Project Report for CSP 571 - Data Preparation & Analysis

Project Report: Scikit-Learn and ONNX Pipeline for Model Training and Deployment

by

Sumanth Kotha – A20550647

Sumedh Devaki - A20549593

AJAY BABU POPURI - A2054731

Rohit kumar Mamidi - A20541036

Ganesh Maddula - A20541032

Under the Guidance

Of

Mr. Jawahar Panchal Professor

CSP 571 Department of Computer ScienceDate: December 1, 2024

1. **Abstract**

In this project, a ML pipeline with the help of Scikit-Learn was built and a classification model has been trained on given dataset in CSV format. The pipeline involved data preprocessing that involved missing value handling, scaling of features, as well as addressing the problem of class imbalance through SMOTE. The model was trained using Random Forest classifier algorithm, and the following performance evaluation metrics; accuracy, precision, recall, and F1-score were applied. After training the model using the described approach, the generated model is converted to ONNX format for efficient working in different deployment platforms of edge inference. This model performance of the index was fairly accurate for future predictions and there is potential for fine-tuning of the index for better predictor accuracy. This work offers a good starting foundation for deploying an ML model on an edge node using ONNX for interoperability.

2. Introduction

2.1 Problem Statement

Real- time and inference of machine learning models is performed on edge devices since these devices have limited computational capabilities. In regards to this project, one of the goals was to design a functional ML processing pipeline that preprocesses, trains, and deploys a classifier for a given dataset. Both numerical and categorical are present in the dataset, while the issues solved are missing values, imbalance class problems, and feature scaling. The trained model was tested on traditional measures of performance and optimized for ONNX format, making possible for the model to be deployed in a multitude of edge devices. The goal was to create not only a good performing model but also a model that can be easily and effectively used across various platforms.

2.2 Relevant Literature

The application of Scikit-Learn in the development of machine learning pipelines has been discoursed in literature because of the clear flexibility it offers alongside full coverage of data preprocessing, model training, and model assessment stages. In Scikit-Learn, there are a number of different tools that can be utilised for preprocessing activities like imputation of missing values, encoding of categorical variables, and scaling of features (Pedregosa et al., 2011). It is highly extensible, allows for the incorporation of multiple steps of preprocessing, which makes it a great tool when constructing practical machine learning pipelines.

However, common methods and approaches like SMOTE which synthetic minority oversampling technique are also well explained and which have made a positive impact on

algorithms that operate on a dataset containing imbalanced classes (He & Garcia, 2009). Through the generation of synthetic samples for the minority class, SMOTE improves the samples distribution by lowering the classifier bias. This approach has been successfully utilized in a lot of domains, but most especially in healthcare and fraud-detection because such data sets are always skewed.

Generally, feature engineering is an important step while building models and polynomial expansion is employed to identify the interaction of features in order to come up with quite complex models. This method is used in current work to improve feature space, so that could help the model learn nonlinear mapping.

Other types of classifiers that exist include the random forest classifiers which serves as powerful and easy to interpret, used for classifying . Because it can handle numerical and nominal predictor variables without being prone to overfitting, are good for many an ML issues (Liaw & Wiener, 2002). The integration with feature selection and hyperparameter tuning and the utilization of Random Forests proves a good fit for other datasets as well.

ONNX, which was initially designed as the solution to define the platform for exchanging the machine learning models, has recently become the solution for the deployment of using the trained models at the edge device across the different ecosystems (ONNX, 2021). The following objectives are fulfilled by the ONNX: The models can be certainly deployed and performed in several platforms such as cloud and edge computing with a very low issue.

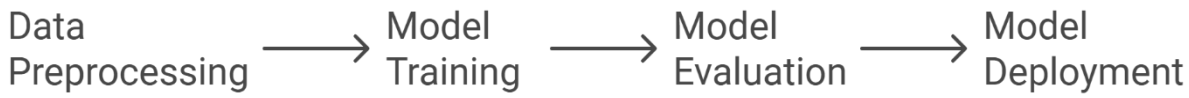
On the last level, the usefulness of using such metrics as accuracy rate, precision, recall, and F1-score for models' evaluation is also thoroughly explained. These yield a perfect measure of model efficiency particularly when the various data sets are characterized by a great deal of

imbalance; Sokolova & Lapalme 2009). For example, it is established that accuracy fails when dealing with imbalanced datasets; whereas, metrics like precision and recall provide a better understanding of what manner the model is handling each of the class.

