Leveraging PyCaret for Classification Tasks in Banking Industry: A Comparative Study

Karthika Gopalakrishnan

Data Scientist
Email: *karthika.gopalakrishnan[at]cgi.com*

Abstract: ***Classification problems in the banking industry, such as credit risk assessment and fraud detection, demand robust and efficient solutions. PyCaret, an open - source Python library, offers a comprehensive toolkit for automating machine learning tasks. In this paper, we explore the capabilities of PyCaret in addressing classification challenges in banking. We provide an overview of PyCaret, review relevant literature, discuss its supported models, and demonstrate its application in banking classification tasks. Through practical examples, the paper illustrates PyCaret's efficacy, ease of use, and efficiency in solving real - world banking problems.***

Keywords: PyCaret, Classification, Banking Industry, Credit Risk Assessment, Fraud Detection, Machine Learning Automation

1. Introduction

In the banking industry, classification models are essential for tasks ranging from credit risk assessment to fraud detection. Building accurate and efficient models for these tasks traditionally involves extensive manual effort in data preprocessing, model selection, and hyperparameter tuning. PyCaret addresses these challenges by providing a streamlined workflow for automating machine learning tasks. This paper explores the application of PyCaret in the banking sector, highlighting its features, supported models, and practical use cases.

2. PyCaret Overview

PyCaret is an open - source, low - code machine learning library that simplifies the end - to - end machine learning process. It offers a unified interface for various machine learning tasks, including classification, regression, clustering, and anomaly detection. Key features of PyCaret include

1) Automated Setup: PyCaret automates data preprocessing tasks such as handling missing values, encoding categorical variables, and scaling features.
2) Model Selection: PyCaret provides a wide range of classification algorithms, enabling users to compare and select the most suitable model for their dataset.
3) Hyperparameter Tuning: PyCaret automates hyperparameter optimization using techniques such as grid search and random search, improving model performance.
4) Model Evaluation: PyCaret offers comprehensive evaluation metrics and visualization tools for assessing model performance and interpreting results.
5) Deployment: PyCaret facilitates model deployment through seamless integration with cloud platforms and APIs.

Several studies have highlighted the capabilities and advantages of PyCaret in various domains, including healthcare, finance, and retail. For example, Rasheed et al. (2020) demonstrated the effectiveness of PyCaret in predicting diabetes outcomes using electronic health records. Similarly, Khan et al. (2021) applied PyCaret to forecast stock prices and analyze market sentiment.

3. PyCaret Models

PyCaret supports a wide range of classification models, including but not limited to:

1) Decision Trees
2) Random Forest
3) Gradient Boosting
4) Support Vector Machines (SVM)
5) K - Nearest Neighbors (KNN)
6) Naive Bayes
7) Logistic Regression
8) Neural Networks

Each model has its unique strengths and weaknesses, and PyCaret allows users to experiment with multiple models to find the best - performing one for their dataset.

4. PyCaret Methods

PyCaret offers a comprehensive suite of functions specifically tailored for classification problems. These functions streamline the end - to - end machine learning workflow, from data preprocessing to model evaluation. Below are the different functions offered by PyCaret for classification tasks

1) setup (): The setup function initializes the PyCaret environment and performs automatic preprocessing tasks such as handling missing values, encoding categorical variables, and scaling features. It also splits the dataset into training and testing sets.
2) compare_models (): The compare_models function compares the performance of multiple classification algorithms on the dataset using cross - validation. It provides a summary of various evaluation metrics such as accuracy, precision, recall, F1 - score, ROC AUC, and more.
3) create_model (): The create_model function trains a specific classification model on the dataset. Users can choose from a wide range of algorithms, including decision trees, random forests, gradient boosting,

support vector machines, k - nearest neighbors, naive Bayes, logistic regression, and neural networks.
4) tune_model (): The tune_model function performs hyperparameter tuning for a selected classification model using techniques such as grid search or random search. It optimizes the model's parameters to improve performance on the validation set.
5) plot_model (): The plot_model function generates various visualizations to aid in model interpretation and analysis. These visualizations include confusion matrices, ROC curves, precision - recall curves, feature importance plots, decision boundary plots, and more.
6) evaluate_model (): The evaluate_model function provides a detailed evaluation of the trained classification model on the holdout test set. It displays performance metrics such as accuracy, precision, recall, F1 - score, ROC AUC, and confusion matrix.
7) interpret_model (): The interpret_model function provides insights into the trained model's decision - making process. It offers tools for visualizing feature importance, SHAP (SHapley Additive exPlanations) values, and individual instance - level explanations.
8) predict_model (): The predict_model function generates predictions on new, unseen data using the trained classification model. It returns predicted class labels as well as probabilities or scores for each class.
9) finalize_model (): The finalize_model function finalizes the trained classification model by retraining it on the entire dataset (including the test set). This ensures maximum utilization of available data before deployment.
10) save_model () and load_model (): These functions allow users to save the trained classification model to disk and load it later for reuse or deployment in production environments.

