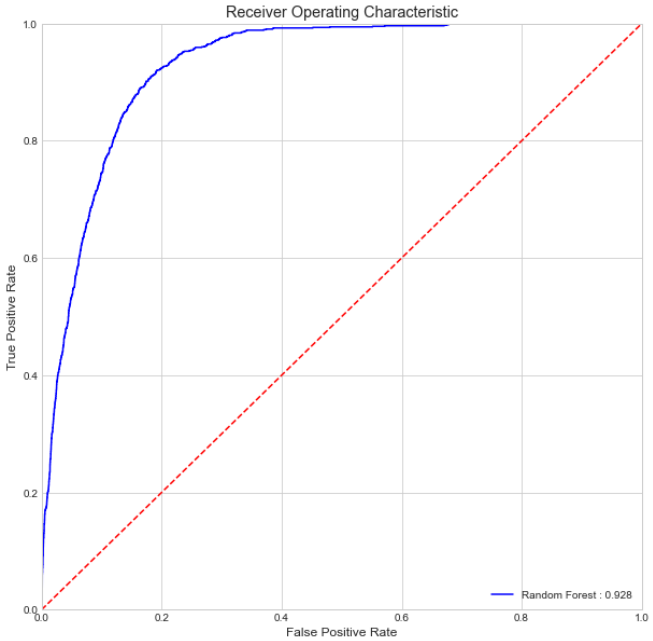
# Random Forest

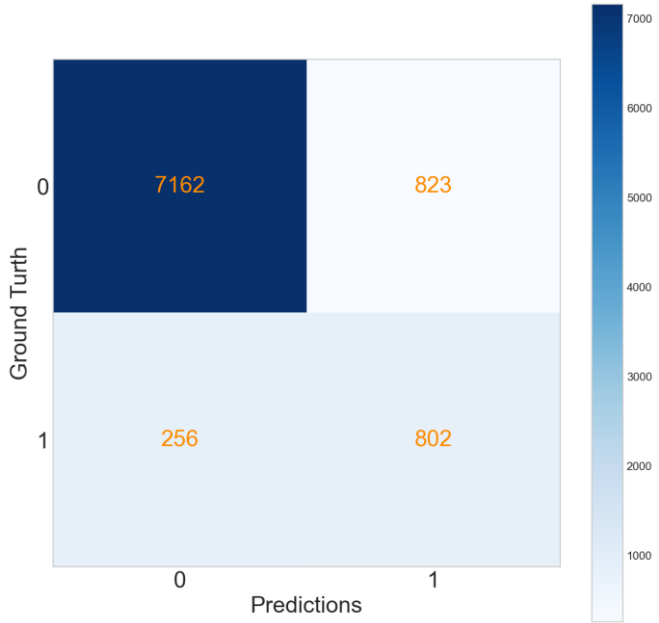
**Random forest is an ensemble learning algorithm which is based on decision tree classifiers on various subset of dataset samples and features to make predictions by majority voting among sub trees.**



# Random Forest

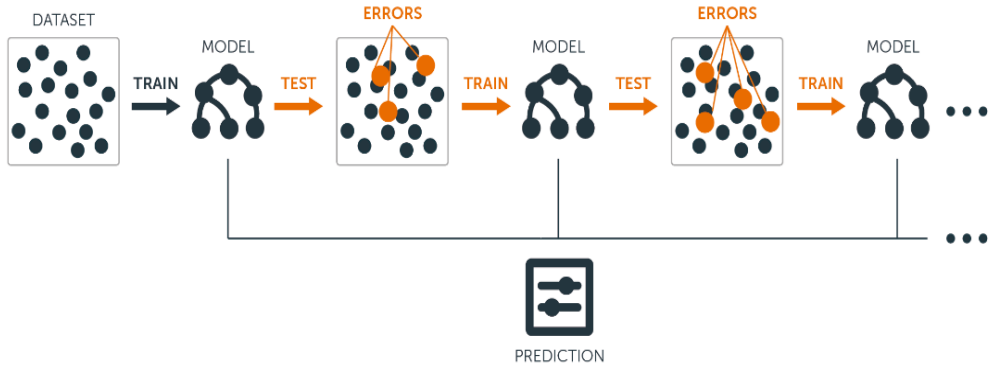


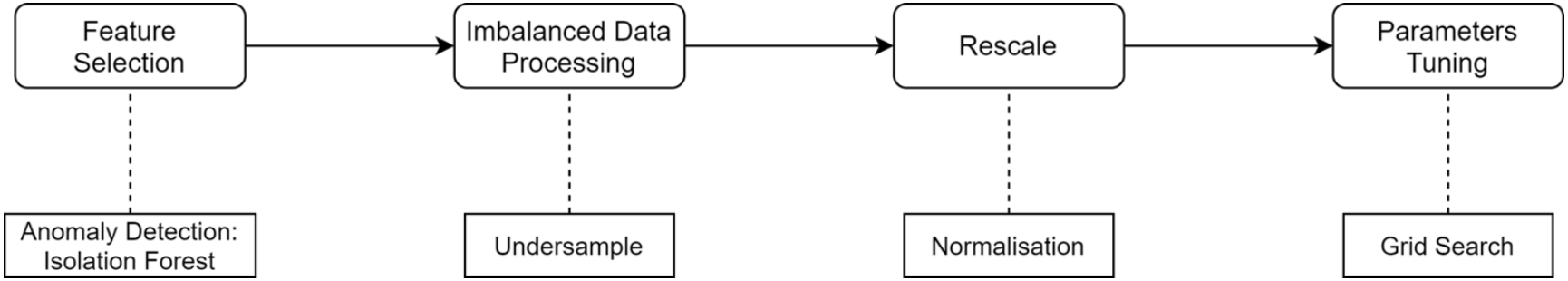




| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| Random Forest | 0.758 | 0.494 | 0.598 | 0.928 | 0.531 | 0.548 |

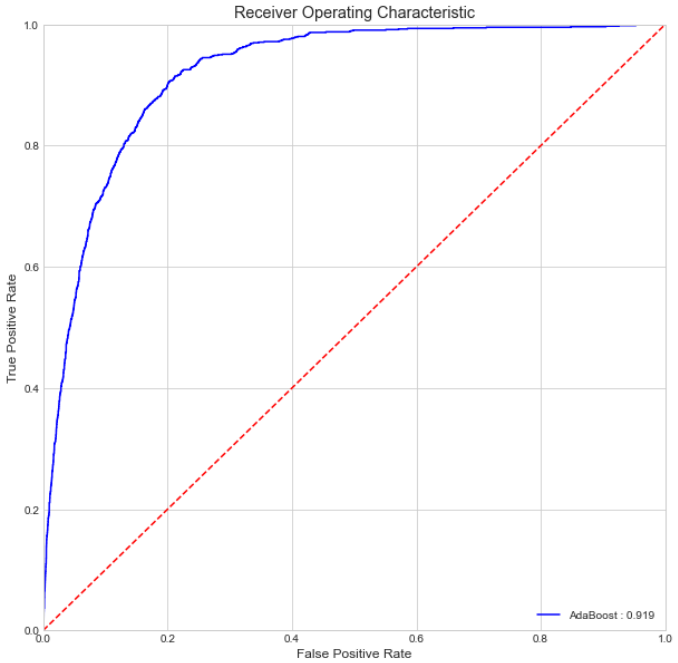
# Adaboost

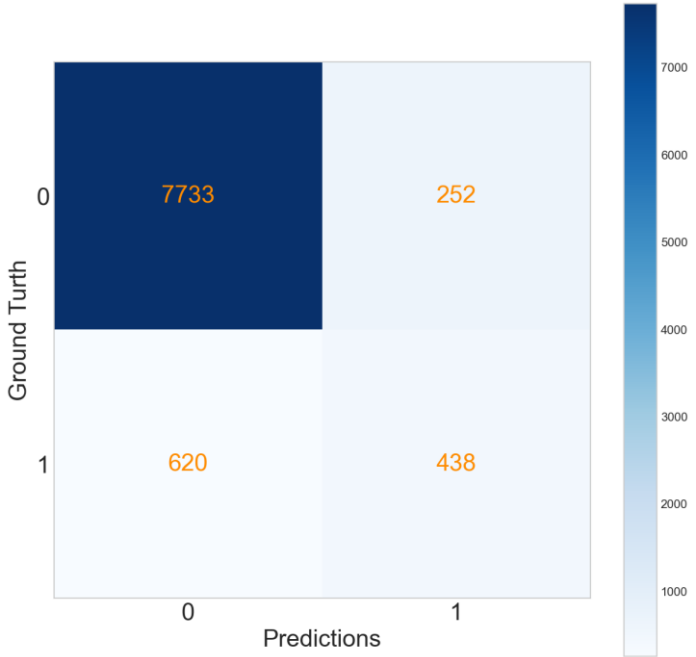
**AdaBoost is a type of algorithm that uses an ensemble learning approach to weight various inputs and attempts to create a strong classifier from a number of weak classifiers.**



# Adaboost

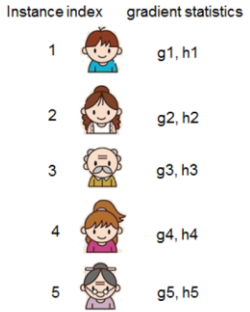


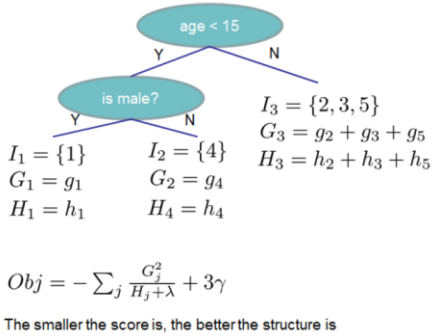



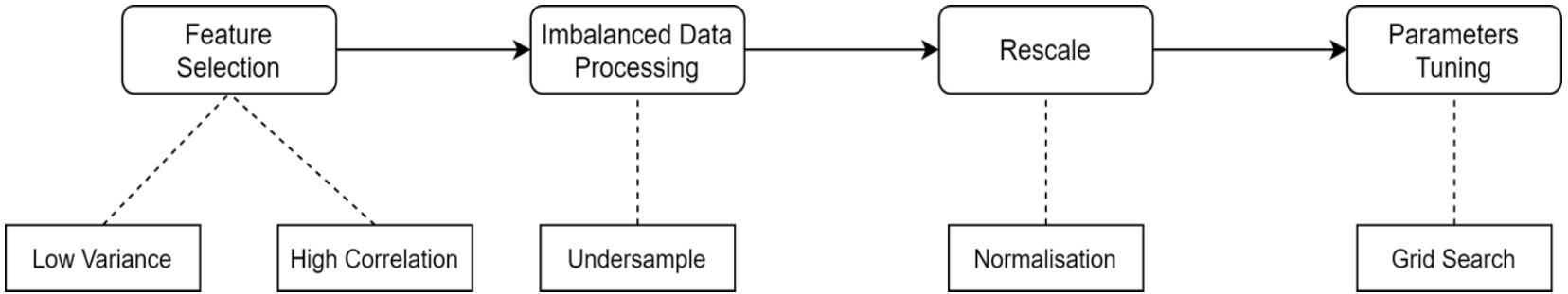

| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| Adaboost | 0.414 | 0.635 | 0.501 | 0.919 | 0.450 | 0.463 |

