Fundamentals of Data Analytics

Course Number: 30539

Assessment Task 3: Data analytics in action

University of Technology Sydney

Nicolas Huber
Student ID: 25061944
Date: July 31, 2024

Contents

1 Introduction

This is the final report on a machine learning (ML) classifier project that aims to create a classifier for a space data related to the Gaia project. The report is structured as follows:

First, a description of the data mining problem and concrete task details is provided in Section 2. Second, the information about the underlying preprocessing is presented in Section 3. Third, a concrete description of how to solve the problem, including the underlying reasoning, can be found in Section 4. Fourth, the different classifiers used are described in Section 5. Fifth, the results of the performance of the different classifiers with respect to their relevant parameters, found through hyperparameter tuning, are demonstrated in Section 6. Sixth, the evaluation and selection of the best classifier is discussed in Section 7. Finally, the report concludes with a personal reflection on data mining, viewing classification as a typical data mining problem, in Section 8. The interested reader can find additional material in the Appendix A.

2 Description

This section provides a comprehensive description of the data mining problem and addressed some of the task details.

The objective of this project is to develop a robust classifier to predict the *SpType-ELS* attribute, representing the estimated spectral class by Gaia. This classification task involves handling a categorical attribute with possible values "A" and "B". Using state-of-the-art data analytics techniques, we aim to construct models that can effectively differentiate between those classes. The challenge lies in preprocessing the data, selecting relevant features, and employing appropriate classification algorithms to achieve high accuracy. The process includes model evaluation and hyperparameter tuning to ensure optimal performance. The underlying goal is to finally make predictions of unseen data, which could be the highly relevant in a business context, such as identifying new objects as either "A" or "B" in large astronomical datasets. This project builds on previous in-depth data exploration, which is essential for understanding the data's structure and identifying patterns. Such preliminary work is crucial in data analytics as it informs the choice of features and models, thereby enhancing the overall performance of the classifiers. This project demonstrates general data analytics practices in the real world, particularly in the context of classification tasks.

3 Preprocessing

This section explains and justifies the preprocessing steps undertaken for this project, ensuring data quality and consistency for model training. The preprocessing steps were implemented using `scikit-learn` [1]. The main steps included feature selection, data encoding, handling missing values, scaling numerical features, and combining preprocessing steps into a comprehensive pipeline. The preprocessing involved the following key steps:

1. Dropping non-relevant columns.
2. Encoding the categorical target variable.
3. Handling missing values using median imputation.
4. Scaling numerical features to standardize the data.
5. Combining all preprocessing steps using `ColumnTransformer`.

3.1 Insights from the past Data Exploration

Initial distribution analysis confirmed the class distribution within the training data was relatively balanced (Class A: 80,088 Class B: 68,450). Given the manageable amount of missing data, median imputation was chosen to be appropriate, ensuring that the central tendency of the data is maintained. Also, we defined the following columns to be excluded from the data analysis, since they do not contribute to the true signal: *ID*, *Unnamed: 0*, and *Source*.

3.2 Data Encoding and Splitting

The target variable, `SpType-ELS`, a categorical variable, was encoded using `LabelEncoder` to convert categorical values into a binary numerical format. This transformation was essential for compatibility with the machine learning algorithms employed:

```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

Subsequently, the data was split into training and validation sets using `train_test_split`, ensuring the model is evaluated on unseen data. This split helps in assessing the generalization capability of the model and is considered to be best-practice:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded,
    test_size=ModelConfig.test_size,
    shuffle=True,
    random_state=Settings.random_state
)
```

The `LabelEncoder` is also crucial for transforming predicted numerical labels back to their original categorical form, althougt this is technically not part of the preprocessing. This reverse transformation is necessary for interpreting the model's predictions in their original context and for preparing the results for submission. After making predictions on the unknown data, the encoded predictions are converted back to categorical values:

```
# Make predictions
y_unknown_pred_encoded = best_pipeline.predict(df_unknown)
y_unknown_pred = label_encoder.inverse_transform(y_unknown_pred_encoded)
```

3.3 Numerical Data Preprocessing

Numerical features were identified and processed using a pipeline. The pipeline included median imputation to handle missing values and standard scaling to normalize the data. These steps are crucial to ensure the data is clean and features are on a comparable scale:

```
numerical_features = X_train.select_dtypes(include=[np.number]).columns.tolist()
numerical_pipeline = Pipeline([
    ('impute', SimpleImputer(missing_values=np.nan, strategy='median')),
    ('scaler', StandardScaler())
])
```

[1] `scikit-learn`: A machine learning library in Python.

3.4 Combining Preprocessing Steps

All preprocessing steps were integrated into a single transformer using `ColumnTransformer`. This approach ensures efficient preprocessing of both numerical and categorical features:

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_pipeline, numerical_features),
    ],
    remainder='passthrough'
)
```

3.5 Full Pipeline Creation

A comprehensive pipeline was created to incorporate preprocessing and classification steps, facilitating end-to-end model training and evaluation. This pipeline ensures that all preprocessing steps are consistently performed, which also applies to potential predictions on new data. Overall, the quality and reliability of the data will benefit:

