Optimizing Laser Welded Steel-copper Lap Joints

An application of statistical learning to reduce cost in manufacturing setting.

John Rey Faciolan

2023-09-16

Table of contents

Preface

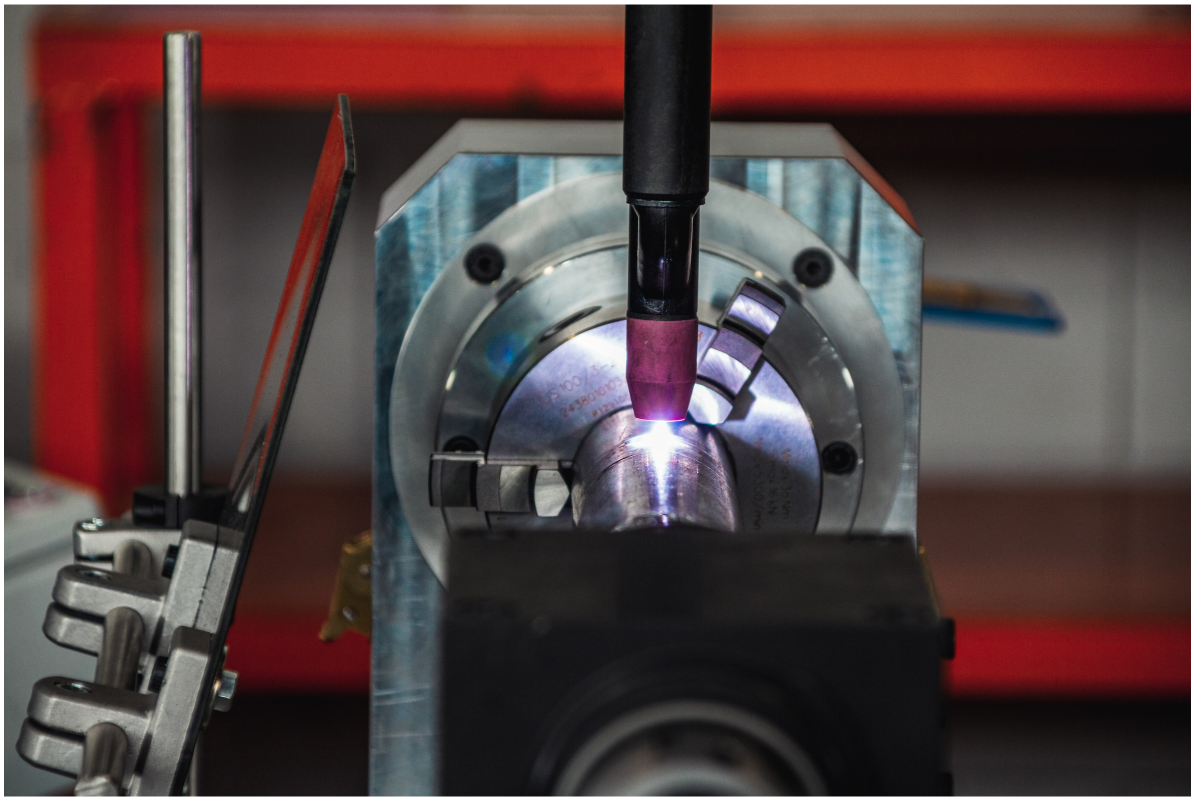


Figure 1: Photo by Sergio Rota on Unsplash

Welcome to this article on laser welding and welding cracks. In this comprehensive guide, we will delve into the fascinating world of laser welding, exploring its applications, advantages, and the challenges associated with welding cracks. Laser welding is a cutting-edge process that has revolutionized various industries, enabling precise and efficient joining of materials. However, welding cracks can pose significant risks to the quality and cost-effectiveness of the welding process. Therefore, our objective in this study is to identify the optimal parameters and factors that can minimize welding cracks, ultimately improving product quality and reducing associated costs. By following the CRISP-DM methodology and analyzing the "Screening datasets for laser welded steel-copper lap joints" dataset, we will uncover valuable insights into the relationship between welding process parameters and welding cracks. Join us on this journey as we explore the intricacies of laser welding and discover effective strategies to mitigate welding cracks.

1 Introduction

Laser welding is a process that utilizes a high-powered laser beam to join materials together. It is commonly used in various industries such as automotive, aerospace, and electronics. The purpose of laser welding is to create strong and precise welds with minimal heat input, resulting in reduced distortion and improved overall quality. Compared to traditional welding processes, laser welding offers several advantages including higher welding speeds, smaller heat-affected zones, and the ability to weld dissimilar materials.

Welding cracks are defects that can occur during the welding process. These cracks can significantly impact the quality and cost of the final product. They can compromise the structural integrity of the weld, leading to potential failures and safety hazards. Additionally, the presence of welding cracks often requires rework or scrap, increasing production costs and causing delays. Therefore, it is crucial to minimize welding cracks to ensure high-quality and cost-effective welding operations.

The aim of this project is to reduce the cost of rework and scrap, as well as minimize the risk of customer/market claims related to welding cracks. By identifying the optimal parameters and factors that can minimize welding cracks, businesses can improve their welding processes, enhance product quality, and reduce associated costs.

The objective of this study is to find the optimal parameters/factors that can minimize welding cracks in laser welding. It involves identifying the important features that affect welding cracks and understanding the effect of each parameter/factor of the laser welding process on the occurrence of welding cracks.

In this project, we will be utilizing the CRoss-Industry Standard Process for Data Mining (CRISP-DM) methodology. CRISP-DM is a widely recognized and proven approach for solving data mining problems. It consists of six phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment. By following this methodology, we can systematically analyze the data, develop models, and evaluate their effectiveness in addressing the problem statement.

The "Screening datasets for laser welded steel-copper lap joints" dataset will be used in this study. This dataset contains various parameters and factors related to laser welding, as well as information on the occurrence of welding cracks. By applying the CRISP-DM methodology to this dataset, we can gain insights into the relationship between the welding process parameters and welding cracks, and ultimately identify the optimal conditions for minimizing welding cracks.

2 Laser Welding Process

The laser welding process described in the dataset involves joining steel and copper lap joints using a high-powered laser beam. The steel-copper lap joints were prepared by overlapping a steel sheet and a copper sheet, creating a joint area where the laser welding would take place. The dataset includes six factors that were varied during the laser welding process.

These factors are:

1. **Laser beam power (W)**: This refers to the power of the laser beam used in the welding process. It determines the intensity of the laser beam and can affect the depth and quality of the weld.
2. **Welding speed (m/min)**: The welding speed refers to the rate at which the laser beam moves along the joint during the welding process. It affects the heat input and the cooling rate, which can influence the formation of welding cracks.
3. **Angular position in welding direction (°)**: This factor represents the angular position of the laser beam in relation to the welding direction. It can affect the distribution of heat and the formation of welding cracks.
4. **Focal position (mm)**: The focal position refers to the distance between the laser beam focus and the joint surface. It determines the spot size and the energy density of the laser beam, which can impact the weld quality and the occurrence of welding cracks.
5. **Gas flow rate (l/min)**: The gas flow rate represents the rate at which shielding gas is supplied during the welding process. Shielding gas is used to protect the weld pool from atmospheric contamination. The gas flow rate can influence the cooling rate and the formation of welding cracks.
6. **Material thickness of the steel sheet (mm)**: This factor represents the thickness of the steel sheet used in the lap joint. The thickness can affect the heat input and the cooling rate, which can influence the formation of welding cracks.

These factors were chosen based on their potential influence on the occurrence of welding cracks. To create the dataset, 18 parameter combinations were selected, representing different combinations of the six factors. Each parameter combination was repeated five times, resulting in a total of 90 welding experiments. Additionally, each sheet was cut four times to generate a total of 360 cross sections.

Each line in the dataset represents a cross section that was evaluated for the dimensions of the weld metal. The dataset provides information on the weld depth and the gesometrical dimensions of the weld metal in the laser welded steel-copper lap joints.

The Screening datasets for laser welded steel-copper lap joints(see Rinne (2021)) has two versions: **V1** and **V2**. In **V1**, the dataset includes information on the occurrence of welding cracks, as well as the dimensions of the weld metal. In **V2**, additional information on the occurrence of partial penetration and the dimensions of the heat-affected zone is included. The **V1.1** and **V2.1** versions of the dataset were created to address some inconsistencies and errors found in the original versions (**V1** and **V2**). Overall, the dataset provides valuable information on the relationship between the welding process parameters and the occurrence of welding cracks in laser welded steel-copper lap joints. By analyzing this dataset, researchers can gain insights into the optimal conditions for minimizing welding cracks and improving the quality of laser welding operations.

3 Project Overview

3.1 Objective

This project is focused on the creation of a machine learning model capable of predicting crack formation in laser-welded steel-copper lap joints. By analyzing various process parameters, the model aims to understand and highlight the correlation between these parameters and the incidence of defects. The ultimate goal is to enhance the welding process, improve joint quality and reliability, and reduce costs associated with rework and scrap in industries utilizing laser welding for steel-copper lap joints.

3.2 Solution Usage

The proposed solution, a predictive machine learning model, will serve as an integral tool in the laser welding industry. The model, once trained and validated, can be incorporated into the manufacturing workflow.

Prior to welding, the model will ingest process parameters such as laser power, welding speed, and focus position. It will then predict the likelihood of crack formation in the welded joint under these conditions.

In case of a high-risk prediction, parameters can be iteratively adjusted and re-evaluated by the model until an acceptable risk level is reached. This facilitates real-time optimization of the welding process, thereby reducing defect occurrence and enhancing product quality.

Furthermore, the model can aid strategic planning and decision-making by offering insights into how different parameters influence weld quality. This can steer research and development towards more efficient welding techniques and technologies.

In essence, this machine learning model is poised to be a valuable asset for operational optimization and strategic planning in industries employing laser welding of steel-copper lap joints.

3.3 Current Solution

The prevention of cracks in laser-welded steel-copper lap joints currently involves understanding the mechanism of solidification cracking and implementing measures to control it (Rinne et al. (2021), Gao et al. (2022)).

Solidification cracking in welds is a synergistic effect of phase liquation, inclusions, and composition segregation (Rinne et al. (2021), Gao et al. (2022)). The welding process can cause grain boundary liquation, reducing cohesion between grains and resistance to intergranular crack propagation (Rinne et al. (2021), Gao et al. (2022)). Composition segregation within grains can induce lattice distortion, reducing the material's plastic deformation capacity and increasing crack susceptibility (Rinne et al. (2021), Gao et al. (2022)).

