CREDIT CARD FRAUD DETECTION

KRISH WERAGALARACHCHI

OUTLINE

- INTRODUCTION
- EXPLORATORY DATA ANALYSIS
- DATA PREPARATION : Treating Class Imbalance with Random Oversampling, SMOTE, Random Undersampling and SMOTESVM
- METHODOLOGY : Logistic Regression, KNN, Naïve Bayes, SVM, Decision Tress, Random Forest and XGBoost
- MODEL EVALUATION : Hyperparameter Tuning, Hypothesis Testing To Compare Baseline Model and The Best Model Selected, Loss Function for Best Model
- RESULTS
- CHALLENGES
- FUTURE DEVELOPMENT

Introduction

Credit card fraud is a form of identity theft in which criminals make purchases or get cash using the credit cards of other people.

There are nearly 400,000 frauds reported to federal trade commission in 2020 and it is a 44% increase from 2019.

Credit card fraud loses billions of dollars of consumers, merchants, and financial intuitions and should be stopped before happening.

(Khilnani,2012)

Data

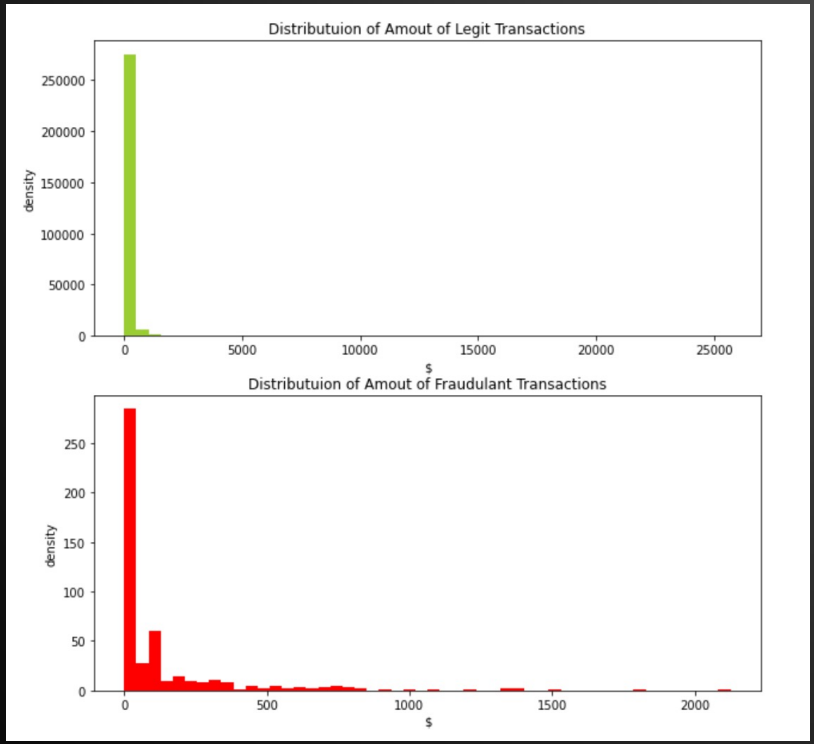
- Credit card transactions in 2013 : from Worldline and the Machine Learning Group at ULB (Université Libre de Bruxelles) on big data mining and fraud detection. Data is publicly available at Kaggle
- Contains 28 PCA components, and two features that haven't been transformed (Amount and Time).

V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232794	0.105915	0.253844	0.081080	3.67	0
.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750137	-0.257237	0.034507	0.005168	4.99	0
.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267	-0.051634	-1.206921	-1.085339	40.80	0

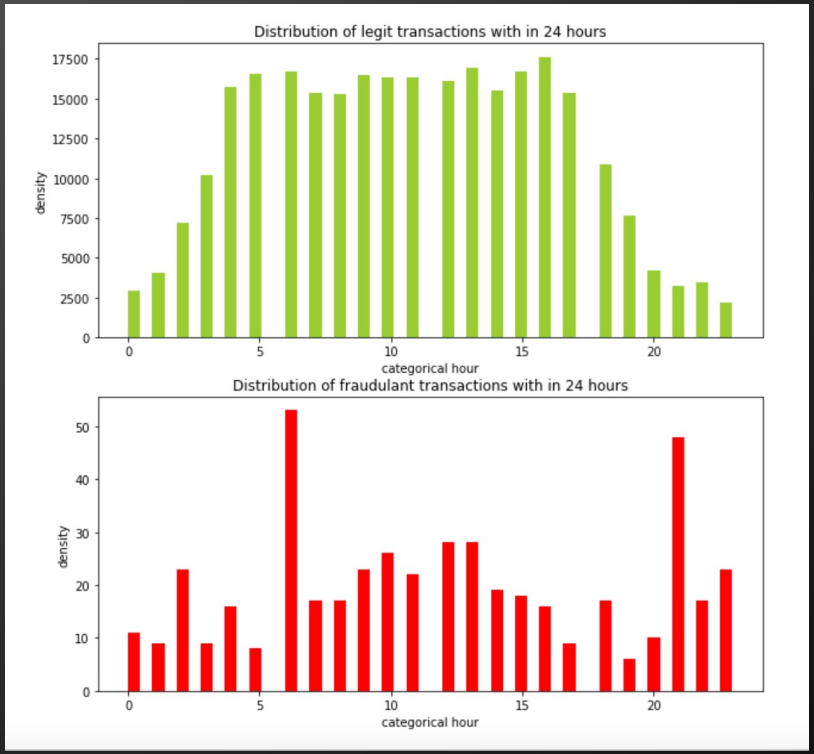
❑ Exploratory Data Analysis

- Check for missing values : none
- Remove duplicates : 1081 duplicates
- Feature Engineering : extract hour from time
- Standardize the ‘amount’ and’ hour’