```
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', None)
])
```

This structured approach to preprocessing sets a solid foundation for the machine learning models, ensuring they receive clean, well-prepared data for training and evaluation.

4 Methodology

This section explains how the problem was approached. First, the overall approach to solve the classification problems is described. Second, initial insights from a preliminary feature importance is provided. Lastly, the resulting pipeline is explained.

4.1 Approach

The objective is to develop a robust classifier for predicting the *SpType-ELS* attribute, representing the estimated spectral class from the Gaia project. This classification task involves predicting a categorical attribute with classes A and B. To achieve this, the following systematic approach was undertaken:

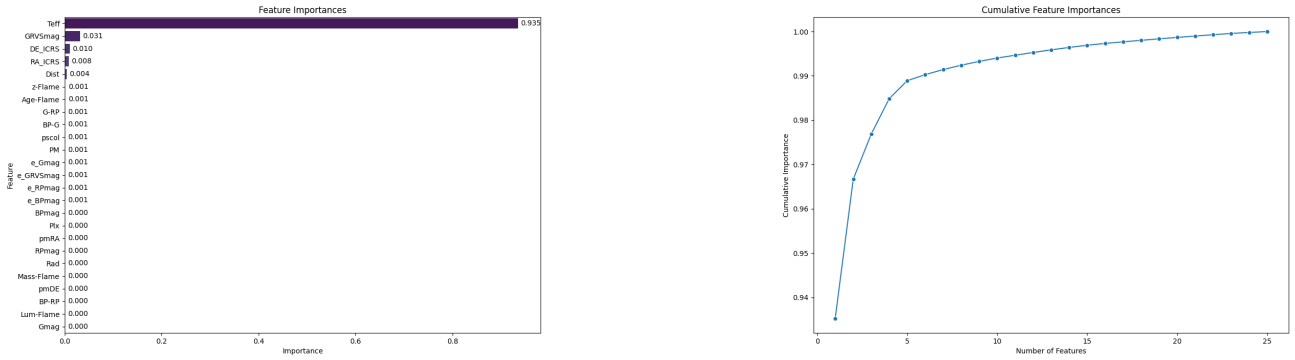
Initial data exploration was conducted to gain insights into data characteristics, distributions, and attribute importance. This preliminary analysis included visualizing distributions and identifying potential correlations, providing a foundational understanding necessary for subsequent steps. Research was then carried out to identify suitable machine learning models and corresponding hyperparameters for tuning. Based on this research, a robust machine learning pipeline was defined to facilitate systematic experimentation. The pipeline incorporated steps for data preprocessing, model training, hyperparameter tuning, and validation, ensuring consistency and reproducibility across multiple experiments. Afterewards, models were trained and evaluated against predefined metrics such as accuracy, precision, recall, and F1-score. This evaluation assessed multiple metrics. Finally, the best-performing classifier from the evaluation phase was applied to predict on unknown data. Table 1 summarizes the described workflow.

Step	**Description**
Initial Data Exploration	Conducted to gain insights into data characteristics, distributions, and attribute importance.
Model Research	Identified suitable models and corresponding hyperparameters for tuning.
ML Pipeline Definition	Established a robust ML pipeline for multiple experiments to optimize parameters and generate performance metrics for model comparison.
Model Evaluation	Evaluated models against each other based on defined metrics.
Final Prediction	Applied the best classifier to predict on unknown data.

Table 1: Systematic Approach for Developing the Classifier

4.2 Initial Insights

We began by analyzing the importance of features using tree-based methods. This hierarchical analysis helped identify the most influential features, which could be used later onduring feature selection. The result of this analysis is shown in Figure 1a and in Figure 1b. Different subgroups of features were defined and tested against each other. This step ensured that the most relevant features were used, potentially improving model performance and reducing overfitting. The feature subsets are shown in the Appendix (Table 8).



(a) Feature Importance

(b) Accumulated Feature Importance

Figure 1: Feature Importance Analysis: *Teff* is responsible for about 0.935 of the signal

4.3 Pipeline

The pipeline for developing a robust classifier for predicting the *SpType-ELS* attribute was carefully designed to ensure a systematic and reproducible approach. This process began with defining a comprehensive parameter grid for multiple

classifiers. This extensive grid allowed for a thorough exploration of model possibilities.

Data preprocessing was a crucial step, involving the use of pipelines to handle missing values and standardize numerical features. This preprocessing ensured that the data quality and consistency were maintained, directly impacting the model's performance. The next step was splitting the data into training and test sets using `train_test_split`, which created a set to validate the model's performance on unseen data, enhancing its robustness and generalizability.

To address the issue of imbalanced classes, `StratifiedKFold` was employed for cross-validation. This technique ensured that each fold had a representative proportion of each class, providing a more reliable estimate of model performance. Hyperparametertuning was managed using `GridSearchCV`, which systematically explored a range of hyperparameters, utilizing accuracy as the primary metric and incorporating `StratifiedKFold` to maintain balanced class distribution during validation.

The best pipeline was then evaluated using a comprehensive set of metrics: Accuracy, precision, recall, F1-score, area under the receiver operating characteristic curve (ROC-AUC). These metrics provided a multi-faceted view of the model's performance, ensuring it met various evaluation criteria. Visual tools like confusion matrices and ROC curves were used to gain insights into the model's accuracy and its ability to distinguish between classes.

Once the optimal model was identified, it was saved using `joblib.dump` for future use, ensuring that the best-performing model was readily available for deployment or further analysis. The optimized model was then applied to make predictions on unknown data, with results formatted appropriately for submission.

Table 2 summarizes the entire workflow, highlighting the systematic approach undertaken to develop a robust classifier for the *SpType-ELS* attribute. The corresponding Python code can be found in the Appendix (Listing C.1).

Step	**Explanation**
Define Parameter Grid	Set up a grid of hyperparameters for different classifiers to explore optimal settings using `param_grid`.
Feature Selection	Removing of non-relevant columns and feature selection to train the data on a feature subset.
Data Preprocessing	Preprocess the data using pipelines for imputing missing values and standardizing numerical features.
Split Data	Split data into training and test sets using `train_test_split`.
Encode Target	Encode target labels to numerical values using `LabelEncoder`.
Cross-Validation Setup	Implement `StratifiedKFold` for cross-validation to ensure balanced class distribution in each fold.
Grid Search	Perform hyperparameter tuning using `GridSearchCV` to find the best model and parameters based on cross-validation accuracy.
Model Evaluation	Evaluate the best model using metrics like accuracy, precision, recall, F1-score, and ROC-AUC, and visualize performance with confusion matrices.
Save Best Model	Save the best model and pipeline using `joblib.dump` for future use.
Generate Predictions	Use the optimal model to make predictions on unknown data and format results for submission.
Document Results	Log the best configurations and their performance metrics for reproducibility and future reference.

Table 2: Pipeline Description

5 Classification Techniques

This section enumerates various classifiers used in the analysis, an overview of each. For each classifier, a brief description, contextual background, key strengths and weaknesses, typical application domains, and relevant parameters are discussed.

In class, we explored the following unsupervised (Clustering in Week 6) and supervised techniques: Decision Trees (Week 7), Neural Networks (Week 9), SVM (Week 10), and Ensemble Methods including Bagging, Random Forest, and Boosting (Week 11).

