Comparison Study of Machine Learning Algorithms for Credit Card Default Prediction

T00728581 Kai Sun T00728937 Yishu Liu

Abstract

This report investigates the application of logistic regression and XGBoost models to predict credit card defaults. We assess the effectiveness of feature standardization and feature selection (LASSO regression) on these two models, and the optimization of hyperparameters using Cross-Validation on XGBoost. We also compare the computational cost vs accuracy of these two models. Our findings indicate that logistic regression benefits significantly from standardization and feature selection, improving its prediction accuracy from 77.87% to 80.97%. In contrast, XGBoost, which handles features intrinsically, shows no improvement, but the set of hyperparameters has influence on XGBoost accuracy (from 81.99% to 82.19%). For prediction accuracy and computational costs, the results suggest that while XGBoost generally outperforms logistic regression in accuracy (82.19% versus 80.97%), logistic regression is notably faster and less computationally intensive (0.9 seconds versus 5.0 seconds). This report provides insights that can guide financial institutions in selecting suitable models based on accuracy needs and computational constraints, highlighting the potential of machine learning to enhance credit risk assessments.

1, Introduction

Accurately predicting defaults is crucial for banks and financial institutions as it assists in making informed decisions about credit issuance, risk management, and debt recovery strategies. The ability to preemptively identify potential defaulters can significantly reduce the financial losses associated with uncollected debts [3].

Machine learning offers a robust framework for tackling the complexities associated with credit default prediction. Machine learning models, particularly those involving XGBoost and ensemble methods, have demonstrated superior performance in numerous studies focused on credit risk assessment [1][2].

This report compares two machine learning methods, logistic regression and XGBoost, to see which is better at predicting credit card defaults. It looks at how well each method predicts defaults and how fast they work. The objectives of this report are:

(1) Explore the influence of feature standardization and feature selection (Lasso Regression) on the prediction accuracy of Logistic Regression and XGBoost.

(2) Explore the effectiveness of utilizing Cross-Validation for selecting the best hyperparameters of XGBoost.

(3) Compare the prediction accuracy for Logistic Regression and XGBoost.

(4) Compare the computation cost of Logistic Regression and XGBoost.

2, Data

The "Default of Credit Card Clients" dataset is extensively utilized within the field of credit scoring to predict the likelihood of default among credit card clients. This dataset encompasses 30,000 observations pertaining to credit card holders in Taiwan and includes 25 distinctive features. These features comprise demographic data, credit limits, payment histories, and amounts billed.

The data was sourced from the credit card clients of a financial institution in Taiwan and is available through the UCI Machine Learning Repository. It spans from April 2005 to September 2005 and contains details on whether each credit card holder defaulted on their payment in the subsequent month. The features and their descriptions are systematically presented in Table 1.

Table 1 Credit Card Default Dataset Features

Feature	Description
ID	ID of each client
LIMIT_BAL	Amount of given credit in NT dollars (includes individual and family/supplementary credit)
SEX	Gender (1=male, 2=female)
EDUCATION	Education (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
MARRIAGE	Marital status (1=married, 2=single, 3=others)
AGE	Age in years
PAY_0	Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, ...)
PAY_2	Repayment status in August, 2005 (scale same as above)
PAY_3	Repayment status in July, 2005 (scale same as above)
PAY_4	Repayment status in June, 2005 (scale same as above)
PAY_5	Repayment status in May, 2005 (scale same as above)
PAY_6	Repayment status in April, 2005 (scale same as above)
BILL_AMT1	Amount of bill statement in September, 2005 (NT dollar)
BILL_AMT2	Amount of bill statement in August, 2005 (NT dollar)
BILL_AMT3	Amount of bill statement in July, 2005 (NT dollar)
BILL_AMT4	Amount of bill statement in June, 2005 (NT dollar)
BILL_AMT5	Amount of bill statement in May, 2005 (NT dollar)
BILL_AMT6	Amount of bill statement in April, 2005 (NT dollar)
PAY_AMT1	Amount of previous payment in September, 2005 (NT dollar)
PAY_AMT2	Amount of previous payment in August, 2005 (NT dollar)
PAY_AMT3	Amount of previous payment in July, 2005 (NT dollar)
PAY_AMT4	Amount of previous payment in June, 2005 (NT dollar)
PAY_AMT5	Amount of previous payment in May, 2005 (NT dollar)
PAY_AMT6	Amount of previous payment in April, 2005 (NT dollar)
default.payment.next.month	Default payment (1=yes, 0=no)

3, Method

In order to assess the questions that we listed in the objectives, we designed five experiments (Table 2). The first experiment and the second experiment are to compare the accuracy of Logistic Regression and XGBoost under the raw data (without feature standardization and selection). Experiment 4 and 5 are designed to show the accuracy and computational cost of the two models with the data that has undergone feature selection with LASSO regression and feature standardization. We also designed experiment 3 to compare the accuracy of XGBoost with the random hyperparameters and the best ones (Experiment 5).