Amount



Hour



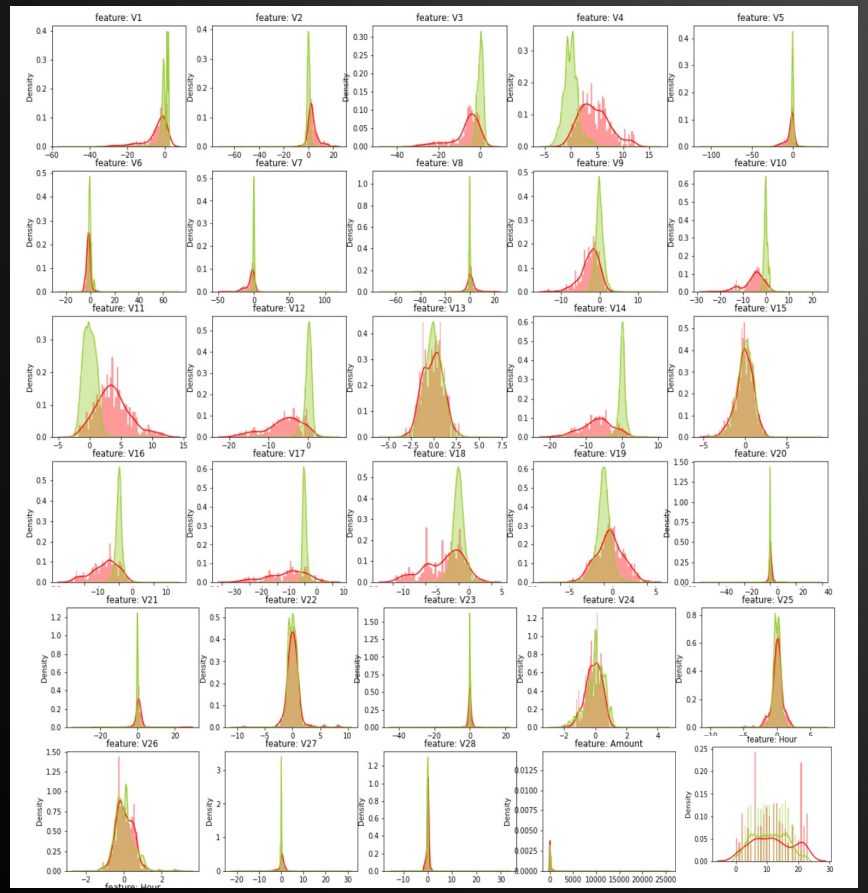
Mean values:

Fraud transactions: $88.41

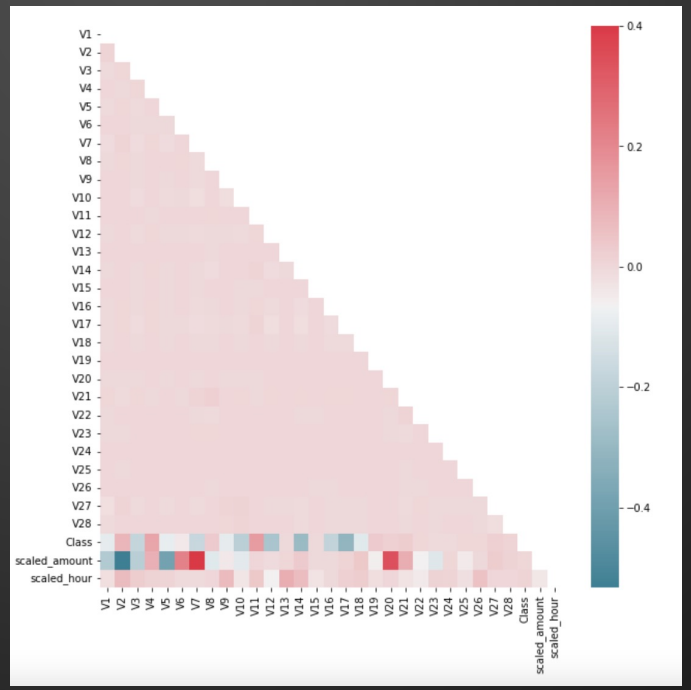
Legit transactions: $123.87

Feature Selection

Class Distribution of 30 features



Correlation Matrix



Mutual information for features with target variable

```
   7  mutual_info_Series
```

V17	0.007873
V14	0.007746
V12	0.007220
V10	0.007150
V11	0.006450
V16	0.005752
V4	0.004587
V3	0.004583
V18	0.003875
V9	0.003854
V7	0.003630
scaled_hour	0.003017
V2	0.002920
V27	0.002166
V21	0.002139
V5	0.002065
V6	0.002001
V1	0.001851
V28	0.001607
V8	0.001600
scaled_amount	0.001467
V19	0.001132
V20	0.000904
V23	0.000471
V24	0.000364
V25	0.000253
V26	0.000239
V22	0.000126
V13	0.000085
V15	0.000023

dtype: float64

Data Preparation

Sample Selection

```
# get a sample maintaining the same imbalance ratio

sample1=df.groupby('Class', group_keys=False).apply(lambda x: x.sample(frac=0.25))
```

Train - Test split

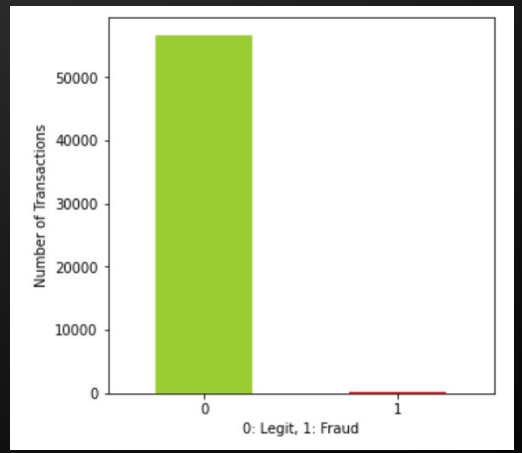
```
# train-test split (80-20)
from sklearn.model_selection import train_test_split

# get 20% of training for testing
Xtrain, Xtest, Ytrain, Ytest  = train_test_split(x,y, test_size=0.2, random_state=42)
```

