Exercise 1 - Classification

Machine Learning

Emil Daub (12143524), Jasper De Landsheere (12228451), Kristian Ristic (51845332)

April 30, 2023

Contents

1 Introduction

The objective of this task is to test a range of classification algorithms on diverse data sets. Different experiments involving various settings, parameters, and other factors are conducted with different classification models. Listed below are the used data sets and a short description of the set task:

- **Drug Consumption**: Given certain information of a person (including personality information) and his/her past drug use, predict the person's experience with ecstasy.
- **Bank Marketing**: Given personal data of a client as well as bank related client data, decide whether a client subscribes to a term deposit.
- **Breast Cancer**: Predict if a tumor is malignant (cancerous) or benign(non cancerous).
- **Loan**: Given application information, predict the score of a loan application.

1.1 Experiment overview

To make this process accessible, we aimed to keep the steps simple and easy to navigate, given the vast number of potential combinations. The approach involved conducting data exploration and initial preprocessing steps for each of the four data sets separately. Once we obtained the "clean" data sets, we merged them into a single notebook, where we trained the models and analyzed the results. In Section 2, Experiment setup, we define our chosen classifiers, preprocessing strategies, and performance metrics. In Section 3, we explore the data sets and prepare them for the experiment. Section 4 handles a case study on the Bank Marketing data set, showcasing the effect of hyperparameters. A summary of the results is given in Section 5. Finally, we end with some findings and further improvements in Section 6.

1.2 Navigation through the code

For each data set, a folder was created containing a notebook and the data needed for training and testing. Data exploration and early preprocessing steps such as categorical data encoding were done individually for each data-set. As a result, the processed dataframes contain n number of features (all numerical) and the target variable with the name "target" and are again exported as csv file in the "preprocessed-datasets" subfolder. In the main folder, named "finale", you can find the notebook with the name "comparison.ipynb", in which the four already processed data-sets have been imported. For the two data-sets involved in the competitions on Kaggle, other steps were carried out such as training and prediction after preprocessing. Functions have been created in the comparison.ipynb to be able to iterate over different types of settings and data, and display the results.

2 Experiment setup

2.1 The chosen classifiers

Regarding the models, we have opted for three different classifiers: K-Nearest Neighbors, Random Forest, and Logistic Regression. We selected these models as they can handle both numerical and categorical variables. Additionally, their diverse nature provides another layer of insight when interpreting the results.

1. Random Forest: *Decision tree based*. This model involves constructing multiple decision trees during the training phase. During classification, the output of the random forest is determined by selecting the class chosen by the majority of the trees. We use the sklearn implementation [link] and the hyperparameters that we will adjust are:
    - n_estimators: The number of trees in the forest.
    - max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
    - min_samples_split: The minimum number of samples required to split an internal node.

- min_samples_leaf: The minimum number of samples required to be at a leaf node.
- max_features: The number of features to consider when looking for the best split.
- bootstrap: Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

2. K-Nearest Neighbors (KNN): *Instance-based classifiers/Lazy learner*. This particular model classifies a sample based on a majority vote from its neighboring samples, with the aim of assigning it to the most common class among its k-nearest neighbors. We use the sklearn implementation [link] and the hyperparameters that we will adjust are:

   - n_neighbors: k, number of neighbors to use by default for kneighbors queries.
   - weights: Weight function used in prediction.
   - algorithm: Algorithm used to compute the nearest neighbors.
   - leaf_size: Leaf size passed to tree. This can affect the speed of the construction and query, as well as the memory required to store the tree.
   - p: Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

3. Logistic Regression: *Linear classifier*. A linear equation is transformed using a sigmoid function, resulting in an S-shaped curve that maps any input to a value between 0 and 1, which represents the probability of the binary outcome. In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme or the cross-entropy loss. We use the sklearn implementation [link] and the hyperparameters that we will adjust are:

   - C: Inverse of regularization strength, smaller values specify stronger regularization.
   - solver: Algorithm to use in the optimization problem.
   - max_iter: Maximum number of iterations taken for the solvers to converge.
   - penalty: We decide to keep it as l2.

Below (Figure 1), the parameter space for each classifier is shown:

```
param_space = {
    'RandomForestClassifier': {
        'model__n_estimators': [10, 50, 100, 200, 500],
        'model__max_depth': [None, 5, 10, 20, 50],
        'model__min_samples_split': [2, 5, 10, 20, 50],
        'model__min_samples_leaf': [1, 2, 5, 10, 20],
        'model__max_features': ['auto', 'sqrt', 'log2', None],
        'model__bootstrap': [True, False]
    },
    'KNeighborsClassifier': {
        'model__n_neighbors': [1, 2, 5, 10, 20, 50],
        'model__weights': ['uniform', 'distance'],
        'model__algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
        'model__leaf_size': [10, 20, 30, 50],
        'model__p': [1, 2]
    },
    'LogisticRegression': {
        'model__penalty': ['l2'],
        'model__C': [0.1, 0.5, 1.0, 2.0, 5.0],
        'model__solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
        'model__max_iter': [10, 50, 100, 200, 500]
    }
}
```

Figure 1: All hyperparameters considered per classifier.