For the determination of hyperparameters, we use 10-fold cross validation to gain the results. **All model accuracy results have been calculated through 10-fold cross validation to get a reliable result.**

(1) Logistic Regression

Logistic regression is a statistical model used for binary classification tasks, where the outcome variable is categorical with two levels (e.g., yes/no, 1/0). It estimates the probability that a given observation belongs to a particular category based on one or more predictor variables. Unlike linear regression, logistic regression uses the logistic function to model the relationship between predictors and the probability of the outcome. It is widely used in various fields, including healthcare, finance, marketing, and social sciences, for tasks such as predicting customer churn, spam detection, and disease diagnosis.

(2) XGBoost

XGBoost, short for Extreme Gradient Boosting, is a powerful and scalable machine learning algorithm that belongs to the gradient boosting family. It is widely used for regression and classification tasks and has gained popularity for its efficiency, accuracy, and flexibility. XGBoost builds a series of decision trees sequentially, where each tree corrects the errors made by the previous ones. It employs a gradient descent optimization technique to minimize a predefined loss function, such as mean squared error for regression or log loss for classification. XGBoost provides various hyperparameters to control the model's behavior and performance, making it suitable for a wide range of applications. It has been used to win numerous machine learning competitions and is a popular choice among data scientists and practitioners.

Table 2 Design of Experiments

Experiment No.	1	2	3	4	5
Data Preprocessing	Missing values	Missing values	Missing values; Standardization	Missing values; Standardization	Missing values; Standardization
Feature Selection	None	None	LASSO	LASSO	LASSO
Training and Validation	Logistic Regression	XGBoost	XGBoost	Logistic Regression	XGBoost with best Hyperparameters
Performance Comparison	Accuracy	Accuracy	Accuracy	Accuracy; Computational cost	Accuracy; Computational cost

Code Link: https://github.com/onmywaysh/new/credit-default.ipynb

****Please run the code in Kaggle Notebook otherwise please edit the dataset file path in**

the code.

Data Link: https://github.com/onmywaysh/new/UCI_Credit_Card.csv

4, Results

In this section, we will discuss the results (Table 3) based on the objectives we mentioned in the introduction.

Table 3 Experiment Results

Experiment No.	Accuracy	Computational Cost
1	77.87%	/
2	81.99%	/
3	81.99%	/
4	80.97%	0.9s
5	82.19%	5.0s

(1) Influence of Feature Standardization and Feature Selection: The application of Lasso Regression helped in identifying crucial features which include 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', and several others related to payment history and bill amounts. Figure 1 and Figure 2 show the correlation matrix before and after the feature selection by LASSO.

The accuracy of logistic regression improved from 77.87% to 80.97% when Lasso Regression was used for feature selection and the data was scaled. This indicates a notable positive impact of feature engineering on logistic regression. However, the accuracy of XGBoost remained the same after these adjustments. This suggests that while feature engineering benefits logistic regression significantly, XGBoost's built-in feature handling capabilities limit the gains from external feature selection methods.