An oscillating laser has been proposed as an effective solution to inhibit solidification cracking (Rinne et al. (2021), Gao et al. (2022)). Laser oscillating welding promotes grain refinement, solute diffusion, and the formation of uniformly distributed -Cu precipitated phases in welds (Rinne et al. (2021), Gao et al. (2022)). This improves intergranular bonding, reduces solidification cracking susceptibility, and increases resistance to plastic deformation (Rinne et al. (2021), Gao et al. (2022)). The tensile strength of joints using laser oscillating welding is 251 MPa, a 35.7% increase compared to 185 MPa using standard laser welding (Rinne et al. (2021), Gao et al. (2022)). The strain of joints using laser oscillating welding is 3.69, a 96% increase compared to 1.88 using standard laser welding (Rinne et al. (2021), Gao et al. (2022)).

While these solutions enhance joint quality and reliability, they require a deep understanding of the welding process and careful parameter control. This is where our machine learning model can add significant value by accurately predicting crack formation based on process parameters and guiding optimization efforts.

3.4 Problem Framing

The problem is structured as a supervised learning task. We have a labeled dataset from Mendeley Data, with known occurrences of cracks in laser-welded steel-copper lap joints (the target variable) for various process parameters (the features). The objective is to train a model on this data to predict the target variable for new, unseen data.

The model will undergo offline training, meaning we will use a static dataset for training, and the model will not learn continuously from new data. Once trained and validated, the model can be deployed to predict crack occurrences based on specified input parameters.

However, it's crucial to acknowledge that while the initial model training is offline, periodic retraining may be required to maintain high predictive performance as welding processes and techniques evolve. This retraining would also be conducted offline, using a new static dataset encompassing the most recent welding data.

3.5 Performance Metric

The model's performance will be evaluated using the F1 Score, a metric that balances precision and recall. Precision quantifies the proportion of positive identifications that were correct, while recall quantifies the proportion of actual positives correctly identified.

The F1 Score is especially useful in scenarios where both false positives and false negatives are present. In this project's context, a false positive implies a prediction of crack formation where none occurs, and a false negative implies a prediction of no crack formation where one does occur. Both scenarios could have significant implications in the laser welding industry, making it crucial to minimize both, which is what the F1 Score aims to achieve.

By employing the F1 Score as our performance metric, we strive to develop a model that not only accurately predicts crack formation but also minimizes both false positives and false negatives.

3.6 Performance Metric and Business Objectives

The F1 Score aligns with the business objective of this project, which is to develop a machine learning model capable of accurately predicting crack occurrence in laser-welded steel-copper lap joints based on process parameters. Understanding the relationship between these parameters and crack formation will optimize the welding process, minimize defects, enhance joint quality and reliability, reduce rework and scrap costs, and ultimately improve the bottom line for industries relying on laser welding of steel-copper lap joints.

The F1 Score is a measure of model accuracy that balances precision (the proportion of positive identifications that were correct) and recall (the proportion of actual positives correctly identified). This balance ensures the model accurately predicts crack formation while minimizing both false positives (predicting a crack will form when it does not) and false negatives (predicting a crack will not form when it does). Both scenarios could have significant implications in the laser welding industry, making it crucial to minimize both.

By employing the F1 Score as our performance metric, we aim to create a model that not only accurately predicts crack formation but also minimizes both false positives and false negatives. This aligns well with our business objective.

3.7 Minimum Performance

The minimum performance required to meet the business objectives would be contingent on the specific needs and constraints of the industries utilizing this model. Generally, a high F1 Score would be desirable, indicating accurate crack prediction while minimizing both false positives and false negatives.

For example, an F1 Score of at least 0.85 could serve as a reasonable benchmark. This implies that the model correctly identifies the presence or absence of cracks 85% of the time, maintaining a balance between precision and recall.

However, this is merely a guideline, and actual minimum performance may vary. Factors such as the cost implications of false positives and negatives, the overall impact on production efficiency, and industry standards for quality and reliability would all influence the acceptable performance level.

The ultimate objective is to create a model that aids in optimizing the welding process and minimizing defects, thereby enhancing product quality and reducing costs. Therefore, the model's performance should be sufficiently high to achieve these goals.

3.8 Similar Problems

There are several comparable problems in the field of manufacturing and materials science where machine learning models have been used to predict outcomes based on various parameters. Here are a few examples:

1. **Predicting Welding Distortion**: Just like predicting cracks in laser-welded steel-copper lap joints, predicting welding distortion is another problem in the welding industry (Rinne et al. (2021), Gao et al. (2022)). Machine learning models can be trained on various welding parameters and the resulting distortion to predict future distortions based on these parameters (Rinne et al. (2021), Gao et al. (2022)).
2. **Predicting Material Properties**: Machine learning models have been used to predict the properties of materials based on their composition and manufacturing processes (Rinne et al. (2021), Gao et al. (2022)). This is similar to predicting cracks in laser-welded steel-copper lap joints, where the outcome (crack formation) is predicted based on various process parameters (Rinne et al. (2021), Gao et al. (2022)).
3. **Quality Control in Manufacturing**: Machine learning models are often used in quality control applications in various manufacturing industries (Rinne et al. (2021), Gao et al. (2022)). These models can predict the quality of a product based on various factors such as machine settings, environmental conditions, and material properties (Rinne et al. (2021), Gao et al. (2022)).

In all these cases, the experience and tools used for developing and training the machine learning models can be reused. The process of data preprocessing, feature selection, model training, validation, and testing are common across these problems. Tools and libraries such as Python's scikit-learn or TensorFlow can be used to develop the machine learning models. Furthermore, techniques for handling imbalanced data, tuning model parameters, and evaluating model performance can also be applied to this problem.

3.9 Human Expertise

human expertise is available in the field of laser welding and machine learning. Experts in laser welding can provide valuable insights into the welding process, the formation of cracks, and the various parameters that affect weld quality. They can also help in collecting and labeling data for training the machine learning model.

Machine learning experts, on the other hand, can help in developing and training the model. They can provide guidance on data preprocessing, feature selection, model selection, and hyperparameter tuning. They can also help in evaluating the performance of the model and suggesting improvements.

Collaboration between these two groups of experts can be highly beneficial for this project. Laser welding experts can provide domain knowledge to guide the development of the machine learning model, while machine learning experts can apply their technical expertise to build a robust and accurate model.

3.10 Methodology

To solve this problem, the following steps would be conducted:

- **Exploratory Data Analysis**: This step involves visualizing and summarizing the data to understand its characteristics, distribution, and patterns. It also helps to identify any outliers, missing values, or anomalies in the data.
- **Data Cleaning & Feature Engineering**: This step involves preparing the data for modeling by removing or imputing any missing values, outliers, or noise. It also involves creating new features or transforming existing ones to capture more information from the data and improve the model performance.
- **Model Selection**: This step involves choosing the most suitable machine learning algorithm for the problem, based on the data type, size, complexity, and desired outcome. Some common algorithms for welding crack detection are convolutional neural networks, support vector machines, and random forests.

- **Hyper-parameter Tuning**: This step involves finding the optimal values for the parameters that control the behavior and performance of the chosen algorithm. This can be done using techniques such as grid search, random search, or Bayesian optimization.
- **Ensembling**: This step involves combining multiple models to create a more robust and accurate model. This can be done using techniques such as bagging, boosting, or stacking.
- **Presentation**: This step involves presenting the results and insights from the model to the stakeholders, using appropriate visualizations, metrics, and explanations. It also involves discussing the limitations, challenges, and future directions of the model.

4 Retrieving the Dataset

The "Screening datasets for laser welded steel-copper lap joints" dataset is a valuable resource for researchers and engineers working in the field of laser welding. This dataset provides comprehensive information on the performance of steel-copper lap joints under different welding conditions (Rinne 2021). In this article, we will explore the different versions of the dataset, where to download them, their file types, and how to load them using the pandas library.

At the time of this writing, this dataset have 4 versions:

- **V1**: The initial version of the dataset.
- **V1.1**: An updated version of V1 with additional data.
- **V2**: A major update to the dataset with expanded features.
- **V2.1**: A minor update to V2, addressing some issues and providing further enhancements.

4.1 Downloading the Dataset

To download the this dataset, the following python function was defined and used.

```python
import os
import urllib.request
import zipfile

def download_file(url, filename):
    # Check if file already exists in the directory
    if os.path.exists(f"datasets/{filename}"):
        # Print the directory tree relative to the datasets folder
        for root, dirs, files in os.walk("datasets"):
            level = root.replace("datasets", "").count(os.sep)
            indent = " " * 4 * (level)
            print(f"{indent}{os.path.basename(root)}/")
            subindent = " " * 4 * (level + 1)
            for f in files:
                print(f"{subindent}{f}")
    else:
        # Download the file from the internet
        urllib.request.urlretrieve(url, f"datasets/{filename}")
        print(f"File '{filename}' downloaded successfully.")

        # Extract the zip file
        with zipfile.ZipFile(f"datasets/{filename}", 'r') as zip_ref:
            zip_ref.extractall("datasets")

        # Print the directory tree relative to the datasets folder
        for root, dirs, files in os.walk("datasets"):
            level = root.replace("datasets", "").count(os.sep)
            indent = " " * 4 * (level)
            print(f"{indent}{os.path.basename(root)}/")
            subindent = " " * 4 * (level + 1)
            for f in files:
                print(f"{subindent}{f}")

url = "https://prod-dcd-datasets-cache-zipfiles.s3.eu-west-1.amazonaws.com/2s5m3crbkd-2.zi
filename = "laser-welding.zip"
download_file(url, filename)
```

```
datasets/
    laser-welding.zip
    V1.csv
    Screening datasets for laser welded steel-copper lap joints/
        V1 and V2/
            Definitive screening steel-copper lap joints V1.xlsx
            Definitive screening steel-copper lap joints V2.xlsx
        V1.1 and V2.1/
            Definitiv screening steel-copper lap joints V1.1.xlsx
            Definitive screening steel-copper lap joints V2.1.xlsx
```

Explanation:

1. The code imports the necessary modules: `os` for file and directory operations, `urllib.request` for downloading files from the internet, and `zipfile` for extracting zip files.
2. The function `download_and_extract_file` takes two parameters: `url` (the URL of the file to be downloaded) and `filename` (the name of the file to be saved in the local directory).