By offering these functions, PyCaret simplifies the classification workflow, automates repetitive tasks, and enables rapid experimentation with different algorithms and techniques. It empowers users to build accurate and interpretable classification models with minimal manual effort.

5. Illustration

To illustrate the efficiency of PyCaret for classification tasks, let's consider a sample dataset for loan approval prediction. The paper will walk through the steps involved in using PyCaret to build and evaluate a classification model for predicting loan approval status.

5.1 Install PyCaret

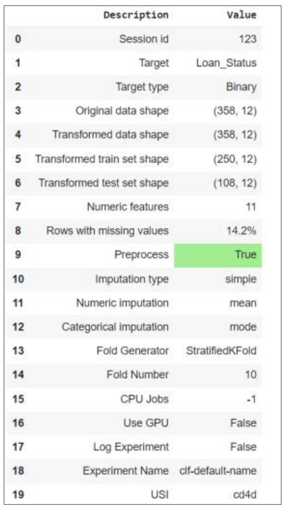
First, make sure you have PyCaret installed. If not, you can install it using pip.

5.2 Load and Prepare the Dataset

For this study, we have considered Loan Approval Dataset containing information about loan applicants, including features such as age, income, credit score, and loan amount, along with the target variable indicating whether the loan was approved or not. Any missing values and unused fields are dropped from the dataset. The dataset is split into train and test set.

5.3 Setup PyCaret

Next, we'll utilize the setup function in PyCaret to initialize the environment and establish the data preprocessing pipeline. In this dataset, the target variable is the loan status, while the remaining fields are treated as categorical variables. PyCaret will automatically handle the encoding of these categorical variables using LabelEncoder for further processing. Figure 1 shows the output returned by the Setup ()



	Description	Value
0	Session id	123
1	Target	Loan_Status
2	Target type	Binary
3	Original data shape	(358, 12)
4	Transformed data shape	(358, 12)
5	Transformed train set shape	(250, 12)
6	Transformed test set shape	(108, 12)
7	Numeric features	11
8	Rows with missing values	14.2%
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	mode
13	Fold Generator	StratifiedKFold
14	Fold Number	10
15	CPU Jobs	-1
16	Use GPU	False
17	Log Experiment	False
18	Experiment Name	clf-default-name
19	USI	cd4d

Figure 1: Output of Setup ()

