Libraries and Dataset

The following standard data science and machine learning libraries were imported: pandas to load the dataset, NumPy for matrix manipulation, seaplot and matplotlib for data visualisation and sklearn for building model classifiers. The dataset used for classification consists of 700 samples with 10 columns/features inclusive of the target variable (class). The features are numerical since they are represented by integers. The dataset is an imbalanced classification problem with class 0 having more samples than class 1 with no missing data. The goal of the classification task is to build a model that can accurately predict the class labels (0 or 1) based on the given features.

Exploratory Data Analysis

The dataset was thoroughly explored through histograms, scatter plot matrices, and correlation coefficient matrices. Histograms revealed diverse distributions across features, with some exhibiting bimodal patterns while others displayed unimodal distributions. Understanding these distributions is crucial for identifying outliers, central tendencies, and selecting appropriate preprocessing or modeling techniques. Scatter plot matrices depicted weak linear relationships among features, with a notable positive correlation between Feature2 and Feature3. Additionally, a correlation coefficient matrix highlighted high positive correlations between Features 2, 3, 6, and 7 with each other and the target variable 'class'. These findings provide valuable insights into the dataset's structure and relationships, guiding further analysis and modelling decisions with other features and the target variable class. Correlation coefficients only capture linear relationship and not complex associations.

Train and Test Splits

Splitting the data into training and testing sets was performed using scikit-learn, ensuring consistency with a fixed random state. Following the standard practice, 20% of the dataset was allocated to the test set for evaluating model performance. Initially, default hyperparameters were utilized for model training, providing a baseline for comparison. However, before proceeding with the final model selection, hyperparameter tuning was conducted to optimize model performance. Hyperparameter tuning plays a crucial role in balancing the trade-off between underfitting (high bias) and overfitting (high variance). To mitigate these issues, cross-validation was employed to assess hyperparameter performance. Rather than splitting the training set into separate subsets for validation, ten-fold cross-validation was chosen for its effectiveness in reducing bias and variance. This method involves dividing the training data into ten equal parts, training the model on nine parts while validating on the remaining part iteratively. The average performance across all iterations provides a robust estimate of the model's effectiveness. Finally, the selected hyperparameters were applied to train the model on the entire training dataset, ensuring comprehensive learning before evaluation on the test set.

Model Evaluation: Decision Trees Classification

The discriminative classification model, known for its transparency in classification rules and ease of interpretation, was chosen for this task. To optimize its performance, a range of options for the max_depth parameter, controlling the number of split points, was explored. This

parameter is crucial for reducing overfitting and complexities inherent in decision trees, especially on imbalanced datasets where the majority class dominates. A ten-fold cross-validation was employed to identify the best max_depth value that strikes a balance between overfitting and generalization. Before tuning, the Decision Tree Classifier achieved a balanced performance with high precision, recall, and F1-Score, as indicated by the weighted averages. The classifier had a higher True Positive rate but also a higher False Negative rate, indicating potential overfitting to class 0. After tuning, there are some changes in the performance metrics. The model correctly predicted 85 instances where the true label was class 0 but incorrectly predicted 9 instances as class 1 when they were class 0. Also, the model incorrectly predicted 2 instances as class 0 when they were class 1 but correctly predicted 44 instances where the true label was class 1. While precision remains stable, there is a slight decrease in recall and F1-Score and a trade-off between True Positive and False Negative rates, suggesting a better balance between the two classes. This suggests that the tuned model may be slightly less effective at correctly identifying class 1 compared to the untuned model. Both before and after tuning, the Decision Tree Classifier demonstrates robust performance, with weighted average scores above 0.9 for precision, recall, and F1-Score. Notably, the model's feature_importances_ attribute facilitated the identification of important predictors, with features 3, 6, 2, and 8 being deemed significant in constructing the decision tree. Visualization of classification rules through plot trees provided insights into the decision-making process, including specific criteria for predicting target classes. Notably, the Gini index at each split and the assigned class were observable, further elucidating the decision tree's behaviour. These findings suggest that while tuning did not lead to significant performance gains, it underscored the suitability of hyperparameters for the dataset, affirming the model's robustness.