# Xgboost

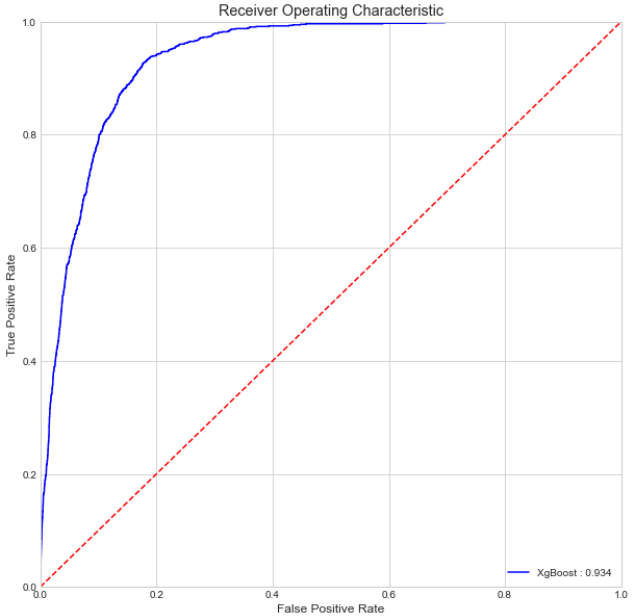
**XGBoost implements Machine Learning algorithms under the Gradient Boosting framework. It provides a parallel tree boosting to solve many data science problems in a fast and accurate way.**

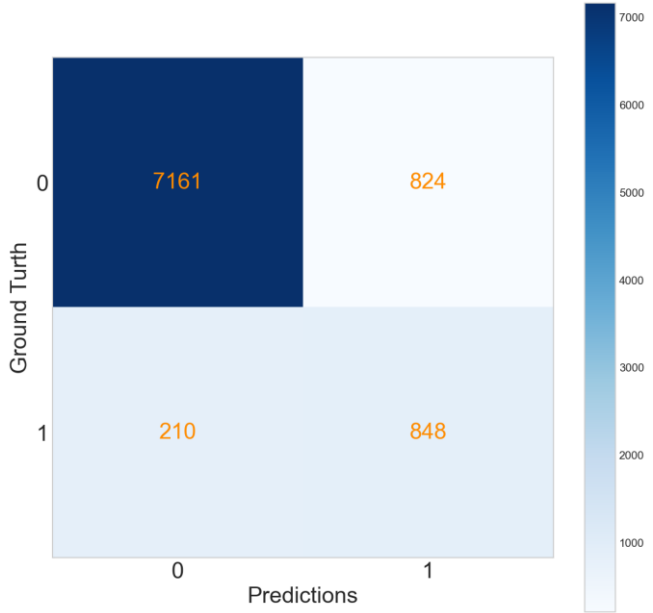


# Xgboost




| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| LightGBM | 0.808 | 0.508 | 0.624 | 0.935 | 0.560 | 0.582 |

# LightGBM

**Algorithm 1:** Histogram-based Algorithm

**Input**: $I$: training data, $d$: max depth
**Input**: $m$: feature dimension
$nodeSet \leftarrow \{0\}$ ▷ tree nodes in current level
$rowSet \leftarrow \{\{0, 1, 2, ...\}\}$ ▷ data indices in tree nodes
**for** $i = 1$ ***to*** $d$ **do**
  **for** *node* ***in*** *nodeSet* **do**
    usedRows $\leftarrow rowSet[node]$
    **for** $k = 1$ ***to*** $m$ **do**
      $H \leftarrow$ new Histogram()
      ▷ Build histogram
      **for** $j$ ***in*** *usedRows* **do**
        bin $\leftarrow I$.f[k][j].bin
        $H$[bin].y $\leftarrow H$[bin].y + I.y[j]
        $H$[bin].n $\leftarrow H$[bin].n + 1
      Find the best split on histogram $H$.
      ...
  Update $rowSet$ and $nodeSet$ according to the best split points.
  ...

**Algorithm 2:** Gradient-based One-Side Sampling

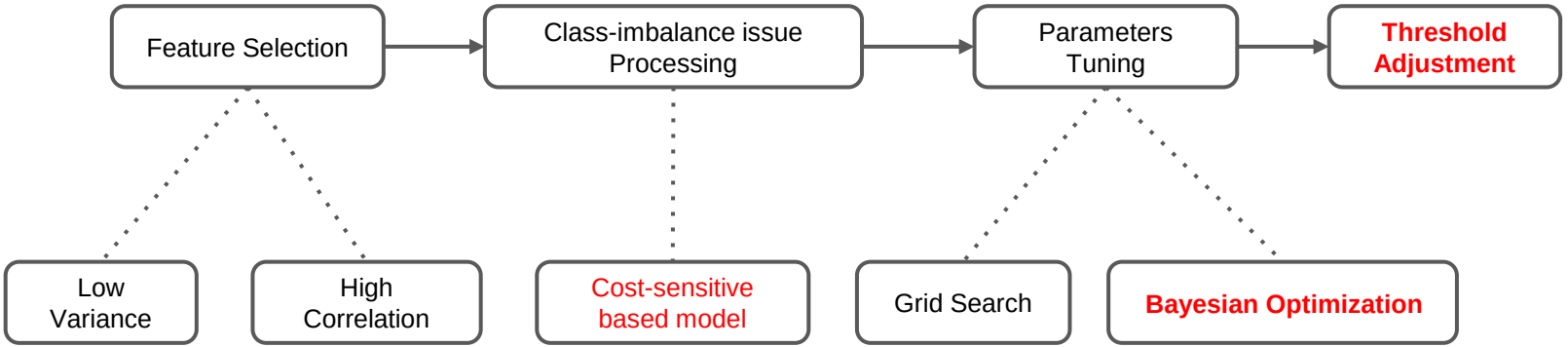
**Input**: $I$: training data, $d$: iterations
**Input**: $a$: sampling ratio of large gradient data
**Input**: $b$: sampling ratio of small gradient data
**Input**: $loss$: loss function, $L$: weak learner
models $\leftarrow \{\}$, fact $\leftarrow \frac{1-a}{b}$
topN $\leftarrow$ a $\times$ len($I$) , randN $\leftarrow$ b $\times$ len($I$)
**for** $i = 1$ ***to*** $d$ **do**
  preds $\leftarrow$ models.predict($I$)
  g $\leftarrow loss(I,$ preds), w $\leftarrow \{1,1,...\}$
  sorted $\leftarrow$ GetSortedIndices(abs(g))
  topSet $\leftarrow$ sorted[1:topN]
  randSet $\leftarrow$ RandomPick(sorted[topN:len(I)], randN)
  usedSet $\leftarrow$ topSet + randSet
  w[randSet] $\times =$ fact ▷ Assign weight $fact$ to the small gradient data.
  newModel $\leftarrow$ L($I$[usedSet], $-$ g[usedSet], w[usedSet])
  models.append(newModel)

**Algorithm 3:** Greedy Bundling

**Input**: $F$: features, $K$: max conflict count
Construct graph $G$
searchOrder $\leftarrow G$.sortByDegree()
bundles $\leftarrow \{\}$, bundlesConflict $\leftarrow \{\}$
**for** $i$ ***in*** *searchOrder* **do**
  needNew $\leftarrow$ True
  **for** $j = 1$ ***to*** *len(bundles)* **do**
    cnt $\leftarrow$ ConflictCnt(bundles[j],$F$[i])
    **if** *cnt + bundlesConflict[i]* $\leq K$ **then**
      bundles[j].add($F$[i]), needNew $\leftarrow$ False
      break
  **if** *needNew* **then**
    Add $F[i]$ as a new bundle to *bundles*
**Output**: *bundles*

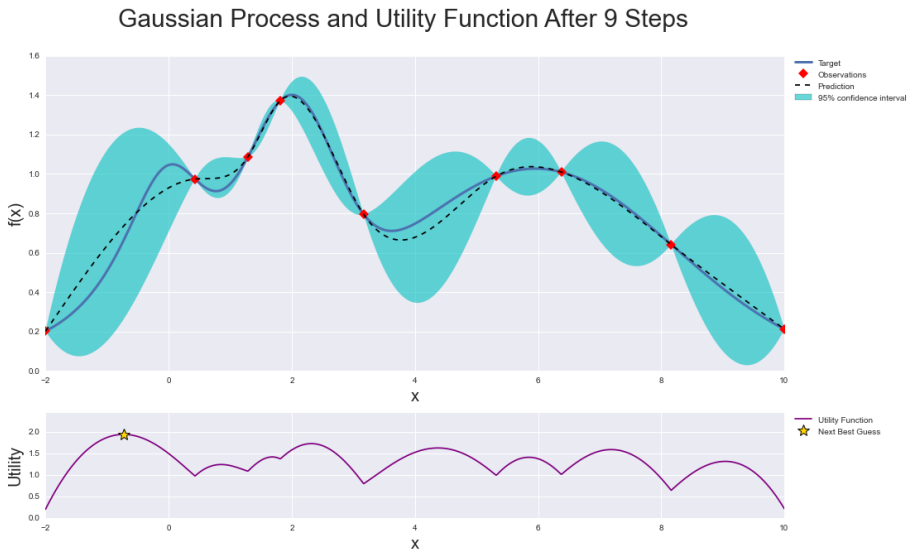
**Algorithm 4:** Merge Exclusive Features

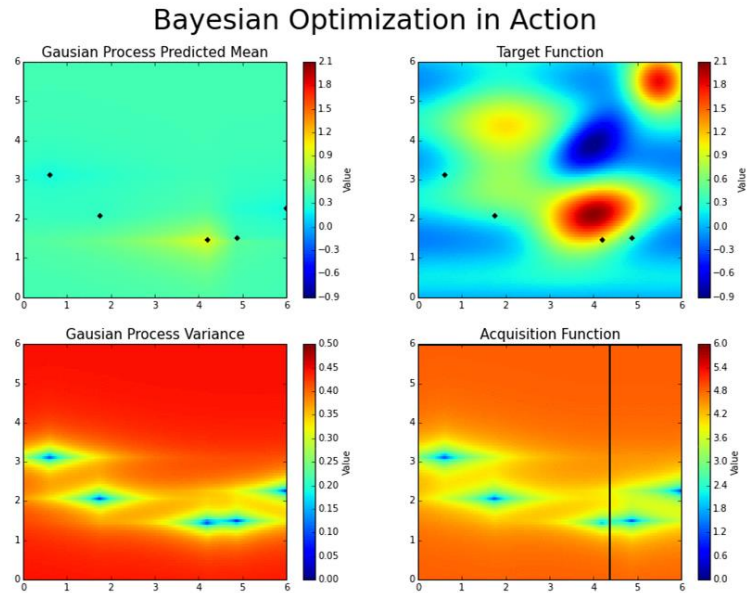
**Input**: $numData$: number of data
**Input**: $F$: One bundle of exclusive features
binRanges $\leftarrow \{0\}$, totalBin $\leftarrow 0$
**for** $f$ ***in*** $F$ **do**
  totalBin += f.numBin
  binRanges.append(totalBin)
newBin $\leftarrow$ new Bin(numData)
**for** $i = 1$ ***to*** $numData$ **do**
  newBin[i] $\leftarrow 0$
  **for** $j = 1$ ***to*** *len*($F$) **do**
    **if** $F$*[j].bin[i]* $\neq 0$ **then**
      newBin[i] $\leftarrow F$[j].bin[i] + binRanges[j]
**Output**: $newBin$, $binRanges$