2.2 Preprocessing strategies

Various approaches and combinations were considered during this stage of the experiment. In order to maintain simplicity and avoid unnecessary complexity, we opted to implement a single feature selection method and a single scaler for scaling the features. Although we initially experimented with different scalers from the sklearn library, we ultimately concluded that additional information gained from using multiple scalers across various datasets was largely redundant. Thus, a single scaler was deemed sufficient to represent the impact of the feature scaling process on our experiment. As a result, the following 8 combinations were generated for each classifier:

Data split	**Feature selection**	**Feature scaling**
5-fold CV	yes	yes
5-fold CV	yes	no
5-fold CV	no	yes
5-fold CV	no	no
train/test	yes	yes
train/test	yes	no
train/test	no	yes
train/test	no	no

Table 1: Overview of all different preprocessing strategies considered.

The effects of these preprocessing steps will be tested across four data sets in terms of F1 score. The results are showcased in Section 5.

2.2.1 Splitting strategy

The validation strategies compared in this study are the commonly used k-fold cross-validation and holdout validation. A 5-fold cross-validation and a 30%-70% train/test split are employed for these respective strategies.

2.2.2 Feature selection

The feature selection process was carried out using statistical testing of the correlation coefficients between each feature and the target variable. To achieve this, the following steps were implemented:

1. Calculate the correlation coefficients between each feature and the target variable.
2. Perform he F-test to assess the significance of the correlation between each feature and the target variable. The F-test is a statistical test used to evaluate the differences between the variances of two populations.
3. Get the p-values from the F-test for each feature.
4. Select only the features with a p-value below the significance level (alpha), which was set to 0.05.

This approach aimed to identify a statistically significant relationship between the features and the target variable while avoiding overcomplication of the feature selection process.

2.2.3 Feature scaling

It is well established in the literature that scaling the training data can improve the performance of a training model. To verify this fact, we decided to test it ourselves. We selected the StandardScaler() from the sklearn library as our scaler of choice, which scales the data of all features (which are already numeric) to have a mean of 0 and a standard deviation of 1. Bringing the data into a similar range is expected to improve the performance of some classifiers. To prevent indirect data leakage, we carefully applied the scaling step in each train split using a pipeline consisting of the preprocessing steps. This pipeline was executed in every train split and k-fold split in CV. It should be noted that scaling was not necessary for Random Forest in all settings, but we still applied it to maintain consistency across the experiment. However, as expected, the results did not show any significant improvement in this case.

2.3 Performance metrics

When dealing with classification problems, we require a quantitative measure to assess a model's performance and determine its effectiveness in executing tasks. This metric enables us to compare, enhance, and standardize the results of any machine learning problem. For this exercise, we seek a generalizable metric that is not specific to any particular scenario since we are evaluating various datasets and models.
Our metric of choice is the F1 score (weighted average), shown in (1). It is calculated as the harmonic mean between precision and recall, which are shown in (2) and (3), respectively. Precision calculates the true 'positive' observation ratio to the total observations predicted positive, while Recall, also called sensitivity, measures the proportion of positive observations correctly labeled. High precision reduces false positives, which is important in avoiding incorrect positive labels. A high recall identifies most positive observations, indicating good class prediction capability. For example, in the Breast Cancer data set, it is necessary to keep an eye on recall and not so much on accuracy, since not identifying positive cases can be fatal for patients.

$$F1\ Score = \frac{2 * (precision * recall)}{precision + recall} \tag{1}$$

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{2}$$

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{3}$$

However, since in this experiment we are mostly concerned with the interactions between models and data sets, we have chosen F1 score over precision and recall, as our metric of reference, because of it takes into account both false positives and false negatives. It is also helpful if classes are unbalanced.

3 Data exploration and preprocessing

3.1 Overview data sets

Table 2 presents a summary of the crucial features of the datasets. The balance of the datasets in terms of the feature types, the number of observations, and the number of classes, whether binary or multiclass, are according to the exercise's expectations.

Data set	Observations	NAs	Total features	Categorical features	Numerical features	Classes of target variable
Loan (Kaggle)	10000	No	90	13	77	A, ... , F
Breast Cancer (Kaggle)	285	No	30	0	30	true/false
Drug Consumption	1885	No	30	18	12	CL0, CL1, ..., CL6
Bank	45211	No	17	9	8	1(No) / 2(Yes)

Table 2: Specifics of the data sets considered in the experiment.