Severe Class imbalance

Fraud : Legit ~ 1:600

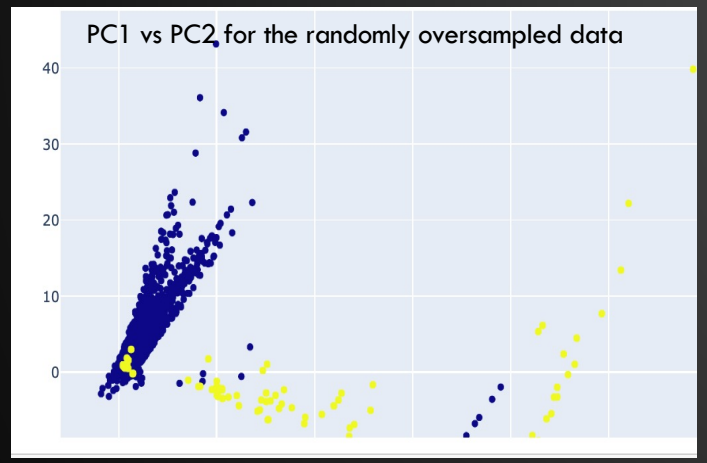
Leads to ML predictions tends to *bias toward the majority class*



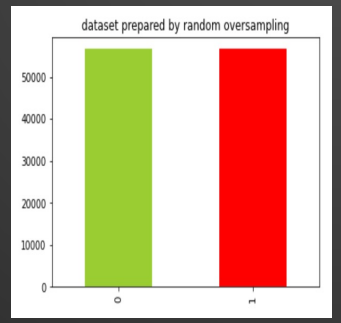
Techniques used to treat class imbalance

- Random Over-sampling
- SMOTE (Synthetic Minority Oversampling Technique) : use K-Nearest Neighbor
- SMOTE + Random Under-sampling
- Borderline-SMOTE SVM (Synthetic Minority Oversampling Technique with Simple Vector Machines)

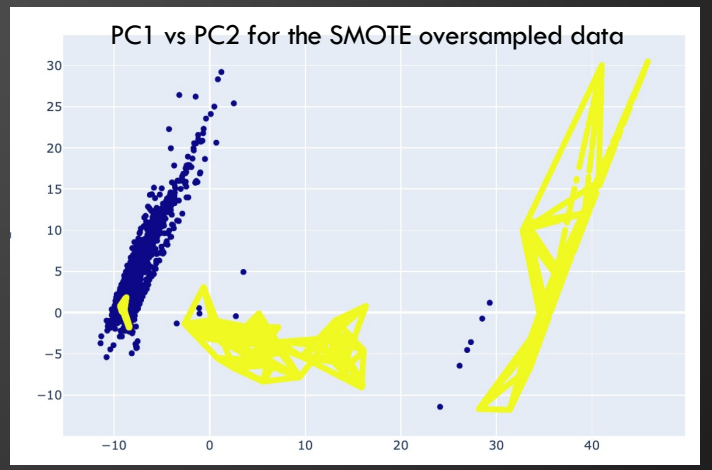
Random Over-sampling



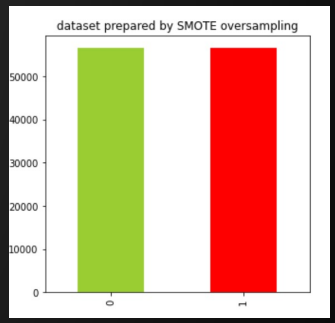
Class distribution for randomly Oversampled data



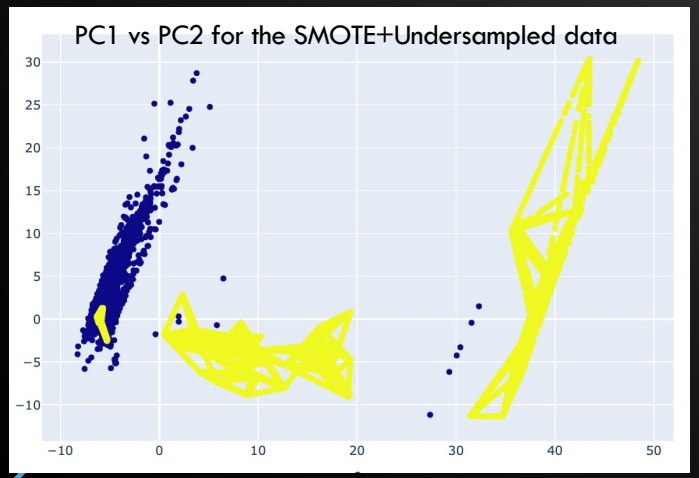
SMOTE (Synthetic Minority Oversampling Technique) : Use KNN



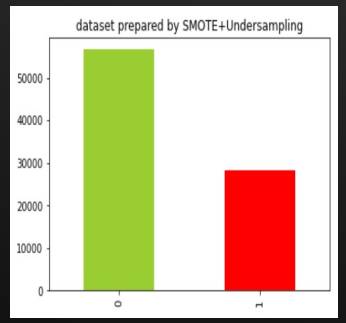
Class distribution for SMOTE Oversampled data