# Hyperparameter Tuning

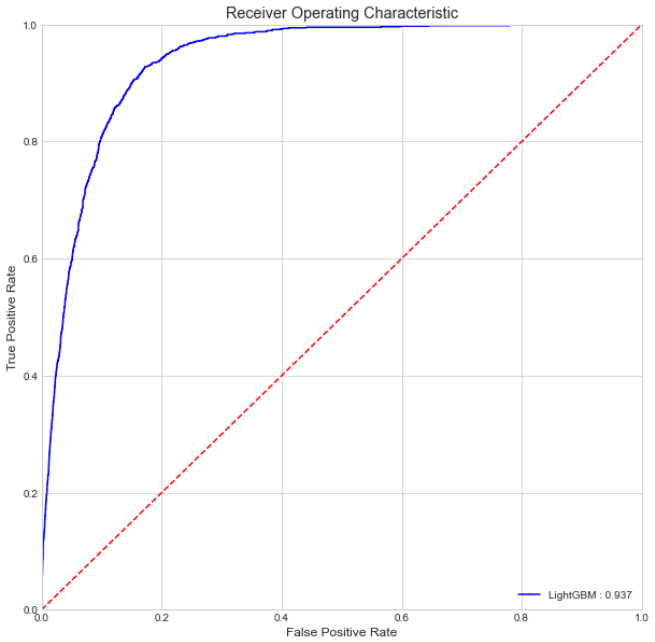
- **Bayesian Optimization**

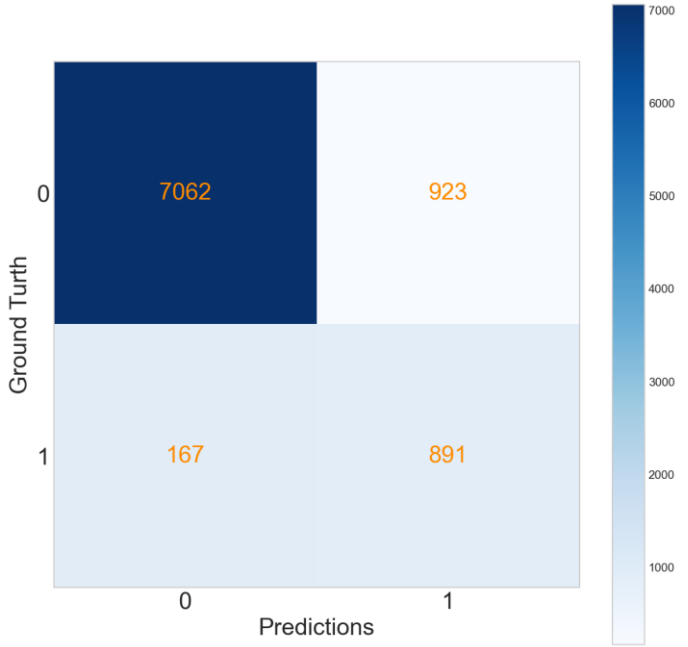



Bayesian optimization works by constructing a posterior distribution of functions (gaussian process) that best describes the function you want to optimize. As the number of observations grows, the posterior distribution improves, and the algorithm becomes more certain of which regions in parameter space are worth exploring and which are not, as seen in the picture above.

# LightGBM




| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| LightGBM | 0.832 | 0.508 | 0.631 | 0.936 | 0.568 | **0.592** |

# CatBoost

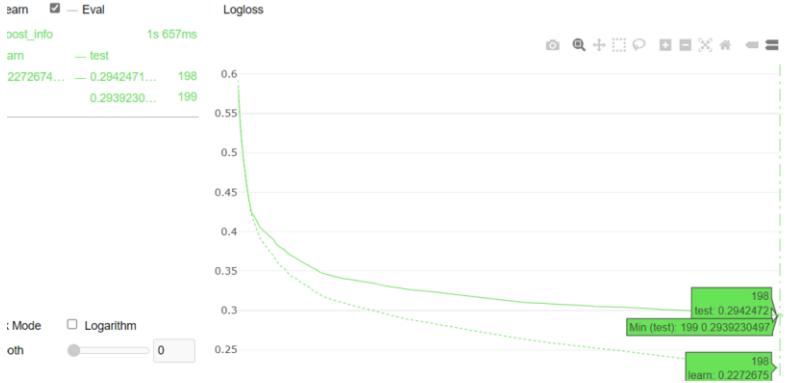
- An innovative algorithm for automatically processing categorical features into numerical features is embedded
- Combined category features are used to take advantage of the connection between features
- Avoided the deviation of gradient estimation and solved the problem of predictive offset

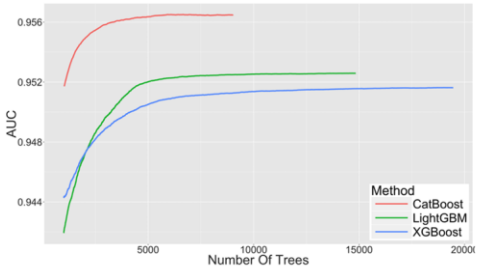
**CatBoost: gradient boosting with categorical features support**

**Anna Veronika Dorogush, Vasily Ershov, Andrey Gulin**
Yandex

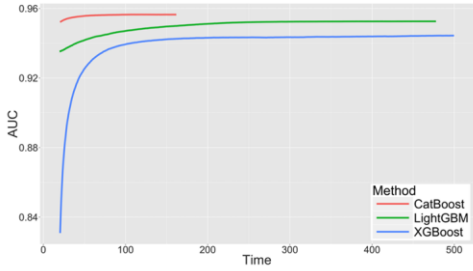
**Abstract**

In this paper we present CatBoost, a new open-sourced gradient boosting library that successfully handles categorical features and outperforms existing publicly available implementations of gradient boosting in terms of quality on a set of popular publicly available datasets. The library has a GPU implementation of learning algorithm and a CPU implementation of scoring algorithm, which are significantly faster than other gradient boosting libraries on ensembles of similar sizes.



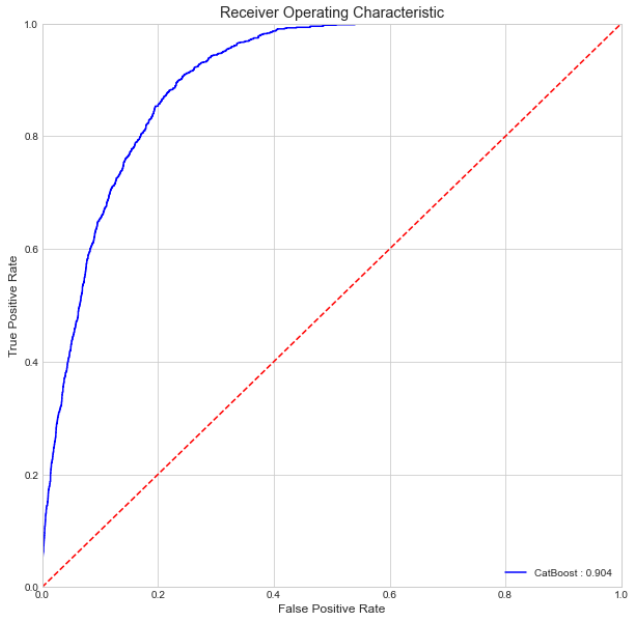



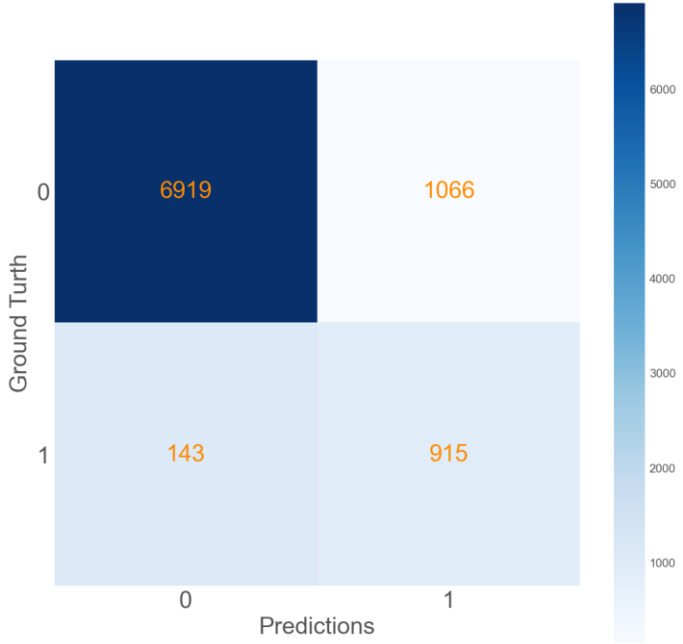

(a) AUC vs Number of trees




(b) AUC vs Time

# CatBoost

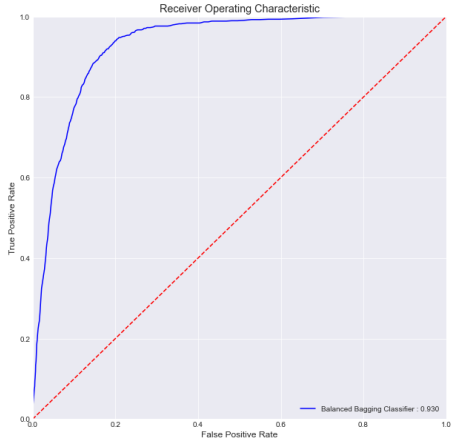


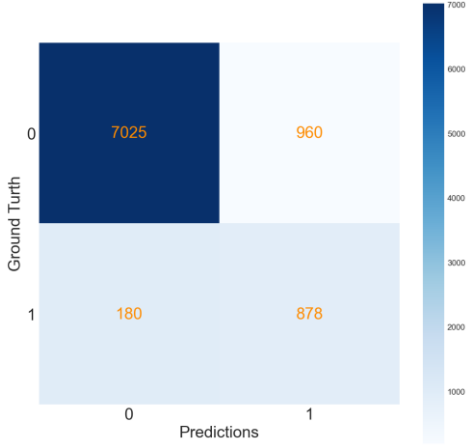




| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| CatBoost | 0.865 | 0.462 | 0.611 | 0.937 | 0.545 | 0.577 |