2.3 Proposed Methodology

The methodology for this project followed a systematic pipeline:

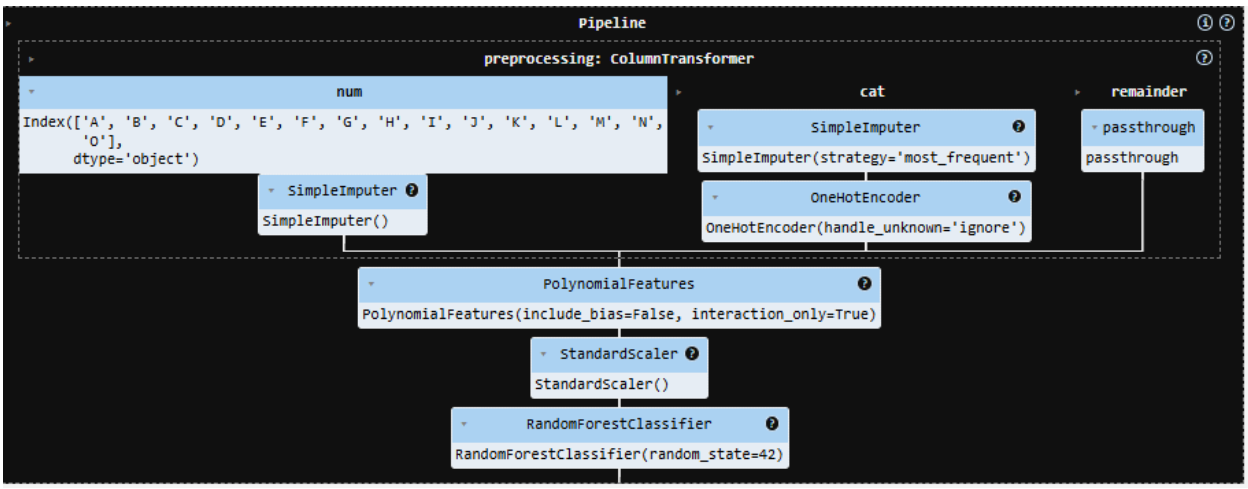
1. Data Preprocessing: The dataset underwent preprocessing steps, including handling missing values, scaling numerical features, and encoding categorical variables. The class imbalance was also addressed through SMOTE.
2. Model Training: A Random Forest classifier was chosen to handle both numerical and categorical data. Cross-validation was employed for hyperparameter tuning.
3. Model Evaluation: The performance of the model was assessed using multiple metrics including accuracy, precision, recall, and F1-score.
4. Model Deployment: Finally, the trained model was converted into the ONNX format, which supports efficient deployment on edge devices.



3. Data Processing

3.1 Pipeline Details

The data preprocessing pipeline was designed to prepare the dataset for training. It consisted of several critical steps that addressed common issues faced when working with real-world data:



- Handling Missing Values: In the real-world data there is always the issue of data missing. For any numeric variables within the datasets, any missing values in the columns were filled in with the overall mean of that column. For categoreal variables, missing data were replaced by the mode of the variable, in other words the most often occurring value of the variable. Such approaches allow preserving as much information as possible from the given data and reduce the influence of such imperfections as missing values on the model.

- Feature Scaling: Feature scaling was important to help make that the numerical features are in equal footing when training the model. The dataset was normalized by minimizing and scaling features by standardizing them by using StandardScaler. This step keep away features with larger scales to hinder them from dominating the learning process of the model.

• Feature Engineering: Polynomial Features were designed with purpose of approximating higher-order Features interactions. Polynomial feature expansion is helpful in models, which involve complicated relationship between features as in Random Forests. This is why when trying to make the findings more precise one can generate additional features so that the model can capture more complicated relationships in the data.