3. It checks if the file already exists in the `datasets` directory using the `os.path.exists` function. If it exists, it prints the directory tree relative to the `datasets` folder using the `os.walk` function.
4. If the file does not exist, it downloads the file from the internet using `urllib.request.urlretrieve` and saves it in the `datasets` directory.
5. It then extracts the zip file using the `zipfile.ZipFile` context manager and the `extractall` method, saving the extracted files in the `datasets` directory.
6. Finally, it prints the directory tree relative to the `datasets` folder using the `os.walk` function.

This code allows for downloading and extracting files from the internet, while also checking if the file already exists in the local directory. It provides a convenient way to manage datasets and avoid unnecessary downloads.

5 Data Wrangling

In order to develop a machine learning model that can effectively highlight the correlation between input parameters and the incidence of defects, we will be using version 1.0 of the dataset. This version contains a set of machine parameter combinations and a binary target variable.

Although version 1.1 of the dataset includes additional information about weld width, weld gap, crack count, and crack length, we have decided not to include these features in our models. This is because they contain information about the occurrence of cracks, which could potentially lead to biased results. These additional features can be considered as leaky features and may negatively impact the performance of our models.

Before we can proceed with data wrangling, it is important to have a deeper understanding of the purpose of each column in the dataset. This will help us identify which columns are features, targets, and unwanted variables, which will be useful during the modeling phase.

```python
import pandas as pd

dataset_uri = "datasets/Screening datasets for laser welded steel-copper lap joints/V1 and

dataset = pd.read_excel(dataset_uri)

dataset.head().T
```

	0	1	2	3	4
power (W)	1050	1050	1050	1050	1050
welding speed (m/min)	1.0	1.0	1.0	1.0	1.0
gas flow rate (l/min)	15	15	15	15	15
focal position (mm)	0	0	0	0	0
angular position (°)	0	0	0	0	0
material thickness (mm)	0.6	0.6	0.6	0.6	0.6
weld number	1	1	1	1	2
cross section positon in the weld (mm)	8	16	24	32	8
cracking in the weld metal	no	no	no	no	no
weld seam width steel (µm)	2097.0	2069.0	2043.0	2010.0	2232.0
weld seam width copper (µm)	286	384	385	414	320

	0	1	2	3	4
weld depth copper (µm)	110	149	177	164	149
gap	58.0	85.0	91.0	58.0	102.0

Importing and showing the 1st 5-rows of the V1.0 of Screening datasets for laser welded steel-copper lap joints

5.1 Dropping Unwanted Variables

Before wrangling the data, its important to have a deeper understanding of the purpose of each columns. Thus, having known which of these columns were features, targets, and unwanted, would be a great help later during the modelling phase.

5.1.1 Identifying the Feature Variables

As highlighted in the Laser Welding Process page, the dataset contains 6 features representing the 6 different factors to be studied. These factors include laser beam power, welding speed, angular position in welding direction, focal position, gas flow rate, and material thickness of the steel sheet. Additionally, there are 2 features for identifying the weld number and cross section position.

5.1.1.1 Factors to be Studied

1. Laser beam power (W)
2. Welding speed (m/min)
3. Angular position in welding direction (°)
4. Focal position (mm)
5. Gas flow rate (l/min)
6. Material thickness of the steel sheet (mm)

5.1.1.2 Experiment Identification

1. weld number
2. cross section positon in the weld (mm)

5.1.2 Identifying the Target Variables

The dataset contains 4 continuous target variables and 1 binary target variable. For this study, we will be focusing on the binary target variable, which indicates the presence or absence of cracking in the weld metal. The remaining 4 continuous target variables will be dropped from our analysis.

5.1.2.1 Binary Target Variable

1. cracking in the weld metal

5.1.2.2 Continuous Variables

1. weld seam width steel (µm)
2. weld seam width copper (µm)
3. weld depth copper (µm)
4. gap

The python code below categorizes each of the columns of the dataset, and drops the unnecessary columns.

```python
FEATURES = [
    'power (W)',
    'welding speed (m/min)',
    'gas flow rate (l/min)',
    'focal position (mm)',
    'angular position (°)',
    'material thickness (mm)',
]

EXPERIMENT_ID = [
    'weld number',
    'cross section positon in the weld (mm)',
]

TARGET = [
    'cracking in the weld metal',
]

UNNECESSARY_COLS = [
    'weld seam width steel (µm)',
    'weld seam width copper (µm)',
    'weld depth copper (µm)',
    'gap'
]

# Drop unnecessary columns
dataset.drop(UNNECESSARY_COLS,inplace=True,axis=1)

# Check the new dataset
dataset.head().T
```

	0	1	2	3	4
power (W)	1050	1050	1050	1050	1050
welding speed (m/min)	1.0	1.0	1.0	1.0	1.0
gas flow rate (l/min)	15	15	15	15	15
focal position (mm)	0	0	0	0	0
angular position (°)	0	0	0	0	0
material thickness (mm)	0.6	0.6	0.6	0.6	0.6
weld number	1	1	1	1	2
cross section positon in the weld (mm)	8	16	24	32	8
cracking in the weld metal	no	no	no	no	no

5.2 Encoding Categorical Variables

Before feeding the dataset into machine learning models, we need to encode the categorical variable "cracking in the weld metal" into a numeric format. This can be done by mapping "yes" to 1 and "no" to 0.

```
dataset["cracking in the weld metal"] = dataset["cracking in the weld metal"].map(
    {
        "no" : 0,
        "yes": 1
    },
)

dataset.head().T
```

	0	1	2	3	4
power (W)	1050.0	1050.0	1050.0	1050.0	1050.0
welding speed (m/min)	1.0	1.0	1.0	1.0	1.0
gas flow rate (l/min)	15.0	15.0	15.0	15.0	15.0
focal position (mm)	0.0	0.0	0.0	0.0	0.0
angular position (°)	0.0	0.0	0.0	0.0	0.0
material thickness (mm)	0.6	0.6	0.6	0.6	0.6
weld number	1.0	1.0	1.0	1.0	2.0
cross section positon in the weld (mm)	8.0	16.0	24.0	32.0	8.0
cracking in the weld metal	0.0	0.0	0.0	0.0	0.0

5.3 Quantifying Missing Data

To ensure that our machine learning models can be trained effectively, we need to address any missing data. Fortunately, this dataset does not contain any missing data.

```
dataset.isna().sum()
```

```
power (W)                                 0
welding speed (m/min)                     0
gas flow rate (l/min)                     0
focal position (mm)                       0
angular position (°)                      0
material thickness (mm)                   0
weld number                               0
cross section positon in the weld (mm)    0
cracking in the weld metal                0
dtype: int64
```

5.4 Data Visualization

In order to gain a deeper understanding of the distribution and relationship of each variable, we will explore the dataset visually.

5.4.1 Data Distribution

Visualizing the distribution of our data will help us understand its central tendency and spread. This can be achieved using histograms.

```
import seaborn as sns
import matplotlib.pyplot as plt

for i in range(len(FEATURES)):
    plt.subplot(3,2,i+1)
    sns.histplot(dataset[FEATURES[i]])

plt.tight_layout()
```

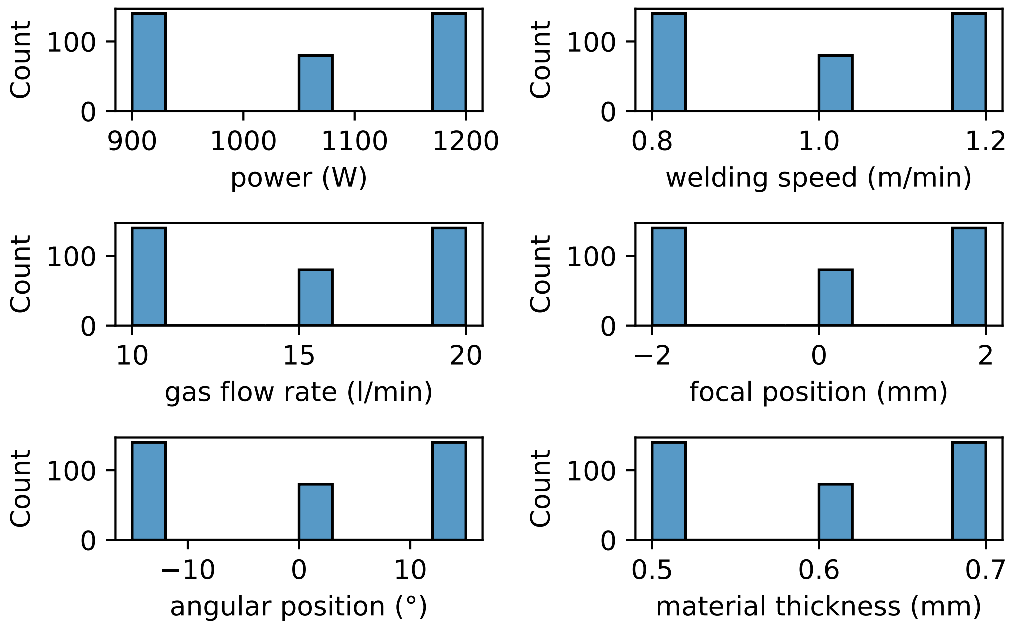


Figure 5.1: Histogram of the 6-parameters to be studied. Based on above plot, only three levels per parameters were gathered.

```
sns.histplot(dataset[TARGET], discrete=True, stat="density");
```

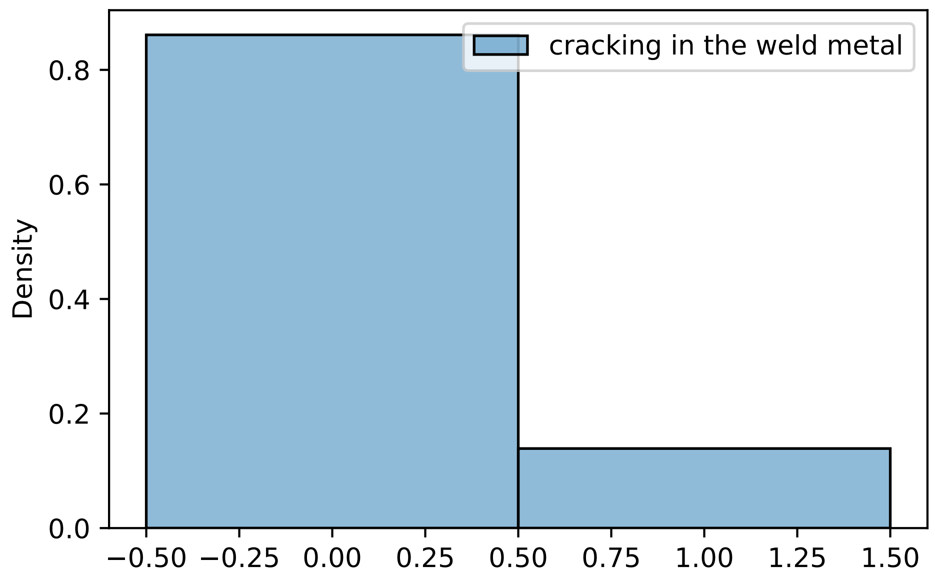


Figure 5.2: Histogram of the target variable. Based on above plot, the target variable was highlighly imbalanced – less than 20% of the dataset has a value of `1`.

