Detailed Rules:

Gordon Dou: Data preprocessing, K-fold validation, and metrics calculation.

Alicia Sheng: Data encoding, model training structure, and coding formats.

Frank Liu: Data encoding, data preprocessing and feature importance

Data Encoding:

After importing the data from CSV format, we first dropped the "CLIENTNUM" column since it is not meaningful for the entire process. We process the non-integer features in two ways: hierarchical and non-hierarchical.

Hierarchical Integer Encoding

For string attributes with hierarchical meanings, we encoded them with integer encoding. For example, we believe income level has a meaning with the data scales, where a higher number means a higher level of income, so we mapped the "Income_Category" attribute from "less than $40K" to "$120K +" as 0 to 4. We also did this mapping to attributes of "Card_Category" and "Education Level."

One-hot Encoding

For string attributes with non-hierarchical meanings, we encoded them with one-hot encoding. For example, we believe females and males are at the same level, so we encoded the "Gender" attribute with one-hot encoding, where the "Gender" column was mapped to two columns of zeros (false) and ones (true), where the first column represents whether the person is male, and the other column represents whether the person is female. We also applied one-hot encoding to "Marital_Status".

Binary Encoding

For the target column "Attrition_Flag", we encoded it as zeros and ones, where zeros represent "Existing Customer" as the negative class, and ones represent "Attrited Customer" as the positive class.

Missing Data Replacement

We tried two different encoding methods for the "Unknown" data. For the first encoding method, we replace the unknown data with the mean value of that column. We use the iterative Imputor method in the SKlearn package for the second encoding method to replace the missing values. After testing the two approaches, we found that the first method led to a relatively better result, **so we finalized our encoding method as the first one, where we replaced the missing values with the mean value of that attribute.**

Data Preprocessing:

After we replace the unknown data points, we first split the data into X and y, in which X is all the features of each corresponding customer while y is the status of the customer. Then we use **labelEncoder** to transform target values into integers (existing customer = 0 and attrited customer = 1). The data was prepossessed by **StandardScaler** to remove the mean and scale to unit variance. We also check the correlation between each feature using the correlation matrix. The columns with a correlation greater than 90% are eliminated to reduce the noise of the prediction.

Identify Feature importance:

This section creates two helper methods to assist in identifying each classification method's feature importance implemented later in the program. There is a corresponding score for each evaluated feature to represent the level of importance. The higher the score is, the more helpful the feature is during the prediction process. On the other hand, the smaller the score is, the less relevant (or possible noise) during prediction. The feature_importance(name, model) method is used to fit the model and then analyze the feature importance of each feature by using the feature_importances_ function in some classification methods (Decision Tree, Random Forest, XGboost, and Adaboost). The case is a little different for SVM (support vector machine) since it needs to evaluate the model's coefficient. In addition, there is no feature_importances_ function for the bagging classifier since it can be used with many different base estimators. The importance score of each classification method is recorded in a dictionary for further analysis. A bar chart is later displayed to give a better sense of the situation. The second method is used to record each classification method's top three highest and lowest features (with detailed name).

Classification process:

We have used six different classifiers to evaluate their performances in this project. They are XGBoost, Decision tree, Support Vector Machine, bagging, random forest, and Adaboost. Each model and its corresponding classifier models are recorded in two separate lists. They are linked together using the zip() function and stored in a variable. A nested for loop is executed in the next part. At the beginning of the outer loop, it runs the feature importance function demonstrated from the previous section and records each classifier's highest/lowest feature. The inner loop mainly runs the stratified Kfold (with n_splits = 5 and Shuffle = True) 20 times. It will first print out the average and the standard deviation of the cross-validation score. Next, cross_val_predict with the method of 'predict_proba' was used to perform a prediction to the y column. Then, precision-recall curves are produced by applying the method precision_recall_curve to y and the predicted y. This step is to compute precision-recall pairs for different probability thresholds. Each classifier's area under the curve (AUC) is created and added to the corresponding dictionary, which provides an aggregate measure of performance across all possible classification thresholds. Then for the outer loop, we print the confusion matrix. In this way, we can present the overall prediction results of each corresponding method. The matrix displays the number of correct and incorrect predictions of each classification method with count values.