Table 3 summarizes the classifiers used, mapping them to the techniques discussed in class where applicable.

Technique	**Concrete Method / Python Function**	**Week**
Decision Tree	`DecisionTreeClassifier()`	Week 7
Logistic Regression	`LogisticRegression()`	new
Random Forest	`RandomForestClassifier()`	Week 11
K-Nearest Neighbors	`KNeighborsClassifier()`	444
Support Vector Classifier	`SVC()`	Week 10
Multi-layer Perceptron	`MLPClassifier()`	Week 9
XGBoost Classifier	`XGBClassifier()`	new
Bagging Classifier	`BaggingClassifier()`	Week 11
AdaBoost Classifier	`AdaBoostClassifier()`	Week 11
Gradient Boosting Classifier	`GradientBoostingClassifier()`	new

Table 3: Mapping of classifiers to discussed techniques in class

5.1 DecisionTreeClassifier

The DecisionTreeClassifier (DCT) constructs a decision tree where nodes represent features, branches represent decision rules, and leaves represent outcomes. This intuitive model handles categorical data well but can overfit. It is commonly used in operational risk assessment and customer segmentation. Key parameters include *max_depth* to limit tree depth and prevent overfitting, *min_samples_split* for the minimum samples required to split a node, and *min_samples_leaf* for the minimum samples required at a leaf node.

5.2 LogisticRegression

LogisticRegression (LR) is utilized for binary classification, estimating probabilities via logistic function. Robust to noise and efficient for small datasets, it may underperform with non-linear problems. It is widely applied in medical fields and financial forecasting. Important parameters are C, the inverse of regularization strength (smaller values indicate stronger regularization), and *solver*, which specifies the optimization algorithm.

5.3 RandomForestClassifier

The RandomForestClassifier (RFC) builds multiple decision trees and aggregates their predictions for accuracy and stability. Versatile and robust against overfitting, it can be slow for prediction. Commonly used in bioinformatics for gene classification, its key parameters are *n_estimators*, the number of trees, and *max_features*, the number of features to consider for the best split.

5.4 KNeighborsClassifier

KNeighborsClassifier (KNC) uses the k-nearest-neighbors of each query point for learning. Simple and effective for small datasets, it struggles with high dimensionality and large datasets. It is often used in recommendation systems and pattern recognition. Key parameters include *n_neighbors*, the number of neighbors used, and *weights*, the weight function for prediction.

5.5 SVC (Support Vector Classifier)

SVC constructs a hyperplane in a high-dimensional space to separate classes with the widest margin. Effective in high-dimensional spaces and versatile, it may be memory-intensive. Common applications include image recognition and text classification. Important parameters are C, the penalty parameter of the error term, and *kernel*, which specifies the kernel type.

5.6 MLPClassifier (Multi-layer Perceptron Classifier)

MLPClassifier (MLPC), a type of neural network, consists of input, hidden, and output layers. Suitable for complex non-linear modeling, it requires careful parameter tuning and is sensible to feature scaling. It is commonly used in speech recognition and image processing. Key parameters include *hidden_layer_sizes* for the number of neurons in hidden layers, *activation* for the hidden layer function, and *learning_rate_init* for the initial learning rate.

5.7 XGBClassifier (XGBoost Classifier)

XGBClassifier (XGBC) utilizes a gradient boosting framework, excelling in performance and speed on large datasets but prone to overfitting if not properly tuned. It is often used in ranking systems and predictive analytics. Important parameters include *n_estimators*, the number of boosted trees, *max_depth*, the tree depth, *learning_rate*, the boosting learning rate, *subsample*, the subsample ratio, and *colsample_bytree*, the column subsample ratio.

5.8 BaggingClassifier

BaggingClassifier (BC) reduces variance and avoids overfitting by fitting base classifiers on random dataset subsets and aggregating their predictions. Robust but computationally expensive, it is typically used in ecological modeling and ensemble methods research. Key parameters include *n_estimators*, the number of base estimators, *max_samples*, the number of samples drawn for each estimator, and *max_features*, the number of features drawn for each estimator.

5.9 AdaBoostClassifier

AdaBoostClassifier (ABC) combines multiple weak classifiers to form a robust classifier, sensitive to noise and outliers. It is often used for binary classification in image recognition. Important parameters are *n_estimators*, the maximum number of estimators for boosting, and *learning_rate*, the weight applied to each classifier at each iteration.

5.10 GradientBoostingClassifier

GradientBoostingClassifier (GDC) builds an additive model, optimizing arbitrary differentiable loss functions. Prone to overfitting without proper tuning, it is widely used in web search ranking and ecology. Key parameters include *n_estimators*, the number of boosting stages, *learning_rate*, the rate at which each tree's contribution shrinks, and *max_depth*, which limits the number of nodes in each tree.

6 Summary

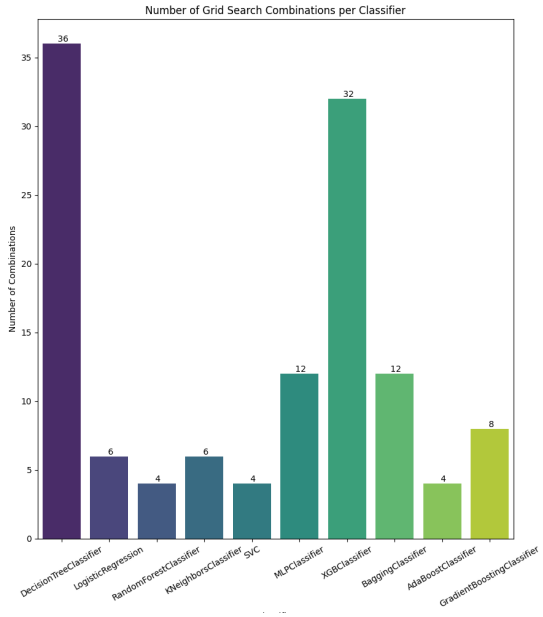
In this section, we first summarise the evaluation workflow before we present the results.

6.1 Evaluation Workflow

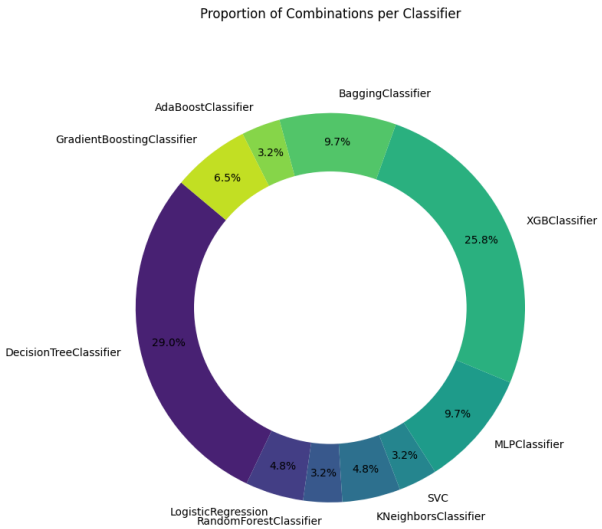
The best model and corresponding parameters were determined as follows: Two distinct settings were utilized, corresponding to two independent script executions, with one using 50% and the other using 100% of the available training data. The complete pipeline was then run for four different subsets of the data. Each pipeline employed `GridSearch` to identify the optimal parameters for ten different models. The parameter grid is shown in Table 4: In total, 124 configurations, an in-depth distribution is depicted in Figure 2, were evaluated using stratified 5-fold cross-validation. For each subset of features, the pipeline identified the best parameters based on model accuracy. The best model was then proposed and evaluated against the test data, with this final evaluation including multiple metrics.

Classifier	**Parameter Grid**
Decision Tree Classifier	max_depth: [None, 10, 20, 30], min_samples_split: [2, 10, 20], min_samples_leaf: [1, 5, 10]
Logistic Regression Classifier	C: [0.1, 1.0, 10], solver: [liblinear, lbfgs]
Random Forest Classifier	n_estimators: [200, 300], max_features: [sqrt, log2]
KNN (K-Nearest Neighbors)	n_neighbors: [1, 3, 5], weights: [uniform, distance]
SVC (Support Vector Classifier)	C: [1, 2], kernel: [rbf, sigmoid]
MLPC (Multi-layer Perceptron Classifier)	hidden_layer_sizes: [(50,), (100,), (50, 50)], activation: [tanh, relu], learning_rate_init: [0.01, 0.1]
XGBC (XGBoost Classifier)	n_estimators: [100, 300], max_depth: [1, 2], learning_rate: [0.1, 0.3], subsample: [0.7, 1], colsample_bytree: [0.7, 1]
Bagging Classifier	n_estimators: [10, 50, 100], max_samples: [0.5, 1.0], max_features: [0.5, 1.0]