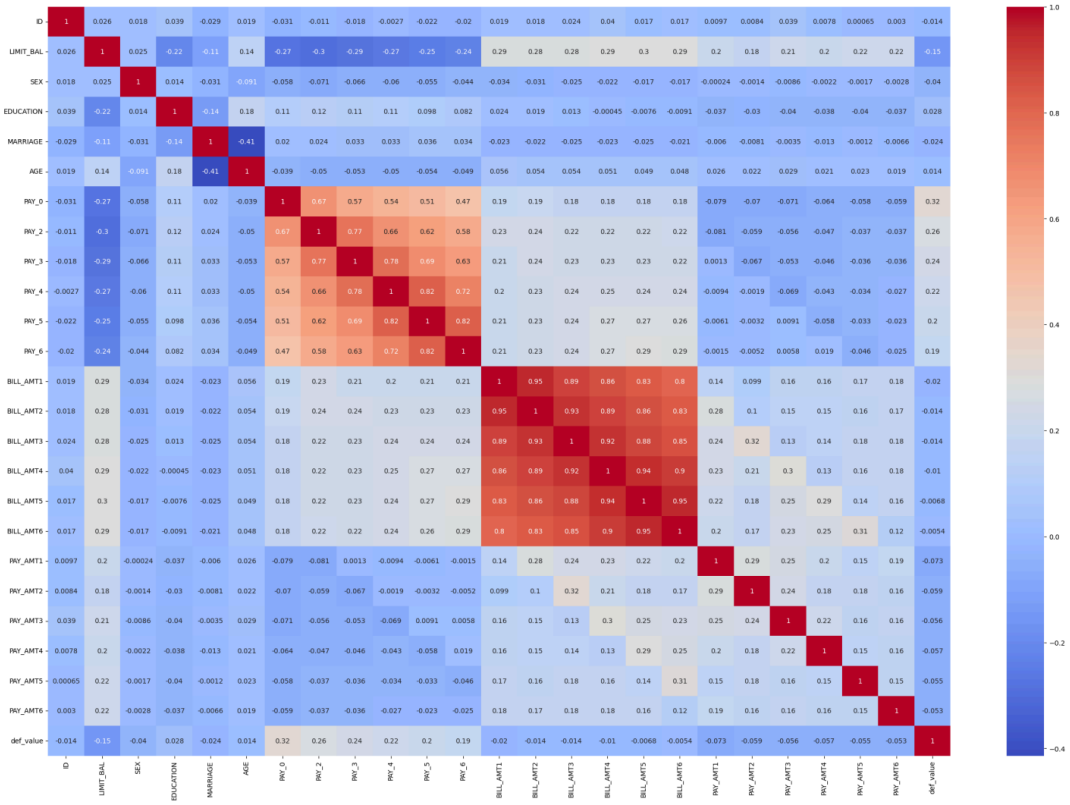


Figure 1 Correlation Matrix before feature selection by LASSO

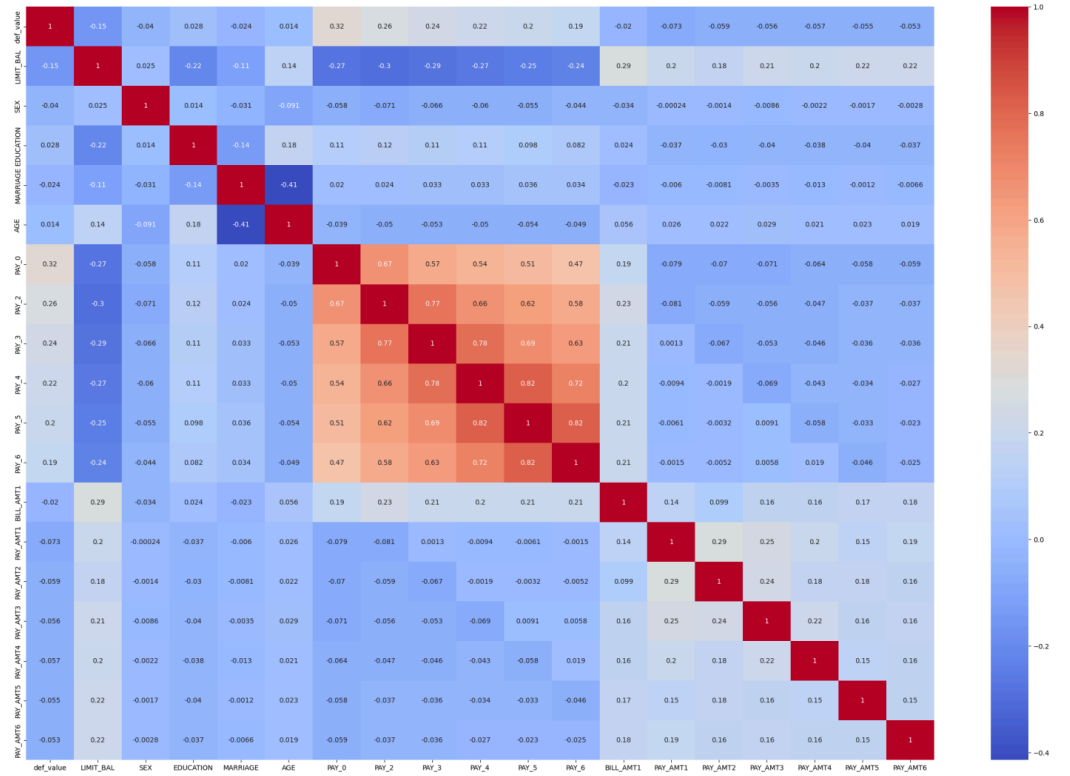


Figure 2 Correlation Matrix after feature selection by LASSO

(2) Utilizing Cross-Validation for Hyperparameter Selection of XGBoost: We employed 10-