3.2 Bank-Marketing

3.2.1 Description

The bank marketing data set is used to classify whether a client of the bank subscribes to a term deposit. The data set consists of personal data of the client like age, job, marital, education. As well as bank related private client data like average yearly balance, housing loan, personal loan and data related to the contact between client and bank via telephone.

3.2.2 Preprocessing

The data set is complete without any missing values. The last contact with the client is captured in three features, including day, month, and pday, which represents the number of days that have passed since the last contact. Given that day and month convey the same information, one of them could be eliminated. In this case, it would be more practical to remove day and month instead of pday since the

latter allows for simpler comparisons. Additionally, five features are categorical, and three are binary (default, housing, and loan), so they need to be converted to numeric values. To prepare the data for analysis, the binary values were transformed into 0s and 1s, while the categorical features were encoded using the one-hot encoding technique. One-hot encoding is a process of converting categorical variables into numerical representation to make them usable for machine learning algorithms. This technique creates a binary vector for each category and assigns a value of 1 to the corresponding vector element, and 0 to all others. This enables the algorithm to understand the relationships between different categories and use them to make predictions.
A imbalance between the two classes are observed in the response variable, the SMOTE method can be used here.
As a result, the dataset now has 45211 instances and 39 features in its new shape.

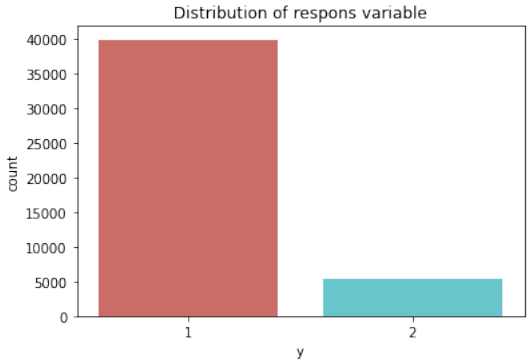

Figure 2: Distribution of response variable

3.3 Drug consumption

3.3.1 Description

The database contains records for 1885 respondents, with each respondent having 12 attributes associated with them. These attributes include NEO-FFI-R (neuroticism, extraversion, openness to experience, agreeableness, and conscientiousness), BIS-11 (impulsivity), ImpSS (sensation seeking), level of education, age, gender, country of residence, and ethnicity, which are all originally categorical and are quantified. Additionally, participants are asked about their use of 18 legal and illegal drugs. For each drug, respondents must select from one of seven answers: "Never used the drug", "Used it over a Decade Ago", "Used in Last Decade", "Used in Last Year", "Used in Last Month", "Used in Last Week", or "Used in Last Day". A fictitious drug, called Semeron, was also included in order to identify over-claimers. The data set does not contain missing values.
For this data set, we defined the task as predict a person's experience with the drug "ecstasy". Hence, it is a multiclass classification problem.

3.3.2 Preprocessing

First, column names were added and the ID column was dropped since it is irrelevant. Next, all instances where users claimed to have used semeron were dropped, since it is a fictitious drug introduced to identify over-claimers and ensure data quality. Subsequently, the whole semeron class was dropped. Then, all categorical values were label encoded, with "CLO" (which is "Never used the drug") becoming 0., "CL1" ("Used it over a Decade Ago") becoming 1., and so on.

3.4 Breast Cancer

3.4.1 Description

The Breast Cancer dataset is composed of 32 attributes, with the first attribute being an ID used to uniquely identify the measurements, and the second attribute being a binary response variable called

"class". The "class" attribute indicates whether the tumor is malignant or benign, with a value of true indicating a malignant tumor and false indicating a benign tumor.
The remaining 30 attributes are derived from digitized images of a fine-needle aspirate (FNA) of a breast mass. Specifically, these attributes describe various features of the cell nuclei present in the images. The attributes include: radius, texture, circumference, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension. Each of these features is stored three times, representing the mean, standard error, and "worst" or largest value (average of the three largest values) for each image.

3.4.2 Preprocessing

The preprocessing steps required for the Breast Cancer dataset are relatively straightforward. All the features in the dataset, except the response variable, are numeric variables. Furthermore, since the data was generated from images, there are no missing values. However, a closer examination of the response variable reveals that the class distribution is unbalanced (see Figure 3), which can lead to biased models.
Additionally, the binary values for the response variable, True and False, were converted into numeric values (1 and 0, respectively) to facilitate analysis. Finally, since the feature ID does not contain any special information beyond uniquely identifying the measured values, it was excluded from the training. Overall, these preprocessing steps help to ensure that the Breast Cancer dataset is well-suited for the development of machine learning models that can accurately predict the type of breast cancer.