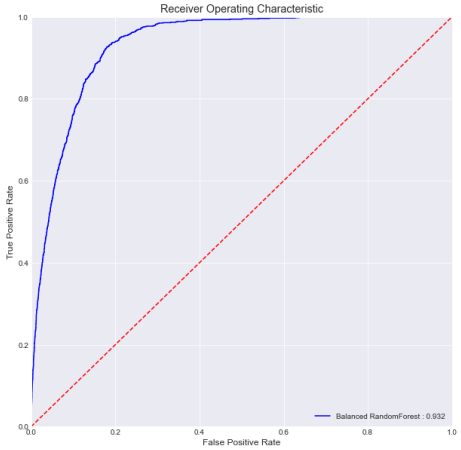
# Easy Ensemble & Voting Classifier

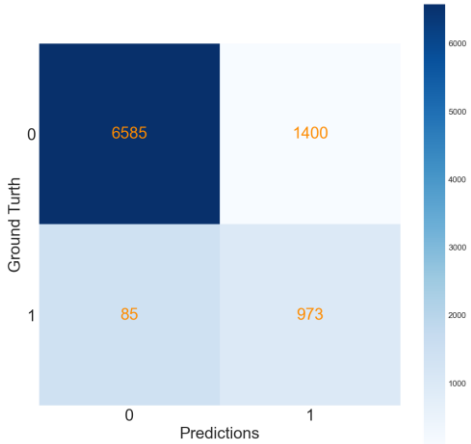
# Balanced Bagging & Balanced RandomForest



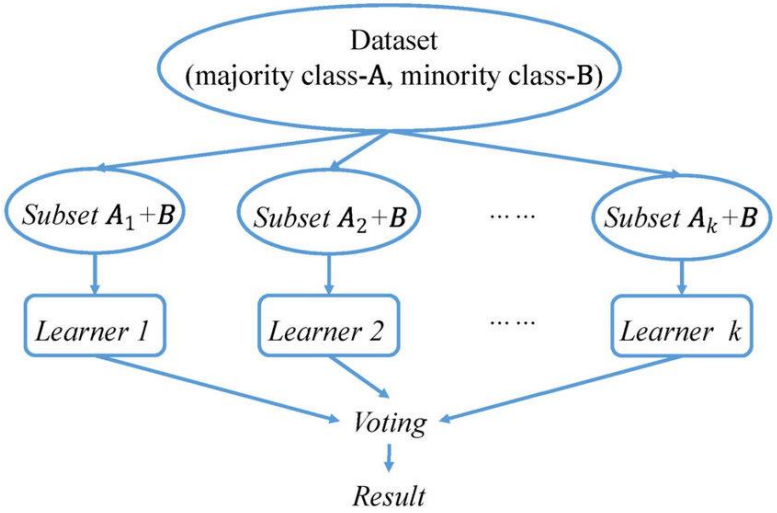

| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| Balanced Bagging | 0.830 | 0.478 | 0.606 | 0.930 | 0.538 | 0.567 |



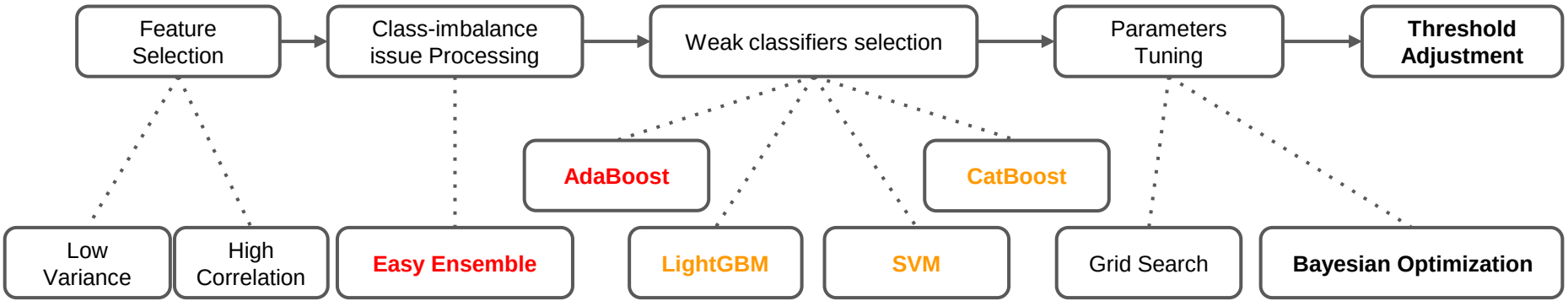

| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| Balanced Random Forest | 0.920 | 0.410 | 0.567 | 0.932 | 0.484 | 0.544 |

# Easy Ensemble

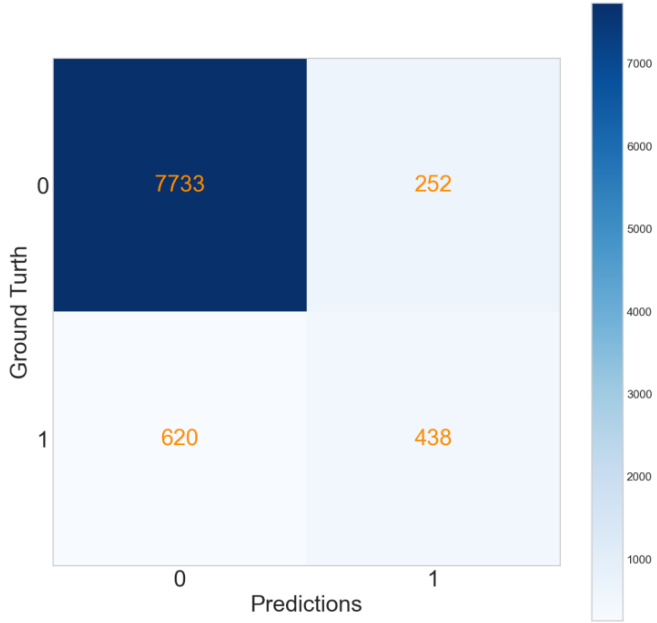


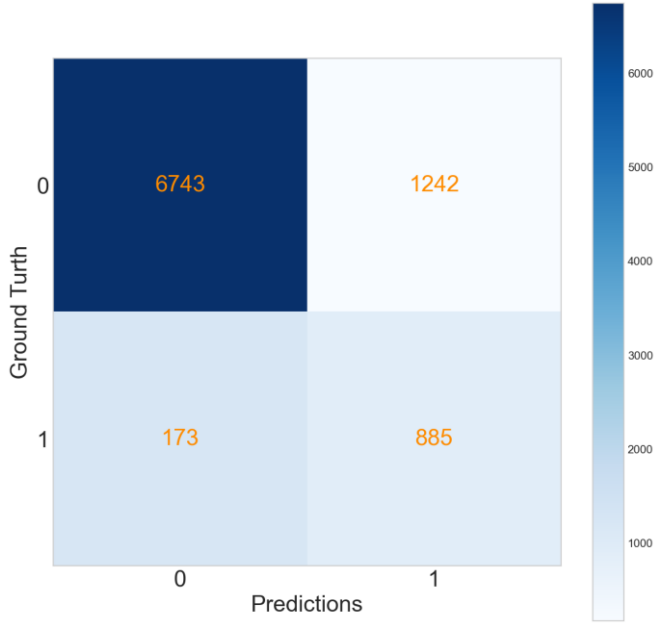

**Algorithm 1** The `EasyEnsemble` algorithm.

1: {Input: A set of minor class examples $\mathcal{P}$, a set of major class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, and $T$, the number of subsets to be sampled from $\mathcal{N}$.}
2: $i \leftarrow 0$
3: **repeat**
4: $\quad i \leftarrow i + 1$
5: $\quad$ Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$.
6: $\quad$ Learn $H_i$ using $\mathcal{P}$ and $\mathcal{N}_i$. $H_i$ is an AdaBoost ensemble with weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$, i.e.
$$H_i(\boldsymbol{x}) = \text{sgn}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\boldsymbol{x}) - \theta_i\right).$$
7: **until** $i = T$
8: Output: An ensemble:
$$H(\boldsymbol{x}) = \text{sgn}\left(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(\boldsymbol{x}) - \sum_{i=1}^{T} \theta_i\right).$$



# Easy Ensemble with AdaBoost



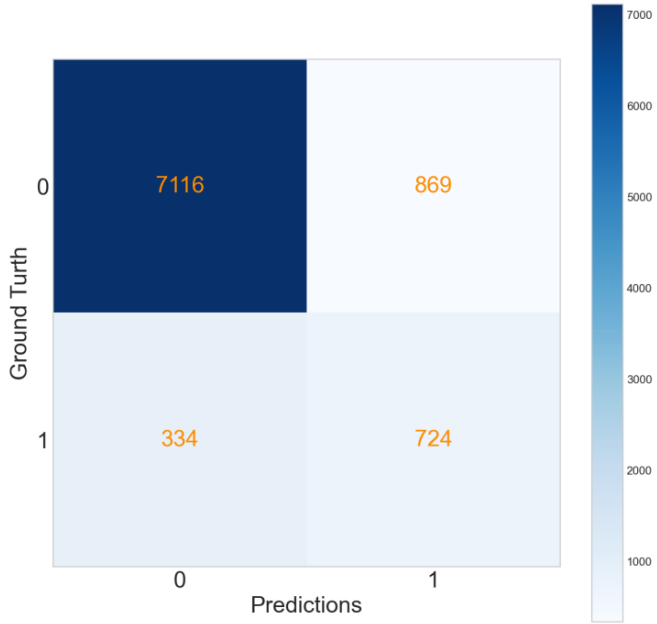

**AdaBoost**

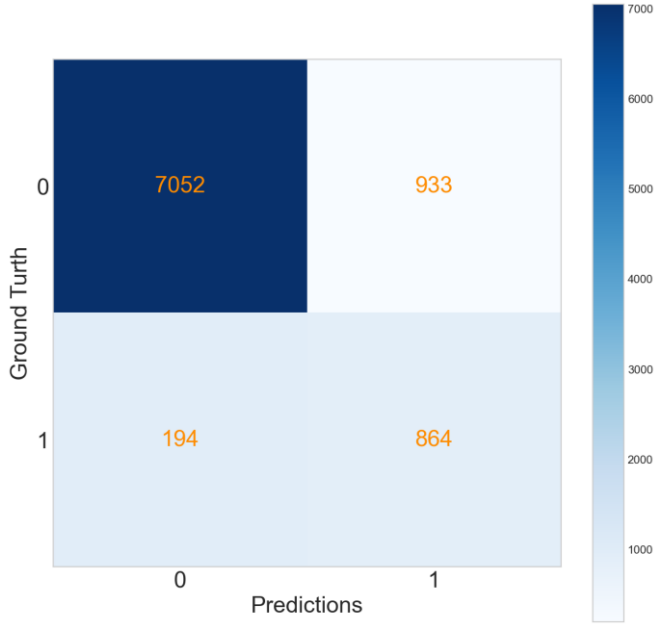
| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| 0.414 | 0.635 | 0.501 | 0.919 | 0.450 | 0.463 |

**Easy Ensemble + AdaBoost**

| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| **0.836** | 0.416 | **0.556** | **0.917** | **0.473** | **0.516** |