fold Cross-Validation to fine-tune the hyperparameters of XGBoost. The best parameters identified were a maximum depth of 3, a learning rate of approximately 0.034, and 167 estimators. The optimized model achieved an improved accuracy of 82.19%.

(3) Comparison of Prediction Accuracy: Before feature selection and standardization, XGBoost outperformed logistic regression with accuracies of 81.99% compared to 77.87%. Post-adjustments, both models saw improvements, but XGBoost maintained a higher accuracy (82.19% versus 80.97%). This illustrates XGBoost's superior ability to handle complex patterns in the data.

(4) Comparison of Computation Cost: The computation time for logistic regression after feature selection and standardization was approximately 0.9 seconds, markedly lower than the XGBoost model with the best hyperparameters, which took about 5.0 seconds. This significant difference highlights logistic regression's advantage in terms of computational efficiency, making it a preferable option in scenarios where prediction speed is critical.

5, Conclusions

In conclusion, this report has addressed the primary objectives surrounding the application of machine learning models to predict credit card defaults, particularly focusing on logistic regression and XGBoost. Through a detailed analysis involving feature engineering, feature selection using Lasso Regression, and optimizing hyperparameters via Cross-Validation, we have drawn several key insights.

Firstly, feature standardization and feature selection significantly enhanced the performance of logistic regression. The use of Lasso Regression proved effective in pinpointing relevant features, which streamlined the model by focusing only on significant predictors. Feature standardization and selection has improved prediction accuracy of Logistic Regression from 77.87% to 80.97%. This change underscores the importance of appropriate feature handling to enhance model effectiveness, particularly for models that do not inherently manage feature relevance like logistic regression. On the other hand, XGBoost, which handles feature selection intrinsically, showed no accuracy improvement. This highlights its robustness and less dependency on external feature engineering compared to logistic regression.

Furthermore, our findings from utilizing 10-fold Cross-Validation for hyperparameter tuning in XGBoost demonstrated that with the appropriate hyperparameters, the model accuracy can be improved to some extent.

The comparative analysis of both models in terms of prediction accuracy and computation costs highlighted that while XGBoost generally outperforms logistic regression in accuracy (82.19% versus 80.97%), logistic regression is notably faster and less computationally intensive (0.9 seconds versus 5.0 seconds). This makes logistic regression a viable option for scenarios where computational resources are limited or where rapid model deployment is crucial.

In essence, the choice between using logistic regression and XGBoost should be guided by the specific needs regarding accuracy, computation efficiency, and available resources for feature engineering and model tuning. This exploration provides a foundational understanding that can assist financial institutions in making informed decisions about deploying machine learning

models for credit risk assessment. As the field evolves, further studies could explore more complex ensemble methods that may offer better trade-offs between accuracy and computational efficiency.

References

1. Ghodselahi, A., & Amirmadhi, A. (2011). Application of artificial intelligence techniques for credit risk evaluation. International Journal of Modeling and Optimization.
2. Sariannidis, N., Papadakis, S., Garefalakis, A., Lemonakis, C., & Kyriaki-Argyro, T. (2020). Default avoidance on credit card portfolios using accounting, demographical and exploratory factors: decision making based on machine learning (ML) techniques. Annals of Operations Research.
3. Yeh, I. C., & Lien, C. H. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Systems with Applications.

