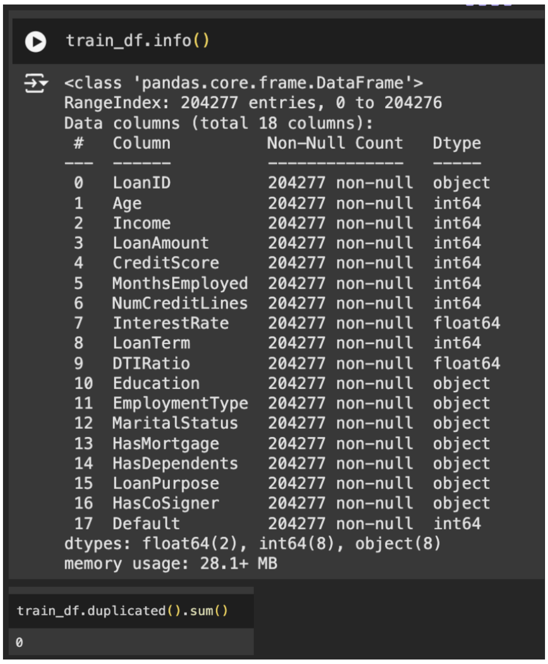
AIM-511: Machine Learning Project Report : Lend or Lose

Team Z *Aditya Saraf*
IMT2022067
Aditya.Saraf@iiitb.ac.in

Preprocessing Steps

Data Cleaning

Data was checked for null values, duplicates, and outliers. None were identified. Exploratory analysis and box plots confirmed the dataset's consistency. The `LoanID` column was dropped as it was a unique identifier with no predictive value, as shown in Figure 1.



```
train_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204277 entries, 0 to 204276
Data columns (total 18 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   LoanID          204277 non-null  object
 1   Age             204277 non-null  int64
 2   Income          204277 non-null  int64
 3   LoanAmount      204277 non-null  int64
 4   CreditScore     204277 non-null  int64
 5   MonthsEmployed  204277 non-null  int64
 6   NumCreditLines  204277 non-null  int64
 7   InterestRate    204277 non-null  float64
 8   LoanTerm        204277 non-null  int64
 9   DTIRatio        204277 non-null  float64
 10  Education       204277 non-null  object
 11  EmploymentType  204277 non-null  object
 12  MaritalStatus   204277 non-null  object
 13  HasMortgage     204277 non-null  object
 14  HasDependents   204277 non-null  object
 15  LoanPurpose     204277 non-null  object
 16  HasCoSigner     204277 non-null  object
 17  Default         204277 non-null  int64
dtypes: float64(2), int64(8), object(8)
memory usage: 28.1+ MB

train_df.duplicated().sum()

0
```

Figure 1: Data Preprocessing: checking for nulls and duplicate values

Encoding Categorical Features

Categorical columns *(Education, EmploymentType, MaritalStatus, LoanPurpose, HasMortgage, HasDependents, HasCoSigner)* were label-encoded to convert them into numerical categories suitable for modeling.

Experimental Design and Feature Selection

Exploratory Data Analysis (EDA)

EDA revealed no significant outliers in features such as *Age, Income, LoanAmount, CreditScore, MonthsEmployed*, etc., based on box plots.

Feature Selection

Feature relevance was assessed using a correlation matrix, identifying key predictors like *Age, Income, MonthsEmployed, InterestRate*, etc. Irrelevant columns were excluded. Correlation of attributes with target value is shown in Figure 3. Attempts to add synthetic features resulted in overfitting and were discarded.

	Default
Age	-0.167484
Income	-0.100515
MonthsEmployed	-0.095429
HasCoSigner	-0.040688
HasDependents	-0.034737
CreditScore	-0.034128
Education	-0.021508
HasMortgage	-0.021174
LoanPurpose	-0.010799
MaritalStatus	-0.008441
LoanTerm	0.000815
DTIRatio	0.018595
NumCreditLines	0.028565
EmploymentType	0.042670
LoanAmount	0.085519
InterestRate	0.129885
Default	1.000000

dtype: float64

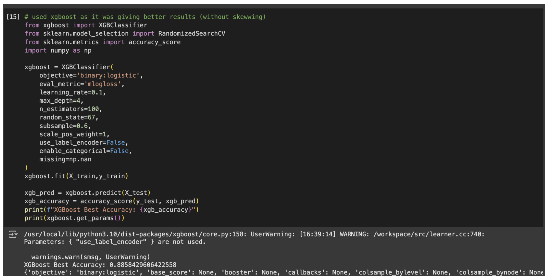
Figure 2: Correlation of attritbutes with targeted value

The dataset had a low defaulter rate ($\leq 5\%$). Following the valuable advice from the Teaching Assistant on skewing techniques, noise addition (ranging from 0.005 to 10) was tested. However, these modifications yielded inferior results compared to the original dataset.