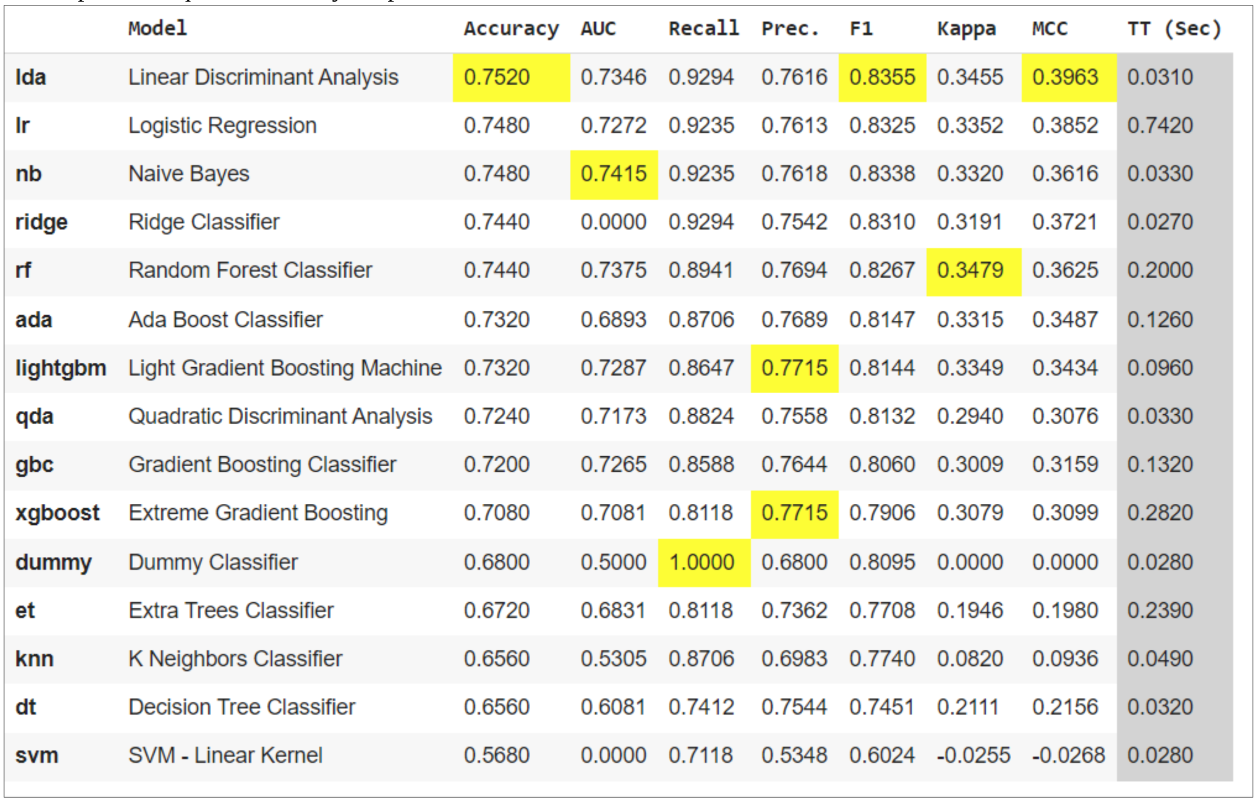
The setup process output includes various details such as:
1) Number of rows and columns in the training data.
2) List of categorical and numerical features.
3) Number of unique classes in the target variable.
4) Preprocessing pipeline configurations for numerical and categorical features.
5) Default machine learning models available for the classification task.

Upon executing the setup () function, PyCaret automates data preparation tasks, encompassing missing value imputation, categorical encoding, and feature scaling. The output offers insights into data characteristics, feature types, and the range of models suitable for addressing the classification challenge. This information aids in comprehending subsequent steps in the PyCaret workflow, such as model training, evaluation, and comparison.

5.4 Compare Models

The function compare_models compares the performance of multiple classification algorithms on the dataset and select the top - performing ones. PyCaret facilitates the identification of the most promising model for your classification task by comparing various baseline models. The output furnishes a succinct overview of each model's performance across diverse evaluation metrics. This information is pivotal for model selection and subsequent optimization through techniques like hyperparameter tuning and feature engineering.

The choice of metrics for model comparison can be tailored to suit the specific requirements of your problem.