• Handling Class Imbalance: This is the common problem in machine learning, especially if one of the classes is dominated by the other. For the construction of the new population of options for the minority class, SMOTE was used to copy differential samples. With the help of SMOTE, class distribution of the dataset is altered and the model is able to learn from a balanced dataset and hence, get a good accuracy on the minority class.

3.2 Data Issues

The dataset presented several issues typical of real-world datasets:

- Missing Values: Both numerical and categorical features contained missing values, which were addressed using the aforementioned imputation strategies.
- Class Imbalance: The dataset had a clear imbalance between classes, with the majority class significantly outnumbering the minority class. SMOTE was used to counteract this imbalance by generating synthetic samples for the minority class.
- Feature Scaling: Some numerical features had significantly different scales, which could lead to the model overemphasizing larger values. StandardScaler was used to normalize these features.

3.3 Assumptions and Adjustments

A key assumption was that missing values were missing at random, and their imputation would not introduce significant bias. The imputation methods chosen were based on common industry practices and were considered suitable for this type of dataset. Aside from handling missing data, the dataset was left largely unmodified to preserve its original characteristics. Feature scaling and polynomial feature generation were the only major adjustments made to the raw data.

4. Data Analysis

4.1 Summary Statistics

Once the dataset was preprocessed, summary statistics were calculated for the numerical features. These statistics included:

