Lawrence Technological University
Department of Math and Computer Science
MCS 5623 Machine Learning
Assignment # 04
Michael Giuliani
10/27/2024

I. INTRODUCTION & DOMAIN KNOWLEDGE

In this assignment, a dataset from University of California Irvine (UCI), called “Default of Credit Card Clients” [1], is used to write two different methods of predicting if a given client will default on payment of a credit card bill. The characteristics of the data are analyzed, the preprocessing required to make the data useful is determined, insights are drawn about what data points may or may not be useful in achieving the prediction goals, and visualizations of the data are explored to gain a better understanding of what relationships can be found in the data to make better predictions.

To make the predictions, two methods are explored: the logistic regression classifier, and the random forest ensemble classifier. A logistic regression classifier attempts to identify two classes from each other by calculating the probability that an input matches one of the classes. The probability is based on a linear combination of the input’s features and the sigmoid function [2]. A random forest ensemble classifier builds a group of decision trees based on randomly selected samples and features in the dataset, where the prediction is the averaged result of all trees [3]. These methods are compared and contrasted to see how their performance compares, what they do better or worse, and what can be changed for each to improve performance.

II. DATASET ANALYSIS & UNDERSTANDING

A. Data Characteristics

The “Default of Credit Card Clients” dataset from UCI provides data on thirty thousand individuals, detailing whether they have defaulted or not, six months of credit payment and balance history, and other statistics like age, sex, education level, and marital status. All of the data is numerically encoded and was preprocessed by UCI for any missing values [1].

The data is very imbalanced, with 77.87% not defaulted and 22.12% defaulted. Training techniques need to be used to avoid skewing predictions too much toward the more represented class [4]. When making predictions on this dataset, the performance is expected to be in the 70% - 80% accuracy range at the least, since if the model predicts “Not Defaulted” for every sample, it can be 77.87% accurate.

After cleaning and analysis, some columns were found that were more useful for classification, and some that were not needed at all. There were some rows that could be deleted as well, which were not found until several preprocessing steps were taken.

B. Feature Analysis & Selection

To select features for classification, the best relationship to look at is how the features correlate to the class labels. Figure 1 shows a correlation heatmap between every column in the dataset, where red is a strong positive correlation, white is no correlation, and blue is strong negative correlation. Since the goal is to classify whether someone will default, the row correlating the class labels to each column should be the main focus.

It becomes immediately clear from the correlation values that most of the data in this dataset is not useful for classifying if someone will default. Sex, education level, marital status, age, bill amounts, and payment amounts all have correlation values in the hundredths, if not thousandths, while 1.00 or -1.00 represent perfect correlation.

It also becomes clear which columns are useful. The columns giving data on how many months behind or ahead the person was on payments each month are ranging 0.19 to 0.32. Relatively high, but still low. The limit balance column has a slightly larger negative correlation to the class labels than other columns at -0.15, but it is not high enough to be useful. For the best classification results with the given data, the first payment history column was used, labeled “PAY_1”. This column was chosen because it has the highest correlation of any other column to the class labels at 0.32.

C. Data Cleaning/Preprocessing

The dataset was already made quite clean and ready for use by UCI, but there was more to be done. Every column was labeled by UCI with an ‘X’ followed by a number starting from one. There is also a row in the dataset that gives a more readable label for the column. For example, column “X5” has the label “AGE” in its description row. The more readable column names from that description row were used to replace the column names given by UCI. Doing this results in the description row being duplicated, so after renaming the columns that row was deleted.

After doing this it was found that the first payment history column was labeled “PAY_0”, while the next was “PAY_2”. The payment and balance size history columns begin at one, so “PAY_0” was renamed to “PAY_1”. The dataset also contains a column labeled “ID” that simply numbers each row sequentially, which is not useful for this application, so that column was deleted.

Only after taking the steps outlined above, checking for duplicate values revealed thirty-five duplicate rows. Duplicate training samples can lead to those samples having undue influence on the predictions, so they were deleted to keep each training sample influences balanced. Since the dataset is relatively large at thirty-thousand samples, deleting thirty-five of them is not an issue.

Since UCI did most of the preprocessing themselves, techniques like encoding of categorical columns did not need to be taken. Encoding non-numeric categories allows them to be used in classifiers and other AI algorithms since these algorithms rely on numeric data [5]. Encoding replaces the categorical data with a numeric representation of some sort to represent each unique category, with that numbering technique varying depending on the application. UCI already did this for this dataset.