AdaBoost Classifier	n_estimators: [50, 100], learning_rate: [0.01, 0.1, 1]
Gradient Boosting Classifier	n_estimators: [10, 100], learning_rate: [0.1, 0.3], max_depth: [3, 5]

Table 4: Hyperparameters for each model to be tuned during `GridSearch`



(a) Number of `GridSearch` Combinations per Classifier (124 in total)



(b) Proportion of Combinations per Classifier

Figure 2: Analysis of `GridSearch` Combinations: The left plot shows the number of grid search combinations for each classifier, while the right plot illustrates the proportion of these combinations relative to the total combinations across all classifiers.

6.2 Results

After identifying the optimal parameters for each of the 10 models, training was conducted for each feature subset (4 in total). This process was repeated twice: once with a sample of 50% of the training data and once with the entire training dataset. In total, this yields $10 * 4 * 2 = 80$ accuracy values, as shown in Table 5. The pipeline also suggests the best classifier among the 10 models for each run, which is displayed in the last column of the table (*Best*). The abbreviations used in the previous table are explained in Table 9. Table 10 summarizes all best-parameter configurations. In addition to that, Figure 3 provides a heat map containing all accuracy values using visual encoding.

Subset	DTC	LR	RFC	KNC	SVC	MLPC	XGBC	BC	ABC	GBC	Average per set	Best
50% Data - Subset 1	0.970	0.538	0.944	0.969	0.476	0.969	0.967	0.966	0.970	0.970	0.8739	DCT
50% Data - Subset 2	0.978	0.976	0.969	0.976	0.968	0.978	0.977	0.974	0.978	0.978	0.9752	ABC
50% Data - Subset 3	0.994	0.991	0.996	0.993	0.995	0.995	0.996	0.995	0.995	0.996	0.9946	XGBC
50% Data - Subset 4	0.995	0.993	0.996	0.989	0.995	0.996	0.997	0.996	0.996	0.997	0.9950	XGBC
100% Data - Subset 1	0.970	0.539	0.944	0.969	0.741	0.969	0.967	0.966	0.970	0.970	0.9005	DCT
100% Data - Subset 2	0.978	0.976	0.973	0.977	0.936	0.978	0.977	0.976	0.978	0.978	0.9727	ABC
100% Data - Subset 3	0.995	0.991	0.996	0.994	0.994	0.995	0.996	0.996	0.995	0.996	0.9947	XGBC
100% Data - Subset 4	0.995	0.993	0.997	0.990	0.995	0.997	0.997	0.997	0.996	0.997	0.9954	XGBC
Average of Classifier	0.984	0.875	0.977	0.982	0.888	0.984	0.984	0.983	0.983	0.983		

Table 5: Accuracy scores of the best model parameters across different subsets and data configurations.

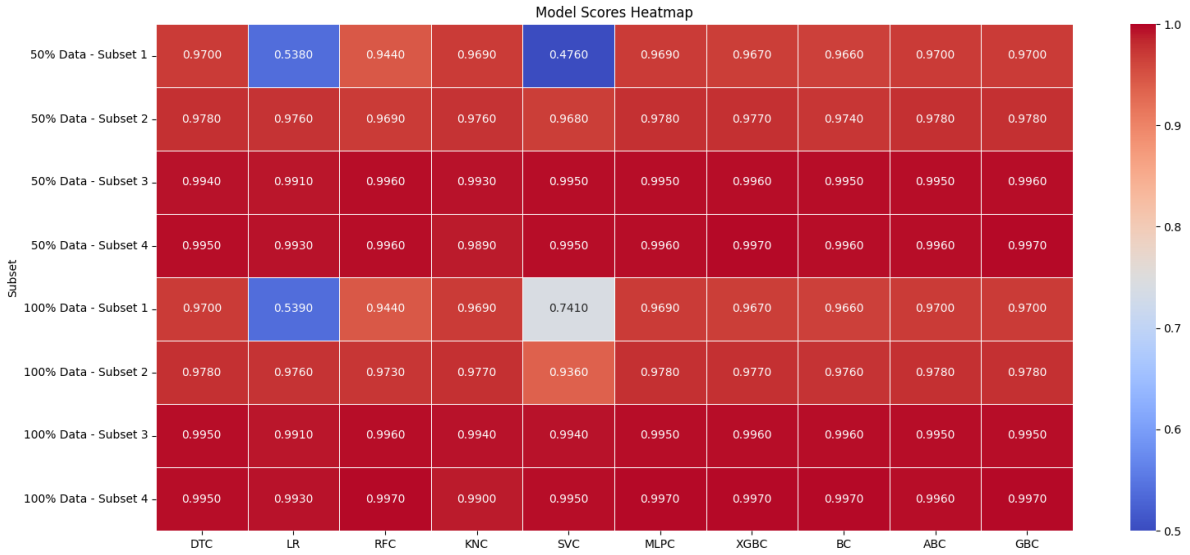


Figure 3: Heatmap visualizing the performance of various models with optimal parameters across two different training data sizes (50% and 100%) and multiple feature sets. The scores are displayed with increased precision to highlight the subtle differences in model performance. This visualization aids in identifying the most effective models and configurations.

To incorporate additional dimensions, Table 6 was created to compare the best performing classifiers on a subset of the data using state-of-the-art machine learning performance metrics.

Subset	Classifier	Accuracy	Precision	Recall	Sensitivity	F1-Score
50% Data - Subset 1	DCT	0.973	0.991	0.932	0.932	0.965
50% Data - Subset 2	ABC	0.975	0.956	0.929	0.981	0.977
50% Data - Subset 3	XGBC	0.992	0.994	0.995	0.995	0.995
50% Data - Subset 4	XGBC	0.994	0.996	0.997	0.997	0.996
100% Data - Subset 1	DCT	0.971	0.994	0.938	0.938	0.968
100% Data - Subset 2	ABC	0.972	0.957	0.931	0.991	0.978
100% Data - Subset 3	XGBC	0.993	0.995	0.996	0.996	0.996
100% Data - Subset 4	XGBC	0.997	0.997	0.998	0.997	0.997

Table 6: In-depth performance metrics of the best model parameters across different subsets and data configurations.

Analyzing the results, we observe several interesting patterns:

1. **Subset Performance Consistency**: For both 50% and 100% data scenarios, subsets 3 and 4 consistently yield the highest accuracy scores across all classifiers. This suggests that these subsets, which use a more comprehensive set of features, may contain particularly relevant or informative features for classification tasks.

2. **Top Performers**: Across all configurations, DTC, MLPC, and XGBC consistently perform at the top, often achieving near-perfect accuracy.

3. **Logistic Regression and SVC**: These classifiers exhibit lower average accuracies, particularly when trained on only 50% of the data. However, they also show substantial improvement with the complete dataset, indicating sensitivity to data size.

4. **Impact of Data Size and Feature Selection**: The transition from 50% to 100% data results in noticeable improvements in accuracy for all classifiers, particularly for LR and SVC. This underscores the importance of

data size in achieving optimal model performance. Additionally, subsets using a more comprehensive set of features (subsets 3 and 4) generally lead to higher accuracy, highlighting the significance of feature selection in model performance.

5. **Average Performance**: The average accuracy across classifiers is high, with most models exceeding 97% on average, except for LR and SVC, which have average accuracies of 87.5% and 88.8%, respectively.

The interested reader can find further material in the Appendix: Both, Figure 5 and Figure 6 visualize the results regarding accuracy of the best classifier, as well as Figure 7 and Figure 8 show the corresponding heatmaps.

7 Best classifier

In this section, we present and justify the best classifier and the corresponding hyperparameters based on our results.

7.1 Justification for Selecting XGBoost Classifier

The best model was identified by training on the entire dataset using the complete feature set. The chosen classifier is the XGBC, which achieved the highest accuracy scores consistently across different data subsets and configurations. We justify this by looking at:

- **Accuracy**: XGBC consistently achieved the highest accuracy across all subsets and data configurations, making it the most reliable model.
- **Precision and Recall**: The near-perfect precision and recall indicate that XGBC effectively balances false positives and false negatives, which is crucial for many applications.
- **F1 Score**: High F1 scores across all subsets reflect the model's overall effectiveness, combining both precision and recall.
- **Performance**: XGBC demonstrated superior performance with the highest average accuracy (99.5%) across all data configurations. This model effectively handled both small and large datasets, showing minimal variance in performance, which is crucial for robustness.
- **Advanced Boosting Mechanism**: XGBC utilizes gradient boosting, an ensemble technique that iteratively improves model predictions by correcting errors from previous iterations. This allows XGBC to capture complex patterns in the data, making it particularly effective for high-dimensional datasets.
- **Comparative Analysis**:
  1. **DTC**: While DTC showed strong performance, especially in subsets 3 and 4, it often overfits the data. XGBC, by contrast, mitigates overfitting through regularization parameters and ensemble learning.
  2. **LR**: LR struggled with lower accuracies, particularly with smaller datasets (50% data). It assumes a linear relationship between features and the target variable, which is often an oversimplification in complex classification tasks. XGBC's ability to model non-linear relationships gave it a clear advantage.