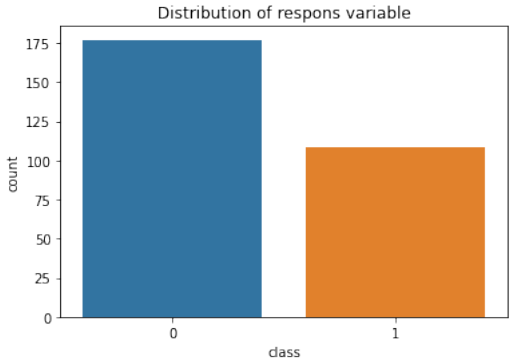

Figure 3: Distribution of response variable

3.5 Loan Dataset

3.5.1 Description

The loan consists of 10k rows representing loan instances. The task it to predict the class of a loan application, which varies from A to F. General loan details and application request's details are collected as features. In total we have 90 features, where 13 are categorical features and 77 numeric.

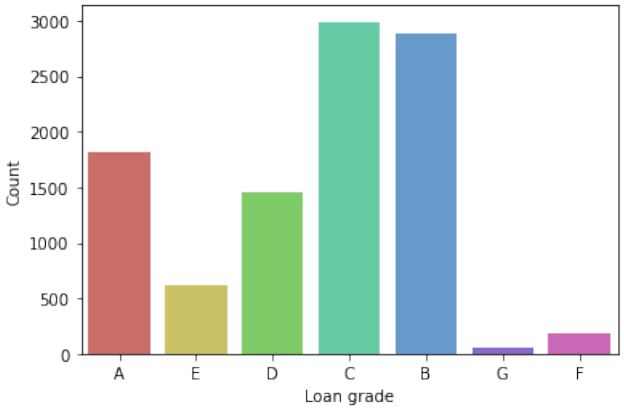

Figure 4: Distribution of "loan grade" variable

From the distribution of the target variable, in Figure 4, we can notice an unbalanced data set. A concern we need to deal with and be careful when evaluating mean metrics.

3.5.2 Preprocessing

The preprocessing step presented some challenges in this study. Initially, the variables of type 'object' were plotted to identify the categories present, revealing an ordered category that was resolved with label encoding. For all other variables, one-hot encoding was used to transform them into floats. It is worth noting that this step increased the number of features by almost 40%, which is not optimal for the models' performance.

4 Case study Bank Marketing data set: Responsiveness to hyperparameters

The analysis that follows provides a demonstration of the results obtained using the Bank dataset. The tables presented below exhibit the performance, measured by the cv F1-Score, of all models executed while exploring the model parameters. This analysis helps to understand the influence of various parameters on the performance and runtime of the models. To ensure consistency, we used the RandomizedSearchCV method throughout the analysis. Moreover, the dataset was scaled using the StandardScaler. RandomForestClassifier is the first classifier, the results are showcased in Table 3. The first entry in the table represents the run with default parameters.

n_estim	max_depth	m_s_s	m_s_l	max_features	boot	F1 score	runtime (ms)
100	None	2	1	sqrt	True	0.6017	4.3528
10	5	10	20	log2	True	0.7295	0.3178
100	10	50	1	log2	False	0.6858	3.9992
500	5	20	10	auto	False	0.6921	95.2382
50	5	20	10	auto	False	0.6126	15.3456
10	None	5	1	sqrt	True	0.5673	0.7567
500	None	20	2	None	False	0.2320	142.5949
50	5	50	2	sqrt	False	0.7178	1.3871

Table 3: F1 score and runtime of RandomForestClassifiers for different hyperparameters. m_s_s denotes min_samples_split, m_s_l denotes min_samples_leaf, and boot denotes bootstrap.

The table shows that the runtime increases as the number of n_estimators increases. This is because each estimator contributes to the overall runtime. Additionally, the max depth of the trees also has a significant impact on runtime since a deeper tree requires more computations. For the min_samples_split parameter, increasing this can lead to fewer splits and a more generalized model

that is less prone to overfitting, but it may also reduce the model's ability to capture complex relationships in the data.
For the min_samples_leaf parameter, the minimum number of samples required to be a leaf node. A split point at any depth will only be considered if it leaves at least min samples leaf training samples in each of the left and right branches. Larger values lead to simpler trees, which can underfit the data, while smaller values allow the tree to become more complex, which can lead to overfitting. Regrettably, the table does not demonstrate any noteworthy changes in the score and runtime, hence it is not feasible to provide a more detailed statement regarding the impact of min_samples_split and min_samples_leaf on the RandomForestClassifiers.
On the other hand, it is evident from the scores that a larger random forest does not necessarily lead to better performance. Instead, the key to achieving optimal performance lies in making a wise choice of individual parameters that are best suited for the dataset. Therefore, it is crucial to conduct a thorough parameter search and fine-tune the parameters to obtain the best results. The choice of appropriate parameters is highly dependent on the characteristics of the dataset, and there is no universal set of parameters that work for all datasets.
Moving forward, we will delve into a more detailed analysis of the KNeighborsClassifier.