5.4.2 Correlation Heatmap

Some machine learning models assume that input features are not correlated. To check for correlations, we will generate a heatmap of the correlation matrix of our input features. This heatmap will also help us identify which features are correlated with the target variable.

```
corr_matrix = dataset[FEATURES].corr()
sns.heatmap(corr_matrix, annot=True, fmt=".2f");
```

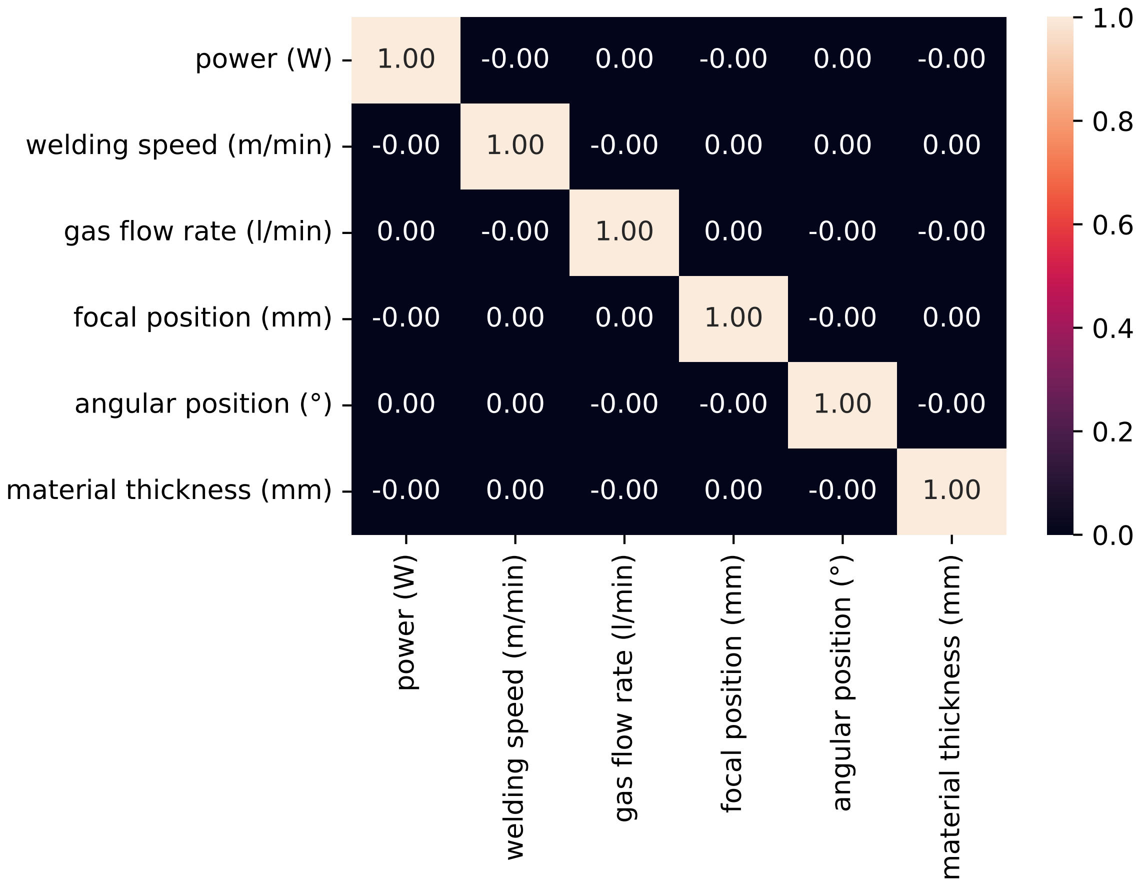


Figure 5.3: Heatmap of the correlation matrix of the feature variables.

Based on the correlation heatmap, we can observe the following:

1. There is no multicollinearity among the input features.
2. The features “power,” “angular position,” and “material thickness” are positively correlated with the presence of cracking in the metal weld.
3. The feature “gas flow rate” is negatively correlated with the presence of cracking in the metal weld.

5.5 Exporting the Dataset

```
dataset.to_csv("datasets/V1.csv")
```

5.6 Conclusion

In conclusion, we have explored the dataset and identified the relevant features and target variables for our machine learning model. We have also encoded the binary target variable and visualized the data distribution and correlations. The next steps will involve training and evaluating our machine learning models using this prepared dataset.

6 Modelling

In the previous section on Data Wrangling, it was discovered that the 6 machine parameters to be studied had discrete values with three levels each, and the target variable was a highly imbalanced binary variable.

In this section, we will feed the dataset into different machine learning models to find the optimal model that can help us understand the relationship between the input and output variables. To do this, we will split the dataset into a training set and a testing set, with a test size of 20%. The training set will be used for model training and cross-validation, while the testing set will be used to evaluate the model's performance on unseen data.

Once the data is split, we will load it into different out-of-the-box machine learning models from the scikit-learn library. At this stage, our goal is to find the optimal model using their default hyperparameters.

Finally, we will compare and evaluate the training and testing performance of each model. The objective is to shortlist the top-performing model.

6.1 Importing Relevant Libraries

The following code contains all the necessary import statements for this section.

```
# Data Wrangling
import numpy as np
import pandas as pd

# Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt

# Dataset Splitting
from sklearn.model_selection import train_test_split, cross_validate

# Feature Engineering Classes
from sklearn.compose import ColumnTransformer
```

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Machine Learning Model Classes
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

# Classification Metrics
from sklearn.metrics import classification_report

# Classification Visualization
from yellowbrick.classifier import ConfusionMatrix, ClassificationReport, ROCAUC

# Base Classes
from sklearn.base import ClassifierMixin
```

6.2 Importing the Dataset

The code below loads the pre-processed dataset from the EDA section into a pandas DataFrame.

```python
dataset_uri = "datasets/V1.csv"
dataset = pd.read_csv(dataset_uri, index_col=0)

dataset.head()
```

	power (W)	welding speed (m/min)	gas flow rate (l/min)	focal position (mm)	angular position (°)
0	1050	1.0	15	0	0
1	1050	1.0	15	0	0
2	1050	1.0	15	0	0
3	1050	1.0	15	0	0
4	1050	1.0	15	0	0

6.3 Vertical Data Splitting

Once the dataset is loaded, we will split it into a feature matrix `X` and a target vector `y`. `X_train` and `y_train` will be soon splitted into `X_train` & `X_cv`, and `y_train` & `y_cv` during model selection.

```python
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    stratify=y,
    random_state=22
)
```

6.4 Horizontal Data Splitting

After defining the feature matrix and target vector, we will split them horizontally into a training set and a testing set. The code below performs the horizontal split with a test size of 20%, ensuring that both sets contain the same proportion of `0`s and `1`s in their target variable.

6.5 Training the Model

The code below will train 5 machine learning models using the preprocessed dataset from the Data Wrangling section.

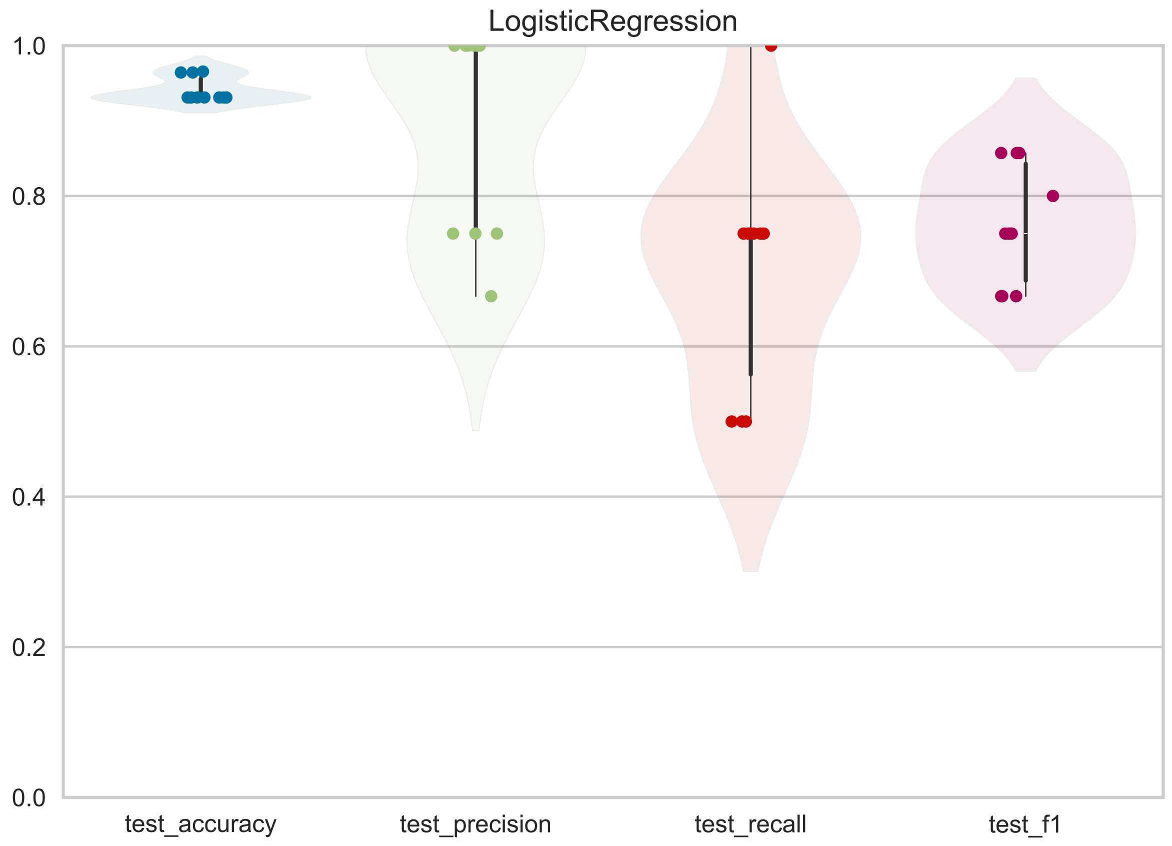
```python
class ClassifierEvaluator():
    def __init__(self, estimators: list[ClassifierMixin]) -> None:
        self.estimators = estimators