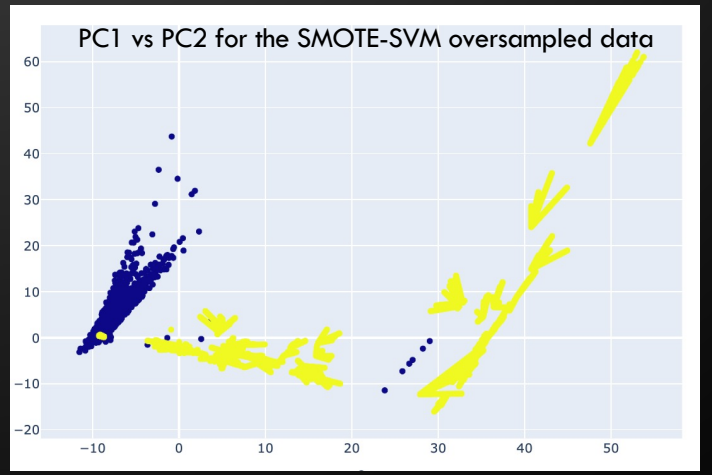
SMOTE + Random Under-sampling



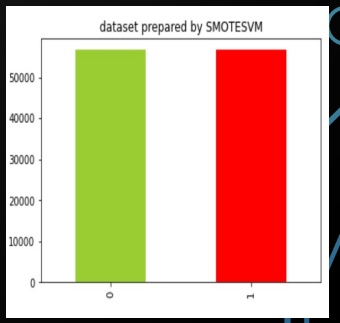
Class distribution for SMOTE + Undersampled data



Borderline-smote SVM



Class distribution for SMOTE SVM Oversampled data



❑ Methodology

Machine Learning Models Tested

- Logistic Regression
- Naive Bayes Algorithm
- K-Nearest Neighbors Algorithm
- Support Vector Machine (SVM)
- Decision Tree Algorithm
- Random Forest (Bagging Classifier)
- XGBoost (Boosting Classifier)

Baseline model

Logistic regression with imbalanced data

PR AUC (average precision) : 0.3973

Recall : 0.4242

Precision : 0.9333

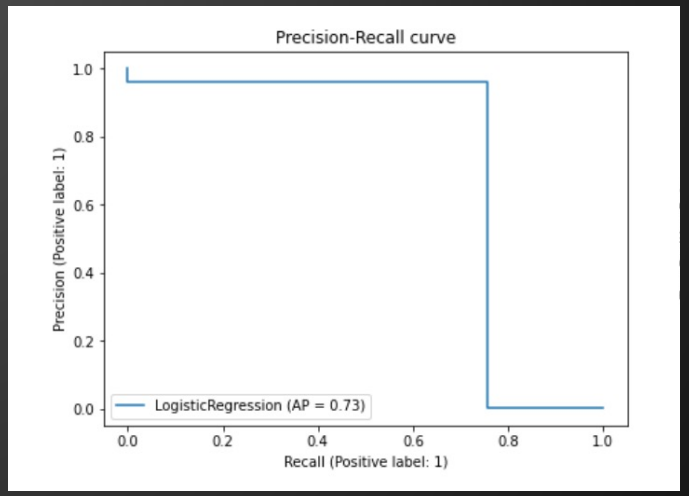
F1 score : 0.5833

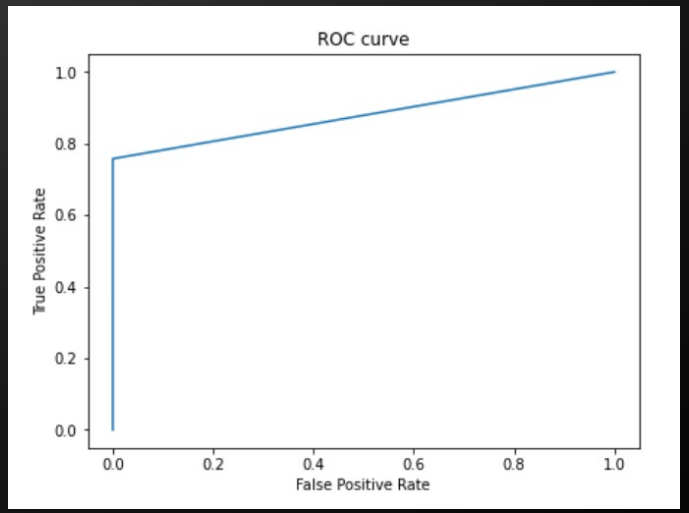
ROC AUC. : 0.8934

Accuracy : 0.9986

Confusion Matrix

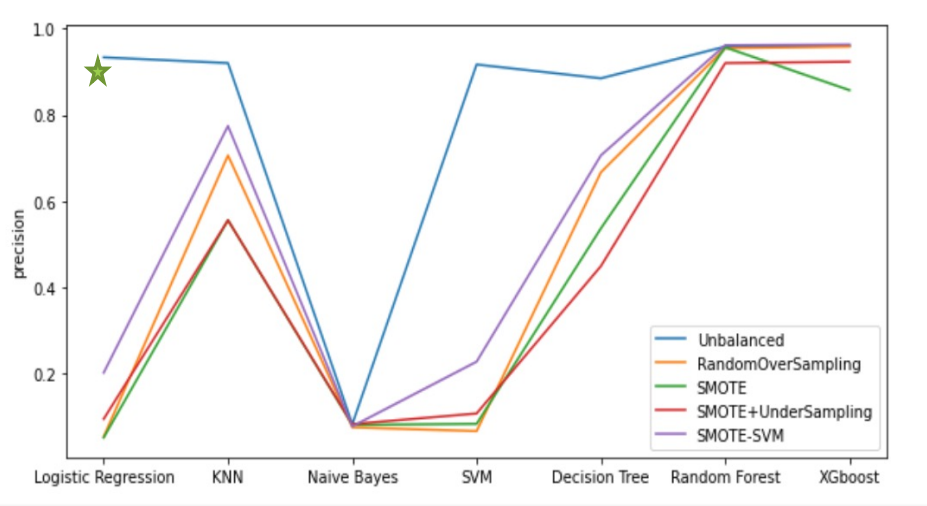
Predicted labels \ True labels	1	0
1	14	1
0	19	14153