Next, the precision, recall, and f1_score are calculated by using the following formulas:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

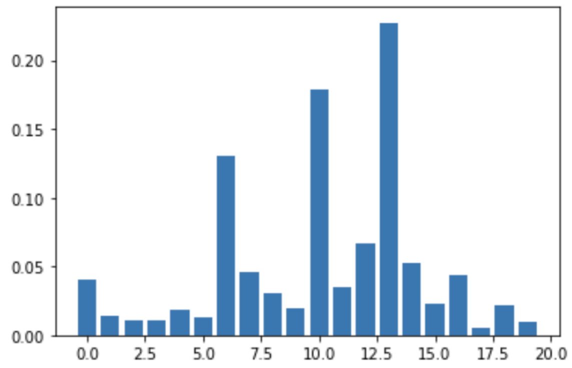
$$\text{F1} = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

All three scores are later printed. Afterward, the Receiver Operating Characteristic curve (ROC) between variables predicted y using the 'predict_proba' method and without. The accuracy score is obtained by calculating the average of the cross-validation score from the 20 times stratified k-fold of each classifier. (So does the standard deviation for each classifier)

Lastly, each classifier's precision, recall, and thresholds are derived by applying the precision_recall_curve method to y and the predicted y. Below is the best performance classifier (XGboost); therefore, we have finalized our classifier with the XGBoost.

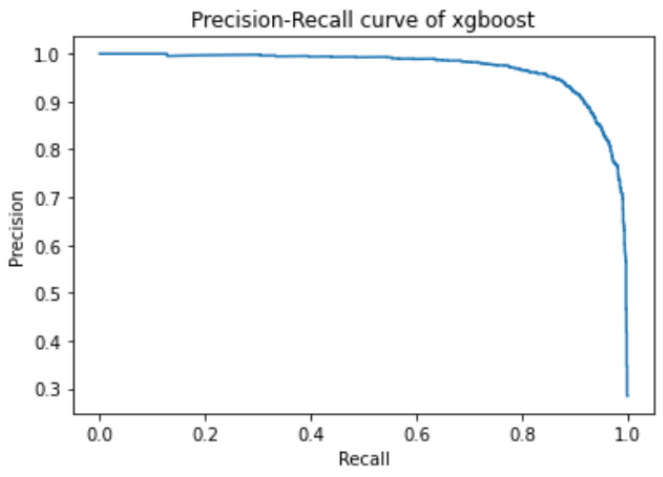
Our final results are as follows:

```
The model running is: xgboost
```



```
The most important feature is Total_Trans_Ct with the importance of 0.22733772

The second most important feature is Total_Revolving_Bal with the importance of 0.17915079

The third most important feature is Total_Relationship_Count with the importance of 0.13029319

The least important feature is Marital_Status_Divorced with the importance of 0.0051091807

The second least important feature is Marital_Status_Single with the importance of 0.010091331

The third least important feature is Income_Category with the importance of 0.010648648
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      8500
           1       0.93      0.89      0.91      1627

    accuracy                           0.97     10127
   macro avg       0.96      0.94      0.95     10127
weighted avg       0.97      0.97      0.97     10127

                 Predicted Negative  Predicted Positive
Actual Negative                8397                 103
Actual Positive                 179                1448

The FN is 179
The tn is 8397
The tp is 1448
The fp is 103

The precision is: 0.9335912314635719
The recall is: 0.8899815611555009
The f1 score is: 0.9112649465072372


The area under the Receiver Operating Characteristic curve is: 0.9389319570483387

The averagve area under the precision-recall curve is 0.9705125996354802 with std of 0.001077541707
2242388

The average accuracy is 0.9710675538980903 with the std of 0.002467415592803737
```



	Precision Recall Curve	Standard deviation of PR curve	Accuracy	Standard deviation of Accuracy
xg boost	0.9695249731709	0.001477083312555	0.971729048298	0.00266622112

	727	4684	0513	10485274
Decision Tree	0.8199890434958746	0.00498188698461395	0.9377948935444165	0.004531802377660425
SVM Linear	0.7594442003901671	0.0008168368859518799	0.9055593531010443	0.004924989098528238
Random Forest	0.953026288832947	0.0010777477383506673	0.9624863929411477	0.003726703205118494
Bagging	0.9409723435817433	0.001651812965024977	0.9616124261148281	0.0037029746052379088
Adaboost	0.9464196379483715	0.0027381822426395864	0.9627729296917845	0.003606848614769461

By looking at the feature importance, we notice that the three most relevant features contributing to the targets are total trans count, total revolving balance, and total relationship count. The least relevant features are marital status, income, education level, month on book, and card category.

In conclusion, the amount of total transactions, the revolving balance of the bank account, and the number of total relationship counts are three key factors that determine if the customer is an existing customer or attrited customer.

Reflection

From this project, we've learned much stuff. First of all, we have learned how to use stratified k-fold to minimize the bias of imbalanced data. Also, in past assignments, we rarely plot graphs or diagrams about the performance of one classifier. We noticed that it is an excellent way to see something peculiar about the performance. We've also learned a new way to deal with "unknown" or NaN data using the impute and the Iterative Imputor method. Both of them can provide a reasonable estimation of the unknown data point based on the result of other known data points. We also learned the difference between each classifier and how preprocessing methods affect the results.

Project Github Link & Readme

https://github.com/franciscoliu/Customer_churning