# Easy Ensemble with SVM



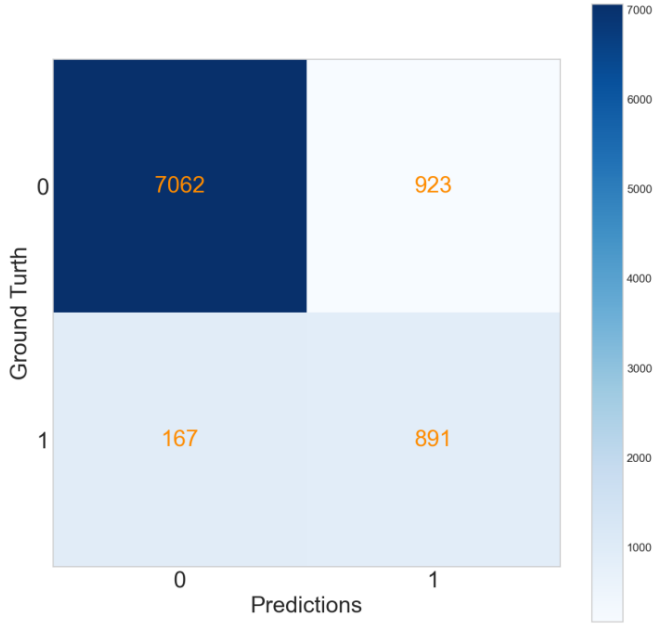

**SVM**

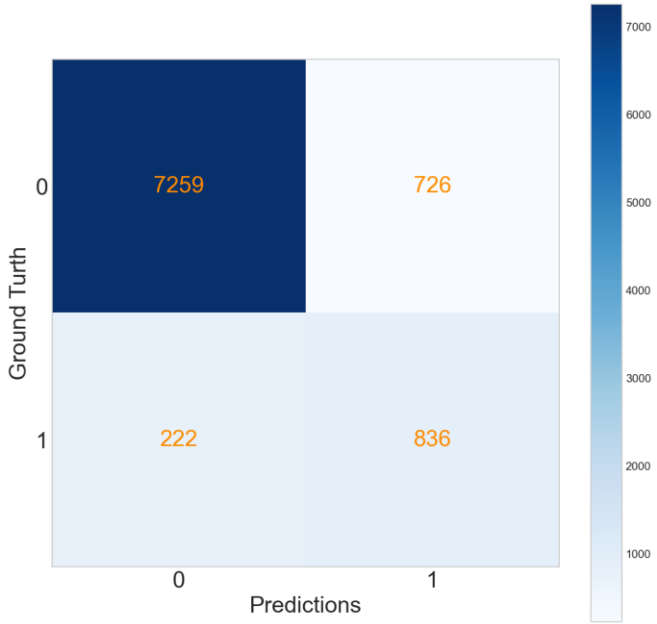
| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| 0.880 | 0.425 | 0.573 | 0.925 | 0.493 | 0.542 |

**Easy Ensemble + SVM**

| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| 0.817 | **0.481** | **0.605** | **0.929** | **0.537** | **0.564** |

# Easy Ensemble with LightGBM



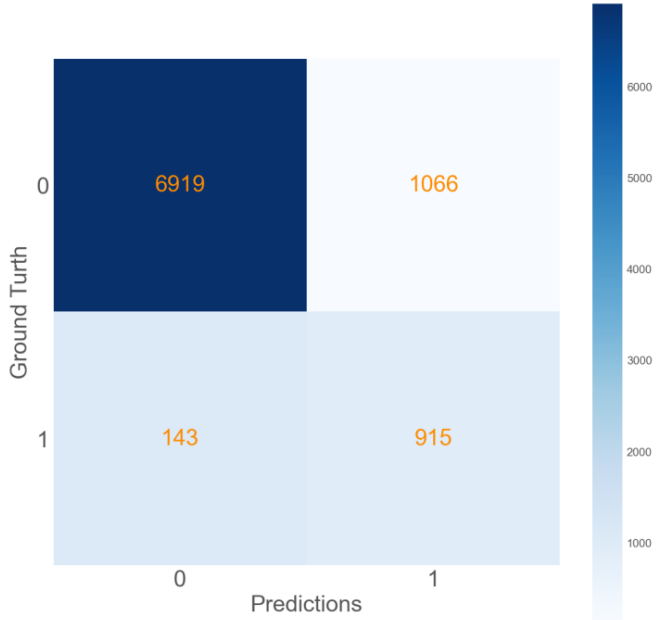

**LightGBM**

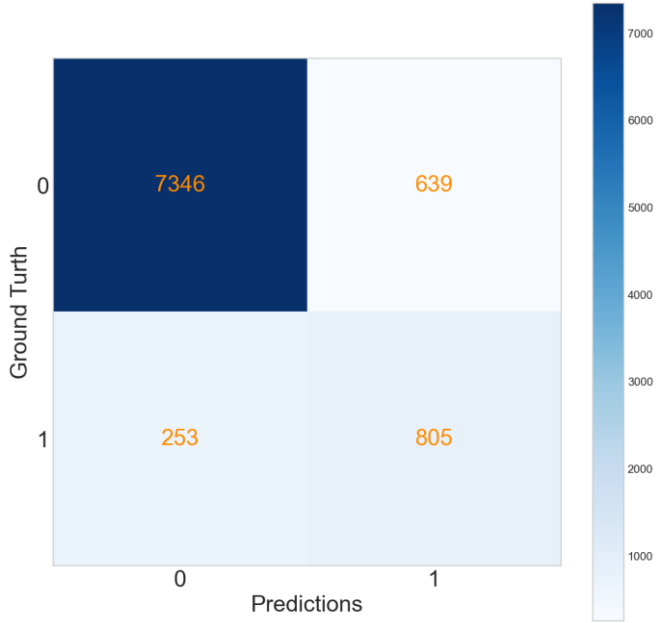
| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| 0.832 | 0.508 | 0.631 | 0.936 | 0.568 | 0.592 |

**Easy Ensemble + LightGBM**

| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| 0.790 | **0.535** | **0.638** | **0.939** | **0.580** | **0.595** |

# Easy Ensemble with CatBoost




**CatBoost**

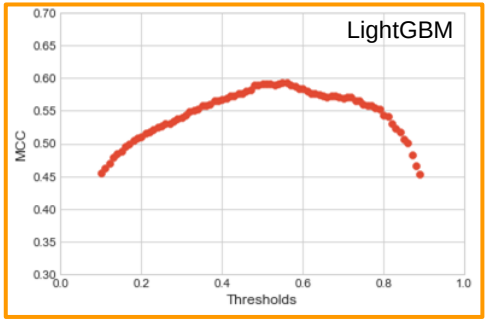
| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| 0.865 | 0.462 | 0.611 | 0.937 | 0.545 | 0.577 |

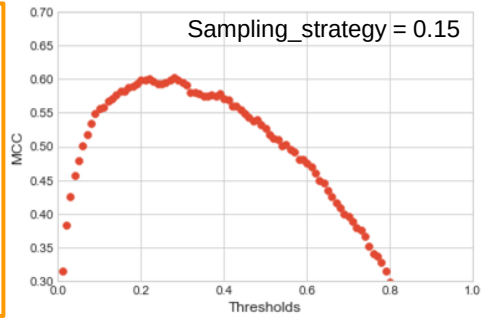
**Easy Ensemble + CatBoost**

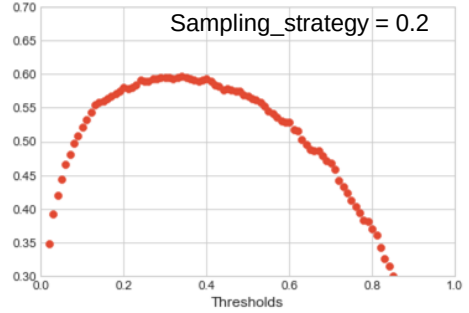
| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| 0.761 | **0.557** | **0.643** | **0.941** | **0.588** | **0.597** |