    def cross_validate_metrics(self, X, y) -> None:
        scorings = ['accuracy', 'precision', 'recall', 'f1']

        for estimator in self.estimators:
            name = type(estimator).__name__
            pipeline = Pipeline(
                steps=[
                    ("std_scaler", StandardScaler()),
                    ("classifier", estimator)
                ]
            )

            metrics = pd.DataFrame(cross_validate(pipeline, X, y, cv=10, scoring=scorings)
            metrics_viz = sns.stripplot(metrics.iloc[:, 2:]).set(title=name)
            metrics_viz = sns.violinplot(metrics.iloc[:, 2:], alpha=0.1).set(title=name)
            plt.ylim([0,1.0])
            plt.show()

    def tabulate_cross_validation_metrics(self, X, y) -> None:
        scorings = ['accuracy', 'precision', 'recall', 'f1']

        df = pd.DataFrame()

        for estimator in self.estimators:
            name = type(estimator).__name__

            pipeline = Pipeline(
                steps=[
                    ("std_scaler", StandardScaler()),
                    ("classifier", estimator)
                ]
            )

            metrics = pd.DataFrame(cross_validate(pipeline, X, y, cv=10, scoring=scorings)
            metrics["Model"] = name
            df = pd.concat([df, metrics])

        return df
```

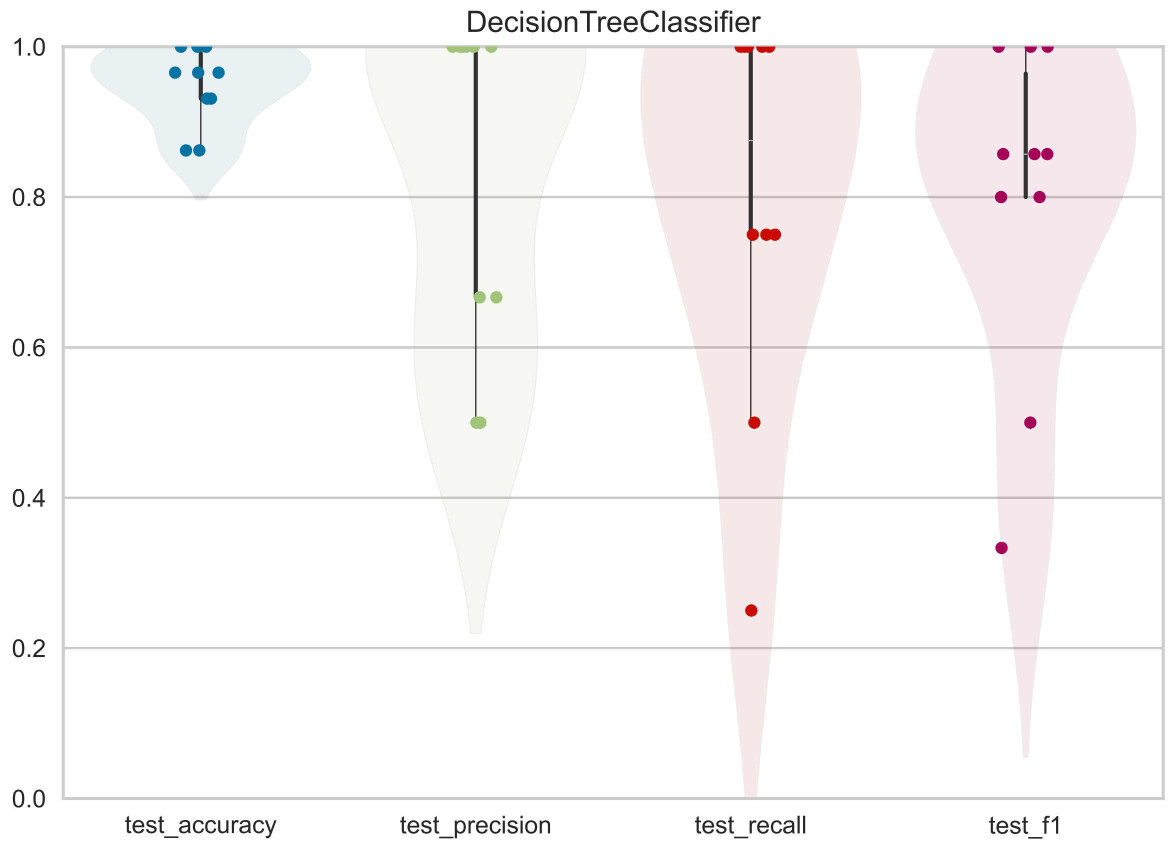
```
evaluator = ClassifierEvaluator(
    estimators=[
        LogisticRegression(),
        DecisionTreeClassifier(),
        SVC(),
        GaussianNB(),
        KNeighborsClassifier()
    ],
)
```

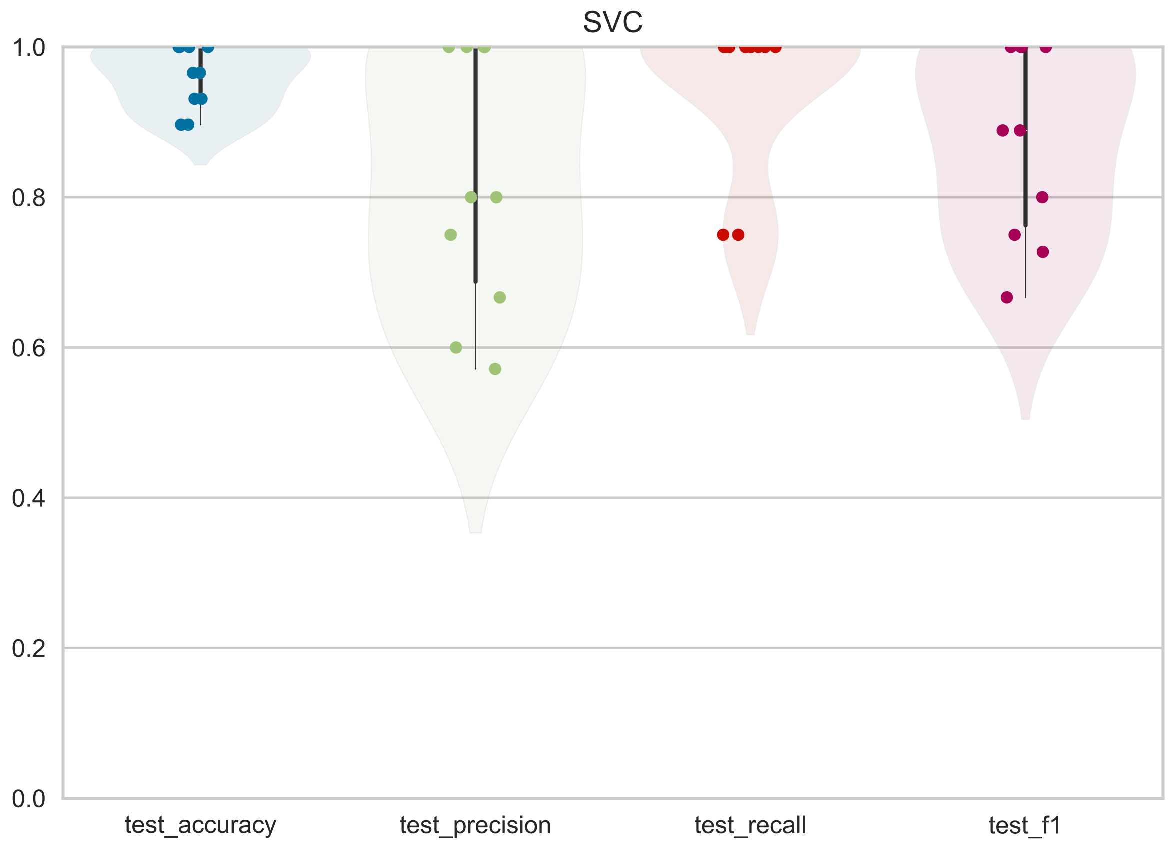
6.6 Baseline Cross-validation Validation Metrics

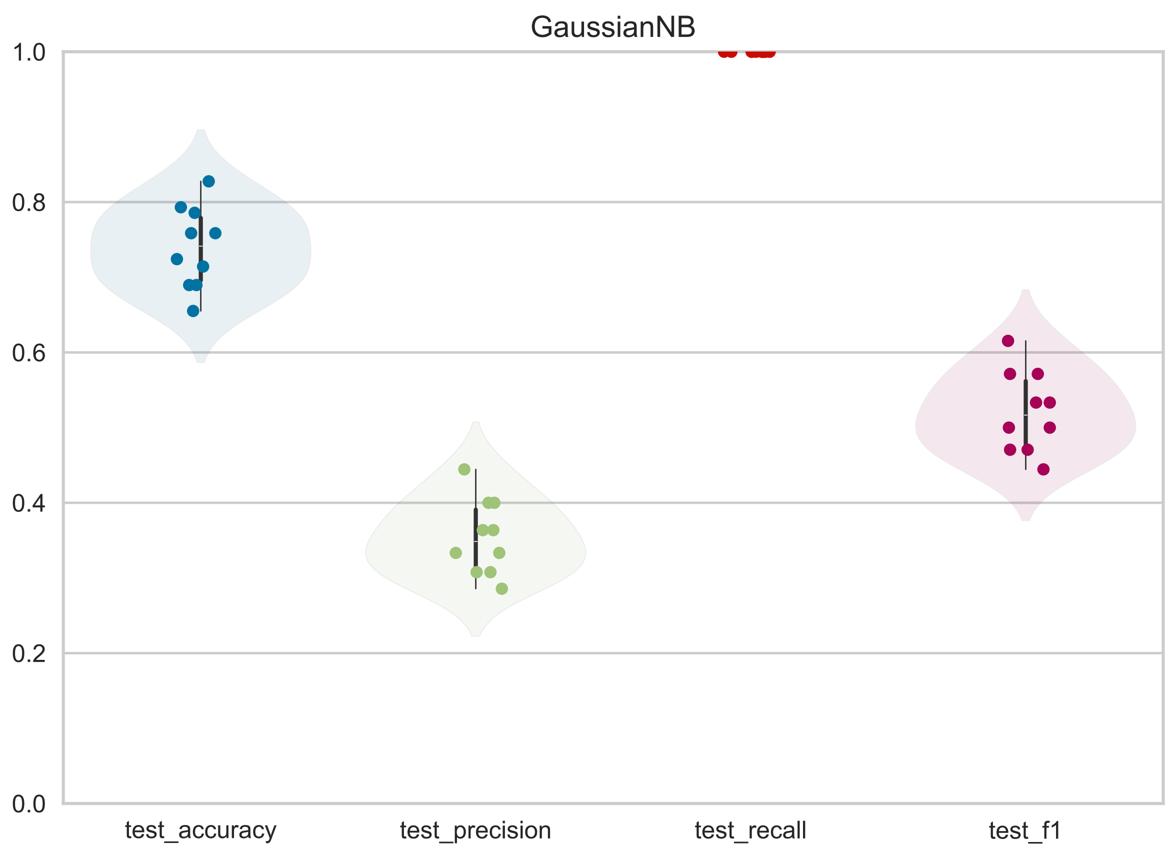
The following series of plots below shows the strip-plot and violin plot of each of the model's accuracy, f1, recall, and precision rates measured during 10-fold cross validation on the training set.

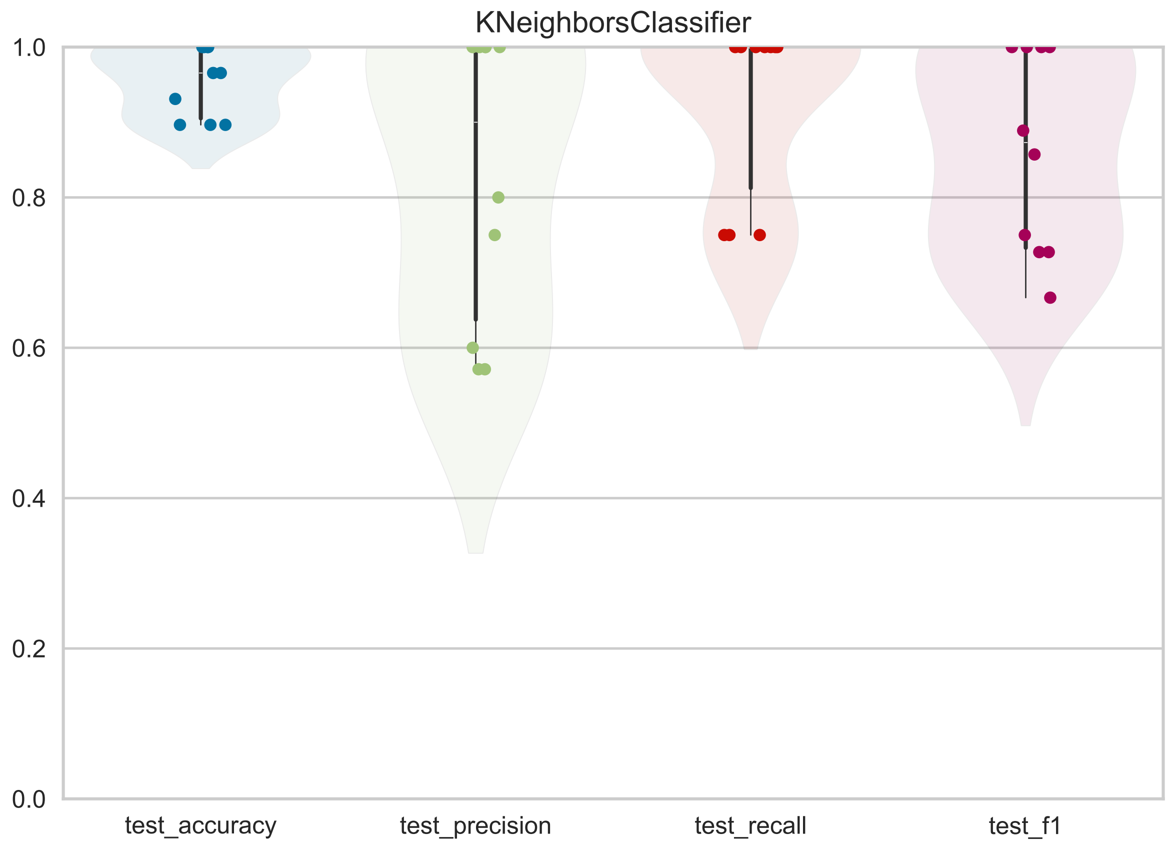
```
evaluator.cross_validate_metrics(X_train, y_train)
```




DecisionTreeClassifier
1.0
0.8
0.6
0.4
0.2
0.0
test_accuracy
test_precision
test_recall
test_f1


SVC
1.0
0.8
0.6
0.4
0.2
0.0
test_accuracy
test_precision
test_recall
test_f1


GaussianNB
1.0
0.8
0.6
0.4
0.2
0.0
test_accuracy
test_precision
test_recall
test_f1