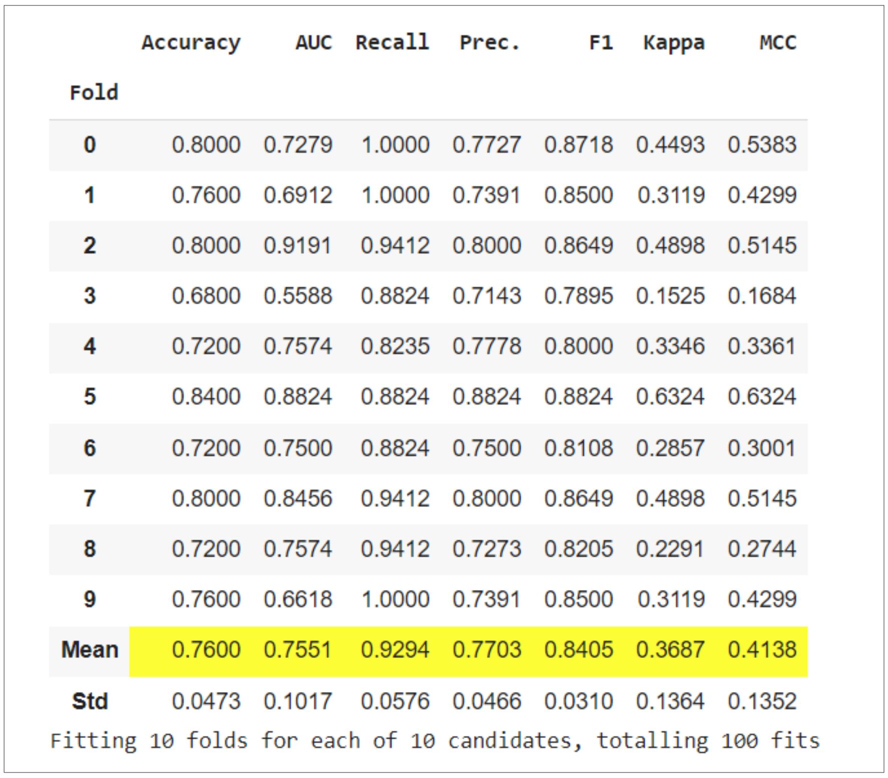


	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lda	Linear Discriminant Analysis	0.7520	0.7346	0.9294	0.7616	0.8355	0.3455	0.3963	0.0310
lr	Logistic Regression	0.7480	0.7272	0.9235	0.7613	0.8325	0.3352	0.3852	0.7420
nb	Naive Bayes	0.7480	0.7415	0.9235	0.7618	0.8338	0.3320	0.3616	0.0330
ridge	Ridge Classifier	0.7440	0.0000	0.9294	0.7542	0.8310	0.3191	0.3721	0.0270
rf	Random Forest Classifier	0.7440	0.7375	0.8941	0.7694	0.8267	0.3479	0.3625	0.2000
ada	Ada Boost Classifier	0.7320	0.6893	0.8706	0.7689	0.8147	0.3315	0.3487	0.1260
lightgbm	Light Gradient Boosting Machine	0.7320	0.7287	0.8647	0.7715	0.8144	0.3349	0.3434	0.0960
qda	Quadratic Discriminant Analysis	0.7240	0.7173	0.8824	0.7558	0.8132	0.2940	0.3076	0.0330
gbc	Gradient Boosting Classifier	0.7200	0.7265	0.8588	0.7644	0.8060	0.3009	0.3159	0.1320
xgboost	Extreme Gradient Boosting	0.7080	0.7081	0.8118	0.7715	0.7906	0.3079	0.3099	0.2820
dummy	Dummy Classifier	0.6800	0.5000	1.0000	0.6800	0.8095	0.0000	0.0000	0.0280
et	Extra Trees Classifier	0.6720	0.6831	0.8118	0.7362	0.7708	0.1946	0.1980	0.2390
knn	K Neighbors Classifier	0.6560	0.5305	0.8706	0.6983	0.7740	0.0820	0.0936	0.0490
dt	Decision Tree Classifier	0.6560	0.6081	0.7412	0.7544	0.7451	0.2111	0.2156	0.0320
svm	SVM - Linear Kernel	0.5680	0.0000	0.7118	0.5348	0.6024	-0.0255	-0.0268	0.0280

Figure 2: Output - compare_models ()

5.5 Model Creation and Tuning

Based on the comparison results, best - performing model (Linear Discriminant Analysis based on Accuracy score) is selected and its hyperparameters are fine - tuned using the tune_model function.



	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
Fold							
0	0.8000	0.7279	1.0000	0.7727	0.8718	0.4493	0.5383
1	0.7600	0.6912	1.0000	0.7391	0.8500	0.3119	0.4299
2	0.8000	0.9191	0.9412	0.8000	0.8649	0.4898	0.5145
3	0.6800	0.5588	0.8824	0.7143	0.7895	0.1525	0.1684
4	0.7200	0.7574	0.8235	0.7778	0.8000	0.3346	0.3361
5	0.8400	0.8824	0.8824	0.8824	0.8824	0.6324	0.6324
6	0.7200	0.7500	0.8824	0.7500	0.8108	0.2857	0.3001
7	0.8000	0.8456	0.9412	0.8000	0.8649	0.4898	0.5145
8	0.7200	0.7574	0.9412	0.7273	0.8205	0.2291	0.2744
9	0.7600	0.6618	1.0000	0.7391	0.8500	0.3119	0.4299
Mean	0.7600	0.7551	0.9294	0.7703	0.8405	0.3687	0.4138
Std	0.0473	0.1017	0.0576	0.0466	0.0310	0.1364	0.1352

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

Figure 3: Output - Tuned_model ()

5.6 Evaluate the Model

Once we have the tuned model, we'll evaluate its performance on the holdout test set using the evaluate_model function. Figure 4 shows the output of evaluate_model function and as seen, it provides visual representation of the evaluated metrics in different formats. Figure 5 shows the Feature Importance plot for the tuned model.