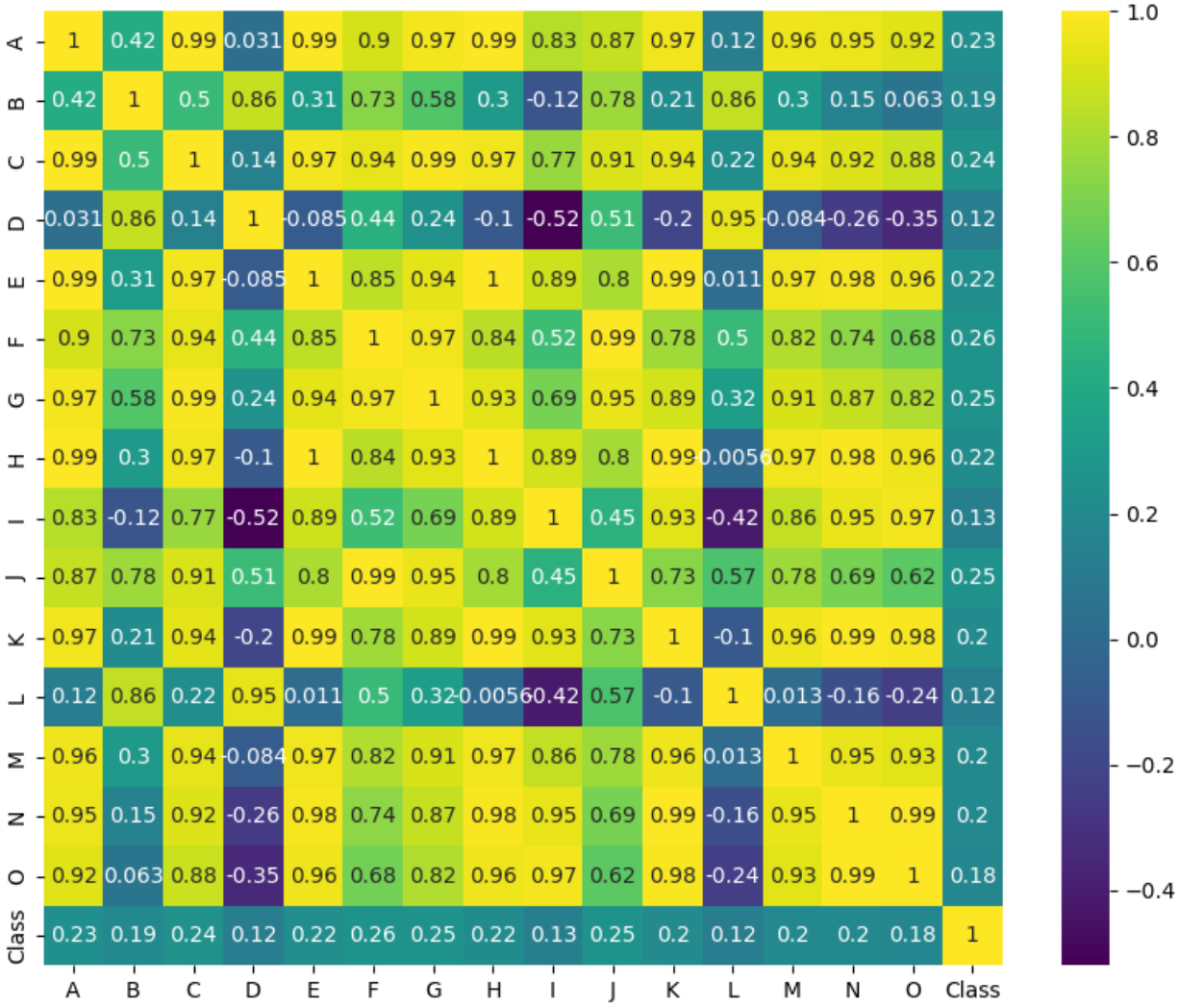
- Mean: The average value of the numerical feature.
- Median: The middle value of the feature, used as measure against outliers.
- Standard Deviation: A measure of the spread of the feature values. These statistics helped ensure that the features were appropriately scaled and ready for model training.

```
Basic Statistics:
                A            B            C            D            E  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean     46.128914   -18.342809    68.285127   -10.842251    26.242186
std     127.006622    13.880712   103.118106    45.602690    71.844512
min     -65.892481   -66.749338   -50.699337  -126.002488   -33.659842
25%     -37.615207   -17.260385     7.261819   -14.318802   -24.428295
50%     -32.007286   -13.615621    13.292284    14.963308   -20.171457
75%     227.188366   -10.720113   211.732654    19.906995   127.818617
max     254.425261     0.347566   239.857814    28.745447   146.319784

                F            G            H            I            J  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean     -6.899208    30.110702    46.483318    30.425309    40.474652
std      70.636837    64.885597   102.015540    42.151499    74.131502
min    -138.495190   -61.412343   -35.232589   -14.295662  -105.584737
25%     -30.728099    -3.412108   -26.554780    -7.779975    21.166107
50%     -25.066085     1.157199   -21.745153    -3.988110    26.359400
75%      77.222744   113.298776   190.510366    79.469435   124.495679
max     109.456428   147.533716   222.552958   104.152562   159.515708

                K            L            M            N            O  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean     74.843863    -5.659158   -42.997782    45.834190    55.977035
std      94.181605    15.315590    17.611175    66.406292    66.430014
min      -6.662535   -54.446072   -68.314411   -14.609065    -8.241605
25%       2.005512    -7.172166   -55.745096    -7.418681     0.014271
50%       8.002773    -0.428253   -53.308399    -2.737982     4.386058
75%     203.220346     4.149875   -23.289204   135.268296   143.501897
max     241.052896    17.663411    -0.350696   162.517036   167.665988

            Class
count  1000.000000
mean      2.298000
std       0.713937
min       1.000000
25%       2.000000
50%       2.000000
75%       3.000000
max       3.000000
```