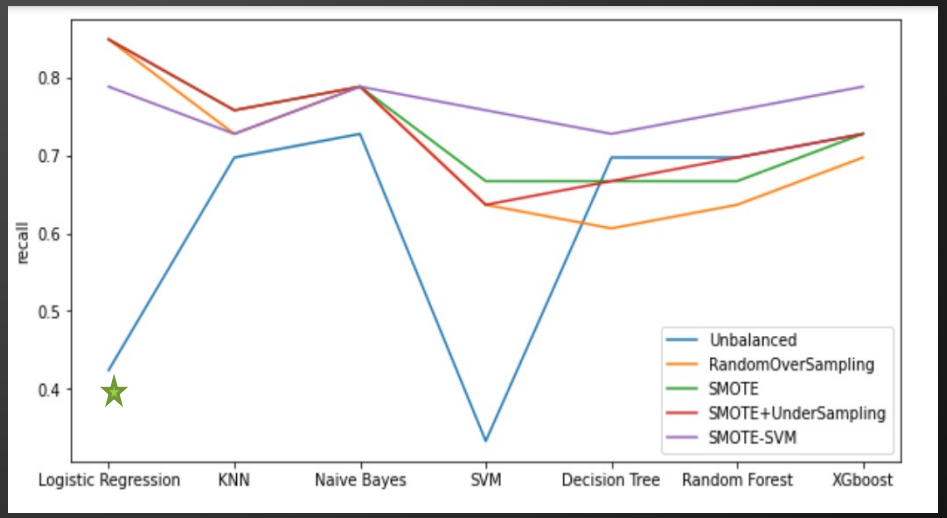
❑ Model Evaluation

Precision



	Logistic Regression	KNN	Naïve Bayes	SVM	Decision Trees	Random Forest	XGBoost
imbalanced	★ 0.9333	0.92	0.0822	0.9167	0.8519	0.9583	0.96
Random Oversampling	0.0534	0.7059	0.0749	0.0662	0.6774	0.9565	0.9583
SMOTE	0.0511	0.5556	0.0807	0.083	0.5	0.9167	0.8571
SMOTE+ Undersampling	0.0946	0.5556	0.0823	0.1071	0.4894	0.9565	0.9231
SMOTESVM	0.2185	0.7742	0.0795	0.2273	0.6765	0.96	0.963

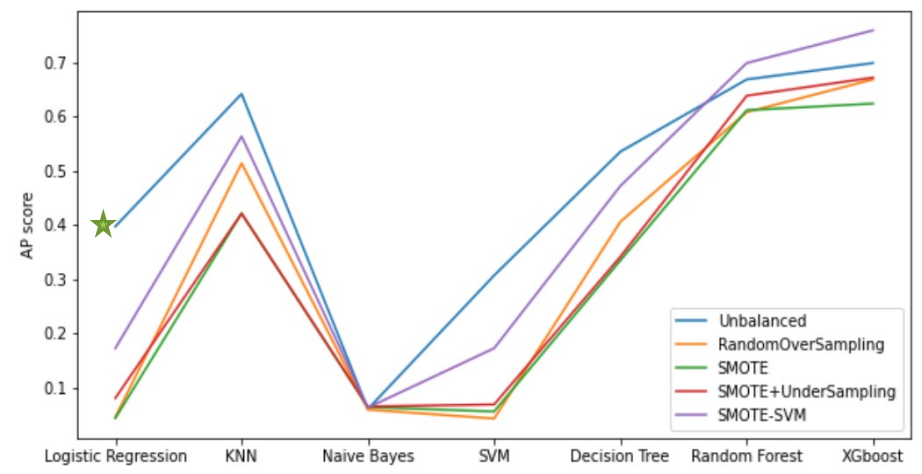
Recall



	Logistic Regression	KNN	Naïve Bayes	SVM	Decision Trees	Random Forest	XGBoost
imbalanced	★ 0.4242	0.697	0.7273	0.3333	0.697	0.697	0.7273
Random Oversampling	0.8485	0.7273	0.7879	0.6364	0.6364	0.6667	0.697
SMOTE	0.8485	0.7576	0.7879	0.6667	0.6061	0.6667	0.7273
SMOTE+ Undersampling	0.8485	0.7576	0.7879	0.6364	0.697	0.6667	0.7273
SMOTESVM	0.7879	0.7273	0.7879	0.7576	0.697	0.7273	0.7879

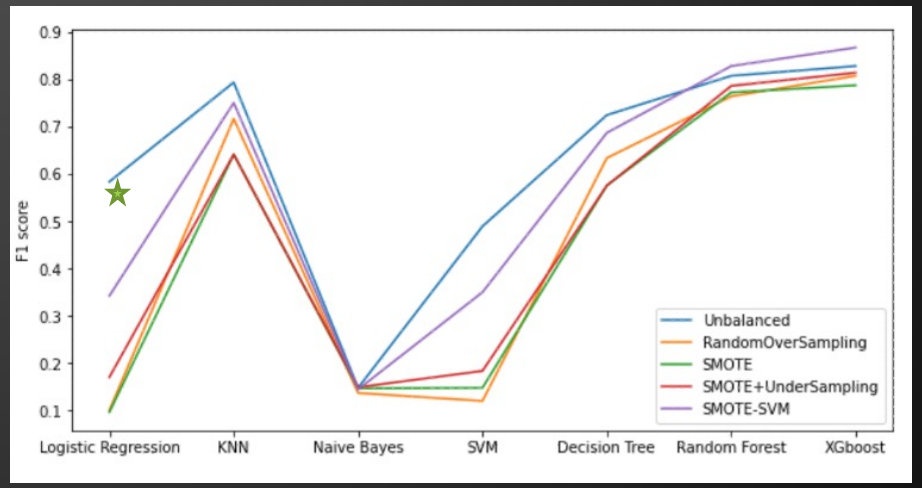
★ Baseline Logistic regression with unbalanced data