In [17]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import sklearn as sk
from sklearn.model_selection import train_test_split, RandomizedSearchCV, KFold,
from sklearn.linear_model import LogisticRegression,Lasso, LassoCV
import seaborn as sns
import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from sklearn.preprocessing import StandardScaler
from scipy.stats import uniform, randint
import time

#read UCI data from Kaggle
data=pd.read_csv('/kaggle/input/default-of-credit-card-clients-dataset/UCI_Credi
```

In [18]:
```
#EDA
#print the summary of data
print(data.describe().T)
data.rename(columns={'default.payment.next.month':'def_value'}, inplace=True)
#print the number of null values in the data
print(data.isna().sum())
```

	count	mean	std	min
ID	30000.0	15000.500000	8660.398374	1.0
LIMIT_BAL	30000.0	167484.322667	129747.661567	10000.0
SEX	30000.0	1.603733	0.489129	1.0
EDUCATION	30000.0	1.853133	0.790349	0.0
MARRIAGE	30000.0	1.551867	0.521970	0.0
AGE	30000.0	35.485500	9.217904	21.0
PAY_0	30000.0	-0.016700	1.123802	-2.0
PAY_2	30000.0	-0.133767	1.197186	-2.0
PAY_3	30000.0	-0.166200	1.196868	-2.0
PAY_4	30000.0	-0.220667	1.169139	-2.0
PAY_5	30000.0	-0.266200	1.133187	-2.0
PAY_6	30000.0	-0.291100	1.149988	-2.0
BILL_AMT1	30000.0	51223.330900	73635.860576	-165580.0
BILL_AMT2	30000.0	49179.075167	71173.768783	-69777.0
BILL_AMT3	30000.0	47013.154800	69349.387427	-157264.0
BILL_AMT4	30000.0	43262.948967	64332.856134	-170000.0
BILL_AMT5	30000.0	40311.400967	60797.155770	-81334.0
BILL_AMT6	30000.0	38871.760400	59554.107537	-339603.0
PAY_AMT1	30000.0	5663.580500	16563.280354	0.0
PAY_AMT2	30000.0	5921.163500	23040.870402	0.0
PAY_AMT3	30000.0	5225.681500	17606.961470	0.0
PAY_AMT4	30000.0	4826.076867	15666.159744	0.0
PAY_AMT5	30000.0	4799.387633	15278.305679	0.0
PAY_AMT6	30000.0	5215.502567	17777.465775	0.0
default.payment.next.month	30000.0	0.221200	0.415062	0.0

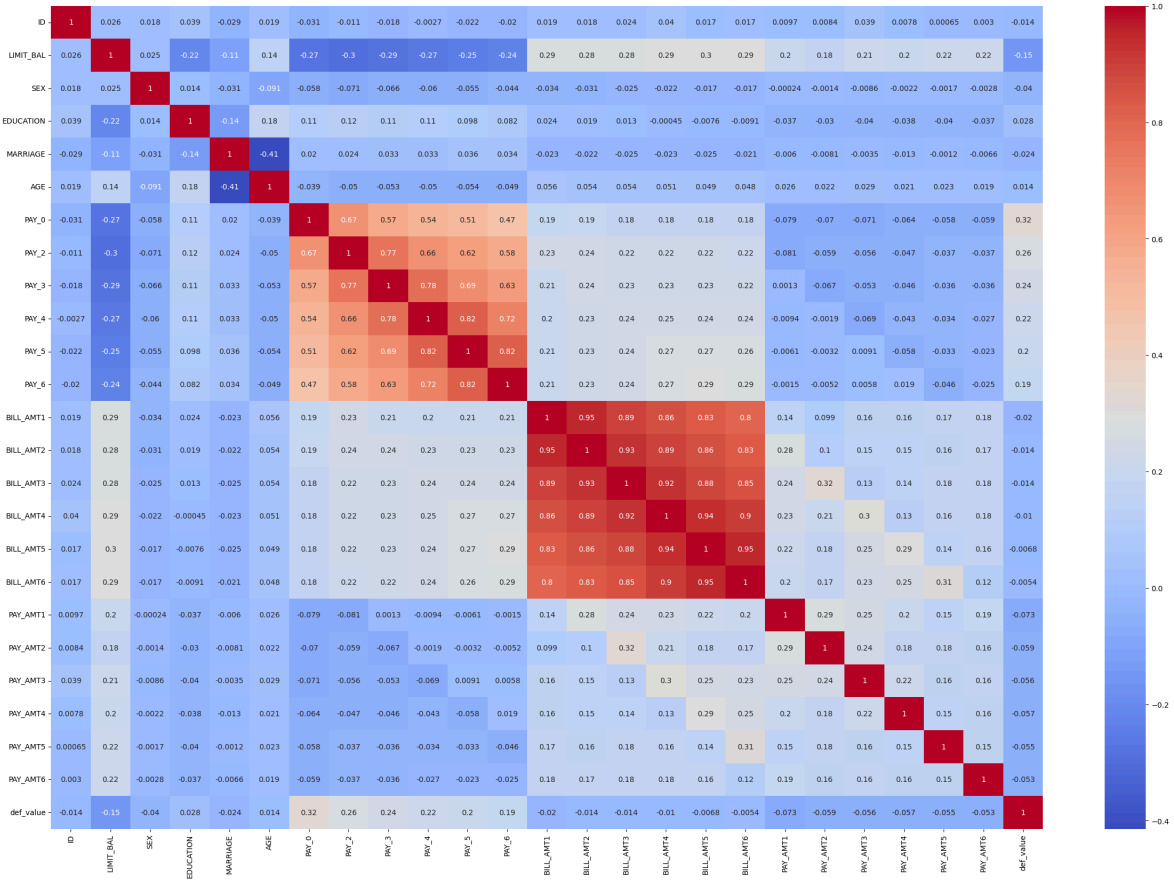
	25%	50%	75%	max
ID	7500.75	15000.5	22500.25	30000.0
LIMIT_BAL	50000.00	140000.0	240000.00	1000000.0
SEX	1.00	2.0	2.00	2.0
EDUCATION	1.00	2.0	2.00	6.0
MARRIAGE	1.00	2.0	2.00	3.0
AGE	28.00	34.0	41.00	79.0
PAY_0	-1.00	0.0	0.00	8.0
PAY_2	-1.00	0.0	0.00	8.0
PAY_3	-1.00	0.0	0.00	8.0
PAY_4	-1.00	0.0	0.00	8.0
PAY_5	-1.00	0.0	0.00	8.0
PAY_6	-1.00	0.0	0.00	8.0
BILL_AMT1	3558.75	22381.5	67091.00	964511.0
BILL_AMT2	2984.75	21200.0	64006.25	983931.0
BILL_AMT3	2666.25	20088.5	60164.75	1664089.0
BILL_AMT4	2326.75	19052.0	54506.00	891586.0
BILL_AMT5	1763.00	18104.5	50190.50	927171.0
BILL_AMT6	1256.00	17071.0	49198.25	961664.0
PAY_AMT1	1000.00	2100.0	5006.00	873552.0
PAY_AMT2	833.00	2009.0	5000.00	1684259.0
PAY_AMT3	390.00	1800.0	4505.00	896040.0
PAY_AMT4	296.00	1500.0	4013.25	621000.0
PAY_AMT5	252.50	1500.0	4031.50	426529.0
PAY_AMT6	117.75	1500.0	4000.00	528666.0
default.payment.next.month	0.00	0.0	0.00	1.0