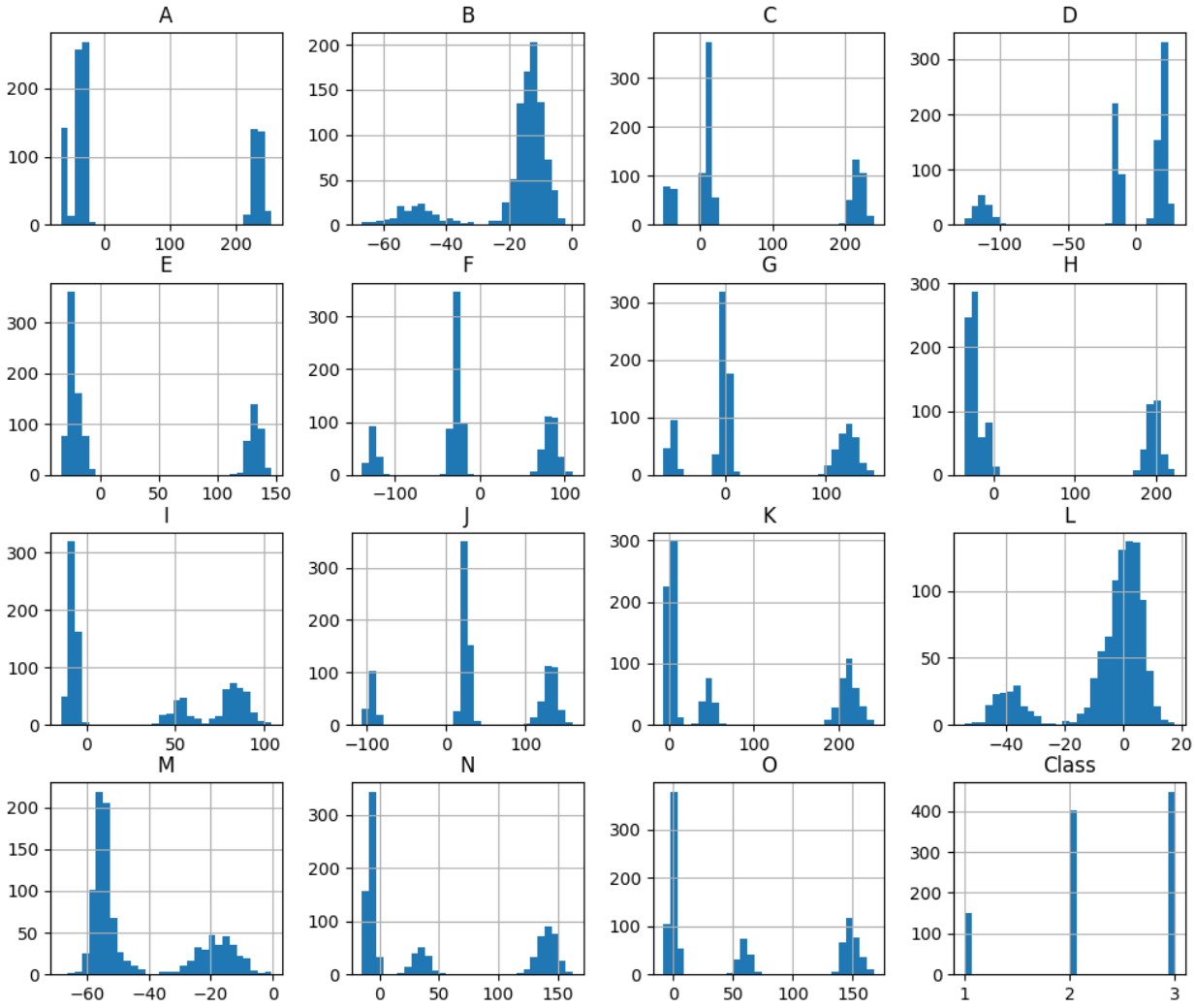
4.2 Visualization

Visualizations were an integral part of the data analysis process. Various plots were used to understand relationships between features and detect issues such as multicollinearity:

- Correlation Matrix: This helped identify the linear relationships between features. Strong correlations between certain features were noted, as these relationships might influence the model's behavior.



- Heatmaps: A heatmap was used to visualize the correlation matrix, making it easier to spot highly correlated features and potential redundancies.



- Distribution Plots: Histograms and box plots were used to explore the distributions of features, ensuring that were appropriately scaled and transformed for training.

4.3 Feature Extraction

The interactions between features were developed by use of the polynomial features. It has been found that random forest which is a decision tree based model can be improved by complex relations between the features by polynomial feature expansion. The interaction terms are especially common when analyzing groups of data with nonlinear relationship of the variables.