D. Data Visualization

Figure 1 shows a cross-correlation heatmap of the dataset. The correlation to the "default payment next month" row is what matters for this assignment. Most columns have negligible correlation with it, and are therefore not useful for classification. The payment history columns have the best correlation, with "PAY_1" having the highest.

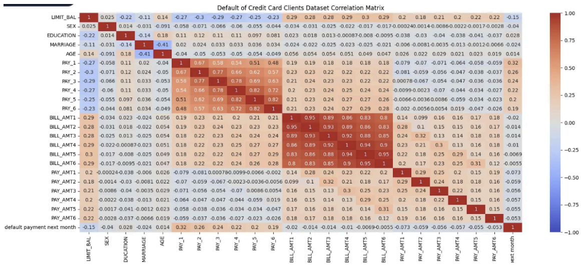

Fig. 1. Cross-Correlation Matrix Heatmap of the "Default of Credit Card Clients" Dataset. [6]

Figure 2 shows the age distribution of the dataset and how many people of each age defaulted or did not default. The ratio of defaulted to not defaulted is consistent and only shrinks a small amount in the older ages. This explains why age had little correlation with whether someone defaulted.

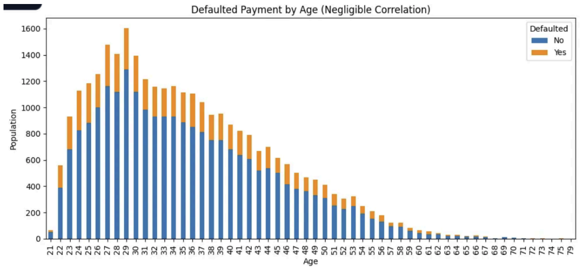

Fig. 2. Population Defaulted (Blue) or Not Defaulted (Orange) by Age. [7]

Figure 3 shows how many months behind or ahead the population was on their payments, and whether the people at each monthly increment defaulted or not. The histories range from two months ahead to eight months behind. As the number of months behind grows, the portion of people that defaulted grows significantly. The columns providing this data had relatively high correlation to whether the person defaulted, so they were most effective for classification training.

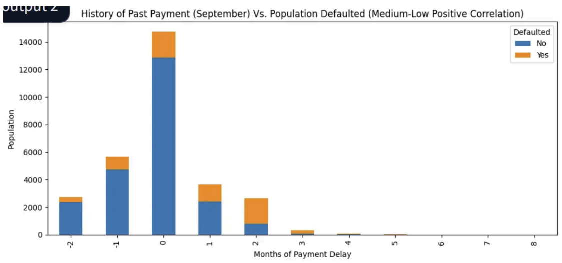

Fig. 3. Population Defaulted (Blue) or Not Defaulted (Orange) Versus Months of Payment Delay. [7]

Figure 4 shows the credit account balance limit distribution of the dataset, and how many people did or did not default at each limit amount. This data had slight negative correlation with whether the person defaulted. It is clear that people with low limits defaulted more often, which explains the correlation.

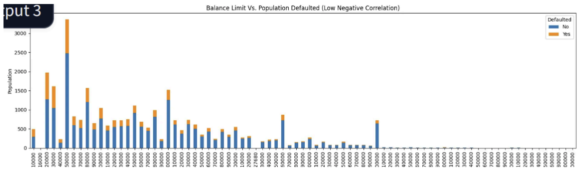

Fig. 4. Population Defaulted (Blue) or Not Defaulted (Orange) Versus Credit Limit Balance. [7]

Figure 5 shows the imbalance of the classes in the dataset. Out of 29,965 samples in the preprocessed dataset, 23,335 samples (77.87%) represent Not Defaulted (0), and 6630 (22.12%) represent Defaulted (1). As a result, the models will be biased towards predicting that someone will not default, since in a worst case, only predicting that someone will not default with this dataset will result in 77.87% accuracy.