```
ID           0
LIMIT_BAL    0
SEX          0
EDUCATION    0
MARRIAGE     0
AGE          0
PAY_0        0
```

```
PAY_2          0
PAY_3          0
PAY_4          0
PAY_5          0
PAY_6          0
BILL_AMT1      0
BILL_AMT2      0
BILL_AMT3      0
BILL_AMT4      0
BILL_AMT5      0
BILL_AMT6      0
PAY_AMT1       0
PAY_AMT2       0
PAY_AMT3       0
PAY_AMT4       0
PAY_AMT5       0
PAY_AMT6       0
def_value      0
dtype: int64
```

It indicates there is no missing values in this data.

In [19]:
```
#draw the heatmap of the correlation between the variables
plt.subplots(figsize=(30,20))
sns.heatmap(data.corr(),annot=True,cmap='coolwarm')
plt.show()
```



We can see that the correlation between PAY** varaibles, BILL_Amount** is high. Others are low.

In [20]:
```
count_def=data.def_value.value_counts(normalize=True)*100
print(count_def)
```

```
def_value
0    77.88
1    22.12
Name: proportion, dtype: float64
```

We can see that 78% of the data are non_default and 22% of the data are default.

In [21]:
```
data_x=data.drop(['def_value','ID'], axis=1)
data_y=data.def_value

model1_LR = LogisticRegression(max_iter=10000)
fold_10=KFold(n_splits=10,shuffle=True,random_state=40)
results_model1=cross_val_score(model1_LR,data_x,data_y,cv=fold_10)
print("Accuracy: %.2f%% (%.2f%%)" % (results_model1.mean()*100, results_model1.s



##print(classification_report(y_pred, y_test))
#print(sk.metrics.confusion_matrix(y_pred, y_test))
#print('\nAccuracy Score for model1: ', sk.metrics.accuracy_score(y_pred,y_test)
```