Average Precision (PR AUC)



	Logistic Regression	KNN	Naïve Bayes	SVM	Decision Trees	Random Forest	XGBoost
imbalanced	0.3973	0.6419	0.0604	0.3071	0.5944	0.6686	0.6988
Random Oversampling	0.0457	0.514	0.0595	0.043	0.4319	0.6385	0.6686
SMOTE	0.0437	0.4214	0.0641	0.0561	0.3039	0.6119	0.624
SMOTE+ Undersampling	0.0806	0.4214	0.0653	0.069	0.3418	0.6385	0.672
SMOTESVM	0.1726	0.5637	0.0631	0.1727	0.4722	0.6988	0.7592

F1 score



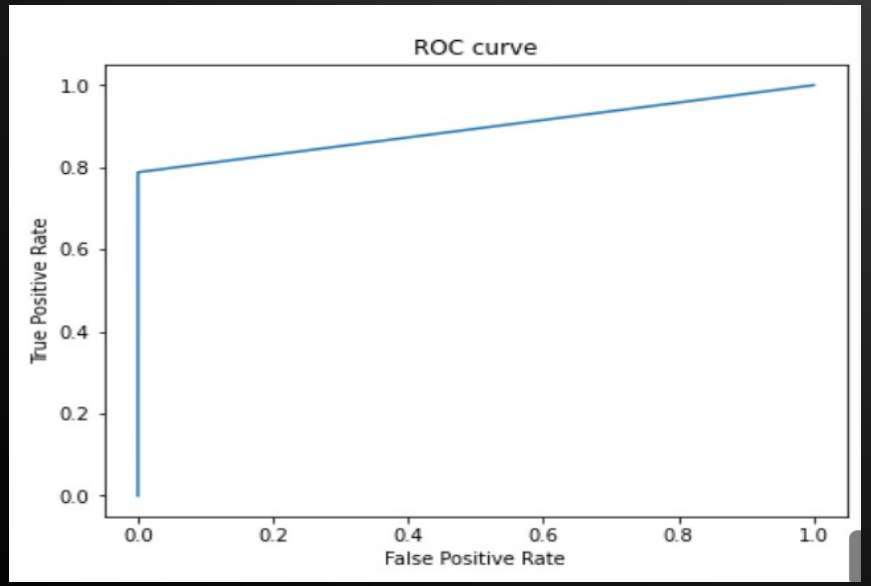
	Logistic Regression	KNN	Naïve Bayes	SVM	Decision Trees	Random Forest	XGBoost
imbalanced	0.5833	0.7931	0.1477	0.4889	0.7667	0.807	0.8276
Random Oversampling	0.1005	0.7164	0.1368	0.12	0.6562	0.7857	0.807
SMOTE	0.0964	0.641	0.1465	0.1477	0.5479	0.7719	0.7869
SMOTE+ Undersampling	0.1702	0.641	0.149	0.1834	0.575	0.7857	0.8136
SMOTESVM	0.3421	0.75	0.1444	0.3497	0.6866	0.8276	0.8667

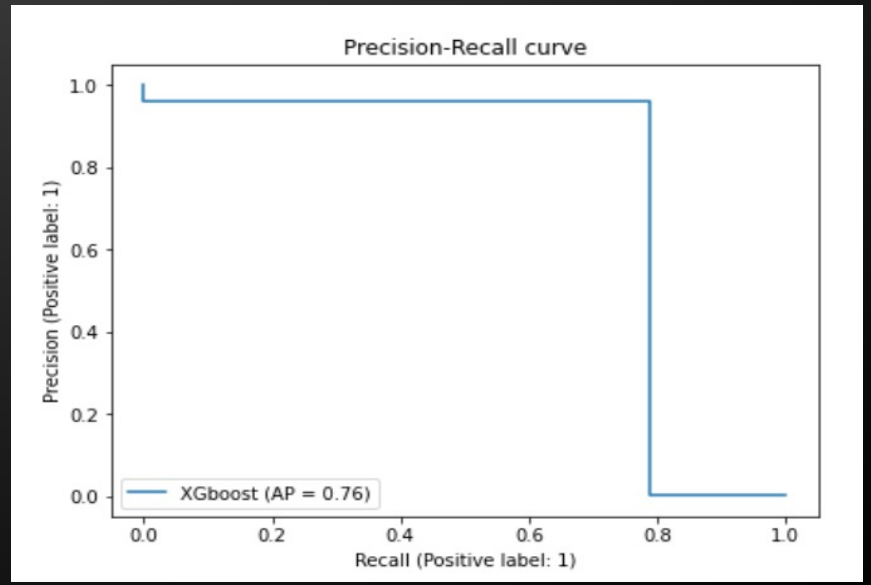
★ Baseline Logistic regression with unbalanced data

The Best Classifier – XGBoost with SMOTSVM treated dataset

PR AUC (average precision) : 0.7592
Recall : 0.7879
Precision : 0.9630
F1 score : 0.8667
ROC AUC. : 0.9474
Accuracy : 0.9994

Confusion Matrix :





Hyperparameter Tuning for XGBoost

Exhaustive GridSearch VS Successive Halving GridSearch

- Exhaustive GridSearch exhaustively generates candidates from a grid of parameter values.
- Successive halving GridSearch start evaluating all the candidates with a small amount of data and iteratively selects the best candidates, using more and more data.
- Used Pipeline method from imblern library to deal with imbalance data.

```
from imblearn.pipeline import Pipeline
model = Pipeline([
        ('sampling', SVMSMOTE()),
        ('classification', XGBClassifier())
    ])
```

- Parameters tested :

Parameter	Value
gamma	0 , 0.25 , 1
max_depth	3 , 4, 5, 6
min_child_weight	1 , 100