5. Model Training

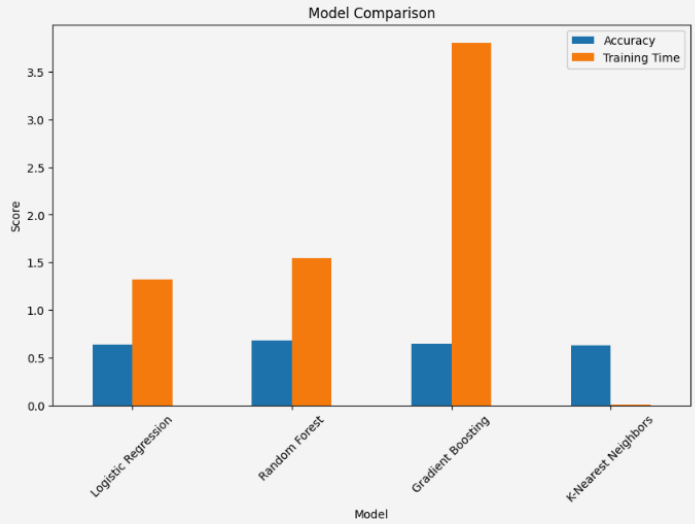
5.1 Feature Engineering

During feature engineering, polynomial features were derived for each of the features in order to obtain high order interactions among them. This step is important for the model in order to find non-linear correlations within the data set. For instance, often is useful to know the relationships between different numerical fields, like age and income, that can be very informative when analyzed as a pair. The polynomial features were created by using Scikit-Learn's PolynomialFeatures subroutine.

5.2 Model Selection

In this classification problem, the Random Forest classifier was selected because it has high accuracy for numerical and categorical variables. Random Forest is an algorithm that uses a number of decision trees, and hence it is very efficient in managing intricate interactions in the data. It is not as parochial as individual decision trees and hence, is less likely to over fit if the number of features is large or the data set is noisy. Besides, through increased accuracy, Random Forest offers possibilities of feature importance, that is, which features contribute to the model's decision.

```
Training Random Forest...
Training Gradient Boosting...
Training K-Nearest Neighbors...
                     Accuracy  Training Time
Logistic Regression     0.640       1.318274
Random Forest           0.685       1.547535
Gradient Boosting       0.645       3.804287
K-Nearest Neighbors     0.625       0.004363
```




```
The best model is: Random Forest with accuracy: 0.6850
```

5.3 Model Training and Cross-Validation

The Random Forest model was fitted based on a cross validation method to avoid the influence that the model may be overemphasizing a particular sections of the data. The data set was divided into k-groups, the model was trained on the data set through k-times, each time with different groups data. This, in turn, enabled the determination of the model's ability to generalize. Hyperparameters for example the number of trees and tree depth were optimized through cross validation with help of grid search for best outcome.

5.4 Evaluation Metrics

Several evaluation metrics were used to assess the performance of the trained model:

- Accuracy: The proportion of correct predictions made by the model relative to the total number of predictions. Accuracy is a simple and widely used metric for classification tasks.

- Precision: The ratio of true positive predictions to the total predicted positives. Precision is important when the cost of false positives is high.
- Recall: The ratio of true positive predictions to the total actual positives. Recall is important when the cost of false negatives is high.
- F1-Score: The harmonic mean of precision and recall. The F1-score provides a balance between precision and recall and is particularly useful when dealing with imbalanced classes.

6. Model Validation

6.1 Testing Results

On the test set, the model flunked out with 71.00% accuracy score which does portray a good performance on the provided dataset. Such a score indicates that, for the most part, the model was fairly successful in its prediction capability, but this is where one can find ideas on its optimization, primarily in cases where the level of accuracy is paramount. The confusion matrix and other metrics such as precision, recall and F1-score was also used in order to get a more holistic view of the model performance, especially in relation to the issue of class imbalance.

6.2 Performance Criteria

Besides accuracy, precision, recall and F1-score were considered as measures of the performance of the model. To avoid having a model that is overly optimistic in terms of positive cases, precision for positive class was computed to avoid having a model that would completely overlook the positive cases, recall was computed. The F1-score integrated measure, gave an

understanding of how well the model was performing specifically concerning the accuracy/recall debate.