```
UNBAL_results = evaluator.tabulate_cross_validation_metrics(X_train, y_train)
```

6.6.1 Logistic Regression

```
UNBAL_results[UNBAL_results["Model"] == "LogisticRegression"].describe()
```

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f1
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	0.006024	0.009311	0.941133	0.891667	0.700000	0.762143
std	0.000956	0.001059	0.016264	0.141912	0.158114	0.079090
min	0.003999	0.008002	0.931034	0.666667	0.500000	0.666667
25%	0.005975	0.008272	0.931034	0.750000	0.562500	0.687500
50%	0.006000	0.009502	0.931034	1.000000	0.750000	0.750000
75%	0.006823	0.010010	0.955973	1.000000	0.750000	0.842857

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f1
max	0.007026	0.011031	0.965517	1.000000	1.000000	0.857143

6.6.2 Decision Tree Classifier

```
UNBAL_results[UNBAL_results["Model"] == "DecisionTreeClassifier"].describe()
```

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f1
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	0.002380	0.009863	0.948276	0.833333	0.800000	0.800476
std	0.000489	0.001093	0.052042	0.222222	0.258199	0.220497
min	0.001998	0.008030	0.862069	0.500000	0.250000	0.333333
25%	0.002000	0.009154	0.931034	0.666667	0.750000	0.800000
50%	0.002008	0.009987	0.965517	1.000000	0.875000	0.857143
75%	0.002939	0.010024	0.991379	1.000000	1.000000	0.964286
max	0.002993	0.011971	1.000000	1.000000	1.000000	1.000000

6.6.3 Support Vector Classifier

```
UNBAL_results[UNBAL_results["Model"] == "SVC"].describe()
```

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f1
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	0.004958	0.009803	0.958621	0.818810	0.950000	0.872172
std	0.001000	0.001035	0.042389	0.172879	0.105409	0.128790
min	0.003000	0.008013	0.896552	0.571429	0.750000	0.666667
25%	0.004334	0.009016	0.931034	0.687500	1.000000	0.762500
50%	0.005004	0.010028	0.965517	0.800000	1.000000	0.888889
75%	0.005853	0.010742	1.000000	1.000000	1.000000	1.000000
max	0.006014	0.011002	1.000000	1.000000	1.000000	1.000000

6.6.4 Gaussian Naive Bayes

```
UNBAL_results[UNBAL_results["Model"] == "GaussianNB"].describe()
```

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f1
count	10.000000	10.000000	10.000000	10.000000	10.0	10.000000
mean	0.003004	0.010806	0.739655	0.353948	1.0	0.521053
std	0.001412	0.001209	0.054213	0.049893	0.0	0.053877
min	0.001988	0.008996	0.655172	0.285714	1.0	0.444444
25%	0.002004	0.009988	0.695813	0.314103	1.0	0.477941
50%	0.002502	0.010298	0.741379	0.348485	1.0	0.516667
75%	0.003020	0.011987	0.778941	0.390909	1.0	0.561905
max	0.006004	0.012534	0.827586	0.444444	1.0	0.615385

6.6.5 K-Neighbors Classifier

```
UNBAL_results[UNBAL_results["Model"] == "KNeighborsClassifier"].describe()
```

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f1
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	0.002756	0.014556	0.955172	0.829286	0.925000	0.861724
std	0.000800	0.002012	0.046120	0.193911	0.120761	0.134958
min	0.001999	0.010986	0.896552	0.571429	0.750000	0.666667
25%	0.002006	0.013249	0.905172	0.637500	0.812500	0.732955
50%	0.002992	0.014780	0.965517	0.900000	1.000000	0.873016
75%	0.003000	0.015017	1.000000	1.000000	1.000000	1.000000
max	0.004547	0.018019	1.000000	1.000000	1.000000	1.000000

Based on the analysis of mean and standard deviation, the models that perform well are LogisticRegression, SVC, and KNeighborsClassifier. These models have relatively high mean accuracy, precision, recall, and F1 score, with low variability (standard deviation). On the other hand, DecisionTreeClassifier and GaussianNB have higher variability and lower mean values, indicating poorer performance.

In addition, when comparing the cross validation metrics, it is evident that LogisticRegression, SVC, and KNeighborsClassifier are good models due to their high mean values and low variability.

7 Oversampling

In this section, we will tackle the issue of class imbalance in the target variable by employing oversampling techniques. By engineering the data itself, we aim to improve the baseline metrics of our models while keeping the default hyperparameters. To achieve this, we will utilize the imblearn library and apply oversampling methods such as SMOTE and ADASYN. We will then conduct cross validation to evaluate the performance of our models in terms of accuracy, f1 score, precision, and recall.

7.1 Importing Libraries

To begin, we will import the necessary libraries for this section.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_validate

# Machine Learning Model Classes
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

# Pipelines and Transformers
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.base import ClassifierMixin

from imblearn.over_sampling import SMOTE, ADASYN
```

7.2 Loading and Splitting The Dataset

Next, we will load the dataset into a pandas data frame and split it into a feature matrix and target vector. We will then further split the data into a training set and a testing set. The training set will be used for training and cross validation, while the testing set will be used for model evaluation.

```
dataset_uri = "datasets/V1.csv"
dataset = pd.read_csv(dataset_uri, index_col=0)

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    stratify=y,
    random_state=22
)
```

7.3 Resampling Minority Class

To address the issue of class imbalance, we will utilize the SMOTE and ADASYN functions from the imblearn library. These techniques will help us oversample the minority class and create a more balanced dataset.

```
X_train_SMOTE, y_train_SMOTE = SMOTE().fit_resample(X, y)
X_train_ADASYN, y_train_ADASYN = ADASYN().fit_resample(X, y)
```

7.4 Cross Validation Metrics

7.4.1 Cross Validation Utility Class

We will use a utility class that conducts a 10-fold cross validation on the training set using five different machine learning models. This class will help us evaluate the performance of each model.