  3. **SVC**: SVC performed well with larger datasets but had higher computational costs and sensitivity to parameter selection. XGBC, with its scalable nature and built-in parameter tuning, offered more consistent and efficient performance.

7.2 Conclusion

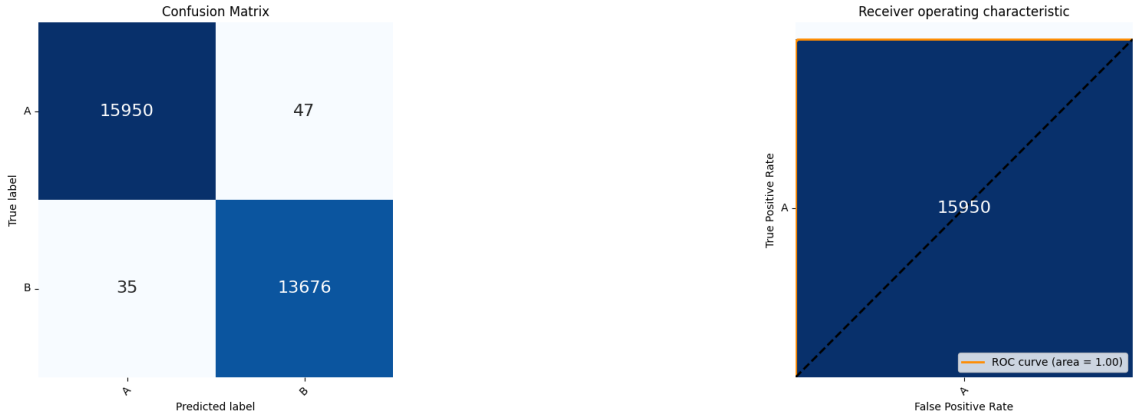
XGBoost Classifier was selected due to its high accuracy, robustness, and ability to handle large, complex datasets efficiently. Its advanced boosting mechanism outperformed simpler models like DTC and LR, which either overfit or underperformed on smaller datasets. Additionally, the comprehensive feature set used in training further leveraged XGBC's strengths in capturing complex data patterns. These factors lead to th selection of XGBC as the best classifier for this problem. Figure 4 underlines those findings with corresponding ROC curves and heatmaps, providing a comprehensive visual analysis of model performance across different configurations. Therefore, we propose XGBC as the best model with the following configuration:

- *colsample_bytree*: 0.7
- *learning_rate*: 0.3
- *max_depth*: 2
- *n_estimators*: 300
- *subsample*: 1

Table 7 further illustrates relevant metrics for the XGBC as best classifier with the aforementioned hyperparameters. In the appendix, the interested reader can find the detailed calculation in Section D.1.

Metric	Formula	Value
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	0.9972
Precision	$\frac{TP}{TP+FP}$	0.9966
Recall (Sensitivity)	$\frac{TP}{TP+FN}$	0.9975
Specificity	$\frac{TN}{TN+FP}$	0.9971
F1 Score	$2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}$	0.9970
False Positive Rate (FPR)	$\frac{FP}{FP+TN}$	0.0029
False Negative Rate (FNR)	$\frac{FN}{FN+TP}$	0.0026
False Discovery Rate (FDR)	$\frac{FP}{TP+FP}$	0.0034

Table 7: Confusion Matrix Metrics of the best classifier



(a) Heatmap of the best classifier XGBC.

(b) ROC curve

Figure 4: Performance of the best and proposed classifier XGBC using 100% of the training data and all features (100% Data - Subset 4)

8 Reflection

Assignment 3 provided a deep dive into the nature of data mining, offering valuable insights and personal development. In this section, I first describe my personal leanings regarding data mining, then I list some learnings about myself and I finally propose some approaches that I want to pursue for future problems.

8.1 Learning about Data Mining

One of the most important lessons learned was the importance of thorough data exploration, which we did as Assignment I, before the modeling phase. This initial step is crucial for uncovering patterns, identifying anomalies, and understanding the dataset's structure. Proper data exploration sets a strong foundation for subsequent steps, ensuring that the chosen models and methods are well-informed and appropriate for the data at hand.

The assignment introduced me to the practice of comparing different classifiers, a task that was highly educational. I could consolidate my understanding to evaluate various classifiers by tuning their hyperparameters and comparing their performance using different metrics. This comparison involved understanding and applying various evaluation metrics, to determine the most effective classifier for the given problem. The process of hyperparameter tuning was particularly beneficial, as it demonstrated how fine-tuning model parameters could significantly impact performance.

8.2 Learning about Myself

Through the course of this assignment, I discovered certain tendencies in my working style. I realized that I sometimes focus excessively on small details, which can lead to overengineering solutions and losing sight of the broader objectives. This unhealthy attention to detail, while beneficial in some contexts, proved to be a downside in maintaining a streamlined and efficient workflow.

Moreover, I found that my initial approach using Jupyter [2] notebooks became quite messy and difficult to manage as the complexity of the project increased. This experience highlighted the necessity of employing pipelines and more structured coding practices from the outset to maintain clarity and organization.

Through this assignment, I also discovered the value of engaging with others. I truly enjoyed helping and discussing the topic with my peers, as these interactions significantly enhanced my understanding. Sharing ideas and insights not only clarified my own thoughts but also introduced me to different perspectives and approaches.

8.3 Approach for Future Problems

If I were to approach this problem again, there are several strategies I would implement to enhance my efficiency and effectiveness:

- **Utilize Pipelines from the Start:** Implementing pipelines at the beginning of the project would help in maintaining a clean and organized workflow. Pipelines facilitate seamless data preprocessing, model training, and evaluation, reducing the chances of errors and improving reproducibility.
- **Invest More Time in Understanding Models and Parameters:** Dedicating more time to reading and comprehending the documentation and theoretical foundations of different models and their parameters would provide deeper insights and enable more informed decisions during model selection and tuning.
- **Valuable Discussions:** Moving forward, I plan to continue leveraging discussions with colleagues and peers as a crucial part of my learning process. Although I am by no means an expert yet, these conversations have proven to be valuable in improving my comprehension.
- **Streamline Hyperparameter Tuning:** Using automated hyperparameter tuning techniques, such as grid search, from the beginning would save time and ensure a more comprehensive search for the optimal parameters.
- **Balance Detail Orientation with Broader Focus:** While attention to detail is valuable, it's crucial to balance it with maintaining a focus on the overarching goals and objectives of the project. This balance would prevent overengineering and ensure that the solutions remain practical and efficient.

In conclusion, Assignment 3 was a great learning experience that enhanced my understanding of data mining and provided insights into my personal working habits. By applying the lessons learned and adopting a more structured and balanced approach, I can improve my future data mining projects and achieve more effective and efficient outcomes.

[2] https://jupyter.org/

A Appendix

A.1 Source Code

The interested reader can find the source code here.

B Data

B.1 Feature Subsets

Feature (Sub)Set	Number of Features	Accumulated Signal	Feature Names
evaluation/e_1_1f	1	0.0315	*GRVSmag*
evaluation/e_2_1f	1	0.9351	*Teff*
evaluation/e_3_5f	5	0.9891	*Teff*, *GRVSmag*, *DE_ICRS*, *RA_ICRS*, *Dist*
evaluation/e_4_25f	25	1.0000	*RA_ICRS*, *DE_ICRS*, *Plx*, *PM*, *pmRA*, *pmDE*, *Gmag*, *e_Gmag*, *BPmag*, *e_BPmag*, *RPmag*, *e_RPmag*, *GRVSmag*, *e_GRVSmag*, *BP-RP*, *BP-G*, *G-RP*, *pscol*, *Teff*, *Dist*, *Rad*, *Lum-Flame*, *Mass-Flame*, *Age-Flame*, *z-Flame*

Table 8: Feature Configurations and their Importance

C Pipeline

C.1 Python Pipeline Code using `Pipeline()`

Listing 1: Python code demonstrating the core of the ML pipeline.