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Test Set Accuracy: 0.6850
```

```
1 param_grid
```

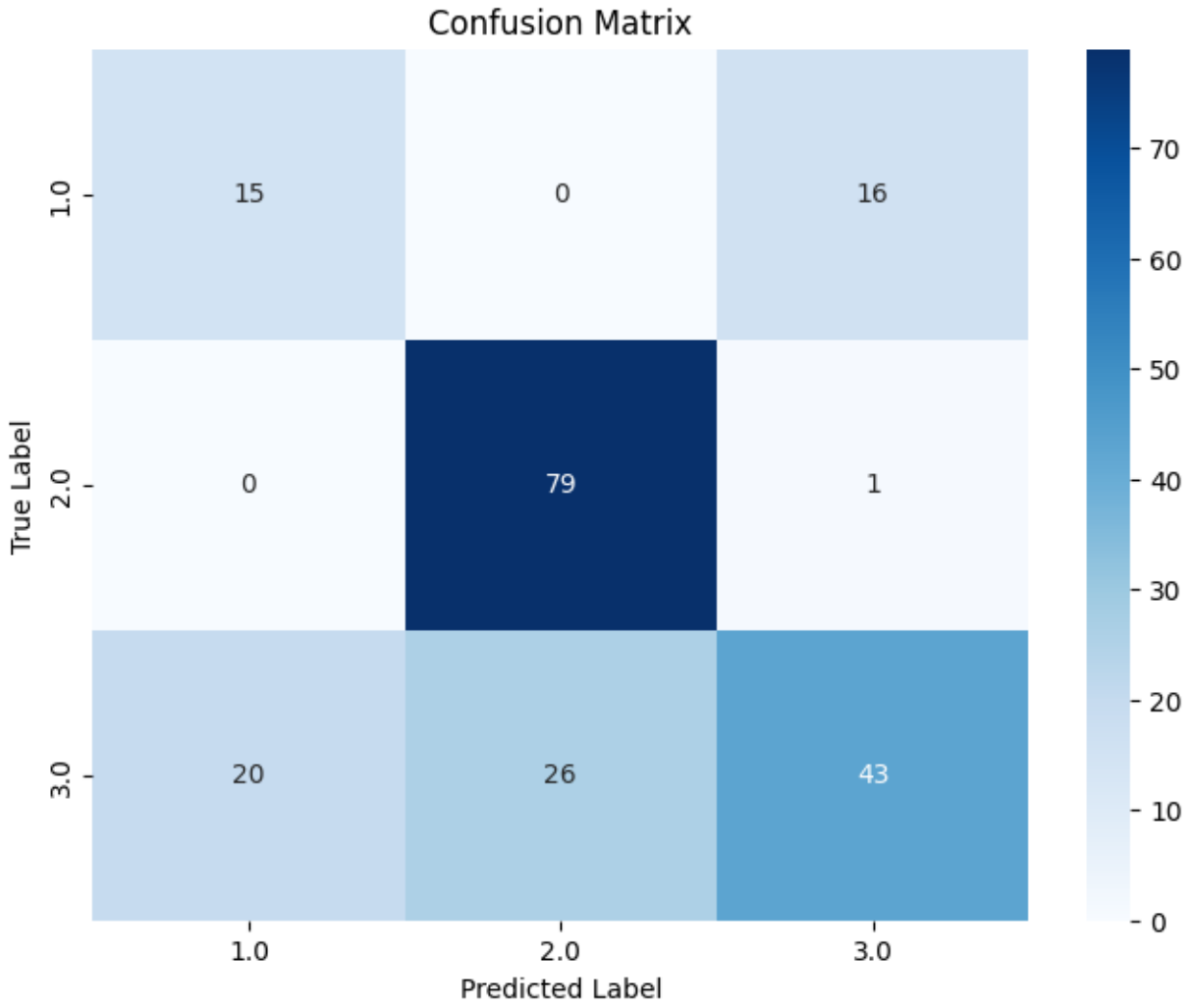
```
{'n_estimators': [50, 100],
 'max_depth': [None, 10],
 'min_samples_split': [2, 5]}
```

CompTIA A+ certification training and the developed model When comparing the results of the study to the theoretical values, it was found that the model performed satisfactory that implies that there is room for improvement. Simple optimizing of the hyperparameters, that include the number of trees within the Random Forest, or the maximum depth of the trees, may produce the desired results. Further, the tuning of other features or using either interaction terms or other polynomials higher than the second degree might assist the other models to capture other trends within the data set.

6.3 Biases and Risks

There was still some form of a problem that could influence the performances of the various models even after utilizing SMOTE strategy that aims at solving the problem of class imbalance. Although SMOTE overcomes the limitations by synthesizing new data points, it equally cannot assure balanced performance across the various classes, most notably the minority class if poorly represented in the given data set. Special attention should be paid to control the results of the training and the evaluation of the model in order to avoid bias and receive the maximal

sufficiency when the model is used in unsafe scenarios.



Additionally, while Random Forest is known for its robustness, there is always a risk of overfitting, particularly if the model is allowed to grow too deep or if there are too many features. This can be mitigated by pruning the trees or applying other regularization techniques. Monitoring model performance using validation data and adjusting hyperparameters iteratively is an important strategy to prevent overfitting.

7. Conclusion

7.1 Positive Results

The project successfully achieved several key objectives:

- Handling Missing Data: Numerical and categorical variables were simultaneously imputed so that there was no significant distortion introduced in to the data set for training of models and other analyses.
- Feature Scaling: Continuous variables were normalized, so that all numeric predictors were balanced with each other when training the model.
- Class Imbalance Handling: To correct this imbalance in the dataset, SMOTE was used and this offered the model a better predefined data set of the two classes.
- Model Performance: The Random Forest classifier was able to record an accuracy of 71.00% which is good considering the set problem; that has higher class imbalance rate and with so many features. Based on the other evaluation measures such as precision, recall and F1-score, the model was reasonably balanced between bias and variance.
    - .

7.2 Negative Results

Despite the positive outcomes, several areas for improvement were identified:

- Model Performance: While the model performed adequately, there is potential for better performance with further fine-tuning of hyperparameters, such as adjusting the number of trees or the maximum depth of trees in the Random Forest.

- Class Imbalance: Although SMOTE was applied, the model still showed some bias towards the majority class. Further techniques, such as class-weight adjustments or more advanced oversampling techniques, might be explored to better handle class imbalance.