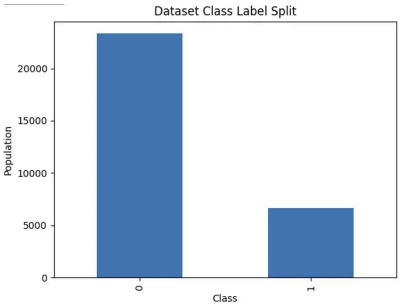

Fig. 5. Population Label Split of Not Defaulted (0) or Defaulted (1). [7]

III. DATA TRANSFORMATION & MODELS USED

A. Feature Scaling

The logistic regression classifier operates with the sigmoid function to calculate its class predictions [2]:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Fig. 6. Sigmoid Function.

The sigmoid function requires its inputs to be scaled between 0 and 1 [2]. Since the dataset is not already scaled this way, a min-max scaler is applied to any data being used as an input to the logistic regression algorithm:

$$X_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Fig. 7. Min-Max Scaler Function.

The random forest ensemble classifier, by nature of being decision-tree-based, is not sensitive to feature scaling [8] and therefore does not require it.

B. One-Hot Encoding for Logistic Regression Classifiers

One-hot encoding is the practice of taking a list of categories in a categorical dataset, assigning an index to each category, then replacing each category with an array of zeros and ones, where every index is a zero except the index that was assigned to the category being replaced [9]. Only the index of the category has a value of one. This is a simple and effective way of making categorical data numeric so it can be used for mathematical operations. For example, if you have categories "High School", "Bachelors", and "Grad School", you would one-hot encode them to [1, 0, 0], [0, 1, 0], and [0, 0, 1] respectively. Since UCI already preprocessed the dataset to be numeric-only for us, doing this with the dataset was not required.

C. Integer Label Encoding for Random Forest Classifiers

Integer label encoding is the practice of simply assigning an integer value to each category in a categorical dataset, then replacing each category with that integer value [10]. This allows the data to be used for mathematical operations. Using the example categories of "High School", "Bachelors", and "Grad School" again, you could encode them to 0, 1, and 2 respectively. UCI already preprocessed the dataset in this way for any categorical data, so no additional encoding was required.

D. Logistic Regression Classifier

Logistic regression classification is a supervised learning algorithm, meaning it takes training data with predetermined class labels as training inputs, and uses the sigmoid function (Figure 6) to predict the class of a test sample [2]. Logistic regression is used for binary classification, so it can only be used for two classes, such as "Defaulted", which was assigned class one, or "Not Defaulted", which was assigned class zero. The output of the logistic regression algorithm is the probability that the given input is of the class assigned to class one. If the probability is greater than or equal to 0.5, the predicted class is one. Less than 0.5, the predicted class is zero.

Logistic regression starts by scaling the input values to be between zero and one using the min-max scaler function (Figure 7). The argument to the sigmoid function, z, is a linear combination of the input features defined by the following function:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

Fig. 8. Linear Feature Combination Function.

β_0 is the y-intercept, β_1 to β_n are the coefficients to the inputs, and x_1 to x_n are the input features of the test sample. Input z is calculated and given to the sigmoid function to get the probability for the prediction.

Once the predictions are made, they can be used along with truth labels to calculate a loss function, which can then be used to tune the coefficients to achieve better predictions. Tuning a coefficient changes how the feature it applies to influences the predicted probability, so more important features will be tuned to have a higher coefficient and vice versa.

This algorithm benefits from its simplicity, making its implementation easy and calculations light. It suffers most when outliers cause overlap between classes, since the algorithm seeks to cleanly divide the classes

E. Random Forest Ensemble Classifier

Random forest ensemble classifiers create a set of decision trees based on the training input features, then the results of all trees are averaged to form a prediction [3]. The randomness of the algorithm comes from the random selection of the features and data points used to create each tree's nodes. Multiple trees are generated and trained independently by running training data through and adding nodes until end conditions are met to create prediction leaves [3]. Once trained, the test data is run through each tree based on the respective features they were formed with, and a prediction is made by each tree. The predictions of all trees are then averaged to get one final prediction.

The main benefit of the random forest ensemble classifier is that it works well against overfitting due to its randomization and averaging. This makes it helpful for small datasets where there are few samples to split between training, validation, and testing. The downside is losing the ease of tracing back the predictions relative to a single tree, and the large number of computations required with several trees plus the randomization.