```
Accuracy: 77.87% (0.62%)
```

The logistic regression result without feature selection and feature scaling. The 10-fold accuracy is 77.87%

In [22]:
```
model2_xgb=XGBClassifier(max_depth=2, n_estimators=10, learning_rate=0.1, eval_m
results_model2=cross_val_score(model2_xgb,data_x,data_y,cv=fold_10,scoring='accu
print("Accuracy: %.2f%% (%.2f%%)" % (results_model2.mean()*100, results_model2.s
```

```
Accuracy: 81.99% (0.86%)
```

We can see that the result of XGBoost is 81.99%. We will know conduct feature selection and feature engineering like scaling and decoding.

In [23]:
```
#Scale the data to mean=0 and SE=1
scaler=StandardScaler()
data_x_scaled=scaler.fit_transform(data_x)
#no need to do scaling for y because it is category.
```

In [24]:
```
#Use Lasso to do feature selection, first determine the best alpha
lasso_cv=LassoCV(alphas=np.logspace(-4, 3, 10000),cv=10)
lasso_cv.fit(data_x_scaled,data_y)
```

Out[24]:
```
LassoCV(alphas=array([1.00000000e-04, 1.00161327e-04, 1.00322914e-04, ...,
       9.96781250e+02, 9.98389328e+02, 1.00000000e+03]),
        cv=10)
```

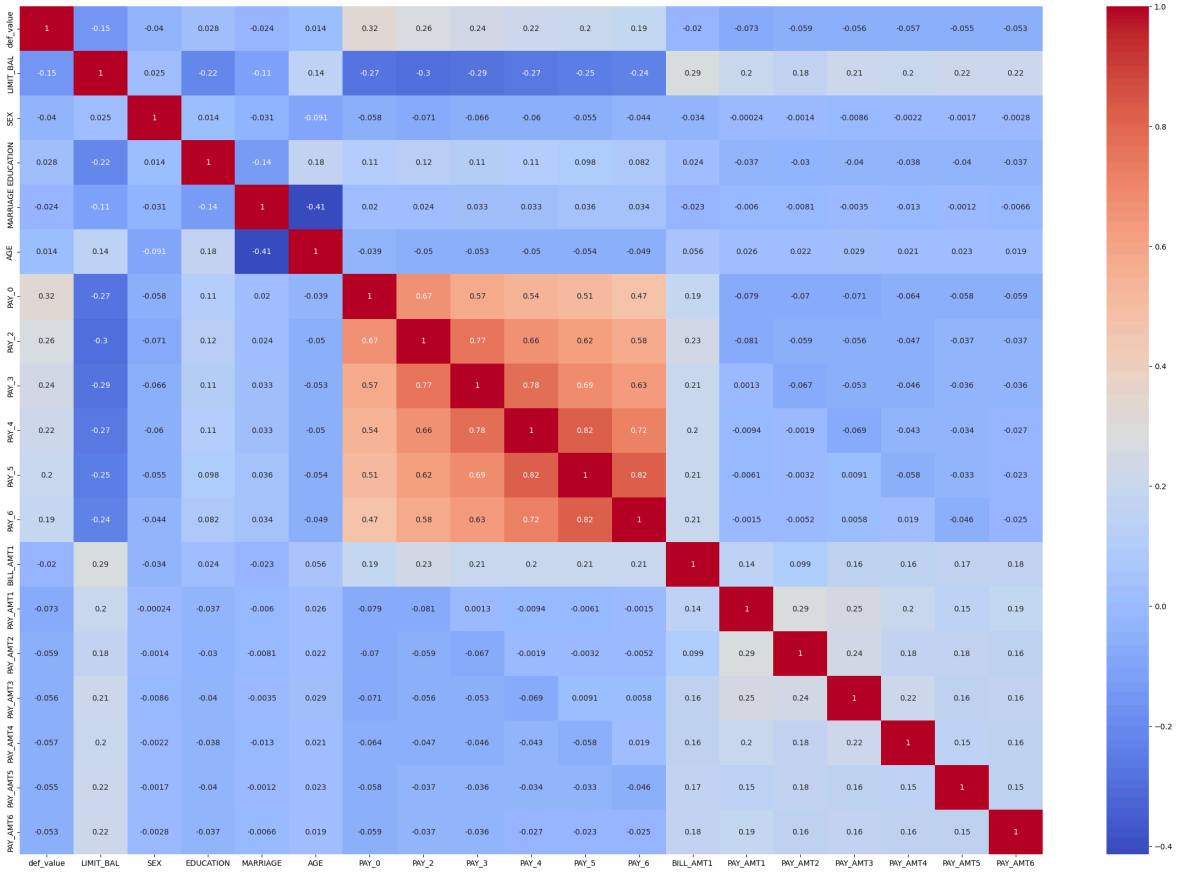
In [25]:
```
feature_mask = lasso_cv.coef_ != 0
selected_features=data_x.columns[feature_mask].tolist()
print(selected_features)
```