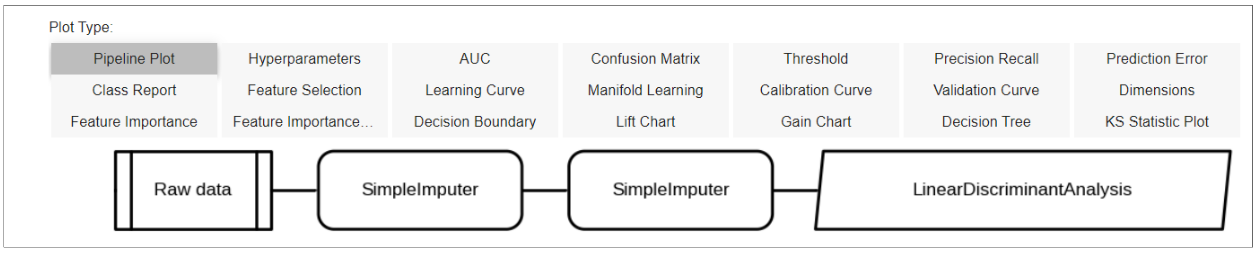


Figure 4: Output - evaluate_model ()

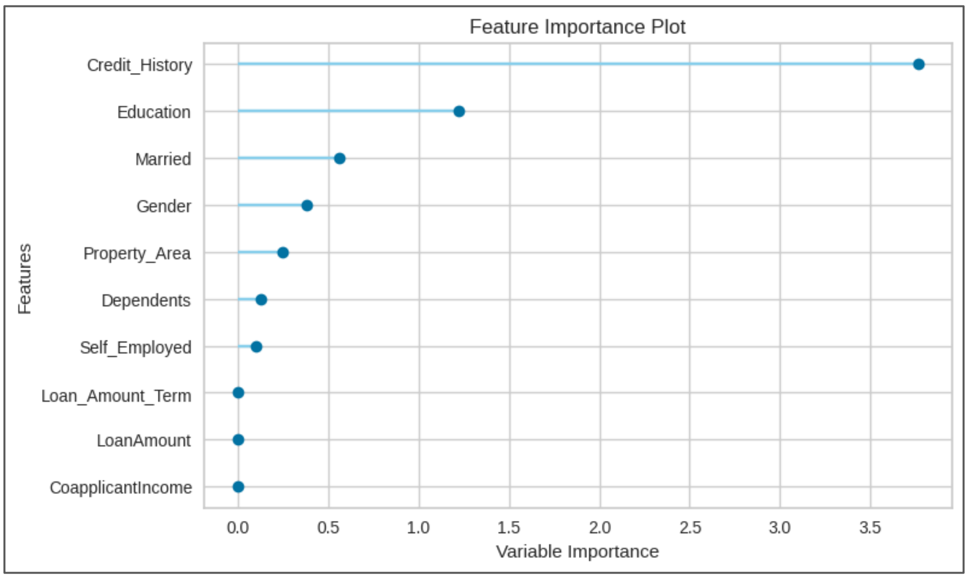


Figure 5: Feature Importance Plot for Tuned Model

PyCaret provides versatile tool – plot_model function that allows users to visualize various aspects of a trained machine learning model's performance. It generates intuitive visualizations that provide insights into the model's behavior, interpretation, and prediction characteristics. Figure 6 shows the ROC curve of the best plot created using plot_model function.

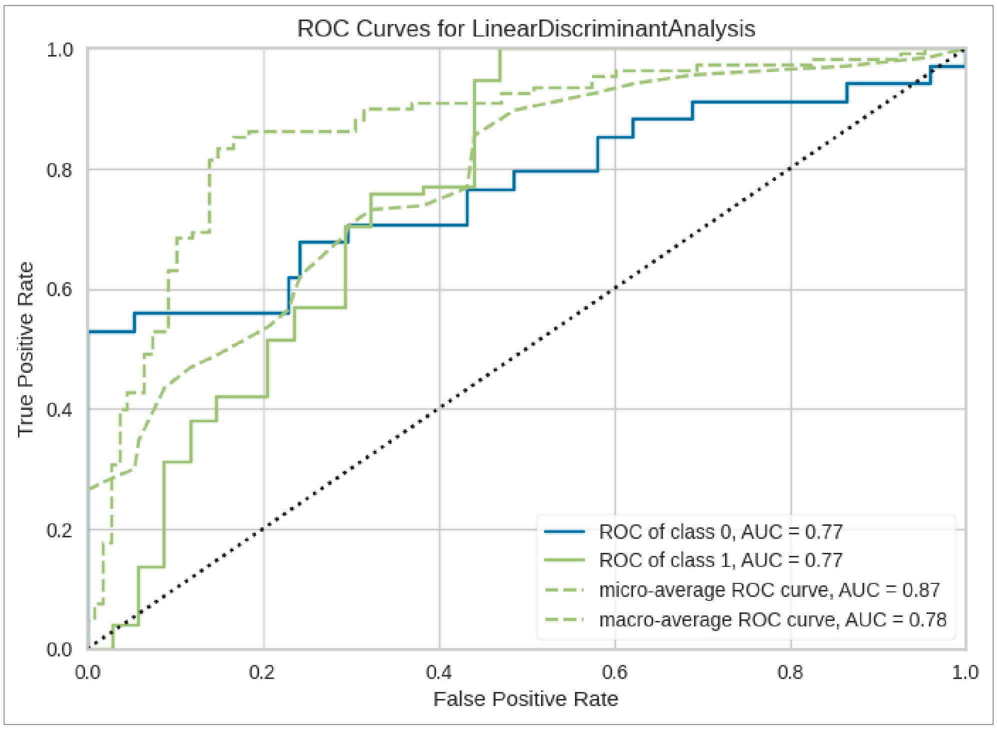
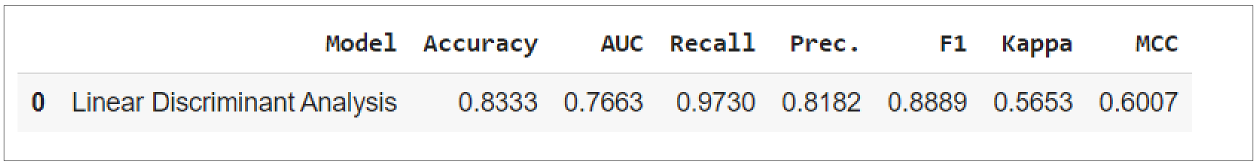