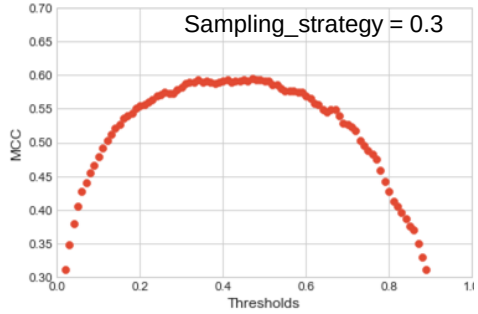
# Sampling strategy

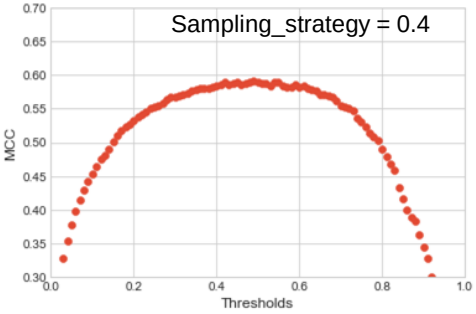
Number of ***B*** / Number of ***Ai***

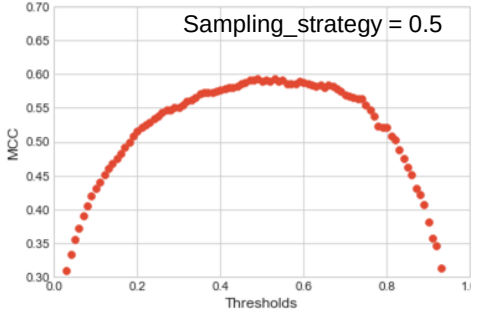


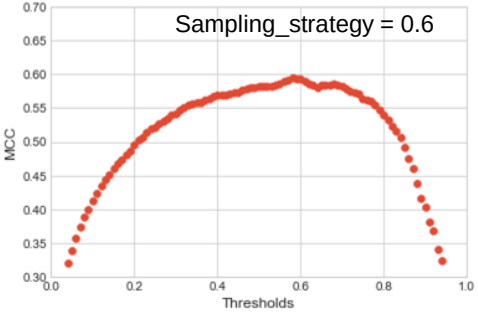



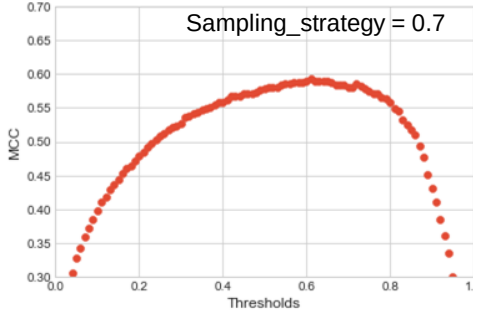



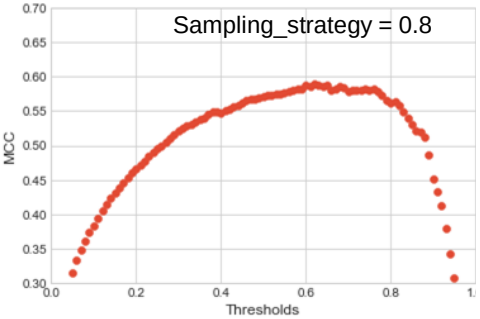



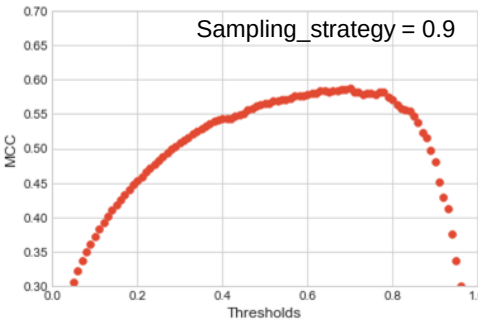



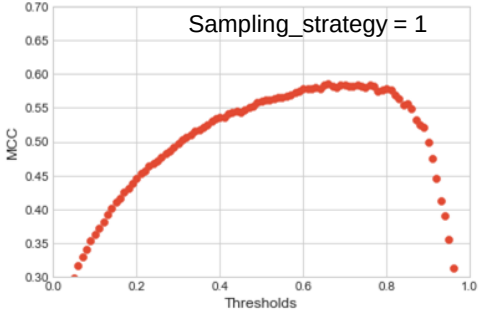



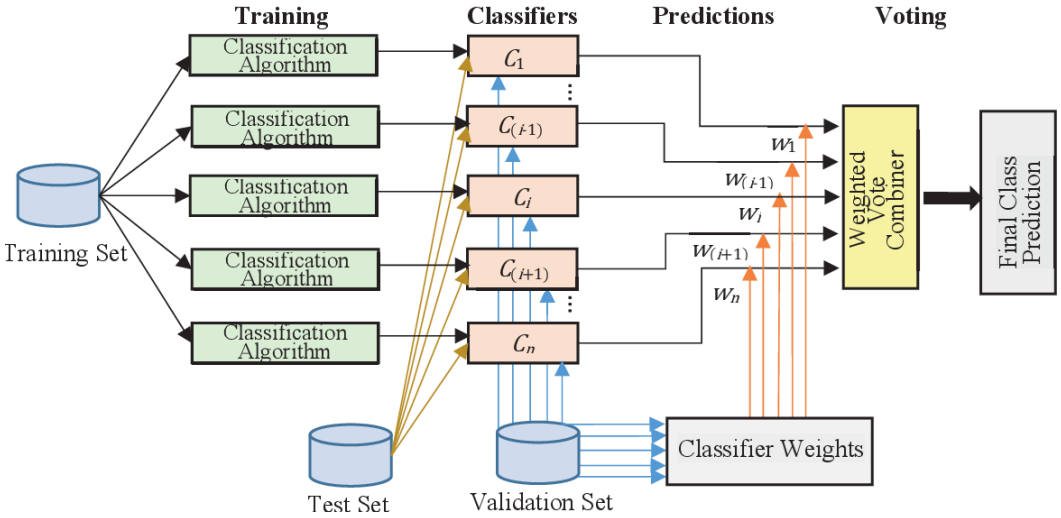



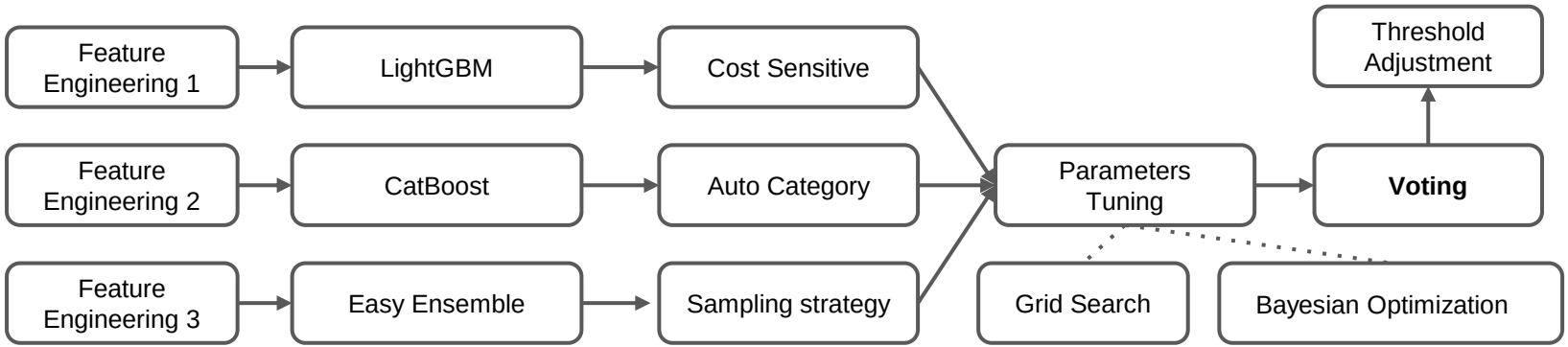



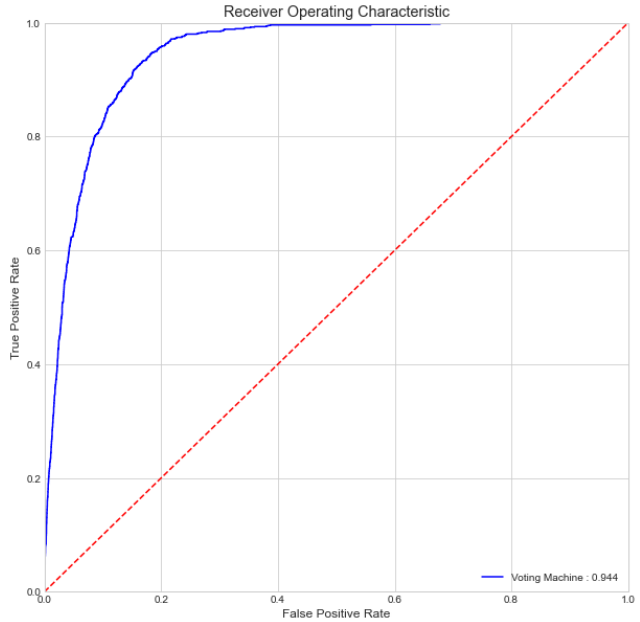



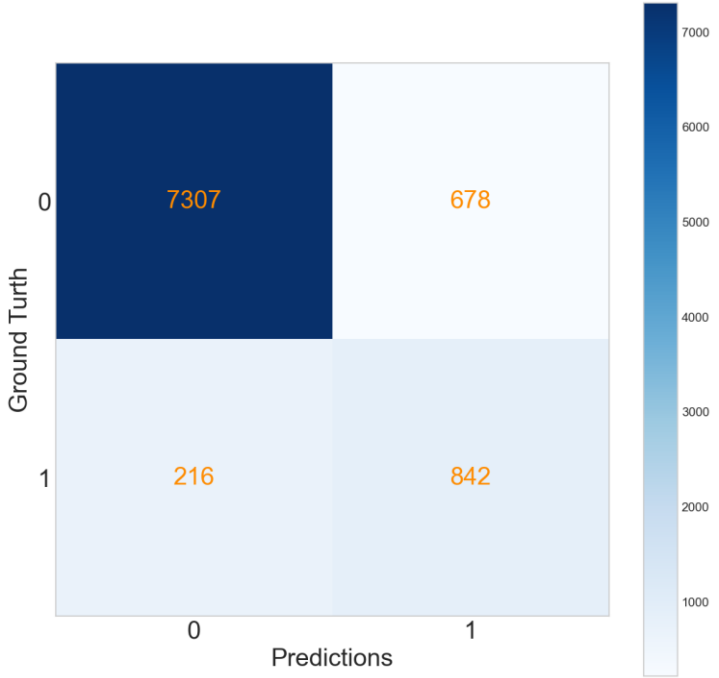
# Voting Machine



# Voting Machine

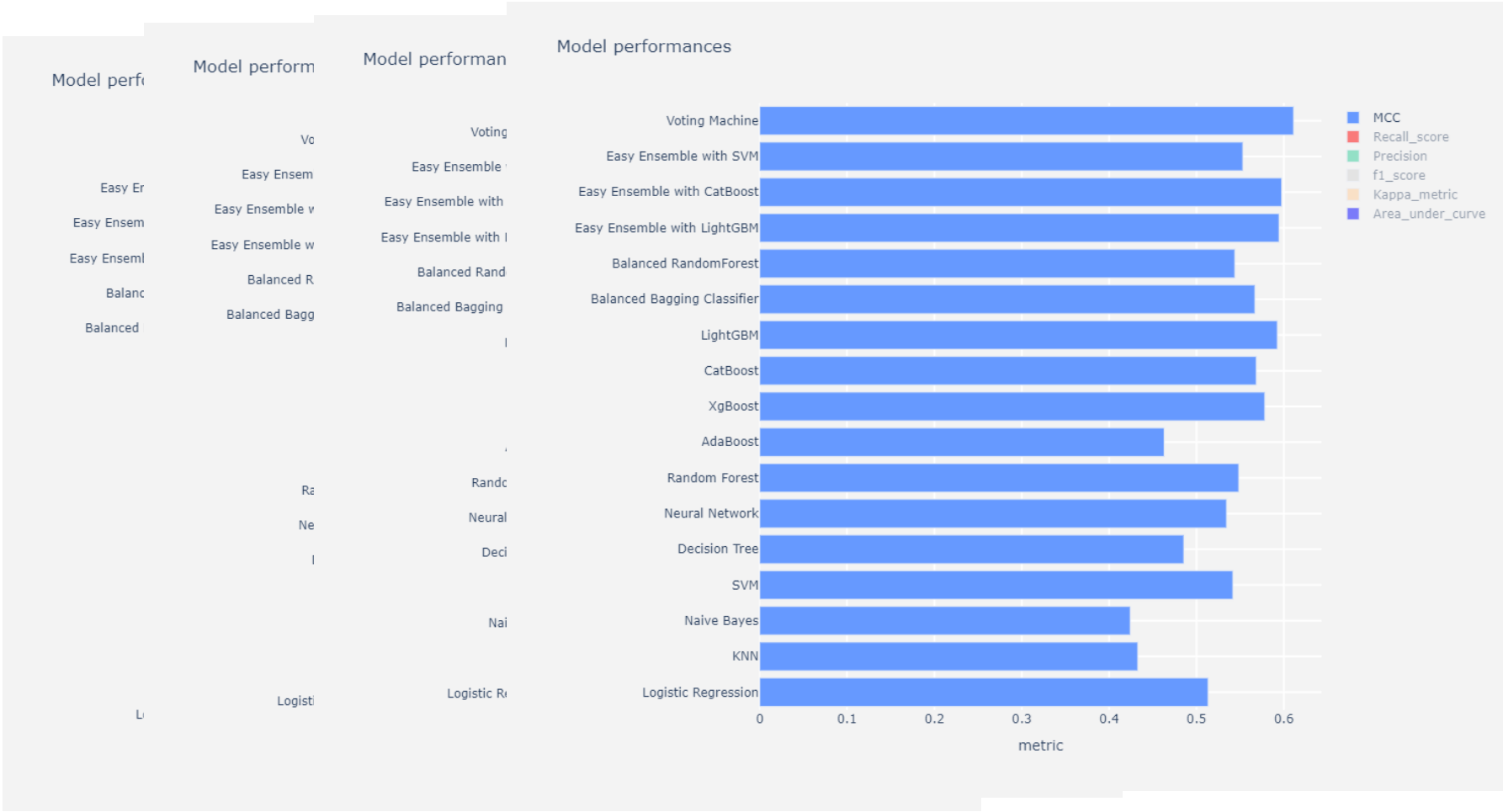