n_neighbors	**weights**	**algorithm**	**leaf_size**	**p**	**F1 score**	**runtime (ms)**
5	uniform	auto	30	2	0.9103	0.0614
2	uniform	kd_tree	50	1	0.8490	0.6963
5	uniform	kd_tree	30	2	0.8450	0.7497
20	distance	brute	30	2	0.8649	0.0711
50	uniform	brute	20	1	0.8792	0.0694
5	uniform	auto	30	1	0.8400	0.0616
10	uniform	brute	50	1	0.8603	0.0609
5	distance	auto	20	1	0.8362	0.0747

Table 4: F1 score and runtime of KNeighborsClassifier for different hyperparameters.

- n_neighbors: The performance of the model varies with the number of neighbors chosen. For instance, a value of 50 for n_neighbors resulted in the best cv f1-Score of 0.9035. However, a lower value of 5 for n_neighbors also gave good results (cv f1-Score of 0.89697).
- weights: The choice of weight function did not seem to have a significant impact on the model's performance. Both 'uniform' and 'distance' weights gave comparable results.
- algorithm: The choice of algorithm can have a significant impact on the runtime. The 'brute' algorithm had the lowest runtime, while the 'kdtree' and 'auto' algorithms had higher runtimes.
- p: The choice of the power parameter did not seem to have a significant impact on the model's performance.

A smaller leaf size will lead to a slower runtime as it will force the algorithm to use brute-force search more often. Overall, the performance of the KNeighborsClassifier model varies significantly with different parameter values. Therefore, it is essential to conduct a thorough parameter search and fine-tune the parameters to achieve the best possible results for a given dataset. We will now discuss the final classifier, which is LogisticRegression.
Based on the information provided in Table 5, we can draw the following conclusions:

- The inverse regularization strength parameter 'C' has a significant impact on the model's performance. Lower values of C lead to stronger regularization and lower performance, while higher values lead to weaker regularization and higher performance. In this case, a value of 1.0 for C gave the best cv f1-Score of 0.9016.
- The choice of solver can have a significant impact on the runtime. The 'lbfgs' solver had the lowest runtime, while the 'sag' and 'liblinear' solvers had higher runtimes.

penalty	C	solver	max_iter	score	runtime (ms)
l2	1.0	lbfgs	100	0.9016	0.2315
l2	0.1	sag	200	0.8440	1.503
l2	0.5	liblinear	200	0.8428	0.5333
l2	0.5	lbfgs	100	0.8427	0.2086
l2	2.0	lbfgs	10	0.8429	0.1459
l2	1.0	sag	10	0.8617	0.3877
l2	2.0	lbfgs	200	0.8425	0.1810
l2	1.0	lbfgs	50	0.8426	0.2219

Table 5: F1 score and runtime of Logistic Regression for different hyperparameters.

- The choice of maximum number of iterations for the solver did not seem to have a significant impact on the model's performance. However, higher values of max_iter can lead to longer runtimes.

5 Summary of results

5.1 Splitting strategy

Figure 5 shows the mean F1-score results from 128 executions outlined in Table 1, with a distinction between (0.7-0.3)-holdout and 5-fold cross-validation as validation criteria. Although there is no significant difference in the metric, the two error measures may not lead to the same best models. As a more stable validation method, cross-validation is generally preferred. Thus, we will use cross-validation to evaluate model performance from now on.