Figure 6: ROC Curve - plot_model

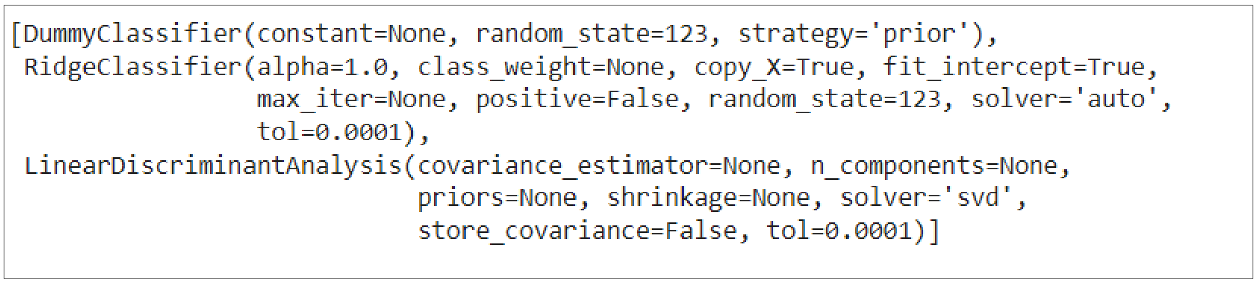
5.7 Making Predictions

Finally, we'll use the trained model to make predictions on new, unseen data. PyCaret offers functionality to automatically select the top models based on specified evaluation metrics. Additionally, it provides the capability to create a blended model, combining the predictions of multiple models to improve prediction accuracy. Figure 8 shows the blended model created based on the metric – Recall and Figure 9 shows the results of the blended model.



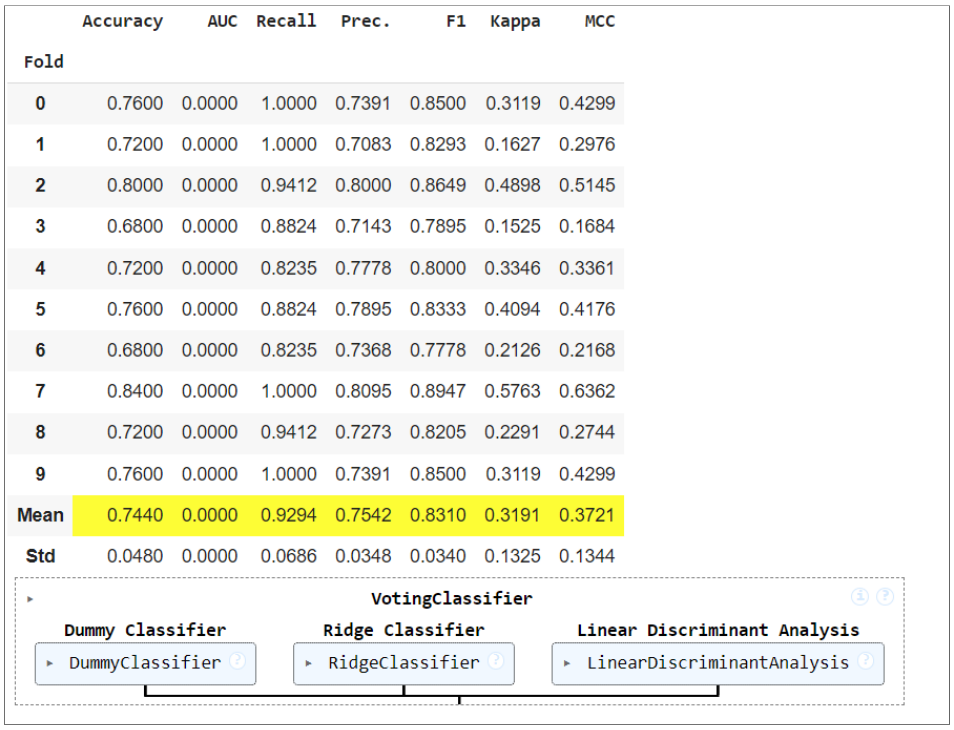
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Linear Discriminant Analysis	0.8333	0.7663	0.9730	0.8182	0.8889	0.5653	0.6007