Next, we will use the utility class to build a metrics dataframe. This dataframe will contain the results of the 10-fold cross validation for each model, using both the original unbalanced data and the resampled data. We will focus on the f1 score, as it is the key metric highlighted in the project overview. Our goal is to achieve a minimum acceptable f1 score of 0.85.

```python
class ClassifierEvaluator():
    def __init__(self, estimators: list[ClassifierMixin]) -> None:
        self.estimators = estimators

    def cross_validate_metrics(self, X, y) -> None:
        scorings = ['accuracy', 'precision', 'recall', 'f1']

        for estimator in self.estimators:
            name = type(estimator).__name__

            pipeline = Pipeline(
                steps=[
                    ("std_scaler", StandardScaler()),
                    ("classifier", estimator)
                ]
            )

            metrics = pd.DataFrame(cross_validate(pipeline, X, y, cv=10, scoring=scorings)
            metrics_viz = sns.stripplot(metrics.iloc[:, 2:]).set(title=name)
            metrics_viz = sns.violinplot(metrics.iloc[:, 2:], alpha=0.1).set(title=name)
            plt.ylim([0,1.0])
            plt.show()

    def tabulate_cross_validation_metrics(self, X, y) -> None:
        scorings = ['accuracy', 'precision', 'recall', 'f1']

        df = pd.DataFrame()

        for estimator in self.estimators:
            name = type(estimator).__name__

            pipeline = Pipeline(
                steps=[
                    ("std_scaler", StandardScaler()),
                    ("classifier", estimator)
                ]
            )
            metrics = pd.DataFrame(cross_validate(pipeline, X, y, cv=10, scoring=scorings)
            metrics["Model"] = name
            df = pd.concat([df, metrics])

        return df

# Instanciate the ClassifierEvaluator
evaluator = ClassifierEvaluator(
    estimators=[
        LogisticRegression(),
        DecisionTreeClassifier(),
        SVC(),
        GaussianNB(),
        KNeighborsClassifier()
    ],
)
```

Based on the metrics, we found that only the KNeighborsClassifier and SVC models surpassed the baseline f1 score of 0.85 by a small margin. However, when using the resampled data, the average f1 score significantly increased. Additionally, the ADASYN data consistently outperformed the SMOTE data.

```
UNBAL_results = evaluator.tabulate_cross_validation_metrics(X_train, y_train)
SMOTE_results = evaluator.tabulate_cross_validation_metrics(X_train_SMOTE, y_train_SMOTE)
ADASYN_results = evaluator.tabulate_cross_validation_metrics(X_train_ADASYN, y_train_ADASY

UNBAL_results["Data"] = "UNBAL"
SMOTE_results["Data"] = "SMOTE"
ADASYN_results["Data"] = "ADASYN"

combined_results = pd.concat([UNBAL_results, SMOTE_results, ADASYN_results])
metrics = combined_results.iloc[:, 2:]
```

7.4.2 Mean Cross Validation Scores

```
metrics.groupby(["Model", "Data"]).mean()
```

Model	Data	test_accuracy	test_precision	test_recall	test_f1
DecisionTreeClassifier	ADASYN	0.925038	0.917092	0.968750	0.935205
	SMOTE	0.916129	0.912282	0.961290	0.927859
	UNBAL	0.944828	0.808333	0.800000	0.789762
GaussianNB	ADASYN	0.922811	0.888134	1.000000	0.935497
	SMOTE	0.854839	0.825370	1.000000	0.892145
	UNBAL	0.739655	0.353948	1.000000	0.521053
KNeighborsClassifier	ADASYN	0.950256	0.928059	0.996875	0.957550
	SMOTE	0.948387	0.926900	0.993548	0.955498
	UNBAL	0.955172	0.829286	0.925000	0.861724
LogisticRegression	ADASYN	0.920097	0.900423	1.000000	0.939324
	SMOTE	0.917742	0.898640	0.996774	0.937081
	UNBAL	0.941133	0.891667	0.700000	0.762143
SVC	ADASYN	0.951843	0.928237	1.000000	0.959135
	SMOTE	0.950000	0.927089	0.996774	0.957135
	UNBAL	0.958621	0.818810	0.950000	0.872172

Based on above metrics, only the KNeighborsClassifier and SVC models surpassed the 0.85 baseline f1 score by a small amount. However upon using the resampled data, the average f1 score increases significantly. It was also observed that using the ADASYN data had higher scores than the SMOTE data.

Overall, the models ability to predict occurrence of welding cracks significantly increased during the use of resampling techniques.

7.4.3 Standard Deviation Cross Validation Scores

```
metrics.groupby(["Model", "Data"]).std()
```

Model	Data	test_accuracy	test_precision	test_recall	test_f1
DecisionTreeClassifier	ADASYN	0.106008	0.134548	0.065881	0.082356
	SMOTE	0.120411	0.145029	0.071000	0.091223
	UNBAL	0.051915	0.215345	0.258199	0.220040
GaussianNB	ADASYN	0.102475	0.135515	0.000000	0.081468
	SMOTE	0.188709	0.193569	0.000000	0.127962
	UNBAL	0.054213	0.049893	0.000000	0.053877
KNeighborsClassifier	ADASYN	0.083646	0.115139	0.009882	0.067886

Model	Data	test_accuracy	test_precision	test_recall	test_f1
	SMOTE	0.085279	0.116162	0.013601	0.069881
	UNBAL	0.046120	0.193911	0.120761	0.134958
	ADASYN	0.146795	0.166302	0.000000	0.105294
LogisticRegression	SMOTE	0.152910	0.170127	0.010201	0.110692
	UNBAL	0.016264	0.141912	0.158114	0.079090
	ADASYN	0.083841	0.115160	0.000000	0.067975
SVC	SMOTE	0.085499	0.116182	0.010201	0.069992
	UNBAL	0.042389	0.172879	0.105409	0.128790

Analyzing the spread of values, we observed that the variability of f1 scores for the SVC, KNeighborsClassifier, and DecisionTree models decreased after using the oversampled data. On the other hand, the variability increased for the LogisticRegression and GaussianNB models.

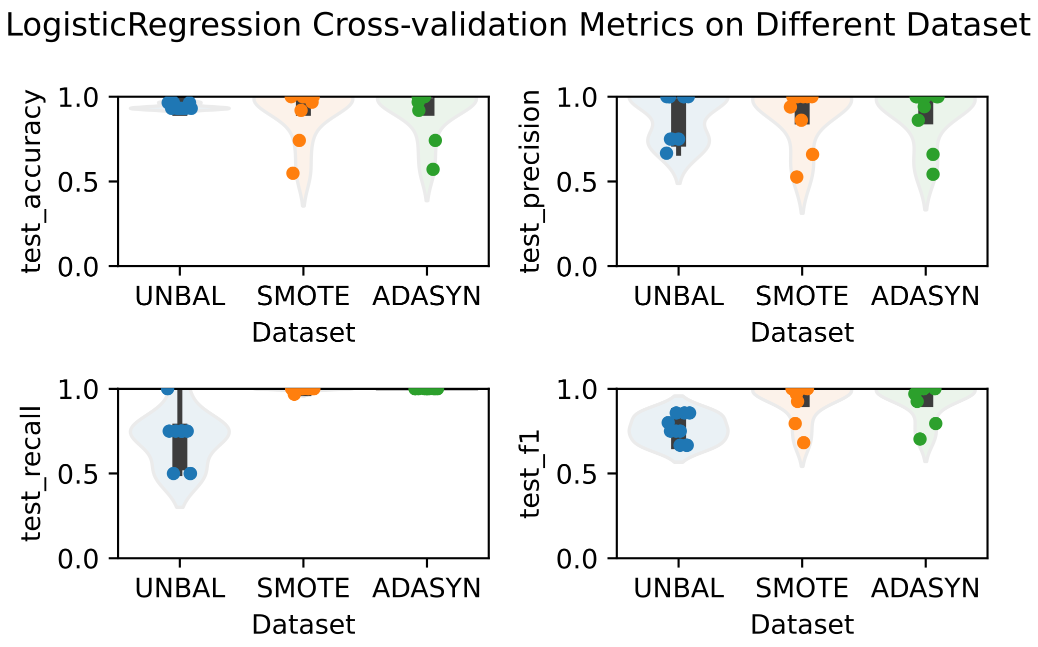
7.4.4 Side-by-side Visualization

To provide a visual representation of the metrics, we created a utility function that plots each metric for each model and training set used.

```python
def compare_metrics(model_name):
    metric_names = metrics.columns
    for i in range(4):
        plt.subplot(2,2,i+1)
        plt.tight_layout()
        sns.stripplot(
            data=metrics[(metrics["Model"] == model_name)],
            x="Data",
            y=metric_names[i],
            hue="Data"
        )
        sns.violinplot(
            data=metrics[(metrics["Model"] == model_name)],
            x="Data",
            y=metric_names[i],
            hue="Data",
            alpha=0.1
        )
        plt.suptitle(f"{model_name} Cross-validation Metrics on Different Dataset")
        plt.xlabel("Dataset")
        plt.ylim([0,1.0])
```