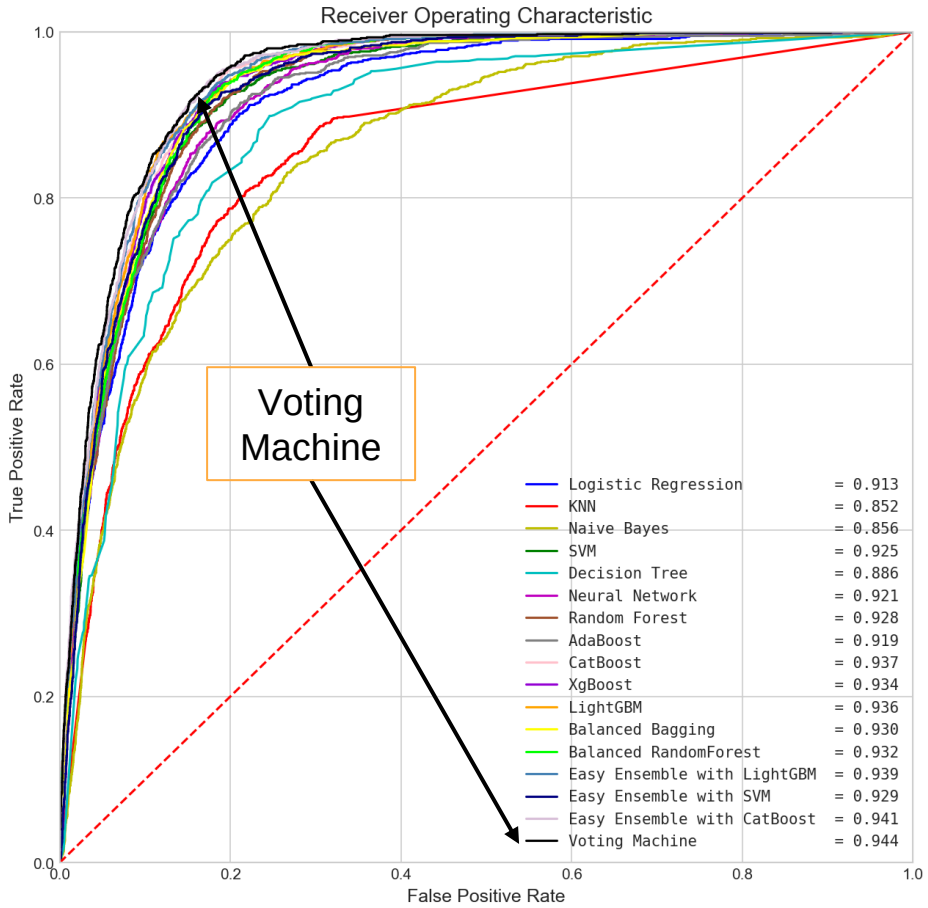

| Model | Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|---|
| CatBoost | 0.796 | 0.554 | 0.653 | 0.944 | 0.598 | **0.611** |

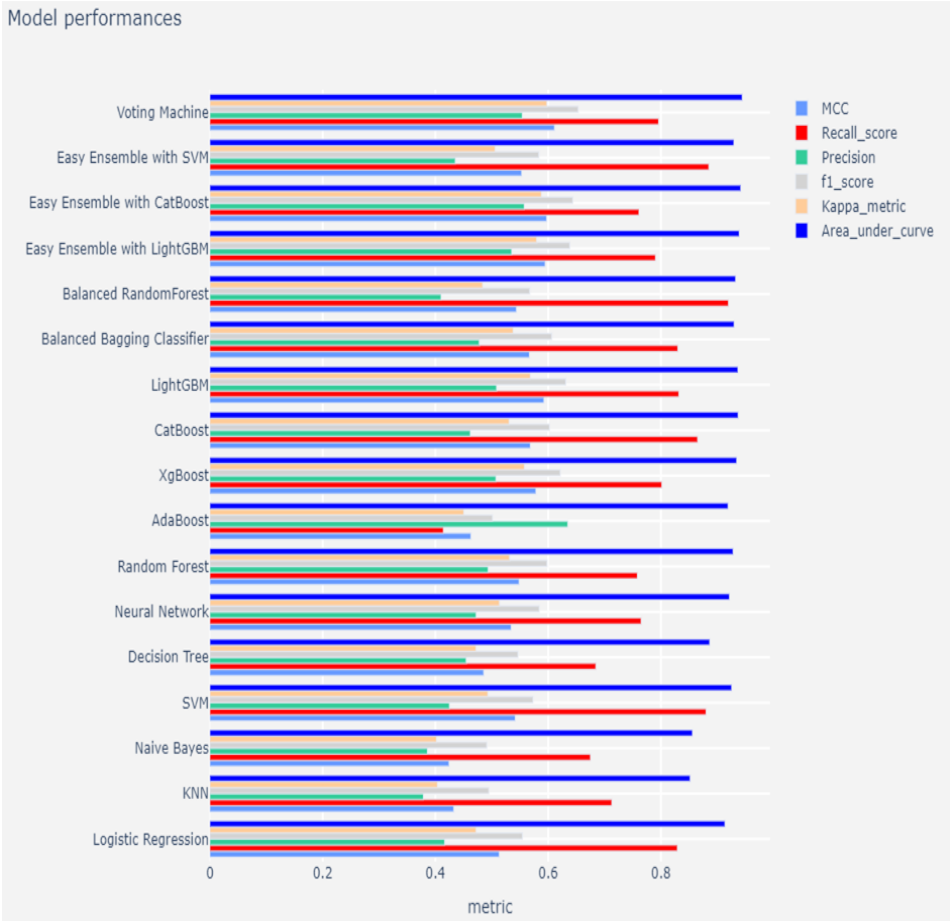
# Comparison & Performance Evaluations

# Model Performance Comparison



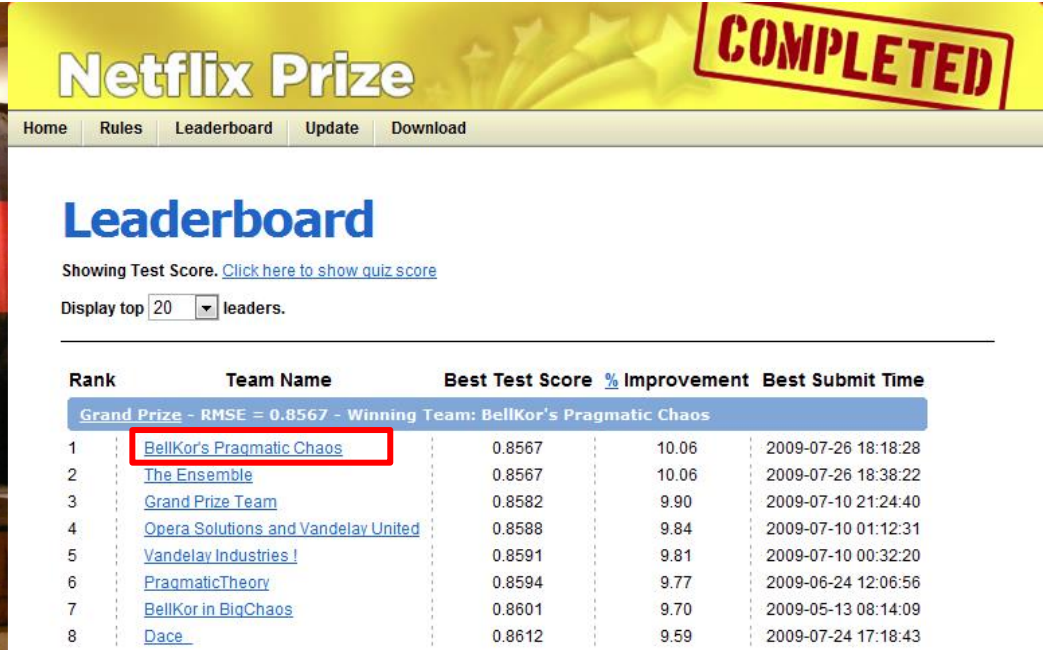
# Model Performance Comparison



# Less is Better

Increased **8.43%** + integrated **107** algorithms + trained **2000** hours + **one million** dollar prize = basically **0** use


## Leaderboard

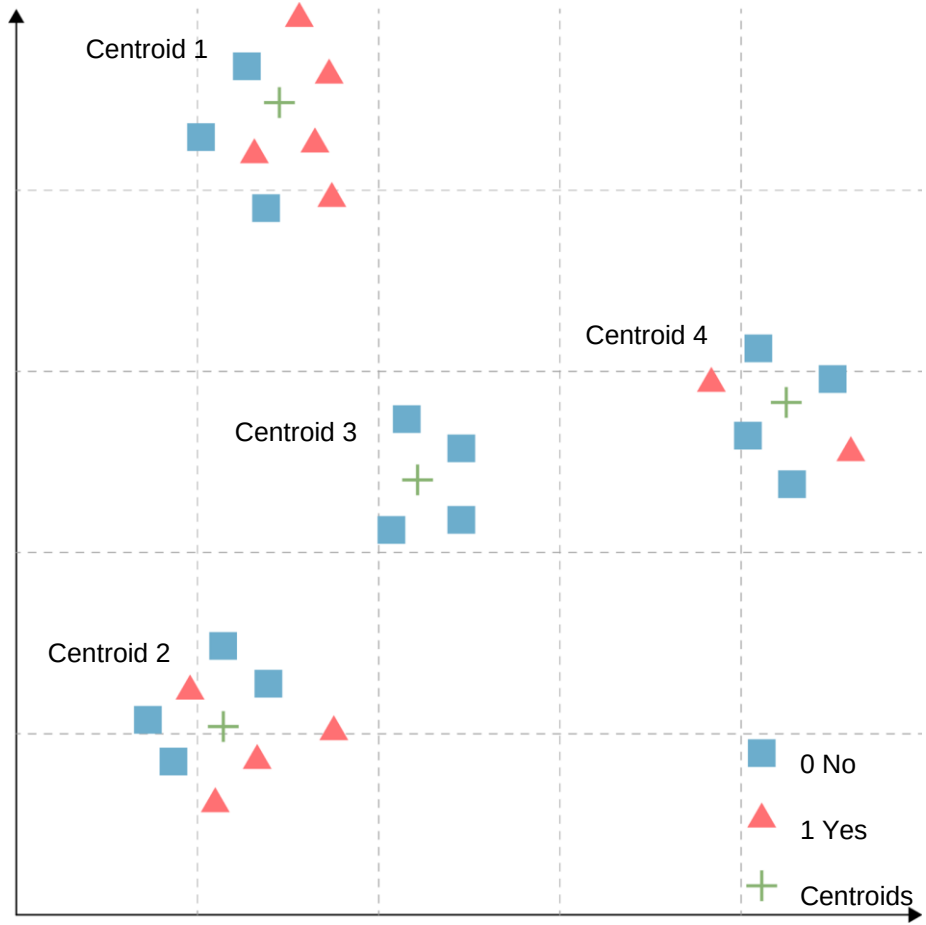
Showing Test Score. Click here to show quiz score

Display top 20 leaders.