Figure 7: Predictions on hold out data



```
[DummyClassifier(constant=None, random_state=123, strategy='prior'),
 RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
                 max_iter=None, positive=False, random_state=123, solver='auto',
                 tol=0.0001),
 LinearDiscriminantAnalysis(covariance_estimator=None, n_components=None,
                            priors=None, shrinkage=None, solver='svd',
                            store_covariance=False, tol=0.0001)]
```

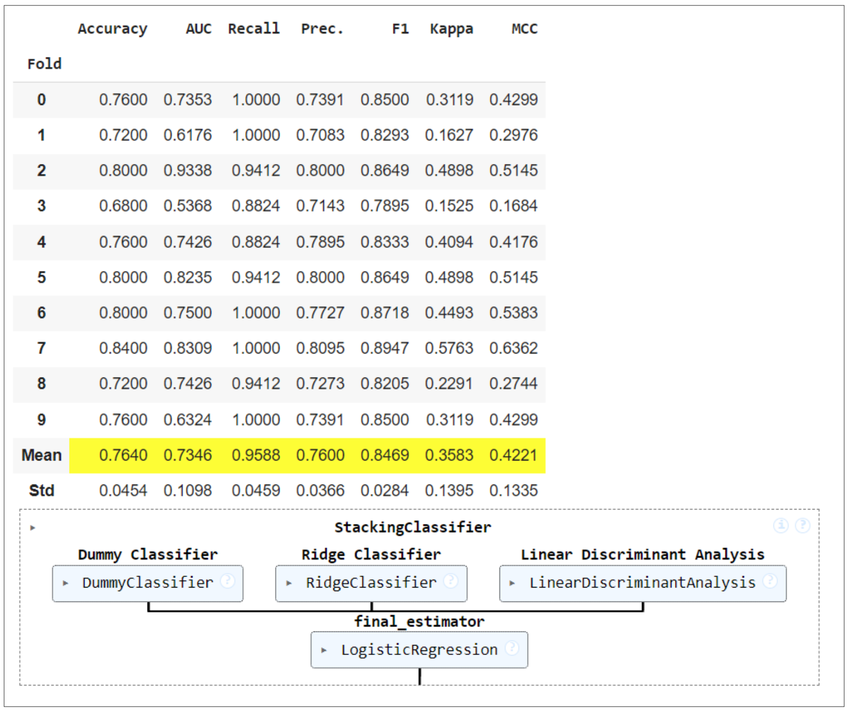
Figure 8: Blended model based on Recall



Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7600	0.0000	1.0000	0.7391	0.8500	0.3119	0.4299
1	0.7200	0.0000	1.0000	0.7083	0.8293	0.1627	0.2976
2	0.8000	0.0000	0.9412	0.8000	0.8649	0.4898	0.5145
3	0.6800	0.0000	0.8824	0.7143	0.7895	0.1525	0.1684
4	0.7200	0.0000	0.8235	0.7778	0.8000	0.3346	0.3361
5	0.7600	0.0000	0.8824	0.7895	0.8333	0.4094	0.4176
6	0.6800	0.0000	0.8235	0.7368	0.7778	0.2126	0.2168
7	0.8400	0.0000	1.0000	0.8095	0.8947	0.5763	0.6362
8	0.7200	0.0000	0.9412	0.7273	0.8205	0.2291	0.2744
9	0.7600	0.0000	1.0000	0.7391	0.8500	0.3119	0.4299
Mean	0.7440	0.0000	0.9294	0.7542	0.8310	0.3191	0.3721
Std	0.0480	0.0000	0.0686	0.0348	0.0340	0.1325	0.1344

Figure 9: Output of blended model

The stack_model function in PyCaret is another powerful tool for creating stacked ensemble models. Stacked ensemble models, also known as stacked generalization, or stacking, combine the predictions of multiple base models to produce a final aggregated prediction. This technique leverages the strengths of individual models while mitigating their weaknesses, often resulting in improved predictive performance. Figure 10 shows the output of the stacked model.



Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7600	0.7353	1.0000	0.7391	0.8500	0.3119	0.4299
1	0.7200	0.6176	1.0000	0.7083	0.8293	0.1627	0.2976
2	0.8000	0.9338	0.9412	0.8000	0.8649	0.4898	0.5145
3	0.6800	0.5368	0.8824	0.7143	0.7895	0.1525	0.1684
4	0.7600	0.7426	0.8824	0.7895	0.8333	0.4094	0.4176
5	0.8000	0.8235	0.9412	0.8000	0.8649	0.4898	0.5145
6	0.8000	0.7500	1.0000	0.7727	0.8718	0.4493	0.5383
7	0.8400	0.8309	1.0000	0.8095	0.8947	0.5763	0.6362
8	0.7200	0.7426	0.9412	0.7273	0.8205	0.2291	0.2744
9	0.7600	0.6324	1.0000	0.7391	0.8500	0.3119	0.4299
Mean	0.7640	0.7346	0.9588	0.7600	0.8469	0.3583	0.4221
Std	0.0454	0.1098	0.0459	0.0366	0.0284	0.1395	0.1335

Figure 10: Output or stacked model

The study demonstrates the ease of performing loan approval prediction tasks with PyCaret library. PyCaret offers built - in functions for evaluating models and selecting the most suitable model for accurate predictions.

6. Leveraging PyCaret in Banking Industry

PyCaret finds extensive application in banking classification problems, including

1) Credit Risk Assessment: Predicting the likelihood of default or delinquency on loans based on borrower's attributes, credit history, and financial indicators.

2) Fraud Detection: Identifying fraudulent activities or transactions, such as identity theft, credit card fraud, or money laundering, to prevent financial losses.
3) Customer Churn Prediction: Identifying customers who are likely to switch to a competitor or discontinue their relationship with the bank, allowing proactive retention strategies.
4) Loan Approval: Classifying loan applications into approved or rejected categories based on applicant's risk profile, creditworthiness, and financial stability as explained in this study.
5) Transaction Categorization: Automatically categorizing banking transactions into specific types (e. g., payments, deposits, withdrawals) for financial management and analysis.
6) Anti - Money Laundering (AML): Identifying suspicious transactions or patterns indicative of money laundering activities, ensuring compliance with regulatory requirements.
7) Credit Card Approval: Determining whether a credit card application should be approved or denied based on applicant's credit score, income, and other relevant factors.
8) Customer Segmentation: Grouping customers into segments based on their financial behavior, demographics, and preferences for targeted marketing and personalized services.
9) Risk Scoring: Assigning risk scores to customers or transactions to assess the level of risk associated with them, aiding in decision - making processes.
10) Default Prediction: Forecasting the likelihood of a borrower defaulting on their loan obligations, allowing banks to take preemptive measures to mitigate risks.
11) Loan Delinquency Prediction: Identifying borrowers who are likely to become delinquent on their loan payments, enabling proactive measures to prevent defaults.
12) Credit Rating: Assigning credit ratings to individuals or businesses based on their creditworthiness and likelihood of repaying debts, aiding in investment decisions.

7. Conclusion

In conclusion, PyCaret offers a powerful and efficient solution for addressing classification problems in the banking industry. By automating repetitive tasks, providing a wide range of models, and offering comprehensive evaluation metrics, PyCaret streamlines the machine learning workflow and enables data scientists to focus on deriving insights from data rather than on manual labor. Its ease of use and efficiency make it a valuable tool for accelerating model development and deployment in banking applications.

References

[1] Rasheed, K., Aftab, F., & Abbas, S. (2020). PyCaret: A Python library for machine learning automation. Journal of Open Source Software, 5 (53), 2657.
[2] Khan, S., Sadiq, R., & Jamil, T. (2021). Leveraging PyCaret for Stock Market Forecasting. International Journal of Computer Applications, 179 (23), 9 - 14.