Comparison between Exhaustive GridSearch VS Succesive Halving GridSearch

	Exhaustive GridSearch	Halving GridSearch
Best parameters	{'gamma': 1, 'max_depth': 3, 'min_child_weight': 1}	{'gamma': 0, 'max_depth': 6, 'min_child_weight': 1}
Best average precision score	0.7973	0.7898
Total search time (minutes)	62.32	14.61

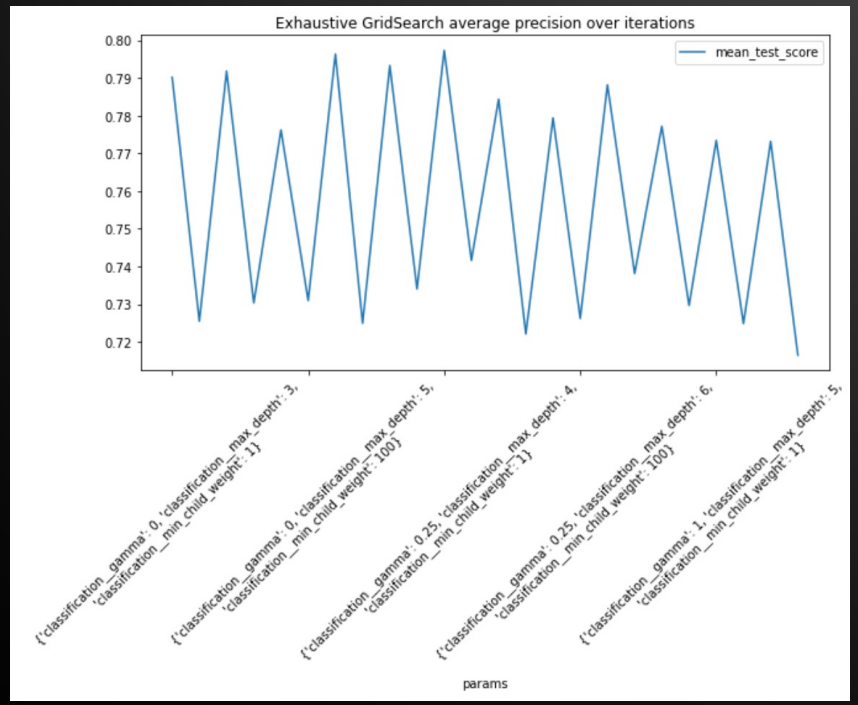
Exhaustive GridSearch vs Halving GridSearch

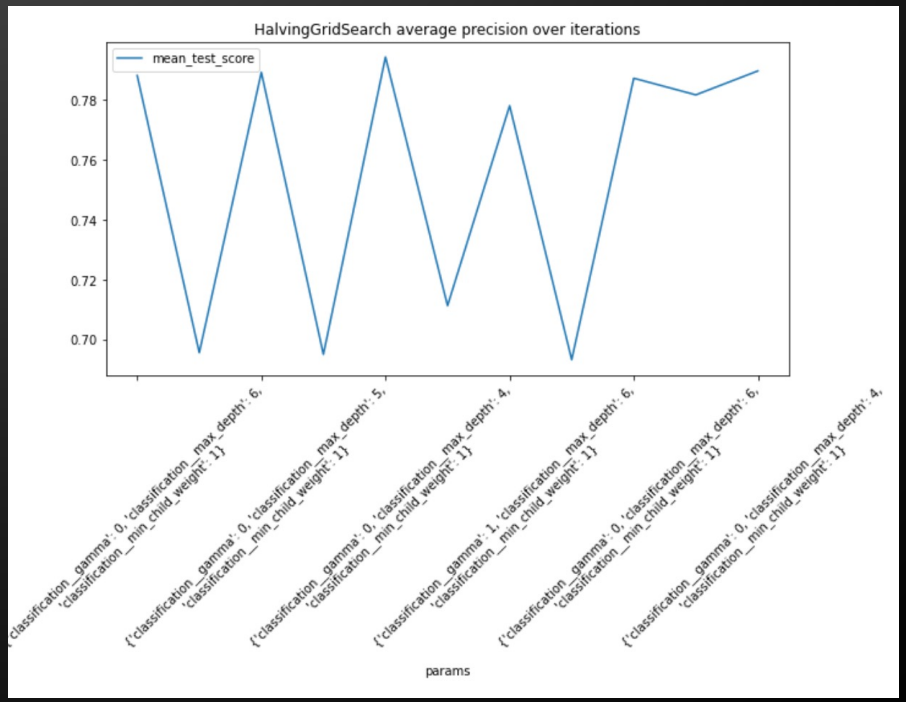
Exhaustive GridSearch

- Exhaustive GridSearch suffers from the curse of dimensionality.
- Not guaranteed to find the best solution, often aliasing over the best configuration.
- inefficient -both in computing power and time.

Successive Halving GridSearch

- Faster
- Gives optimal solution most of the time.



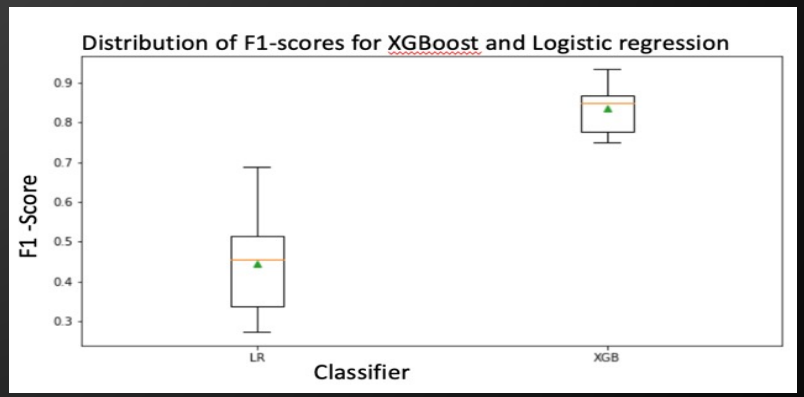


Hypothesis Testing To Compare Baseline Model (Logistic Regression) and The Best Model Selected (XGBoost)

- The mean classification F1 score was calculated for both Logistic Regression and XGBoost models.
- Results suggest that XGBoost has better performance over Logistic Regression when modeled with SMOTESVM treated dataset.