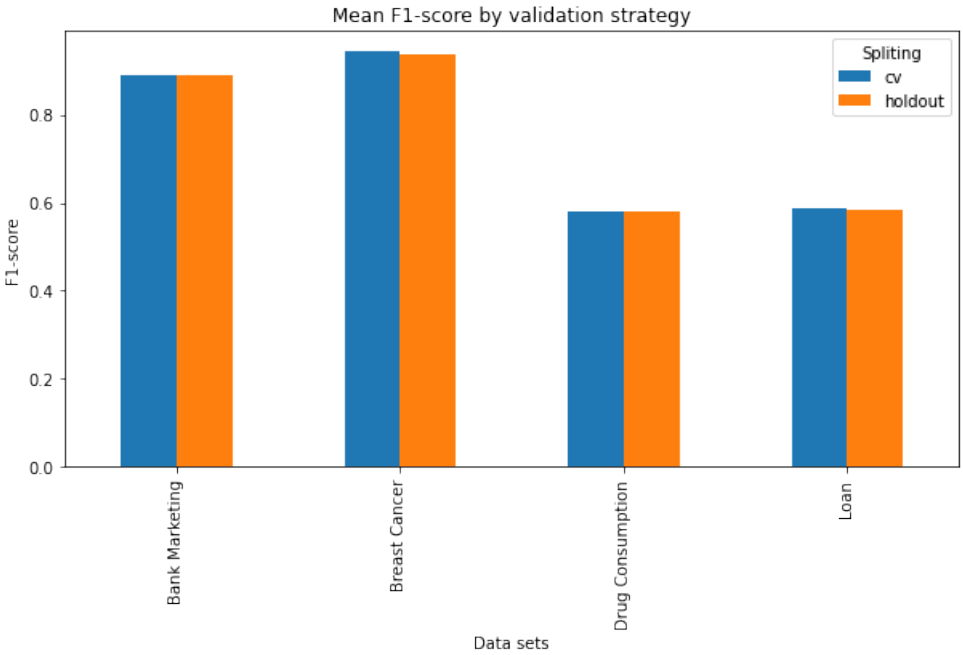

Figure 5: F1 score comparison between different validation strategies among classifiers.

5.2 Classifiers comparison

It is a good practice to examine the distribution of F1 scores across different folds in CV training to gain insight into the stability and reliability of the classifier's performance. Ideally, we want to avoid

a very large deviation in F1 scores as this can be a warning sign that the model may not generalize well to new, unseen data. We show the deviations of the F1-score in Figure 6.

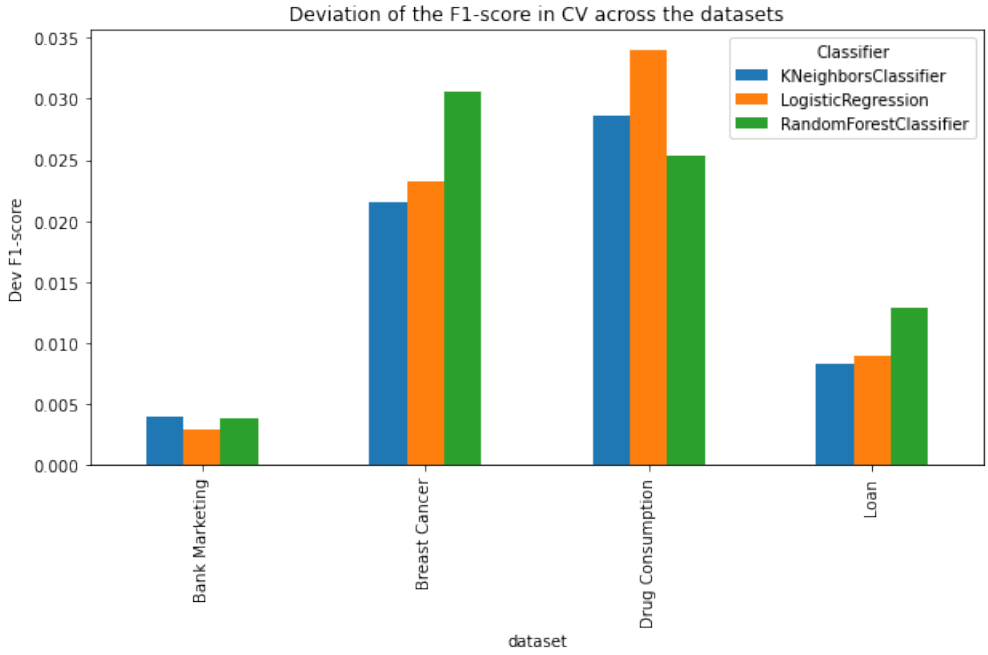

Figure 6: Deviation of the F1 score in k-fold CV.

One thing that can be noted (but which was expected) is the much higher stability of the model for larger data sets compared to the smaller ones, given the possibility of training on more instances for each fold. Analyzing the classifiers we do not notice any significant difference between the models in terms of the f1 score deviations.

5.3 Preprocessing strategies

Figure 7 showcases the impact of various preprocessing techniques, specifically feature selection and feature scaling, on the average F1 scores of the models for each dataset. The initial notable observation when examining the results is that all models perform exceptionally well on the two datasets that feature a binary classification problem (Bank Marketing and Breast Cancer), with accuracies approaching 0.9 and above. The multiclass classification problems perform worse. Both the models on the Bank Marketing and the Drug Consumption data set are not strongly influenced by the different preprocessing methods. The model trained on the Breast Cancer data set shows slight favor for either both feature scaling and selecting, or just scaling. However, the biggest effect of different strategies can be seen on the Loan data set. Here, scaling makes a notable difference.