```
['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3',
'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_
AMT4', 'PAY_AMT5', 'PAY_AMT6']
```

We can see that the selected feature names by Lasso are 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6'

In [26]:
```
#create new predicator data based on select features
selectedF_data_x=data_x.loc[:,feature_mask]
#plot the correlation between the new variables
combined_data=pd.concat([data_y,selectedF_data_x],axis=1)
plt.subplots(figsize=(30,20))
sns.heatmap(combined_data.corr(),annot=True,cmap='coolwarm')
plt.show()

#scale the new data to mean=0 and SE=1
scaler=StandardScaler()
selectedF_data_x_scaled=scaler.fit_transform(selectedF_data_x)
```



We can see that the correlation between predicators have been reduced. Now we will use selected features to train the model.

In [27]:
```
#XGBoost
model3_xgb=XGBClassifier(max_depth=2, n_estimators=10, learning_rate=0.1, eval_m
results_model3=cross_val_score(model3_xgb,selectedF_data_x_scaled,data_y,cv=fold
print("Accuracy: %.2f%% (%.2f%%)" % (results_model3.mean()*100, results_model3.s
```

```
Accuracy: 81.99% (0.86%)
```

In [28]:
```
#Logistic Regression
start_time=time.perf_counter()
model4_LR = LogisticRegression(max_iter=10000)
fold_10=KFold(n_splits=10,shuffle=True,random_state=40)
results_model4=cross_val_score(model4_LR,selectedF_data_x_scaled,data_y,cv=fold_
end_time=time.perf_counter()
```

```
timecost=end_time-start_time
print("Accuracy: %.2f%% (%.2f%%)" % (results_model4.mean()*100, results_model4.s
print(f"Time cost: {timecost} seconds")
```

```
Accuracy: 80.97% (0.78%)
Time cost: 0.8737618099999054 seconds
```

After conducting the feature selection(by Lasso) and scaling. The accuracy for Losgistic Regression has increased from 77.87% to 80.97%, and that of XGBoost increased from 81.99% to 82.17%. This is mainly because Logistic Regression relay on feature selection but XGBoost can shrink the features by itself, so the feature selection has less effect on XGBoost than Logistic Regression.

Next we will do CV to determine the hyperparameters for XGBoost.

In [33]:
```
#select the hyperparameter of XGBoost
xgb_model_set = XGBClassifier()
param_dist = {
    'max_depth': randint(3,10),
    'learning_rate': uniform(0.01,0.2),
    'n_estimators': randint(100,300),
    'subsample':uniform(0.8,0.2)
}
random_search=RandomizedSearchCV(estimator=xgb_model_set,param_distributions=par
random_search.fit(selectedF_data_x_scaled,data_y)

print(random_search.best_params_)
print(random_search.best_score_)
```

```
Fitting 10 folds for each of 50 candidates, totalling 500 fits
{'learning_rate': 0.041565134097084726, 'max_depth': 3, 'n_estimators': 162, 'sub
sample': 0.9665240310129963}
0.8220333333333334
```

In [38]:
```
best_params=random_search.best_params_
#XGBoost with the best hyperparameters
start_time=time.perf_counter()
model5_xgb=XGBClassifier(max_depth=best_params['max_depth'],n_estimators=best_pa
results_model5=cross_val_score(model5_xgb,selectedF_data_x_scaled,data_y,cv=fold
end_time=time.perf_counter()
timecost=end_time-start_time
print("Accuracy: %.2f%% (%.2f%%)" % (results_model5.mean()*100, results_model5.s
print(f"Time cost: {timecost} seconds")
```

```
Accuracy: 82.19% (0.77%)
Time cost: 5.043828054999722 seconds
```

We can see by using the best hyperparameters the accuracy increased by 0.2%.