| Rank | Team Name | Best Test Score | % Improvement | Best Submit Time |
|---|---|---|---|---|
| Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos | | | | |
| 1 | BellKor's Pragmatic Chaos | 0.8567 | 10.06 | 2009-07-26 18:18:28 |
| 2 | The Ensemble | 0.8567 | 10.06 | 2009-07-26 18:38:22 |
| 3 | Grand Prize Team | 0.8582 | 9.90 | 2009-07-10 21:24:40 |
| 4 | Opera Solutions and Vandelay United | 0.8588 | 9.84 | 2009-07-10 01:12:31 |
| 5 | Vandelay Industries ! | 0.8591 | 9.81 | 2009-07-10 00:32:20 |
| 6 | PragmaticTheory | 0.8594 | 9.77 | 2009-06-24 12:06:56 |
| 7 | BellKor in BigChaos | 0.8601 | 9.70 | 2009-05-13 08:14:09 |
| 8 | Dace | 0.8612 | 9.59 | 2009-07-24 17:18:43 |

# Experiments with Clustering Method

# [Experiment] Customised Clustering Classification




Map each centroid to a class label based on the ratio between two classes

**Centroid 1:**

**Number of No: 3 Number of Yes: 5**

Class label mapping: **{1} : 1**

Probability of Yes mapping: **{1} : 0.625**

**Centroid 2:**

**Number of No: 4 Number of Yes: 4**

Class label: **{2} : 1** (based on the experiments)

Probability of Yes mapping: **{2} : 0.5**

**Centroid 3:**

**Number of No: 4 Number of Yes: 0**

Class label: **{3} : 0**

Probability of Yes mapping: **{3} : 0**

**Centroid 4:**

**Number of No: 4 Number of Yes: 2**

Class label: **{4} : 0**

Probability of Yes mapping: **{4} : 0.333**

# [Experiment] Customised Clustering Classification


**Workflow**

Choose K

↓

Data Resampling

↓

Centroids Finding

↓

Mapping Creation

↓

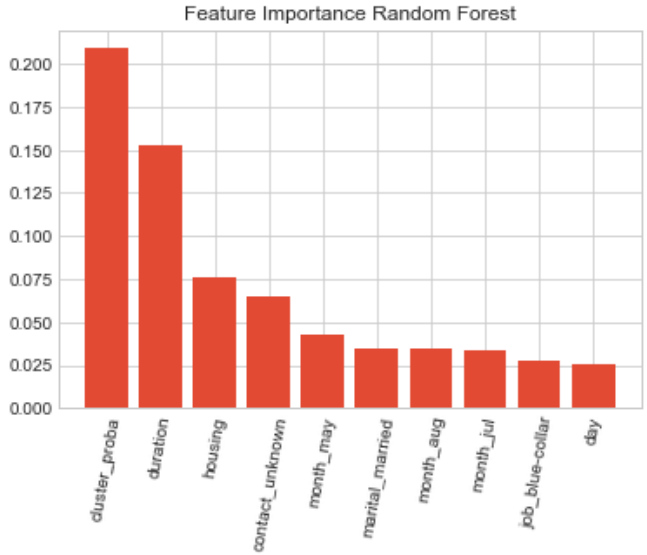
Cluster and map test set based on the centroids

# [Implication] Clustering Feature Engineering

Train Set

| age | balance | ... | cluster_label | cluster_proba |
|---|---|---|---|---|
| 58 | 0 | ... | 1 | 0.852 |
| 44 | 0 | ... | 0 | 0.225 |
| 33 | 0 | ... | 0 | 0.309 |
| 47 | 0 | ... | 0 | 0.429 |
| ... | ... | ... | ... | ... |
| 57 | 1 | ... | 1 | 0.756 |
| 37 | 1 | ... | 0 | 0.445 |

**Engineered New Features**




Random Forest with SMOTE

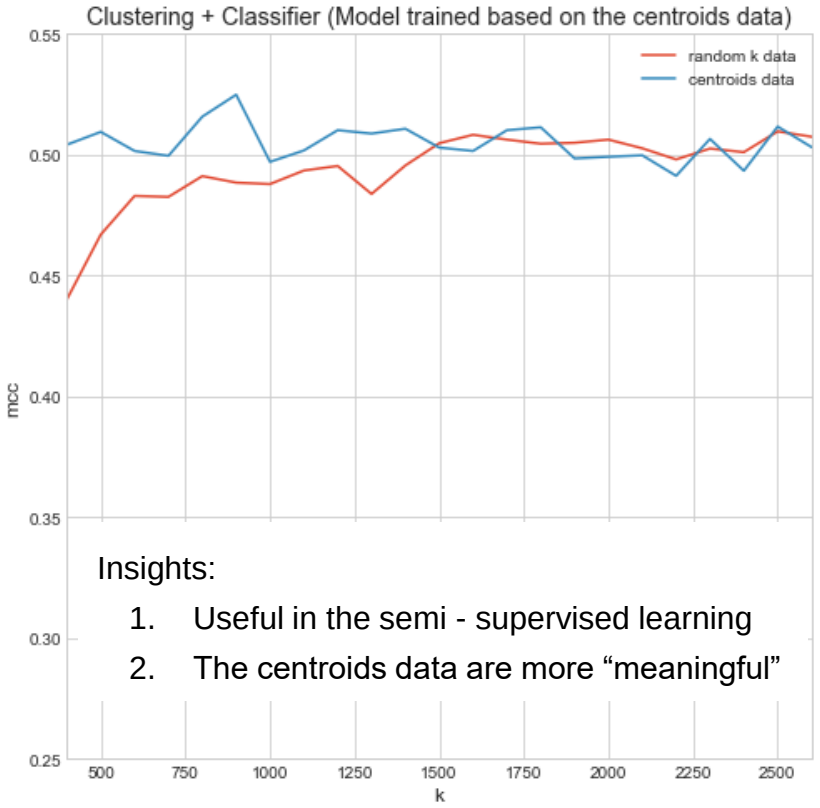
| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| 0.589 | 0.533 | 0.559 | 0.921 | 0.498 | 0.499 |

Random Forest with SMOTE + Clustering Feature Engineering

| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| **0.627** | **0.525** | **0.572** | **0.920** | **0.509** | **0.512** |

Feature Construction: Either use one or both depending on the performance

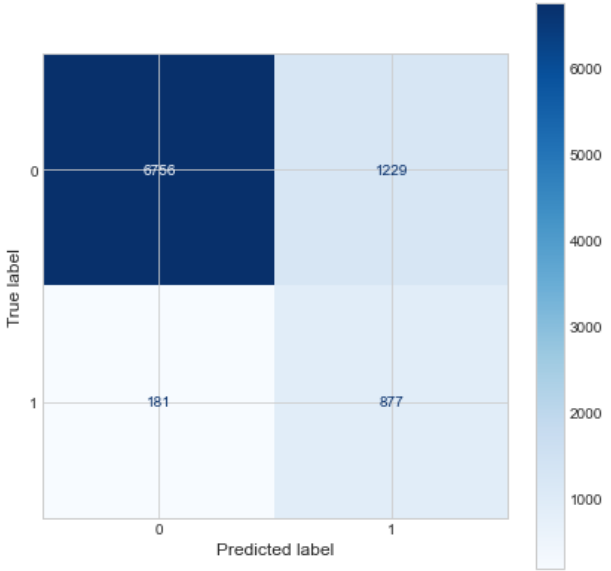
# [Experiment] Clustering Centroids Based Training


Training based on random selected data VS Training based on clustering centroids data

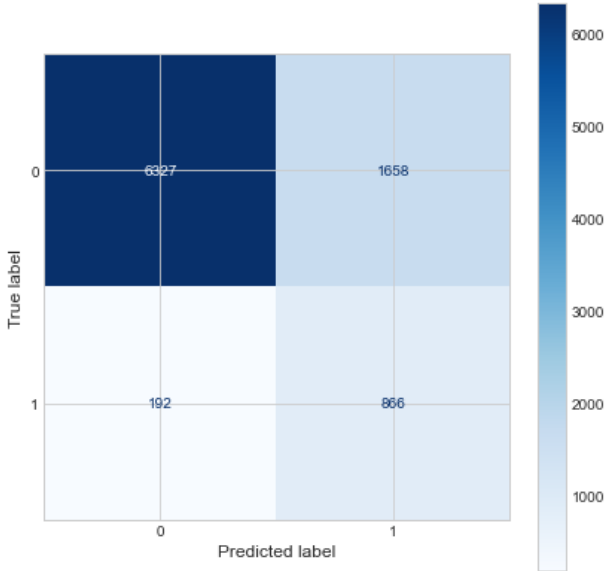
# [Implication] Clustering Centroids Based Undersampling

Random Undersampling




| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| 0.829 | 0.416 | 0.554 | 0.913 | 0.472 | 0.513 |

Centroids Based Undersampling



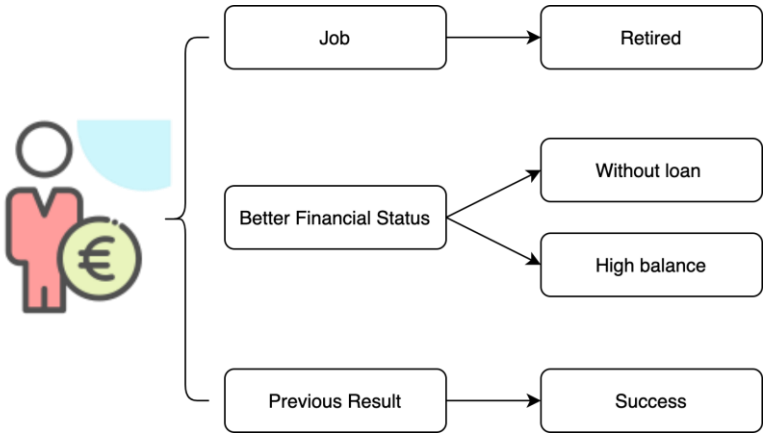

| Recall | Precision | F1 | AUC | Kappa | MCC |
|---|---|---|---|---|---|
| 0.819 | 0.343 | 0.483 | 0.884 | 0.382 | 0.438 |

*** Important records are far away from the centroids**

# Conclusions

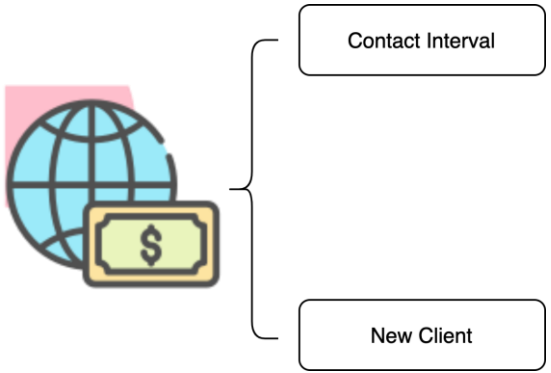
# Business Insights

## Target Client's Profile




- Retired individuals
- Individuals with better financial status
- Individuals with successful previous campaign result

## Campaign Strategy




- Less frequent contact interval
- Incentive for new clients

# Further Improvement

**Feature ‘duration’**

Feature ‘duration’ is considered as the most important feature, try and construct more features with ‘duration’ to see whether the performance gets improved.

**Domain Knowledge Involvement**

Involve more domain knowledge in the feature engineering process. (feature construction, feature discretization...)

**Clustering Feature Engineering**

This method is effective while working with simple models, try and implement with more complex models to see whether any further improvements.

**Model Simplification**

Less is Better, seek better balance between accuracy and efficiency.

# Thank you !