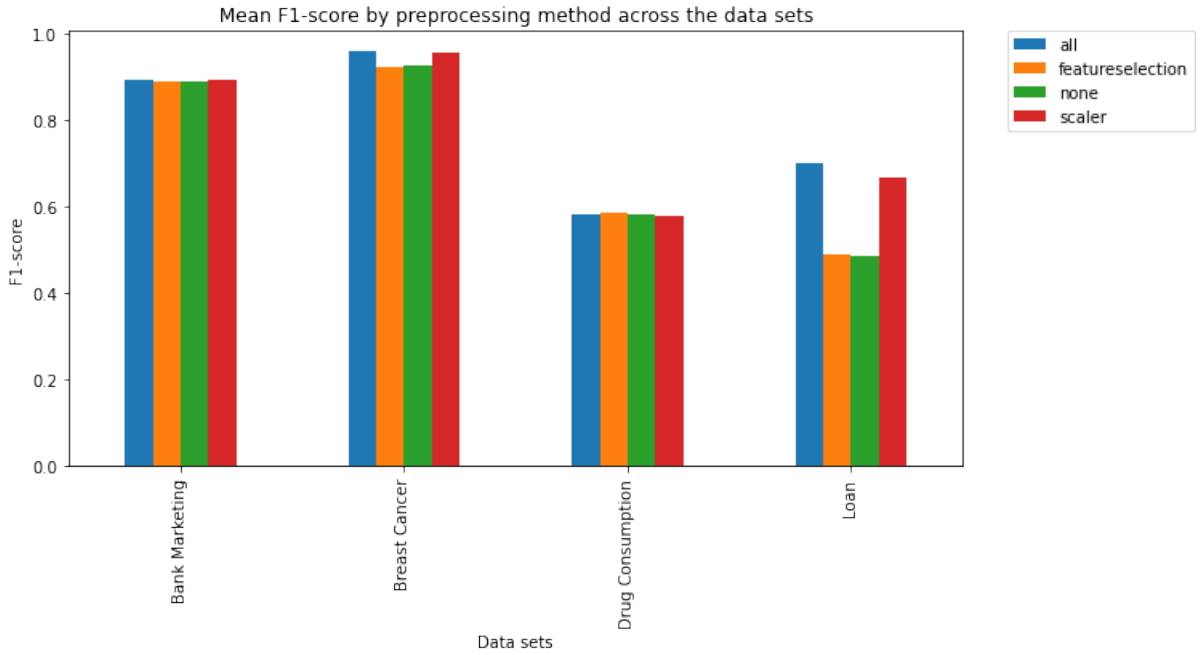

Figure 7: Graph showcasing the mean F1 score by using different preprocessing strategies across the data sets.

5.4 Training time comparison

Training time represents a fundamental characteristic in the field of ML, especially if we have a huge amount of data that must be processed in a distributed system. In our case it was obviously possible to do everything in a local environment on a machine without a GPU. Nonetheless, we want to analyze the training time of different classifiers on different data sets to get a feel for how long classifiers spend training.
As we could already guess, the number of values present in the dataset then went on to define the cost of the training. The datasets (Loan and Bank) containing 10k and around 45k rows, respectively, performed on average substantially slower than the smaller datasets (Drug Consumption and Breast Cancer) containing 1885 and 365 rows, respectively. The training times can be seen in Figure 8. As for the different types of classifiers, the type name "lazy learner" doesn't really fit the time. kNN may be lazy but it is fast. Please notice, that we had to log the y axis, otherwise we could not plot all three classifiers given the large difference in the training time.