```
for config in ModelConfig.configs:
    df_train = pd.read_csv(f"{DataConfig.data_path}/{DataConfig.training_data_filename}")
    selected_features = config['features']
    config_nr = config['config_nr']

    selected_features.append(ModelConfig.target_column)
    print(config['features'])

    df_train = df_train[selected_features]
    print(df_train.columns)

    # Speed up during development
    if Settings.dev:
        df_train = df_train.sample(frac=Settings.sample_size)

    kf = StratifiedKFold(n_splits=Settings.stratified_k_fold_n_splits, shuffle=True, random_state=Settings.random_state)

    # Separate features and target
    X = df_train.drop(columns=[ModelConfig.target_column])
    y = df_train[ModelConfig.target_column].str.strip()

    # Define Scoring Metrics
    scoring = {
        'accuracy': make_scorer(accuracy_score),
        'precision': make_scorer(precision_score, average='weighted'),
        'recall': make_scorer(recall_score, average='weighted'),
        'f1_score': make_scorer(f1_score, average='macro'),
        'roc_auc': 'roc_auc_ovr' if len(np.unique(y)) > 2 else 'roc_auc'  # Handle binary and multi-class
    }

    # Encode the target
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)

    # Split the data into training and validation sets
    X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=ModelConfig.test_size,
```

```
                                                    shuffle=True,
                                                    random_state=Settings.random_state)

    # Setting up preprocessing for numerical columns
    numerical_features = X_train.select_dtypes(include=[np.number]).columns.tolist()
    numerical_pipeline = Pipeline([
        ('impute', SimpleImputer(missing_values=np.nan, strategy='median')),
        ('scaler', StandardScaler())
    ])

    # Combining preprocessing steps into a single transformer
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numerical_pipeline, numerical_features),
        ],
        remainder='passthrough'
    )

    # Creating the full pipeline
    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('classifier', None)
    ])

    # Create the GridSearchCV object
    grid_search = GridSearchCV(pipeline, param_grid, cv=kf, scoring='accuracy', verbose=3, n_jobs=-1)

    # Fit GridSearchCV
    grid_search.fit(X_train, y_train)

    # Use the best estimator for predictions
    best_pipeline = grid_search.best_estimator_
    results = cross_validate(best_pipeline, X, y_encoded, cv=kf, scoring=scoring)
    # Evaluate the model (how it performs on unseen data)
    pprint(results)

    # Use best model and parameters to calculate confusion matrix
    y_pred = best_pipeline.predict(X_test)  # best_model = grid_search.best_estimator_
    cm = confusion_matrix(y_test, y_pred)

    # Set up the matplotlib figure
    plt.figure(figsize=(8, 6))

    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', cbar=False, square=True,
                annot_kws={"size": 16})  # 'fmt' specifies numeric formatting to integers

    # Labels, title, and ticks
    plt.title('Confusion Matrix')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    class_labels = label_encoder.classes_
    plt.xticks(np.arange(len(class_labels)) + 0.5, class_labels, rotation=45)
    plt.yticks(np.arange(len(class_labels)) + 0.5, class_labels, rotation=0)
    heatmap_path = f"{config_nr}/{EvaluationConfig.heatmap_plot_path}"
    ensure_dir(heatmap_path)
    plt.savefig(heatmap_path)
    print(f"Heat map plot saved as '{heatmap_path}'.")

    # Save pipeline
    pipeline_path = f"{config_nr}/{EvaluationConfig.pipeline_path}"
    ensure_dir(pipeline_path)
    dump(best_pipeline, pipeline_path)
    print(f"Pipeline has been saved to {pipeline_path}")

    # Dictionary to hold the best configuration for each classifier type
    logfile_path = f"{config_nr}/{EvaluationConfig.logfile_path}"
    ensure_dir(logfile_path)
    with open(logfile_path, 'w') as logfile:
        # Write a header for the log file
        logfile.write("Model and Parameters - Scores\n")
        logfile.write("-" * 50 + "\n")

        #  Write confusion matrix
        logfile.write(f"Confusion Matrix:\n{cm}\n")
        logfile.write("-" * 50 + "\n")

        # Iterate over each set of parameters and corresponding results
        for i in range(len(grid_search.cv_results_['params'])):
            # Extract the parameters and scores
            params = grid_search.cv_results_['params'][i]
            mean_test_score = grid_search.cv_results_['mean_test_score'][i]
            std_test_score = grid_search.cv_results_['std_test_score'][i]

            # Create a log entry for this parameter set
            model_details = f"Model: {params['classifier'].__class__.__name__}, Params: {params}, "
            scores_details = f"Mean Score: {mean_test_score:.3f}, Std Dev: {std_test_score:.3f}\n"

            # Write the combined details to the log file
            logfile.write(model_details + scores_details)
    print(f"GridSearchCV results have been saved to {EvaluationConfig.logfile_path}")
```