F. Handling of Data Imbalance

77.87% of the samples in the data are of the "Not Defaulted" class, leaving 22.12% as "Defaulted". To fight against this significant imbalance, the training and test data are divided using the stratified K-fold cross validation algorithm [11]. K-Fold cross validation uses 1/K samples as the testing dataset while the rest are used for training. If K=5, then 20% of the data is used for testing. For each training epoch, a new "fold" is created, using an entirely different portion of data for testing. This allows for a more efficient use of data and prevents overfitting by changing what data is used for training each time.

Stratification being introduced to this method helps to combat the class imbalance. Stratification maintains the percentage of the class representations in each fold to match their representations in the full dataset. Without this there could frequently be too many or too few of one class in the training or test data. If K=5, and stratification is not used, a scenario could easily exist in this dataset where only 2.12% of the "Defaulted" samples are left in the training dataset, leading to an inappropriately large bias toward predicting "Not Defaulted". Stratification keeps the classes balanced in training and testing so the more represented feature is not overly dominant.

IV. EXPERIMENTS & MODEL RESULTS

A. Logistic Regression Tuning & Evaluation

Logistic regression was performed on the dataset using sklearn's LogisticRegression class [12]. After thorough experimentation with the hyperparameters available for the class, the maximum average accuracy achieved was 81.96%. The standard deviation of the accuracy across the folds was 0.0036, so it was very consistent in its accuracy. The execution time for this algorithm was as low as 0.059 seconds, while generally ranging from 0.059 to 0.15 seconds.

Several combinations of columns were tested for accuracy, but ultimately the best results were found by classifying based only on the "PAY_1" column. This column has the highest correlation to the class labels overall. Including any other columns with "PAY_1" reduced the performance.

When experimenting with the hyperparameters of the LogisticRegression class, the default parameters did well in achieving the maximum accuracy of 81.96%. Altering the "class_weight" hyperparameter to attempt to further fight the class imbalance resulted in the same or worse performance. The "solver" hyperparameter, which chooses an optimization algorithm to use for training, was tested for each available option: 'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', and 'saga'. None of these achieved higher accuracy than the default ('lbfgs'), but the 'newton-cholesky' solver was found to reduce the execution time by about 0.1 seconds while maintaining the accuracy and standard deviation.

B. Random Forest Tuning & Evaluation

Random forest ensemble classification was performed on the dataset using sklearns's RandomForestClassifier class [12]. The maximum performance achieved with this algorithm was an average accuracy of 81.96%. The standard deviation of the accuracy across the folds was 0.0036, so it was also very consistent. The runtime was as low as 0.049 seconds, ranging from 0.049 to 0.13 seconds.

For this dataset, many different combinations of columns were tested for classification, but the best results were found again using only the "PAY_1" column, which is the column with the highest correlation to the class labels. Once again, any other columns being included with "PAY_1" would reduce the performance.

To achieve the maximum accuracy, many different hyperparameters were experimented with. The performance was the same or worse in most cases when changing the hyperparameters from their defaults. Given the nature of the data, performance was improved by changing two hyperparameters: "max_depth" and "n_estimators".

The "max_depth" hyperparameter limits the depth of each tree. Limiting this to a depth of only one creates a decision "stump", which is a tree with only a root node and two leaves. This increased the performance from 81.93% to 81.96%. "PAY_1" simply provides whether someone was behind on payments or not, so it is simple to make a prediction based on that fact alone, which means depth is not required for the trees. The complexity added by deeper trees made performance worse.

The "n_estimators" column limits the number of trees to one. Only one tree is needed because the prediction by the decision tree can be as simple as predicting "Not Defaulted" if payment was up to date or "Defaulted" if the payments were behind when only using the "PAY_1" column as done here. Limiting the tree count to one does not increase accuracy, but it reduces the execution time significantly due to how simple the model becomes. If the original results improved by using more features, then limiting the number of trees in this way would harm performance.

V. CONCLUSION

A. Lessons Learned

In the overall dataset, the majority of the data is not useful for classifying whether someone will default on their next payment. Even the more helpful information in the dataset had low correlation to the classes at a maximum of 0.32. When selecting a dataset, it is very important to consider how the output of the model is determined by *natural* intelligence, meaning how we could make a prediction using all of the data available in the real world. This dataset does not include income or personal expense information, for example, which would be much more useful than anything in this dataset for predicting a default on payment. A different dataset, or adding more information to this dataset, could significantly improve the model's performance.