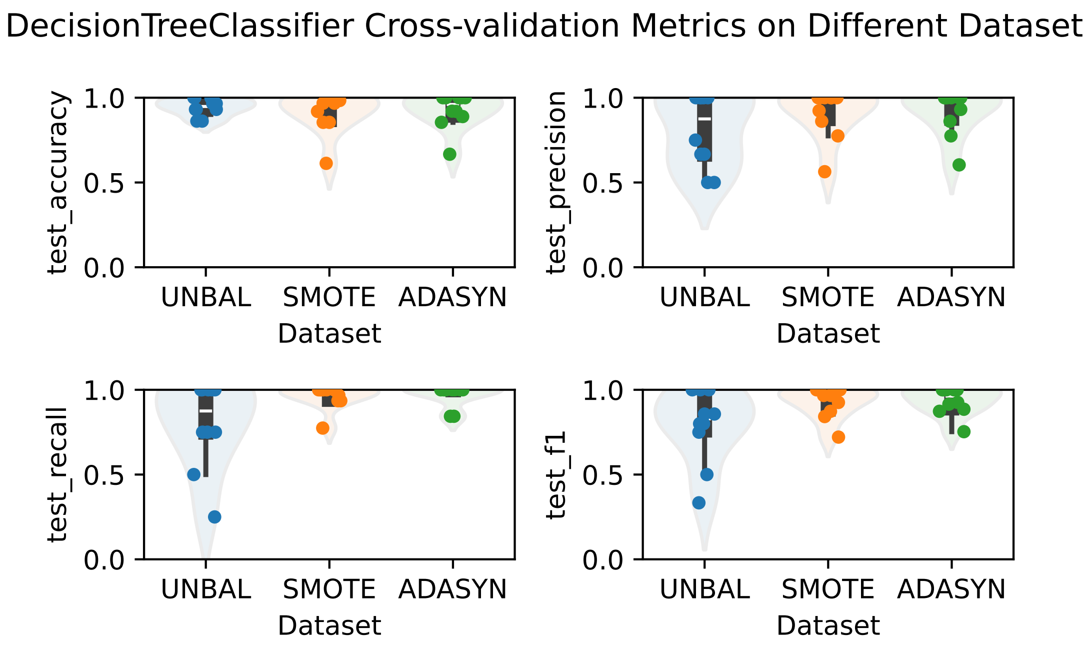
7.4.5 Logistic Regression

```
compare_metrics("LogisticRegression")
```



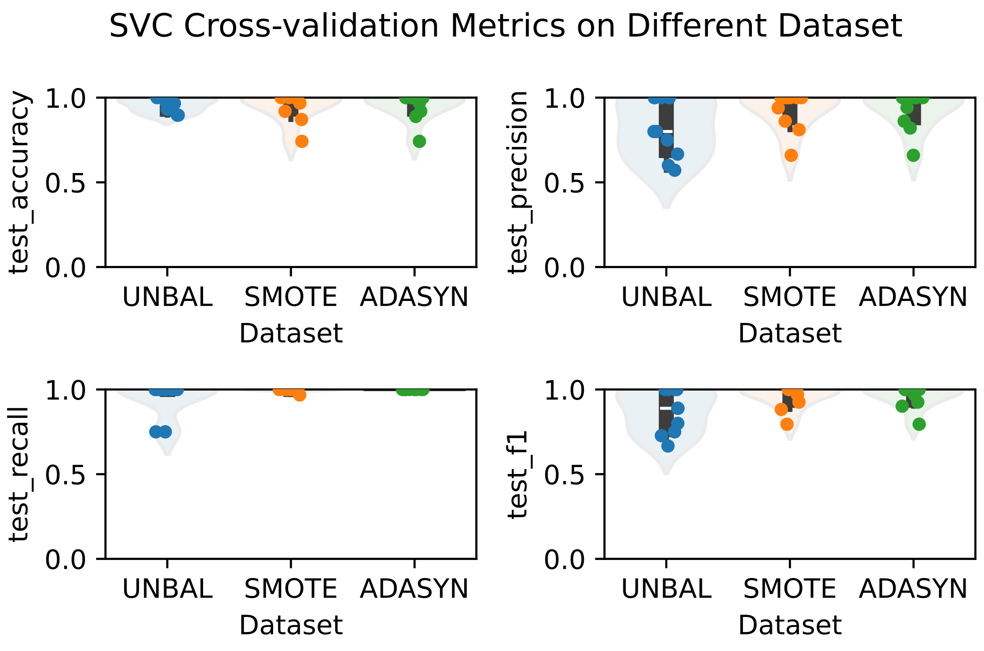
7.4.6 Decision Tree Classifier

```
compare_metrics("DecisionTreeClassifier")
```



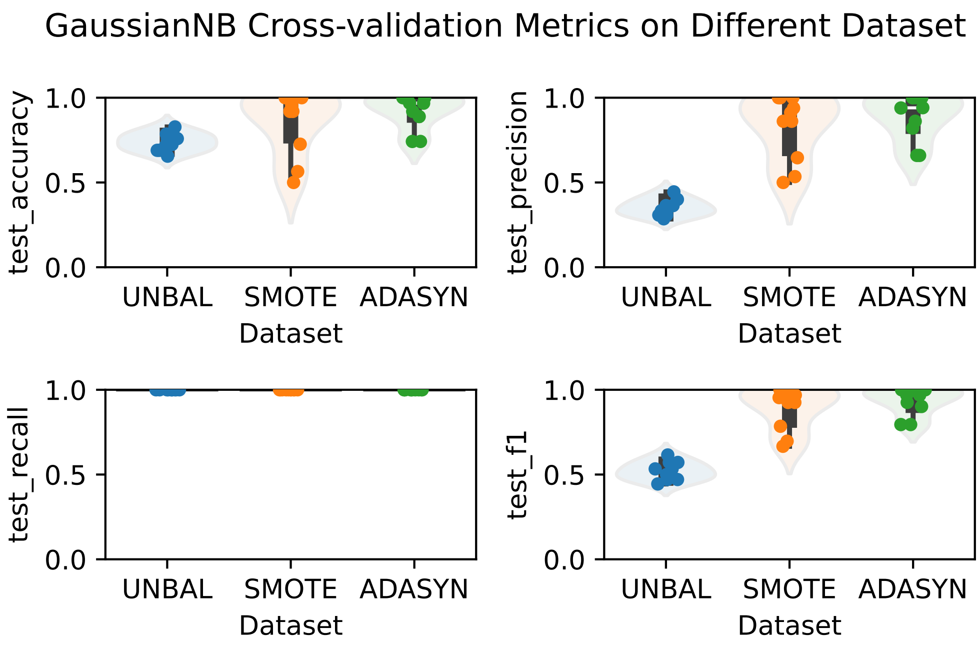
7.4.7 Support Vector Classifier

```
compare_metrics("SVC")
```



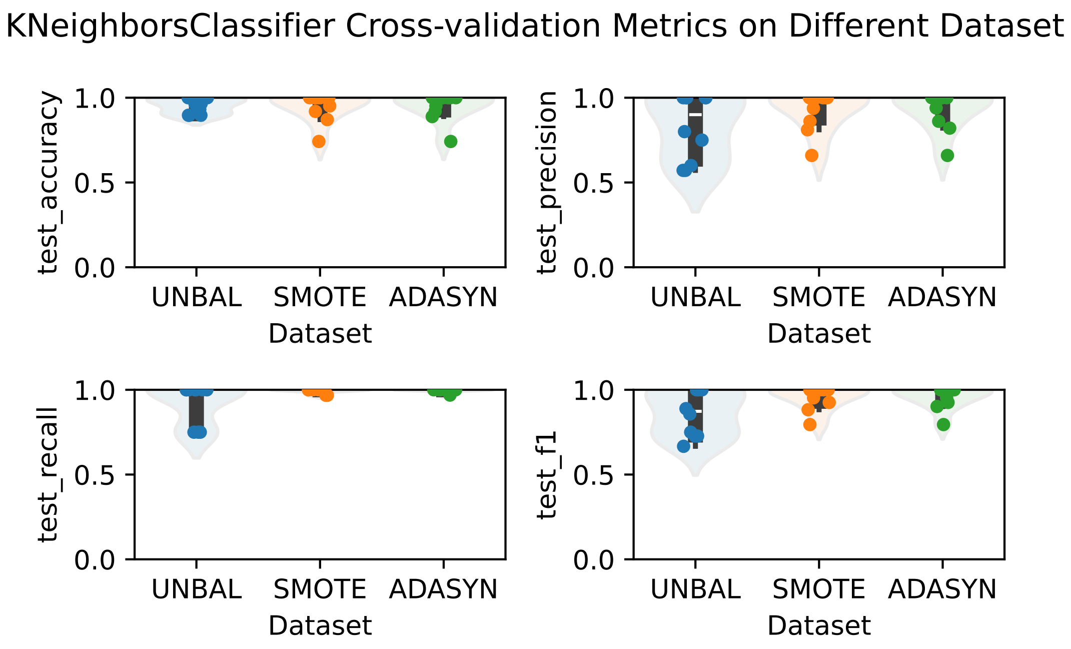
7.4.8 Gaussian Naive Bayes

```
compare_metrics("GaussianNB")
```



7.4.9 K-Neighbors Classifier

```
compare_metrics("KNeighborsClassifier")
```



7.5 Conclusion

In conclusion, addressing the issue of class imbalance through oversampling techniques resulted in a significant performance boost for our models. The use of the ADASYN resampling technique yielded higher average f1 scores compared to SMOTE. This improvement was achieved without tuning the hyperparameters of the models. Moving forward, we will explore ensembling techniques in the next section to further enhance the predictability of our models.

8 Summary

In summary, this book has no content whatsoever.

References

Gao, Zhongmei, Yuye Yang, Lei Wang, Bin Zhou, and Fei Yan. 2022. "Formation Mechanism and Control of Solidification Cracking in Laser-Welded Joints of Steel/Copper Dissimilar Metals." *Metals* 12 (7): 1147. https://doi.org/10.3390/met12071147.

Rinne, Jonas. 2021. "Screening Datasets for Laser Welded Steel-Copper Lap Joints." Mendeley Data. https://doi.org/10.17632/2s5m3crbkd.2.

Rinne, Jonas, Sarah Nothdurft, Jörg Hermsdorf, Stefan Kaierle, and Ludger Overmeyer. 2021. "Investigations on Laser Welding of Dissimilar Joints of Stainless Steel and Copper for Hot Crack Prevention." *Journal of Laser Applications* 33: 042042. https://doi.org/10.2351/7.0000489.