```
    # Analyze the results to find the best configuration per model type
    best_configs = {}
    for i in range(len(grid_search.cv_results_['params'])):
        model_type = grid_search.cv_results_['params'][i]['classifier'].__class__.__name__
        model_score = grid_search.cv_results_['mean_test_score'][i]

        if model_type not in best_configs or model_score > best_configs[model_type]['score']:
            best_configs[model_type] = {
                'params': grid_search.cv_results_['params'][i],
                'score': model_score
            }

    best_configs_file_path = f"{config_nr}/{EvaluationConfig.best_configs_file}"
    ensure_dir(best_configs_file_path)
    # Write the best configurations to the file
    with open(best_configs_file_path, 'w') as f:
        for model, config in best_configs.items():
            f.write(f"Best configuration for {model}:\n")
            f.write(f"Parameters: {config['params']}\n")
            f.write(f"Score: {config['score']:.3f}\n")
            f.write("-" * 50 + "\n")

    print(f"Best configurations have been saved to {best_configs_file_path}")

    # Best parameters and score
    print("Best parameters overall:", grid_search.best_params_)
    print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))

    # Test accuracy
    print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))

    # Prepare to plot ROC curves
    y_prob = best_pipeline.predict_proba(X_test)
    if len(np.unique(y_encoded)) > 2:  # Multi-class case
        # We don't need this case.
        y_test_binarized = label_binarize(y_test, classes=np.unique(y_encoded))
        n_classes = y_test_binarized.shape[1]
        # Compute ROC curve and ROC area for each class
        fpr = dict()
        tpr = dict()
        roc_auc = dict()
        colors = cycle(['blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black'])
        for i in range(n_classes):
            fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_prob[:, i])
            roc_auc[i] = auc(fpr[i])
            # Plot ROC curve
            plt.plot(fpr[i], tpr[i], color=next(colors), lw=2,
                     label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i]))
    else:  # Binary case
        fpr, tpr, _ = roc_curve(y_test, y_prob[:, 1])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, color='darkorange', lw=2,
                 label='ROC curve (area = {0:0.2f})'.format(roc_auc))

    # Plotting the diagonal line
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")

    roc_plot_path = f"{config_nr}/{EvaluationConfig.ROC_plot_path}"
    ensure_dir(roc_plot_path)
    plt.savefig(roc_plot_path)
    print(f"ROC curves plot saved as '{roc_plot_path}'.")

    # Read the unknown (test) data
    df_unknown = pd.read_csv(f"{DataConfig.data_path}/{DataConfig.submission_data_filename}")

    # Make predictions
    y_unknown_pred_encoded = best_pipeline.predict(df_unknown)
    y_unknown_pred = label_encoder.inverse_transform(y_unknown_pred_encoded)

    # Format predictions for submission
    y_unknown_pred_formatted = [f"{label}      " for label in
                                y_unknown_pred]  # Adding spaces to match the expected format

    # Create submission DataFrame
    submission_df = pd.DataFrame({
        'ID': df_unknown['ID'],
        'SpType-ELS': y_unknown_pred_formatted
    })

    submission_file_path = f"{config_nr}/{DataConfig.data_path}/submission_file.csv"
    ensure_dir(submission_file_path)
    # Save the submission DataFrame to a .csv file
    submission_df.to_csv(submission_file_path, index=False)
```