KNN-Model Classification

Hyperparameter tuning was performed to select the optimal value of the hyperparameter 'k', which represents the number of nearest neighbors to consider during the classification. Before tuning, the KNN classifier was initialized with a default value of k=5. After tuning, a range of k values from 1 to 40 was evaluated using cross-validation with 10 folds. The mean cross-validation score was calculated for each k value, and the value of k that yielded the highest mean score was selected as the optimal hyperparameter value. In this case, the best value of k was found to be k=9. The various cross validation values for each K values was generated to identify other possible values of K with high mean score. The KNN classifier was trained on the training dataset using the selected value of k=9 and the trained classifier was then used to predict labels for the testing dataset. The model correctly predicted 93 instances where the true label was class 0 but incorrectly predicted 1 instance as class 1 when they were class 0. Also, the model incorrectly predicted 2 instances as class 0 when they were class 1 but correctly predicted 44 instances where the true label was class 1. The model's performance slightly decreased after tuning, as indicated by the lower F1 score, recall and precision. The changes in the confusion matrix were minimal, with only one additional misclassification in the category. This suggests that the tuning process did not significantly alter the model's overall performance but may have introduced some minor variations in its predictions.

SVM Classification

The SVM classification algorithm was utilized for this task. Hyperparameters were optimized using GridSearchCV with a parameter grid consisting of different combinations of kernel types (linear, rbf), regularization parameter (C), and gamma values. The Gridsearch is used because you can specify a bunch of values and test all possible combination of hyperparameters. Using the ten-fold cross validation, the mean score selected the best hyperparameters: Kernel: RBF, C: 0.5, Gamma: Scale. The model correctly predicted 91 instances where the true label was class 0 but incorrectly predicted 3 instances as class 1 when they were class 0. Also, the model incorrectly predicted 2 instances as class 0 when they were class 1 but correctly predicted 44 instances where the true label was class 1. The recall, precision, F1-score, and confusion matrix showed consistency before and after tuning, indicating stable performance in correctly identifying positive instances, making accurate predictions, and achieving a balance between precision and recall. This suggests that hyperparameter tuning had minimal impact on the classifier's behaviour, likely due to the dataset not requiring substantial adjustments for improved performance. The decision boundary was visualized in the reduced dimensional space through dimensionality reduction with PCA, and tuning SVM hyperparameters using GridSearchCV. This approach helped in understanding how the SVM classifier separates different classes in the transformed feature space.

Conclusion

In the dataset, KNN achieved the highest performance metrics, followed by SVM, and Decision Tree as displayed in the table below. KNN's performance may be attributed to its ability to capture complex patterns in the data without making strong assumptions, but it comes with the trade-off of increased computational cost. SVM strikes a balance between complexity and performance but may require more tuning effort. On the other hand, Decision Trees are interpretable but may suffer from overfitting even after tuning. Hence, using ensemble methods like Gradient Boosting can often improve the performance of decision trees on imbalanced data.

Performance Metrics	Model Classifiers Using the Best Hyperparameters		
	Decision Tree	KNN	SVM
Accuracy	0.92	0.98	0.96
F1 Score	0.92	0.98	0.96
Precision	0.93	0.98	0.96
Recall	0.92	0.98	0.96

Reflection on the Project: While higher metric performance before tuning may suggest a well-performing initial model, it does not necessarily make it the best model overall. Through this project, I learned the importance of selecting the right classification method and tuning hyperparameters for optimal performance. Practical experience in analyzing data, evaluating model performance, and interpreting classification results was gained. In the future, I aim to conduct more in-depth feature engineering, explore additional classification algorithms, and experiment with ensemble methods or advanced techniques like deep learning to enhance model accuracy and robustness.

REFERENCES

Zhang, D. (2020, July 13). Exploring Classifiers with Python Scikit-learn — Iris Dataset. Towards Data Science. https://towardsdatascience.com/exploring-classifiers-with-python-scikit-learn-iris-dataset-2bcb490d2e1b#f4ab

Koehrsen, W. (n.d.). machine-learning-project-walkthrough. GitHub. https://github.com/WillKoehrsen/machine-learning-project-walkthrough

Coursera Staff. (2024, February 9). 10 Machine Learning Algorithms to Know in 2024. Coursera. https://www.coursera.org/articles/machine-learning-algorithms

StatQuest. (n.d.). YouTube. Retrieved from https://www.youtube.com/@statquest

Géron, A. (2023, January 20). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media. https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032647/