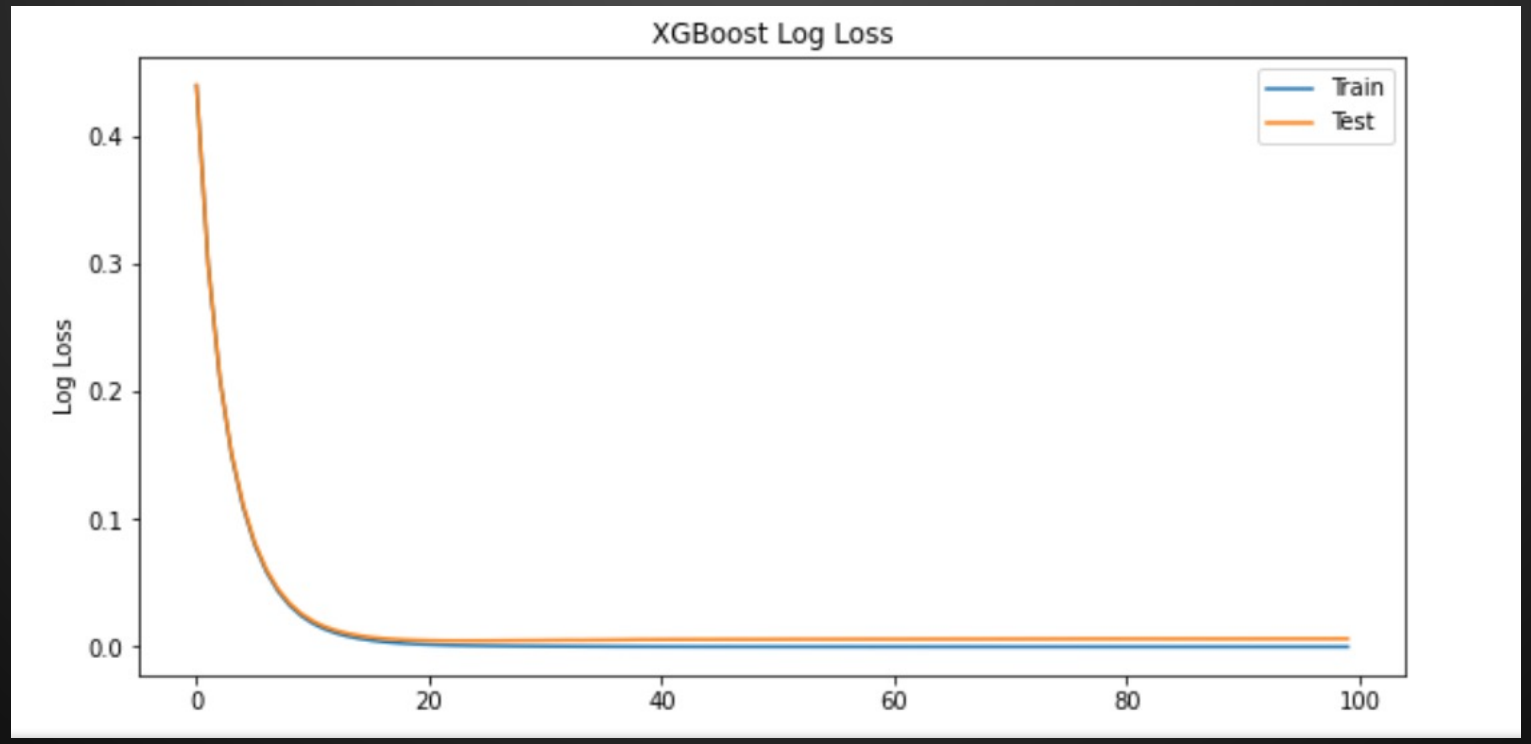
Classifier	Mean F1 score
Logistic Regression	0.445
XGBoost	0.834

Distributions of F1-score for two classifiers



- Paired student t-test was performed to see whether the difference in the performance of the two algorithms is significant or not. Here I used paired_ttest_5x2cv() method from Mlxtend library.
- The p_value = 0.036 < 0.05
- We can reject the null hypothesis and that means the observed difference between the performance of two algorithms is statistically true.

Loss function for XGBoost when used with SMOTE-SVM treated data



Results

- The XGBoost classifier when used with SMOTESVM technique can successfully predict fraud and non fraud of the 2013 European bank data over Logistic Regression classifier. P-value = 0.036
- Succesive Halving Gridsearch is more effective over Exhaustive Grid search for Hyperparameter tuning.

Challenges

- Little time for brainstorming and reading.

- Could not test Parameters in a wide range as it takes longer to finish. I tried 6 parameters all together 576 combinations and it did not finish searching even after 2 days and I had restart Exhaustive gridsearch with less parameters (3 parameters and 24 combinations) to efficiently deliver the project in time.

```
params_XGB = {
    "gamma":[0,0.5, 1, 5],
    "max_depth":[3,4,5,6],
    "min_child_weight": [1,100],
    "subsample": [0.6, 0.8, 1],
    "colsample_bytree": [0.6, 0.8, 1],
    "learning_rate": [0.1, 0.01]
    }
```

- Could not use whole data set due to limited resources capabilities.

Future Work

- The whole dataset can be used to see whether the model performance can be increased.
- The parameter grid can be expanded to find a better model.
- Features that has low mutual information score and same class distributions can be removed to see whether the performance score increases.
- I used SMOTE technique with default k = 5 and we can find the optimal k for SMOTE technique and see whether it is better than other method to treat class imbalance.

THANK YOU