```
219     print("Submission file created successfully!")
```

C.2 Best Model Parameters

Abbreviation	Full Model Name
ABC	AdaBoostClassifier
BC	BaggingClassifier
DTC	DecisionTreeClassifier
GBC	GradientBoostingClassifier
KNC	KNeighborsClassifier
LR	LogisticRegression
MLPC	MLPClassifier
RFC	RandomForestClassifier
SVC	SVC
XGBC	XGBClassifier

Table 9: Legend for the classifier abbreviations.

Dataset	Feature Set	Classifier	Parameters
50%	e_1_1f	DTC	max_leaf_nodes: None min_samples_leaf: 10 min_samples_split: 2
50%	e_2_1f	ABC	estimator: DTC (n_estimators: 50) learning_rate: 0.1
50%	e_3_5f	XGBC	colsample_bytree: 0.7 learning_rate: 0.1 max_depth: 2 n_estimators: 300 subsample: 0.7
50%	e_4_25f	XGBC	colsample_bytree: 1 learning_rate: 0.3 max_depth: 2 n_estimators: 300 subsample: 1
100%	e_1_1f	DTC	max_leaf_nodes: None min_samples_leaf: 10 min_samples_split: 2
100%	e_2_1f	ABC	estimator: DTC (n_estimators: 50) learning_rate: 0.1
100%	e_3_5f	XGBC	colsample_bytree: 1 learning_rate: 0.1 max_depth: 2 n_estimators: 300 subsample: 0.7
100%	e_4_25f	XGBC	colsample_bytree: 0.7 learning_rate: 0.3 max_depth: 2 n_estimators: 300 subsample: 1

Table 10: Best Classifier Parameters and Feature Sets for Different Datasets

D Evaluation

Figure 5: Bar chart illustrating the performance differences of different models on different feature sets (50% of the training data)

Figure 6: Bar chart illustrating the performance differences of different models on different feature sets (100% of the training data)

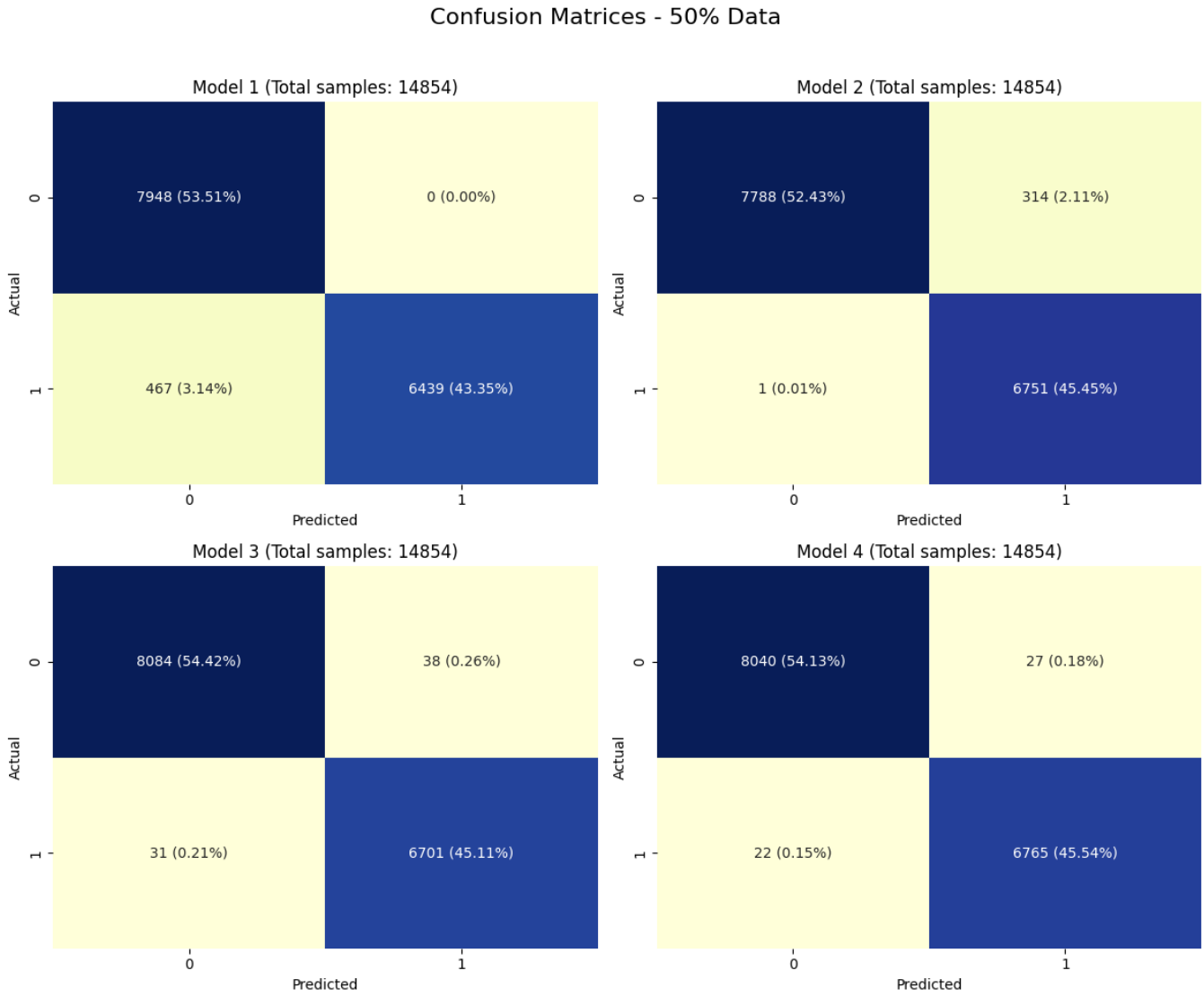


Figure 7: Heat maps visualizing the performance of the best model for the corresponding best classifiers, trained on 50% of the training data

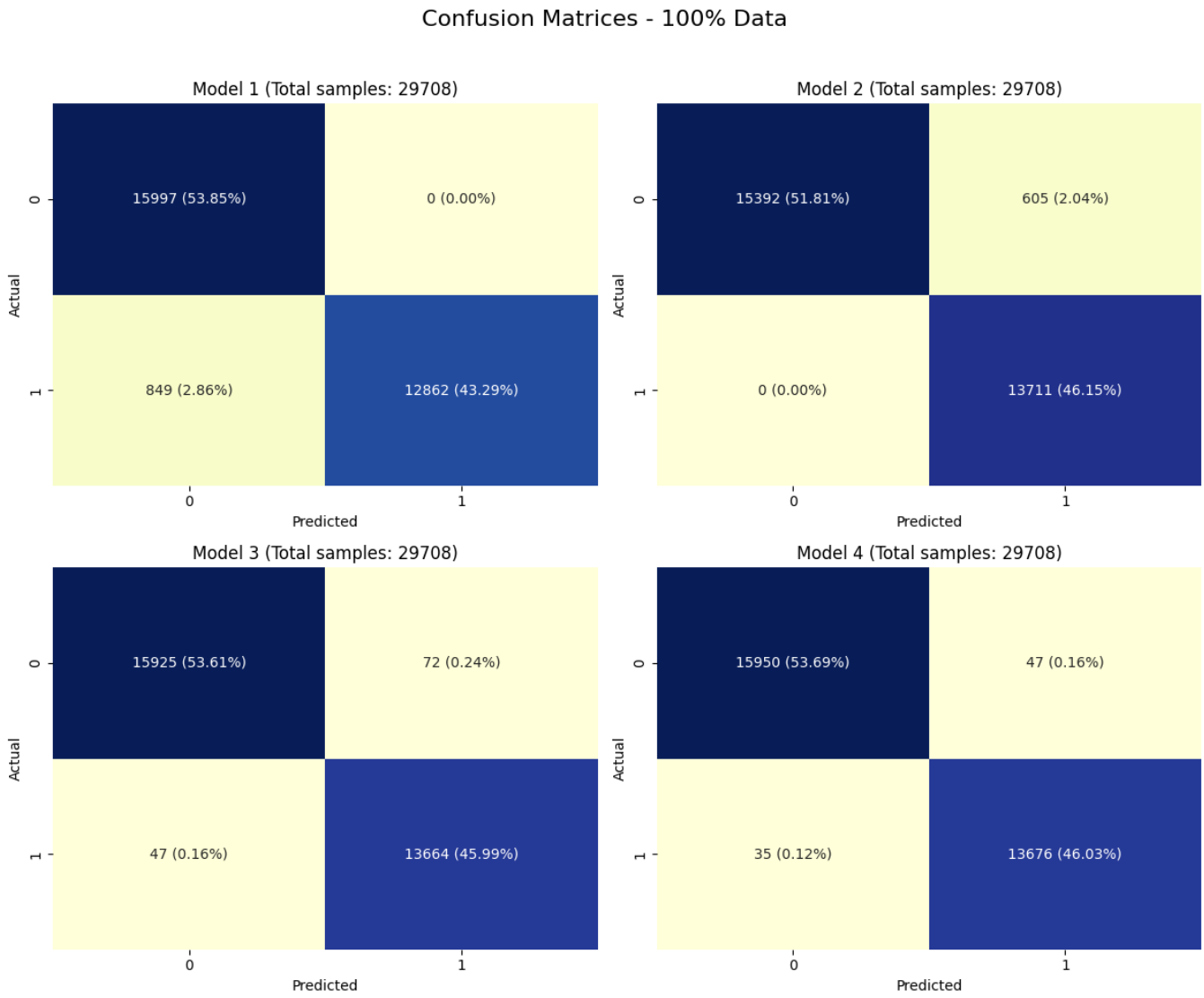


Figure 8: Heat maps visualizing the performance of the best model for the corresponding best classifiers, trained on 100% of the training data

D.1 In-depth Metrics calculations for the best model

Based on the following heat map,

$$\begin{bmatrix} 15950 & 47 \\ 35 & 13676 \end{bmatrix}$$

we evaluate the corresponding scores as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{13676 + 15950}{13676 + 15950 + 47 + 35} = \frac{29626}{29708} \approx 0.9972$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{13676}{13676 + 47} = \frac{13676}{13723} \approx 0.9966$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{13676}{13676 + 35} = \frac{13676}{13711} \approx 0.9975$$

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{15950}{15950 + 47} = \frac{15950}{15997} \approx 0.9971$$

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \cdot \frac{0.9966 \cdot 0.9975}{0.9966 + 0.9975} \approx 0.9970$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN} = \frac{47}{47 + 15950} = \frac{47}{15997} \approx 0.0029$$

$$\text{False Negative Rate (FNR)} = \frac{FN}{FN + TP} = \frac{35}{35 + 13676} = \frac{35}{13711} \approx 0.0026$$

$$\text{False Discovery Rate (FDR)} = \frac{FP}{TP + FP} = \frac{47}{13676 + 47} = \frac{47}{13723} \approx 0.0034$$