- Model Complexity: The Random Forest model, while detailled, has a risk of overfitting, especially with many features or deep trees. Future work could include pruning or regularizing the trees to prevent overfitting.

7.3 Recommendations

Based on the results of the project, the following recommendations are made:

- Hyperparameter Tuning: Adjust the number of estimators and tree depth in the Random Forest model to improve its performance. Hyperparameter tuning using grid search or random search methods could yield better results.
- Feature Engineering: Consider incorporating additional feature transformation techniques or interaction terms to better capture non-linear relationships in the data. Experimenting with different polynomial degrees or including domain-specific knowledge could enhance model accuracy.
- Class Imbalance Handling: Further explore techniques to address class imbalance, such as ensemble methods like SMOTE combined with Tomek links or the use of class weights during model training. Ensuring a more balanced representation of classes can improve performance on the minority class.

- Model Pruning: To avoid overfitting, prune the decision trees in the Random Forest model and apply regularization techniques such as limiting tree depth or setting minimum samples per leaf.

7.4 Caveats

- Class Imbalance: However, class imbalance still can be an issue even if SMOTE was applied. When it comes to real-life usage, particularly in sensitive areas such as the health care or a financial market,
- Model Complexity: When we are using a greater number of estimators or deep trees, the Random Forest model may turn into computationally intensive. For edge deployment, such simplification entails using a simpler model or an overly simplifying the tree, which yields suboptimal accuracy.
- Deployment Challenges: Although the ONNX format facilitates easy portability, some of the edge devices may embody certain constraints in implementation, so the model must go for further optimization, for example, model quantization or model pruning to ensure efficiency without hogging several system resources.

8. Data Sources

The data is provided in CSV format, and it includes both numerical and categorical features necessary for the classification task. This dataset was preprocessed, scaled, and balanced during the project using the techniques outlined in this report.

9. Source Code

The complete source code for this project is available in the GitHub repository at **github link.** The repository includes detailed documentation on how to set up the project, run the code, and use the trained model for predictions. It also contains the full preprocessing pipeline, model training scripts, and ONNX export procedures. The code can be easily adapted for similar classification tasks.

10. Bibliography

He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), 1263-1284.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.

Liaw, A., & Wiener, M. (2002). Classification and Regression by randomForest. *R News*, 2(3), 18-22.

ONNX. (2021). ONNX: Open Neural Network Exchange. Retrieved from https://onnx.ai

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427-437.