Model Selection and Performance

Chosen Model

`XGBoost` was selected for its superior handling of mixed data types and robustness against overfitting. The model achieved a Kaggle score of **0.88789**. Model parameters is shown in Figure 3.



```
[15] # used xgboost as it was giving better results (without skewwing)
     from xgboost import XGBClassifier
     from sklearn.model_selection import RandomizedSearchCV
     from sklearn.metrics import accuracy_score
     import numpy as np

     xgboost = XGBClassifier(
         objective='binary:logistic',
         eval_metric='mlogloss',
         learning_rate=0.1,
         max_depth=4,
         n_estimators=100,
         random_state=67,
         subsample=0.6,
         scale_pos_weight=1,
         use_label_encoder=False,
         enable_categorical=False,
         missing=np.nan
     )
     xgboost.fit(X_train,y_train)

     xgb_pred = xgboost.predict(X_test)
     xgb_accuracy = accuracy_score(y_test, xgb_pred)
     print(f"XGBoost Best Accuracy: {xgb_accuracy}")
     print(xgboost.get_params())

/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [16:39:14] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
XGBoost Best Accuracy: 0.8858429606422558
{'objective': 'binary:logistic', 'base_score': None, 'booster': None, 'callbacks': None, 'colsample_bylevel': None, 'colsample_bynode': None
```

Figure 3: XGBoost Model used in predicting

Hyperparameter Tuning

`RandomizedSearchCV` optimized hyperparameters efficiently. The final parameters were:

```
{'objective': 'binary:logistic', 'eval_metric': 'mlogloss',
'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 100,
'random_state': 67, 'subsample': 0.6, 'scale_pos_weight': 1,
'use_label_encoder': False}
```

Evaluation

K-Fold Cross-Validation ensured robust evaluation. `XGBoost` outperformed other models like Random Forest, Decision Tree, Logistic Regression, AdaBoost, ANN, and SVM in terms of accuracy and generalizability.

Additional Insights

Outlier Treatment

No significant outliers were detected in the dataset.

Insights on Defaulters

Defaulters typically had:
Lower income, younger age, higher loan amounts, and higher interest rates compared to non-defaulters.

Feature Encoding Strategy

Through the above analysis, feature addition was performed using the mentioned parameters as in Figure 4:

```
df['LoanToIncomeRatio'] = df['LoanAmount'] / df['Income']
df['CreditAgeInteraction'] = df['CreditScore'] * df['Age']
df['LoanInterestInteraction'] = df['LoanAmount'] * df['InterestRate']
df['DTIIncomeInteraction'] = df['DTIRatio'] * df['Income']
```

Figure 4: Feature addition techniques

Model Comparison

However, it was found that this led to a downgrade in performance, as it might have caused overfitting of features. This was observed in the additional section and in the Kaggle submissions, where the performance could not outperform the mode used in this.

Label encoding was employed to convert categorical data into numerical form suitable for tree-based models.

Model	**Noise (Skewing)**	**Accuracy**	**Parameters**
Decision Tree	No	0.80313	`{criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, ccp_alpha=0.0}`
Decision Tree	Yes	0.80252	`{criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, ccp_alpha=0.0}`
AdaBoost	No	0.88511	`{base_estimator=None, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None}`

Continued on next page

Model	Noise (Skewing)	Accuracy	Parameters
AdaBoost	Yes (Noise = 1)	0.88314	{`base_estimator=None, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None`}
XGBoost	No	**0.88789**	*Optimized Parameters (as specified earlier)*
XGBoost	Yes (Noise = 0.1)	0.8866	*Optimized Parameters (as specified earlier)*
Logistic Regression	No	0.88541	{`penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None`}
ANN	No	0.88447	`As mentioned in notebook`
ANN	Yes (Noise = 1)	0.87523	`as mentioned in notebook`
SVM	No	0.88709	{`C=1.0, kernel='linear',gamma='auto'`}

Conclusion

The `XGBoost` model, with optimized hyperparameters and no skewing, achieved the best performance. Its ability to generalize and accurately predict the target variable makes it the optimal choice for this dataset.

Acknowledgments

I would like to extend my special thanks to the Teaching Assistant, Sathvik Bhat for providing valuable advice on skewing techniques. Their guidance helped me gain a better understanding of the data and improve the training of the model. This insight significantly contributed to the quality of the results in this project.

References

- XGBoost Documentation: `https://xgboost.readthedocs.io/en/stable/`
- Scikit-learn Documentation: `https://scikit-learn.org/stable/`

- RandomizedSearchCV: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
- ANN Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
- SVM Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- Decision Tree Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- AdaBoost Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
- Logistic Regression Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- Kaggle: https://www.kaggle.com/