The importance of cleaning and double-checking the data was seen as well. There were many duplicate rows that were not revealed until redundant or useless information was removed, which would not have been found if duplicates were only checked for at the start of preprocessing.

The models used for this assignment had identical accuracy and standard deviation, with similar execution speeds, but the highest performing implementation of each model made it clear that one is not appropriate for this application. The logistic regression model needed very little to be changed from sklearn's default hyperparameters to achieve its maximum performance. The random forest classifier ended up simpler computationally, and slightly faster, but the algorithm performed best as a single decision stump. Setting the hyperparameters like this defeats the purpose of the random forest algorithm. It is clear that the random forest algorithm is not necessary for classification with this dataset.

B. Mistakes Made, Challenges, & Future Considerations

There are a large number of features, and at first glance many seem like they could be useful. The correlation matrix was critical in quickly and clearly identifying what data was relevant to the goal of the assignment. One look at the correlation matrix made it clear that "PAY_1" alone would give the best results, as was proven by the prediction accuracy data.

Another challenge was identifying how to best set the hyperparameters of the classification algorithms. Weighing the classes seemed like it could help combat the class imbalance, but giving the smaller "Defaulted" class more weight only hurt performance for both algorithms.

The random forest algorithm seemed to be a good model for this dataset, and it had near maximum performance with default hyperparameter settings, but tuning and exploration revealed the single decision stump to be the best configuration. Without identifying the "PAY_1" column by itself as being best for classification, and without exploring the hyperparameters of the random forest algorithm, it would not have been found that the random forest algorithm is not a good model for this data. Reaching that conclusion had no clear path from the start, and required a good understanding of the relationships in the data, the inner workings of the random forest classifier, and how those relationships in the data affected the random forest classifier's performance.

To improve the prediction performance, more models should be explored to find a true best model for the application. The dataset should also be improved. The given data could only reach less than 82% accuracy, but with more information on income, personal expenses, and other personal social statistics, a much better prediction could be made. With more time, more thorough exploration of hyperparameters could be done to optimize the models. The ranges of each hyperparameter could be fully explored for their effect on performance, and interactions between hyperparameters could be explored.

REFERENCES

[1] I. Yeh. "Default of Credit Card Clients," UCI Machine Learning Repository, 2009. [Online]. Available: https://doi.org/10.24432/C55S3H.

[2] GeeksforGeeks, "Understanding Logistic Regression," GeeksforGeeks, May 09, 2024. https://www.geeksforgeeks.org/understanding-logistic-regression/

[3] Wikipedia Contributors, "Random forest," *Wikipedia*, Apr. 09, 2019. https://en.wikipedia.org/wiki/Random_forest

[4] J. Brownlee, "A Gentle Introduction to Imbalanced Classification," *Machine Learning Mastery*, Dec. 22, 2019. https://machinelearningmastery.com/what-is-imbalanced-classification/

[5] K. N. Jarapala, "Categorical Data Encoding Techniques," *AI Skunks*, Mar. 27, 2023. https://medium.com/aiskunks/categorical-data-encoding-techniques-d6296697a40f

[6] M. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, Apr. 2021, Available:

[7] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.https://joss.theoj.org/papers/10.21105/joss.03021

[8] D. Team, "Decision Trees: An Overview and Practical Guide," DataHeroes, Sep. 20, 2023. https://dataheroes.ai/blog/decision-trees-an-overview-and-practical-guide/

[9] L. Ganji, "One Hot Encoding in Machine Learning," GeeksforGeeks, Jun. 12, 2019. https://www.geeksforgeeks.org/ml-one-hot-encoding/

[10] aakarsha chugh, "ML | Label Encoding of datasets in Python," GeeksforGeeks, Oct. 15, 2018. https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

[11] "Stratified K Fold Cross Validation," GeeksforGeeks, Aug. 06, 2020. https://www.geeksforgeeks.org/stratified-k-fold-cross-validation/

[12] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, 2011, doi: https://doi.org/10.5555/1953048.2078195.

[13] "3.8.20 Documentation," *Python.org*, 2024. https://docs.python.org/3.8/index.html (accessed Oct. 13, 2024).

[14] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

[15] "API reference — pandas 1.3.3 documentation," pandas.pydata.org. https://pandas.pydata.org/docs/reference/index.html#api

[16] G. Team, "Gradio Documentation," *www.gradio.app*. https://www.gradio.app/docs