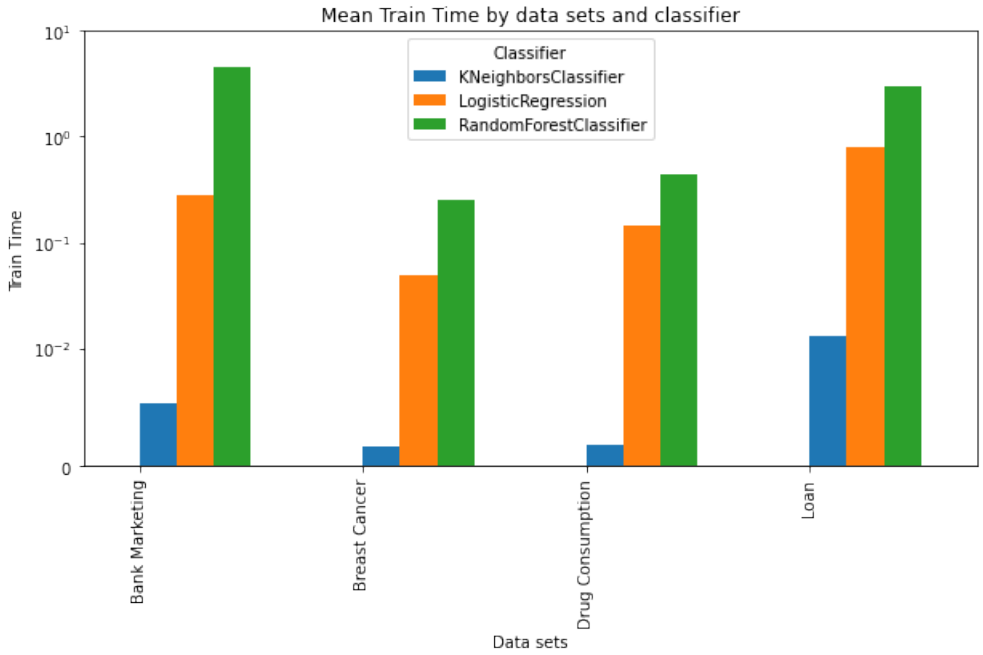

Figure 8: Mean training time across the classifiers

In the figure 9 we see the difference in the training time between the two validation strategies and across the three types of preprocessing i.e. (feature selection + scaling, only feature selection, only scaling) and not preprocessing at all. Generally we can see an improvement in the training time if some pre-processing steps have been implemented. Another fact that is also confirmed in practice is the computation difference between CV and the hold out strategy simply given by the number of repetitions that is performed in the CV to train the data on k-folds.

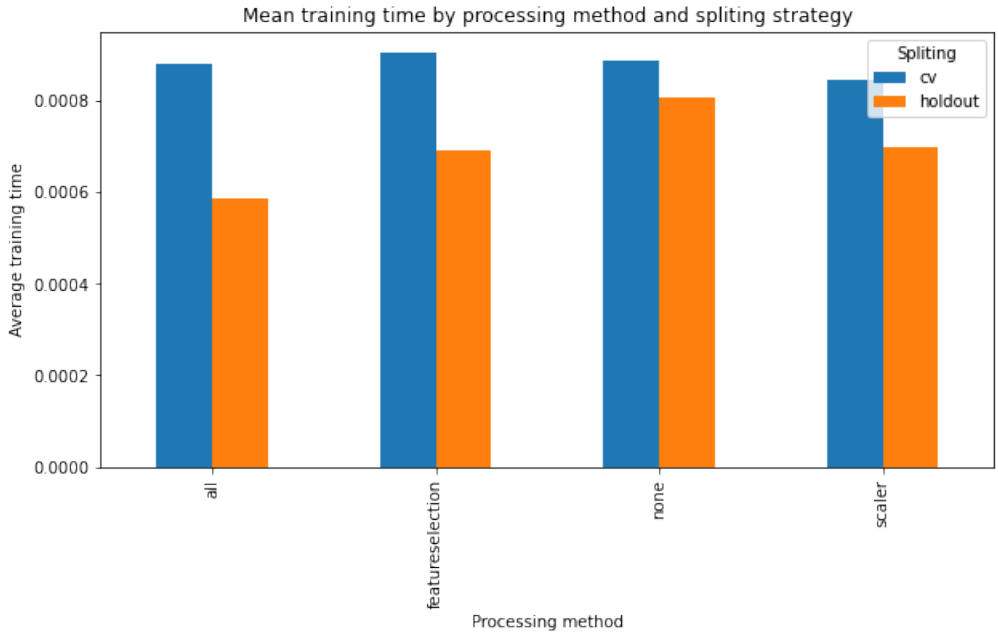

Figure 9: Training time compared for different steps of preprocessing

6 Summary

6.1 Findings

- A standardized pipeline across all datasets and models proves to be effective. Although it requires more complex coding, it simplifies result analysis and ensures comparability of the models.
- The presented case study demonstrated that utilizing either cross-validation or holdout validation can potentially yield distinct optimal hyperparameter configurations. This underscores the significance of defining validation strategies unambiguously and maintaining their consistency.
- Initially, some of us hypothesized that the holdout validation strategy would outperform cross-validation for smaller datasets. However, our results indicate that F1 scores were comparable, with cross-validation slightly outperforming the holdout strategy.
- Among the three classifiers, the Random Forest was found to be the most computationally expensive, with longer training times.
- Preprocessing techniques such as feature selection and scaling do not always demonstrate improvement (however, a decrease in performance was never demonstrated) in the overall performance of the models. However, in certain cases results did substantially improve.
- Our experiment demonstrated that the most effective preprocessing approach was a combination of feature selection and scaling, followed by feature scaling alone. No preprocessing was found to be not recommended.
- Our results indicate a direct correlation between the training time and the number of rows in the dataset for all three classifiers and all four data sets.

6.2 Further improvements

- Using a single evaluation metric (in this case, the F1 score) is not ideal across multiple data sets. Other metrics could have been explored, according to the data set (as explained previously).
- It is possible to experiment with different splits when using the holdout validation strategy, however, for consistency, we used the same split ratio of 70/30 throughout our experiment.
- The StandardScaler from sklearn was used for scaling the data in our experiment. Although it is a commonly used scaler, it is not as robust as some other options and may be sensitive to outliers.
- While there are more complex classifiers based on neural networks, we chose to focus on classical models for this study due to their suitability for the given datasets and the lack of theoretical knowledge about neural networks among most members of our team.